UNIVERSITY OF CALIFORNIA
Santa Barbara

Scalable Analytics Systems for Multi-Tier IoT Deployments with Applications in Agriculture

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Nevena Golubović

Committee in Charge:

Professor Chandra Krintz, Co-Chair

Professor Rich Wolski, Co-Chair

Professor Amr El Abbadi

December 2019

The Dissertation of
Nevena Golubović is approved:

Professor Amr El Abbadi

Professor Rich Wolski, Committee Co-Chairperson

Professor Chandra Krintz, Committee Co-Chairperson

December 2019

Scalable Analytics Systems for Multi-Tier IoT Deployments with Applications in Agriculture

by

Nevena Golubović

To my parents, brother, and the memory of my grandmother.

Acknowledgements

One of the first classes I attended at UCSB was Chandra Krintz's Cloud Computing class. We were discussing the newest research papers where Chandra would lead the discussion and we would raise a hand to join. It felt like merging on a high way from a very slow lane to the left much faster one. Her enthusiasm for the topic was contagious. She would demystify cryptic research papers into a set of steps you could take to try out this newly proposed technology. Rich Wolski's Cloud Computing and Operating Systems classes were even more challenging. It wasn't up to you to give the signal and merge, you were told when to merge and you better do it right. Each class would end in a way a theater performance does. Both of them were so enthusiastic when teaching and I wanted to learn how to talk about computer science topics like they do. I soon joined the lab planning to extend the knowledge about topics they presented in their classes.

This time came with Major Area Exam when learning curve was steep not only in the topics we covered but also in the way we discussed relevant papers. I had no idea how many questions could be lurking in a single ten pages paper. They taught me how to reason about the relevance of research and how to evaluate and communicate the work of others. Chandra and Rich spent countless hours discussing many research ideas with me, helping me understand the background and supporting me to find my own topics to work on. They taught me how to approach a new research topic, write and talk about it. Merging left became easier over time though still challenging.

I am deeply grateful to my advisors for giving me the opportunity to learn from them and for spending countless hours teaching me about research and life. Thank you Amr El Abbadi for teaching me about distributed systems and for joining my PhD committee and offering support and insightful conversations about research. Thank you, professors, from whom I had the opportunity to learn: Ambuj, Ben, Divy, Elizabeth, John, Jon, Linda, Pete, Stefano, Tao, Xifeng, and Yufei.

Life is a continuous learning process and one has never enough mentors and co-workers on this journey. I thank my mentors and colleagues during internships at AppFolio and Google: Andrew, Brynjar, Benji, Harish, John, Kurtis, Rakesh, Siddhartha, and Subhadip. It was a pleasure to work in such a supportive and innovative environments. Big thanks go to many collaborators from UC Davis, PowWoW, CalPoly. Univ. of Landshut, CSUF, Cornell University, Univ. of Buffalo, and a tremendous help from patient IT and CS office at UCSB: Adam, Amy, Andreas, Balaji, Benji, Bo, Bruce, Caleb, Fletcher, Greta, Jennifer, Jill, Jim, Karen, Markus, Nikhil, Olivier, Senal, and Sheryl. Thanks go to my professors from Univ. of Belgrade who strengthened my scientific curiosity and all of my unmentioned teachers and mentors.

Visiting United States and working in Silicon Valley gave me an opportunity to meet people who became close friends, mentors, and an extremely supportive community. They offered countless hours of encouragement and advising, sharing some low moments and celebrating the high ones. Thank you Aleksa, Ana, Ankur, Anton, Filip, Hyungwoon, Jelena, KD, Lakshmi, Lazar, Marija, Marissa, Metrri, Milena, Milutin, Neda, Nikola, Pedja, Rich, Sofija, Sri, Shreedhar, Tea, Thomas, Tom, Wendy, Zoran, and Zorica. It helps tremendously to know that others share the path you are taking and having your labmates to discuss research and daily life is irreplaceable. Thank you, Alex, Andy, Angad, Benji, Carly, Chris, Dani, Dev, Emre, Fatih, Gareth, Hiranya, John, Kevin, Kyle, Kyle, Mengyuan, Michael, Nazmus, Paz, Stratos, and Wei-Tsung for being honest friends and inspiring collaborators. UCSB made me feel welcome and gave me a chance to meet wonderful colleagues coming from all over the world: Adam, Ahmed, Aiwen, Arlei, Burak, Can, Cetin, Deeksha, Divya, Elizabeth, Esther, Faisal, Fracois, George, Goksu, Haraldur, Ivan, Karim, Kevin, Krithika, Leah, Mai, Marija, Martina, Meni, Michael, Morgan, Neeraj, Omid, Petko, Rachel, Sherri, Shivapriya, Sujaya, Theodore, Vaibhav, Veronika, Victor, Xiaofei, Yanick. My family away from home was a Toastmasters group that helped me learn how to express myself and offered the most comfortable environment to grow. Thank you Albert, Bob, Eve, David, Gabe, Gene, Grace, Herb, Jatzibe, Marie, Mike, Nena, Ray, Rhonda, Ron, and

Tracy. Time zone difference did not prevent my closest friends Gaga, Irena, Ivana, Ivan, Jeka, and Natasa from joining some healing phone calls on weekends.

Thank you, Frankel family and Google, for reminding me of how important financial support is. I hope to one day be able to extend it to others in the same generous manner you extended it to me.

Finally, I want to thank my family: my mom, who taught me the importance of organization and scheduling and how to reason beyond just natural and logical consequences; my dad, who inspired me to develop a curiosity for philosophy and science and love for mathematics; and my brother whose wit can lighten the darkest of days. Thank you for your love and selfless support. Thank you to my extended family for their humor, encouragement, and love.

This work is supported in part by National Science Foundation CNS-1703560, OAC-1541215, CCF-1539586, ACI-1541215 and the California Energy Commission (PON-14-304).

Curriculum Vitæ

Nevena Golubović

Education

2019	Doctor of Philosophy in Computer Science, University of California, Santa Barbara
2019	Master of Science in Computer Science, University of California, Santa Barbara
2009	Graduate Mathematician, Mathematics and Computer Science, Faculty of Mathematics, University of Belgrade

Selected Professional Experience

2013 – 2019	Research Assistant, Univ. of California, Santa Barbara, CA
2015, 2017	Software Engineer Intern, Google, Mountain View, CA
2014	Software Engineer Intern, AppFolio, Santa Barbara, CA
2011-2013	Software Engineer, Sony Mobile, Redwood City, CA
2010-2011	Software Engineer, PSTech, Belgrade, Serbia

Selected Awards

2019	Best Student Paper Award, 2019 IEEE International Congress on Internet of Things
2018	IEE Frenkel Foundation Fellowship
2014	The Google Anita Borg Memorial Scholarship

Publications

- N. Golubovic, R. Wolski, C. Krintz and M. Mock; Improving the Accuracy of Outdoor Temperature Prediction by IoT Devices, IEEE Conference on IoT (ICIOT), July 2019 Golubovic et al. (2019)
- N. Golubovic, C. Krintz, R. Wolski, B. Sethuramasamyraja, and B. Liu; A Scalable System for Executing and Scoring K-Means Clustering Techniques and Its Impact on Applications in Agriculture, International Journal of Big Data Intelligence, Vol. 6, Nos. 3/4, 2019 Golubovic et al. (2018)
- C. Krintz, R. Wolski, N. Golubovic, and F. Bakir; Estimating Outdoor Temperature from CPU Temperature for IoT Applications in Agriculture, International Conference on the Internet of Things (IoT), Oct 2018 Krintz et al. (2018*a*)

- N. Golubovic, A. Gill, C. Krintz, and R. Wolski; CENTAURUS: A Cloud Service for K-means Clustering, IEEE DataCom, November, 2017 Golubovic et al. (2017)
- A. Rosales Elias, N. Golubovic, R. Wolski, and C. Krintz; Where's The Bear? – Automating Wildlife Image Processing Using IoT and Edge Cloud Systems, ACM Conference on IoT Design and Implementation, April, 2017 Elias et al. (2017)
- N. Golubovic, C. Krintz, R. Wolski, S. Lafia, T. Hervey, and W. Kuhn; Extracting Spatial Information from Social Media in Support of Agricultural Management Decisions, ACM SIGSPATIAL Workshop on Geographic Information Retrieval, October, 2016 Golubovic et al. (2016)
- C. Krintz, R. Wolski, N. Golubovic, B. Lampel, V. Kulkarni, B. Sethuramasamyraja, B. Roberts, and B. Liu; SmartFarm: Improving Agriculture Sustainability Using Modern Information Technology, KDD 2016 Workshop on Data Science for Food, Energy, and Water, August, 2016 Krintz et al. (2016)

Please Note: Text and figures from these papers are used and appear in this dissertation.

Abstract

Scalable Analytics Systems for Multi-Tier IoT Deployments with Applications in Agriculture

Nevena Golubović

Recent technological advances in environmental and personal sensing, monitoring and data analytics are fueling remarkable innovation in data-driven actuation, decision support, and adaptive control that is based on the "Internet of Things" (IoT). However, to become truly transformative, IoT must exploit a vastly heterogeneous combinations of compute, storage, and networking capabilities provisioned at multiple scales, from low-cost, resource-restricted devices to the public clouds. Low-latency applications and continuous telemetry from a devices in our environment require services to be distributed to "the edge" of the network while, at the same time, both security and programmer-productivity requirements require uniform, efficient, and end-to-end systems.

With this dissertation, we investigate the design and implementation of a scalable, end-to-end system for data-driven IoT applications, which spans IoT tiers – sensors, edge, and cloud. Moreover, we do so in a problem-driven fashion and target the domain of agriculture, specializing the system for data analytics and machine learning techniques that are applicable to precision farming. Our work is novel in that our system integrates popular cloud services and machine learning technologies using open source and provides the scalable building blocks for common analysis tasks, e.g. clustering and regression, in a way that can be tailored to specific problems that growers and farm consultants face. In addition, the system automates placement of analytics deployments across edge and cloud tiers. For the sensing tier, we develop a novel approach that extends the capability of sensor platforms by "synthesizing" information from multiple, other sensors. The result, we believe, is a holistic, easy to use system for data ingress, analysis, and visualization, that integrates new insights in sensing and distributed and scalable systems, and that is applicable to a range of agricultural settings, applications, and low latency, sustainable solutions.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Recent technological advances have driven down the cost of off-the-shelf compute power, storage, and network bandwidth, and have simplified large scale resource use (i.e. cloud computing). As a result, applications have become data-centric and the resulting data resources and products have grown explosively in both number and size. Moreover, the environment in which we live has become increasingly accessible via sensor, observation, and monitoring systems that produce and collect vast amounts of data from ordinary objects, which comprise the “Internet of Things” (IoT). IoT applications attempt to extract actionable insights from this data to drive innovation, forge new industries, and facilitate new scientific discovery across society and our economy, e.g. in health care, manufacturing, education, transportation, the military, agriculture, the energy sector, and manufacturing, among others.

In order to provide actionable insights in time, IoT developers rely on increasing connectivity and wast amount of data being uploaded by IoT devices in real time, making it available for data analytics services from a cost-effective cloud. At the same time, a dynamically changing and heterogeneous set of IoT devices with their communication protocols, and power/battery requirements makes software integration, deployment, and maintenance more challenging given the scale and energy requirements.

Moreover, because IoT applications often exhibit significantly better locality than their e-commerce or social networking counterparts, they leverage "multi-tier" execution environments to achieve low latency response. Specifically, devices increasingly communicate with "edge computing" infrastructure that is "nearby" in terms of network transfer latency. The edge tier communicates with public clouds for data and computation aggregation. Between these two tiers, Content Distribution Networks (CDNs), regional clouds, private clouds, etc., provide computing, networking, and storage services for IoT applications. This tiered hierarchy facilitates efficient and cost-effective network use, localized decision making, controlled data sharing, and real-time and low-latency response for edge devices and services Elias et al. (2017), Krintz et al. (2018*b*, 2016), Foukas et al. (2017), Satyanarayanan (2013)

Particularly exciting is the potential for multi-tier IoT systems to have large-scale societal impact by providing the next generation of digital technologies for optimizing agricultural and food security outcomes. Increasing yields sustainably while using fewer resources (i.e. “precision” agriculture) is essential to ensure that food production is not out-paced by population growth. However, smallholder agriculturists (as opposed to industrial-scale farming concerns) and their rural communities are strikingly under-served by technology, with few solutions becoming commonplace USDA (2017). Thus, new advances in IoT systems research are critical for IoT in general, and agriculture in particular, if we are to achieve the transformational societal impact that IoT promises. The goal of our work is to provide such advances using a problem-driven approach that is motivated by the needs of growers and agricultural operations.

Current precision agriculture technologies fall short in three key ways that have severely limited their impact and widespread use: (i) they fail to provide growers with easy management and control over their data, (ii) they lock growers into proprietary, closed, inflexible, and potentially costly technologies in order to extract actionable insights from their farm and sensor data, and (iii) few systems facilitate cross-vendor sensor and data integration. In addition, many farms have slow, intermittent, or no connectivity to the cloud, precluding the use of cloud-only solutions. As such, growers and farm consultants require new technological advances that provide them with automated farm data management, data-driven recommendations and decision support, and low-cost, integrated sensing in a unified, easy to use platform that works with or without Internet connectivity.

Toward this end and given the limitations in the state of the art in IoT analytics systems for agricultural settings, we ask the following Thesis question:

Can we build an open, scalable, end-to-end-system that provides low latency, reliable, and actionable analytics, and automated error analysis from sensor data, that is useful for addressing analytics challenges that agricultural experts face?

To answer this question, we investigate the design, implementation, and deployment of an end-to-end, multi-tier system for IoT applications in agriculture. Our system, called HYPATIA bridges the gap between sensors and cloud to provide automatic, low-latency deployment of IoT applications across edge and cloud tiers. In addition, it can operate at the edge (on-farm) when the connection to the cloud is intermittent or non-existent. HYPATIA automates ingress of sensors and data sets, exports services for common machine learning and data analytics capabilities, enables users to tailor the system to their analytics and visualization preferences, and integrates scoring metrics which it uses to automate model selection and to provide decision support and a recommendation system for users.

To enable this, we investigate three novel advances that target key challenges from the agriculture sector. First, we investigate the necessary cloud service support required for clustering multivariate and highly correlated data collected (common in a wide variety of IoT analytics). We use the system to study farm soil electrical conductivity (EC). EC is fast and inexpensive to repeatedly measure (a Veris device is pulled across a field behind a tractor Ver (2019), Lund et al. (1999)), and can be used to identify management zone boundaries Moral et al. (2010), Fortes et al. (2015), Corwin & Lesch (2003) and to estimate the number of different soil metrics including salinity, water holding capacity, and texture Bell et al. (1995), Kitchen et al. (2006), Adamchuk et al. (2004). The service simplifies the selection of analytics parameters from which growers choose, and provides a recommendation of the best variant (management zone boundary) that can be easily understood by experts and novices alike. Moreover, the service enables users to visualize the data and results in multiple ways.

Second, we investigate how to extend the capabilities of on-farm sensing. To enable this we use integrated, on-farm sensors to `estimate` the measurements of other sensors, a technique that we call *sensor synthesis*. Since the number of sensors per device is generally bounded by design constraints (e.g. space or power limitations), sensor synthesis makes it possible to free up resources in IoT devices for other sensors, particularly those that are less amenable to synthesis, and to reduce the monetary cost of sensing. We apply sensor synthesis to measure micro-climates across a farm (e.g. temperature and humidity variations due to topography and physical objects). Microclimate data is useful for more precise application of water (irrigation) and frost control. To enable this, we estimate outdoor temperature using the processor (CPU) temperature of simple and low-cost single board computers deployed in outdoor settings. We combine data smoothing techniques and multiple linear regression methods, which we apply to nearby SBC processor and weather station data. We empirically evaluate this approach using a wide range of experiments and we investigate its accuracy with and without the computational load on the SBCs. We find that we can accurately estimate microclimate temperature from combinations of nearby devices on-farm, thereby reducing the number of temperature sensors required to capture temperature variation across a field.

Finally, we develop a new scheduling system that automates distributed deployment of data analytics applications across IoT tiers (edge and cloud). The scheduler accounts for both computation and communication of the applications and automatically splits the execution between edge systems and cloud computing systems, to minimize time to completion and to prioritize edge use. The scheduler uses execution histories to estimate time to completion and uses these predictions to automatically place application "jobs". We find that by doing so, the scheduler is able to significantly reduce time to completion over always using just the edge or just the cloud. In addition, the scheduler simplifies IoT deployment by automatically executing and autoscaling jobs (using multiple virtual servers) on any system on which it is deployed (local or remote).

We combine each of these advances into a single scalable end-to-end system called HYPATIA available through an intuitive user interface of a web browser. The result is an open-source, end-to-end system that enables users to collect data from multiple sensors sources, including user's files, web API's, and other publicly available datasets. HYPATIA provides abstractions for data management algorithms and implements multiple variants of the two frequently used ones: clustering and linear regression, allowing other algorithms to be easily "plugged in" following the same abstractions. For the given algorithms, HYPATIA provides scoring, and model selection. To facilitate model selection and to better understand data coming from various sources, HYPATIA provides different visualization solutions. HYPATIA implements the scheduler described above, which minimizes time to completion of algorithms while considering the type of the algorithm, data transfer and computation time requirements, and whether most of the time will be spent on model training or its use for inference/analysis. We design HYPATIA as a distributed system that executes on any virtualized system (e.g. edge, private, and public clouds) over which IoT applications can be deployed without modification (and without knowledge about or expertise with the underlying systems). We evaluate Hypatia using a number of different IoT analytics applications and show that it enables low latency, reliability, machine learning model selection, error analysis, data visualization, and scheduling, in a unified scalable system.

In the Chapters that follow, we provide background on existing technologies and research that is relevant to our work (Chapter 2). In Chapter 3, we present our advances for automating clustering and the efficacy of its use on correlated data and soil EC analysis. In Chapter 4, we present “sensor synthesis" and show how it can be used to predict outdoor temperature from the CPU temperature of SBCs. We further extend the system to ingress multiple sensors and use multiple linear regression providing scores for each model for model selection. Chapter 5 details the overall Hypatia system which we integrate with each of our advances including sensor data ingress (synthesized and actual), statistical clustering and scoring, multiple linear regression and model selection, and scalable schedule and automated deployment IoT applications across edge and cloud resources. Finally, in Chapter 6, we present our conclusions and plans for future work.

Chapter 2

Background and Related Work

Today IoT developers increasingly combine IoT devices with the scale, data and analytics services, and the cost-effectiveness of the cloud. However, at present, the heterogeneous (in terms of hardware, software, and APIs), asynchronous, highly scalable, dynamically changing, and geographically distributed nature of IoT-cloud applications, makes their infrastructure complex and difficult to provision, program, and optimize for high performance, energy efficiency, and scale.

In an attempt to overcome this challenge, cloud providers are investing heavily in new cloud services. Unfortunately, while effective as platforms for autoscaling web services, extant cloud offerings have not been able to ameliorate the complexities facing reliable and pervasive IoT application deployment, which must be overcome if we are to achieve the transformational societal impact that IoT promises. First, the volumes and velocity of data produced by IoT systems Int (2019*b*) has forced a movement from a centralized model of computing to an "edge" connected model for IoT, in which computation and analysis must be performed *near* where data is generated. The centralized (*cloud-direct*) approach imposes significant request latency and power consumption on remote devices at the network "edge" – prohibiting real-time, data-driven response. Instead, co-location of processing infrastructure and IoT devices significantly reduces the latency between data acquisition and device actuation enables the extension of device capability via local offloading, and alleviates the cost, power consumption, and congestion of network use of the cloud-direct model Floyer (2015), Bonomi et al. (2012), Satyanarayanan et al. (2009), Satyanarayanan (2013), Verbelen et al. (2012).

Some cloud vendors offer restricted versions of cloud services for edge devices AWS IoT Core (2019), AWS IoT Greengrass (2019), Azure IoT Hub (2019), Azure IoT Edge (2019), Cloud IoT Core (2019), Edge TPU (2019), Bosch IoT Suite (2019), General Electric IoT (2019). However, these solutions are not portable across cloud vendors (e.g. Amazon and Azure public clouds), they do not allow arbitrary computations and data analytics at the edge, they are hard to use due to complex configuration, and not being open source precludes extension and reproducibility.

Despite the many advances in cloud services and cloud-based data analytics, few advances have made their way to the agriculture community. Such techniques, however, are critical for lowering the cost of farm operations, reducing labor needs via automation, and increasing yields sustainably. However, smallholder agriculturists (as opposed to industrial-scale farming concerns) and their rural communities are strikingly under-served by technology, with few solutions becoming commonplace USDA (2017).

In this dissertation, we focus on the scalable analytics building blocks that are key for a wide range of applications. We then tailor the system and solutions to agricultural problems and settings so that they may provide growers with decision support as well as data-driven actuation and control for precision agriculture. Precision agriculture (ag) Committee on Assessing Crop Yield: Site-Specific Farming, Information Systems, and Research Opportunities, Board on Agriculture, National Research Council (1997) is a set of farm management techniques that use data from environmental sensors, historical records, and models, and farming operations, to provide decision support to growers and farm consultants. Precision farming integrates cyberinfrastructure and computational data analysis to overcome the challenges associated with extracting useful information and actionable insights from the vast amount of information that surrounds the crop life cycle. Precision ag attempts to help growers answer key questions about irrigation and drainage, plant disease, insect and pest control, fertilization, crop rotation, and soil health, weather protection, and crop production. Existing precision ag solutions include sensor-software systems for irrigation, mapping, and image capture/processing (including via unmanned aerial vehicles (UAVs), intelligent implements (planters, harvesters, steering systems), and more recently, public cloud software-as-a-service (SaaS) solutions that provide visualization and analysis of farm data over time OnFarm (2019), Climate Corporation (A Monsanto Company) (2014), MyAgCentral (2014), gThrive (2014), WatrHub (2014), PowWow (2019).

Current precision ag technologies fall short in three key ways that have severely limited their impact and widespread use: First, they fail to provide growers with control over the privacy of their data and second, they lock growers into proprietary, closed, inflexible, and potentially costly technologies and methodologies. In terms of data privacy, extant solutions require that farmers relinquish control over and ownership of their most valuable asset: their data. Farm data reveals private and personal information about grower practices, crop input (chemicals, fertilizers, pesticides), and farm implement use, purchasing and sales details, water use, disease problems, etc., that define a grower's business and competitiveness. Revealing such information to vendors in exchange for the ability to visualize it puts farmers at significant risk Federation (2014), Russo (2013), Vogt (2014).

The second limitation of extant precision ag solutions is "lock-in". Lock-in is a well-known business strategy in which vendors seek to create *barriers to exit* for their customers as a way of ensuring revenue from continued use, new or related products, or add-ons in the future. In the precision ag sector, this manifests as proprietary, closed, and fragmented solutions that preclude advances in sustainable agriculture science and engineering by anyone other than the companies themselves. Lock-in also manifests as a lack of support for cross-vendor technologies, including observation and sensing devices, farm implements, and data management and analysis applications. Since farmers face many challenges switching vendors once they choose one, the one they choose can charge fees for training, customizations, add-ons, and use of their online resources without limit because of the lack of competition.

The third limitation is that most precision ag solutions today employ the centralized (*cloud-direct*) approach described above. As solutions become increasingly on-line (with the move to SaaS), the lock-in also requires that farmers upload all of their data to the cloud giving vendor full control and access, and leaving growers without recourse when vendors go out of business Rodrigues (2013). In addition to these risks, such network communication of potentially terabytes of image and sensor data is expensive and time consuming for many because of poor network connectivity and costly data rates that are typical of rural areas. Finally, many of these technologies impose high premiums and yearly subscriptions ArcGIS (2019).

The goal of our work is to address these limitations and to provide such a scalable, data analytics platform that facilitates open and scalable precision ag advances. To enable this, we leverage recent advances in Internet of Things (IoT), cloud computing, and data analytics and extend them them to contribute new research that defines a software architecture that tailors each to agricultural settings, applications, and sustainability science. These constituent technologies cannot be used off-the-shelf however because they require significant expertise and staffing to setup, manage, and maintain – which are show stoppers for today's growers. We attempt to overcome these challenges with a comprehensive, end-to-end system for scalable agriculture analytics that is open source and that can run anywhere (e.g. edge, public, and private clouds), precluding lock-in. To enable this, we contribute new advances in scalable analytics, low-cost sensing, easy to use data visualization, data-driven decision support, and automatic edge-cloud scheduling, all within a single, unified distributed platform. In the next chapter, we begin by focusing on an important analytics building block (statistical clustering) and tailoring its use for farm management zone identification using soil electrical conductivity data.

Chapter 3

K-Means Clustering and Its Use for Agriculture Analytics

Statistical clustering, also known as a separation of measurements into related groups, is a key requirement for solving many analytics problems. Lloyd's algorithm Lloyd (1982), commonly called *k-means*, is one of the most widely used approaches Duda et al. (1973). K-means is an unsupervised learning algorithm, requiring no training or labeling, that partitions data into K clusters, based on their "distance" from K centers in a multi-dimensional space. Its basic form is simple to implement and has become an indispensable component of pattern recognition, data mining, image processing, information retrieval, and recommendation applications across fields ranging from marketing and advertising to astronomy and agriculture.

While conceptually simple, there are a myriad of k-means algorithm variants based on how distances are calculated in the problem space. Some k-means implementations also require “hyper parameters” that control for the amount of statistical variation in clustering solutions. Identifying which algorithm variant and set of implementation parameters to use in a given analytics setting is often challenging and error prone for novices and experts alike.

In this chapter, we present Centaurus as an approach to simplifying the application of k-means through the use of cloud-computing. Centaurus is a web-accessible, cloud-hosted service that automatically deploys and executes multiple k-means variants concurrently, producing multiple models. It then scores the models to select the one that best fits the data – a process known as model selection. It also allows for the experimentation with different hyper parameters and provides a set of data and diagnostic visualizations so that users can best interpret its results.

From a systems perspective, Centaurus defines a pluggable framework into which clustering algorithms and k-means variants can be chosen. When users upload their data, Centaurus executes and automatically scales the execution of concurrently executing k-means variants using public or private cloud resources. To perform model selection, Centaurus employs a scoring component based on information criteria. Centaurus computes a score for each result (across variants, cluster sizes, and repeat runs) and provides a recommendation of the best clustering to the user. Users can also employ Centaurus to visualize their data, its clusterings, and scores, and to experiment with different parameterizations of the system (e.g., the number of repeat runs, the combination of features to cluster, and the dimensions to display).

We implement Centaurus using production-quality, open-source software and validate it using synthetic datasets with known clusters. We also apply Centaurus in the context of a real-world, agricultural analytics application and compare its results to the industry standard clustering approach. The application analyzes fine-grained soil electrical conductivity (EC) measurements, GPS coordinates, and elevation data from a field to produce a "map" of differing soil zones. These zones can then used by farmers and farm consultants to customize management of different zones on the farm (application of water, fertilizer, pesticides, etc.) Fridgen et al. (2004), Moral et al. (2010), Fortes et al. (2015), Corwin & Lesch (2003). We compare Centaurus to the state of the art clustering tool (MZA Fridgen et al. (2004)) for farm management zone identification and show that Centaurus is more robust, obtains more accurate clusters, and requires significantly less input and effort from its users.

In the sections that follow, we provide some background on the use of EC for agricultural zone management. We then describe the general form of the k-means algorithm, variants for computing covariance matrices, and scoring method that Centaurus employs (Section 3.2). Following this, we present our datasets, an empirical evaluation of Centaurus, related research specifically related to Centaurus, and summarize our contributions.

3.1 Clustering for Agricultural Zone Management

The soil health of a field can vary significantly and change over time due to human activity and forces of nature. To optimize yields, farmers increasingly rely on site-specific farming in which a field is divided into contiguous regions, called zones, with similar soil properties. Agronomic strategies are then tailored to specific zones (versus using the same strategy across the entire field) to apply inputs more precisely, to lower costs and input use, and to ultimately increase yields.

Management zone boundaries can be determined with many different procedures: soil surveys with or without other measurements Bell et al. (1995), Kitchen et al. (2006); spatial distribution estimates of soil properties by interpolating soil sample data Mausbach et al. (1993), Wollenhaupt et al. (1997) fine-grain soil electrical conductivity (EC) measurements Mulla et al. (1992), Jaynes et al. (1995), Sudduth (1997), Rhoades et al. (1989), Sudduth et al. (2005), Corwin & Lesch (2003), Ver (2019), and a combination of sensing technologies Adamchuk et al. (2004). EC-based zone identification is widely used because it addresses many of the limitations of the other approaches: it is inexpensive, it can be repeated over time to capture changes, and it produces useful and accurate estimates of many yield-limiting soil properties including compaction, water holding capacity, and chemical composition. As a result, EC-based management tools are used extensively for a wide variety of field plants (trees, crops, apples, vines, etc.) Peeters et al. (2015), Aggelopooulou et al. (2013), Gili et al. (2017).

To collect EC data, EC sensors are typically attached to a GPS-equipped tractor or all-terrain vehicle and pulled across a field to collect measurements at multiple depths and at a very fine grain spatially (a few feet). EC maps generated from this data can either be used to directly define management zones (visually) or to inform the future, more extensive, soil sampling locations Ver (2019), Lund et al. (1999). Alternatively, EC values can be clustered into related regions (management zones) using fast, automated, unsupervised statistical clustering techniques (e.g. k-means Lloyd (1982) and its variants Bezdek (2013), Murphy (2012)) Fridgen et al. (2004), Molin & Castro (2008), Fraisse et al. (2001), et al (2003).

Given the potential and wide spread use of EC-based zone identification tools that rely on automated unsupervised algorithms, in this chapter we investigate the impact of using different k-means implementations and deployment strategies for EC-based management zone identification. We consider different algorithm variants, different numbers of randomized runs, and the frequency of *degenerate* runs – algorithm solutions which are statistically questionable because they include empty clusters, clusters with too few data points, or clusters that share the same cluster center Brimberg & Mladenovic (1999). To compare k-means solutions (models), we define a model selection framework that uses the Bayesian Information Criterion (BIC) Schwarz (1978) to score and select the best model. Past work has used BIC to score models for the univariate normal distribution Pelleg et al. (2000). Our work extends this use to multivariate distributions and multiple k-means variants.

3.2 Methodology

The k-means algorithm attempts to find a set of cluster centers that describe the distribution of the points in the dataset by minimizing the sum of the squared distances between each point and its cluster center. For a given number of clusters K, it first assigns the cluster centers by randomly selecting K points from the dataset. It then alternates between assigning points to the cluster represented by the nearest center, and recomputing the centers Lloyd (1982), Bishop (2006), while decreasing the overall sum of squared distances Linde et al. (1980).

The sum-of-squared distances between data points and their assigned cluster centers provides a way to compare local optima – the lower the sum of the distances, the closer to a global optimum a specific clustering is. Note, that it is possible to use distance metrics other than Euclidean distance to compute per-cluster differences in variance, or covariance between data features (e.g. Mahalanobis distance Mahalanobis (1936)). Thus, for a given data set, the algorithm can generate a number of different k-means clusterings – one for each combination of starting centers, distance metrics, and a method used to compute the covariance matrix. Centaurus integrates both Euclidian and Mahalanobis distance. Computation of Mahalanobis distance requires computation of a covariance matrix for the dataset.

In Centaurus we integrate six different methods for computing covariance matrices for k-means algorithm: *Full-Tied*, *Full-Untied*, *Diagonal-Tied*, *Diagonal-Untied*, *Spherical-Tied*, and *Spherical-Untied* Murphy (1998, 2012), Bishop (2006), Cerioli (2005). Each of these methods is defined as:

- *Full*: Compute the entire covariance matrix $\mathbf{\Sigma}$ and use all of its elements to compute distance between points $\boldsymbol{x}$ and $\boldsymbol{y}$:

  $$D(\boldsymbol{x}, \boldsymbol{y}) = \left((\boldsymbol{x} - \boldsymbol{y})^T \mathbf{\Sigma}^{-1} (\boldsymbol{x} - \boldsymbol{y})\right)^{1/2}$$

  This variant is commonly associated with the use of Mahalanobis distance.

- *Diagonal*: Compute the variance matrix, i.e., the covariance matrix with its off-diagonal elements set to zero. This approach ignores the covariance observed between the dimensions of the dataset.

- *Spherical*: Set covariance matrix diagonal elements to the variance computed across all dimensions and set off-diagonal elements to zero. This method is commonly referred to as using Euclidean distance.

In addition, each of these approaches for computing the covariance matrix can be *Tied* or *Untied*. *Tied* means that we compute a covariance matrix per cluster, take the average across all clusters, and then *use the averaged covariance matrix* to compute distance. *Untied* means that we compute a *separate covariance matrix for each cluster*, which we use to compute distance. Using a tied set of covariance matrices assumes that the covariance among dimensions is the same across all clusters, and that the variation in the observed covariance matrices is due to sampling variation. Using an untied set of covariance matrices assumes that each cluster is different in terms of its covariance between dimensions.

We implement k-means in its general form using Mahalanobis distance in CENTAURUS using the following steps:

1. We use k-means++Arthur & Vassilvitskii (2007) to randomly select K points from the data and assign these as the initial cluster centers $\boldsymbol{\mu}^{(k)}$, where K is the number of clusters, k is the cluster index, and $k = 1, \ldots, K$.

2. For data points having d dimensions, compute initial covariance matrix $\boldsymbol{\Sigma}$ using all data points:

$$\Sigma_{ij} = \frac{1}{n}\sum_{p=1}^{n}(x_i^{(p)} - \mu_i)(x_j^{(p)} - \mu_j)$$

   where, Σ_{ij} is (i,j)-th component of the matrix $\boldsymbol{\Sigma}$, $x_i^{(p)}$ is the i-th component of the p-th data point, and μ_i is the i-th component of the global mean.

3. Assign all the points to the closest cluster center using Mahalanobis distance metric:

$$D(\boldsymbol{x}^{(p)}, \boldsymbol{\mu}^{(k)}) = \left((\boldsymbol{x}^{(p)} - \boldsymbol{\mu}^{(k)})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}^{(p)} - \boldsymbol{\mu}^{(k)})\right)^{1/2}$$

   where, $\boldsymbol{x}^{(p)}$ is the d-dimensional vector of components of the p-th data point, $\boldsymbol{\mu}^{(k)}$ is the center of the k-th cluster.

4. Compute covariance matrix $\boldsymbol{\Sigma}^{(k)}$ for each cluster using their cluster center $\boldsymbol{\mu}^{(k)}$.

5. Compute the cluster centers $\boldsymbol{\mu}^{(k)}$: for each point in a cluster, calculate the sum of its distances to all the other points in the same cluster. Assign the point with the minimum sum as the new center.

6. Repeat (4) and (5) until convergence or completion of a maximum number of iterations. The convergence criteria is calculated by summing up the distances of new cluster centers from the old cluster centers.

The output of the algorithm is a list of cluster labels, one per data point, indicating the cluster index to which the data point belongs and a list of cluster centers that correspond to the maximum likelihood estimates of the cluster means. We use the interpretation of k-means as the "hard" cluster assignment of Gaussian Mixture Model (GMM) to compute the maximum log-likelihood (for use by the Bayesian Information Criterion Schwarz (1978) or the Akaike Information Criterion Akaike (1974)) in order to compare the local optima generated from different variants of k-means and, ultimately, to choose the "best" one Pelleg et al. (2000). We discuss the use of information criteria as a "scoring" method across multiple runs of multiple variants in Section 3.3.2.

Once the labels are computed for each data point, we can compute the likelihood (a function of the data given the model) using the equation for GMM with hard assignment Bishop (2006), Murphy (2012), as:

$$f\left(\boldsymbol{X}|\boldsymbol{\mu},\boldsymbol{\Sigma}\right)=\prod_{p=1}^{n}\prod_{k=1}^{K}\pi_k^{\mathbb{1}_{pk}}\cdot\mathcal{N}\left(\boldsymbol{x}|\boldsymbol{\mu}^{(k)},\boldsymbol{\Sigma}^{(k)}\right)^{\mathbb{1}_{pk}}$$

where, p is a data point having d dimensions, k is a cluster index, π_k is the ratio of the number of points in cluster k and the total number of points, and $\mathbb{1}_{pk}$ is an identity coefficient that is 1 if the point p belongs to the cluster k and 0 otherwise, $\boldsymbol{\mu}^{(k)}$ is the k-th cluster center, $\mathcal{N}\left(\boldsymbol{x}|\boldsymbol{\mu}^{(k)}, \boldsymbol{\Sigma}^{(k)}\right)$ is the Gaussian probability density function with $\boldsymbol{\mu}^{(k)}$ mean and $\boldsymbol{\Sigma}^{(k)}$ covariance.

The log-likelihood function is needed to compute information criteria that Centaurus uses to score a particular clustering. We compute the log-likelihood function as:

$$\begin{aligned}
&l\left(\boldsymbol{X}|\boldsymbol{\mu}, \boldsymbol{\Sigma}\right) = \ln f\left(\boldsymbol{X}|\boldsymbol{\mu}, \boldsymbol{\Sigma}\right) \\
&= \sum_{k=1}^{K} n_k \cdot \left(\ln\left(\frac{n_k}{n}\right) - \frac{d}{2}\ln\left(2\pi\right) - \frac{1}{2}\ln|\boldsymbol{\Sigma}^{(k)}|\right) \\
&\quad - \frac{1}{2}\sum_{p=1}^{n}\sum_{k=1}^{K} \mathbb{1}_{pk} \cdot (\boldsymbol{x}^{(p)} - \boldsymbol{\mu}^{(k)})^T (\boldsymbol{\Sigma}^{(k)})^{-1} (\boldsymbol{x}^{(p)} - \boldsymbol{\mu}^{(k)})
\end{aligned}$$

3.3 Centaurus

We design and implement Centaurus as a software service for k-means clustering that takes advantage of cloud-based, large-scale distributed computation, automatic scaling (where computational resources are added or removed on-demand), data management to support visualization, and browser-based user interaction. Centaurus implements six different variants of k-means (described in Section 3.2). Centaurus provides a scalable execution environment and automatic deployment of multiple, concurrent k-means algorithm parameterizations and variants.

3.3.1 Implementation

The Centaurus implementation consists of a user-facing web service and distributed cloud-enabled backend. Users upload their datasets to the web service frontend as files in a simple format: as comma-separated values (CSV files). Advanced users can modify (or accept default values for) the following Centaurus parameters: maximum number of clusters to fit to the data (K); number of experiments (N) per K to run; number of times to initialize the k-means clustering (M); type(s) of covariance matrix to use for the analysis (all options – *Full-Tied*, *Full-Untied*, *Diagonal-Tied*, *Diagonal-Untied*, *Spherical-Tied*, and *Spherical-Untied* – are selected by default.); whether to scale the data so that each dimension has zero mean and unit standard deviation (*scale*).

Centaurus considers each parameterization that the user chooses (including the default) as a "job". Each job consists of multiple tasks (experiment runs) that Centaurus deploys. Users can check the status of a job or view the report for a job (when completed). The status page provides an overview of all the tasks with a progress bar for the percentage of tasks completed and a table showing task parameters and outcomes available for download and visualization.

Centaurus has a report page to provide its recommendation. The recommendation consists of the number of clusters and k-means variant that produced the best score. This page also shows the cluster assignments and spatial plots using longitude and latitude (if included in the original data set). For additional analysis users can select “advanced report” to see the correlation among features in the dataset, scores for each k-means variant, best clusterings for each one of the variants, etc.

We implement Centaurus using Python and integrate a number of open source software, packages, and cloud services. These services include the Python Flask Pyt (2019) (v0.12.1) web framework, RabbitMQ Rab (2019) (v3.2.4) and Python Celery Cel (2019) (v4.0.2) for messaging and queuing support, and an PostgreSQL (v9.5.11) SQL database Pos (2019) and MongoDB Community Edition (v3.4.4) NoSQL database Mon (2019), which we use to store parameters and results for jobs and tasks. Other packages include Numpy Walt et al. (2011) (v1.12.1), Pandas McKinney et al. (2010) (v0.19.2), SciKit-Learn Pedregosa et al. (2011) (v0.18.1), and SciPy Jones et al. (2001–) (v0.19.0) for data processing and Matplotlib Hunter (2007) (v2.0.1) and Seaborn Sea (2019) (v0.7.1) for data visualization. Centaurus can execute on any virtualized cluster or cloud system and autoscales deployments by starting and stopping virtual servers as required by the computation. In our evaluation in this chapter, we deploy Centaurus on a private cloud that runs Eucalyptus v4.4 Nurmi et al. (2009), Ari (2019), which has multiple virtual servers with different CPU, memory, and storage capabilities.

We build upon, generalize, and extend this system (and describe its design and implementation in greater detail) in Chapter 5 of this dissertation.

3.3.2 Centaurus Scoring

Centaurus performs N experiments for a particular K value (where $K = 1, ..., max\_k$), each of which consists of M initial cluster assignments to the k-means algorithm. Each algorithm iterates until convergence or a maximum number of iterations is reached (in Centaurus this value is 30). Thus, Centaurus executes $N * M$ runs of an algorithm for each value of K. Across M initial cluster assignments, Centaurus chooses the best performing one using the maximum log likelihood.

Centaurus scoring takes label assignments from a clustering result for a particular K value and returns a score based on the Bayesian Information Criterion (BIC). BIC uses the log maximum likelihood to rate a particular clustering and then subtracts a "penalty" function that captures the number of parameters that must have been estimated to generate the clustering, scaled by the sample size.

We compute the BIC score for a Full Tied clustering with K clusters as:

$$BIC_K = l(\boldsymbol{X}|\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}}) - \frac{r_K}{2} \log n$$

where, $\hat{\boldsymbol{\mu}}$ is the maximum likelihood estimator for the cluster centers, $\hat{\boldsymbol{\Sigma}}$ is the d-dimensional maximum likelihood estimator for the cluster covariance matrices,

$l(\boldsymbol{X}|\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}})$ is the maximum log likelihood, and r_K is the number of free parameters in the model. r_K is computed as the sum of $K-1$ cluster probabilities (π_k), $(K \cdot d)$ coordinate parameters for all the cluster centers, and $(K \cdot \frac{d \cdot (d+1)}{2})$ parameters for each of the K symmetric cluster covariance mattrices:

$$r_K = (K-1) + (K \cdot d) + (K \cdot \frac{d \cdot (d+1)}{2})$$

When a single covariance matrix is used for all clusters (the Untied variants), the factor of K in the third term is set to 1. Similarly, when the off-diagonal elements are zero, the fraction in the third term is either d (for the Diagonal variants) or 1 (for the Spherical variants). For BIC, the penalty function is r_K multiplied by $\frac{\log n}{2}$ where n is the total number of points.

Note that because these techniques require estimates of the covariance matrix for each cluster, there must be a minimum number of data points per cluster for this estimate to be meaningful. As a result, Centaurus discards (does not score or consider in the scoring average) any clustering result which has one or more clusters with fewer elements than this minimum. This minimum threshold is user configurable with a default setting of 30 data points in the current system.

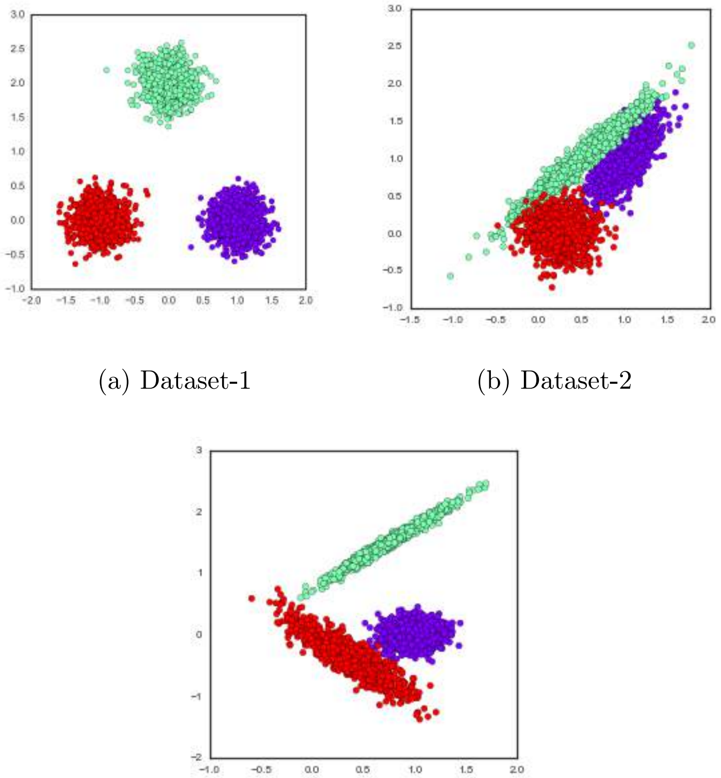
3.4 Datasets

We use both synthetic and real-world datasets to evaluate CENTAURUS empirically. We generate the synthetic data sets with known clusters (as "ground truth"), which we use to validate and measure the accuracy of the CENTAURUS implementation. Using the real-world application data from precision agriculture, we also compare the results generated by CENTAURUS for management zone determination to the industry standard and use them to illustrate the CENTAURUS visualization capabilities.

3.4.1 Synthetic Datasets

We first create multiple 2-dimensional synthetic datasets using multivariate Gaussian distributions. The datasets have three clusters with 1,000 points per cluster and varying degrees of inter-dimensional correlation in each cluster. Figure 3.1 shows these datasets with their ground truth cluster assignments.

- *Dataset-1* clusters have no correlation and equal standard deviations of 0.2 for each dimension. Cluster centers are set at positions $(1, 0)$, $(-1, 0)$, and $(0, 2)$ as can be seen in Figure 3.1a.
- In *Dataset-2* cluster centered at $(0.25, 0)$ has two dimensions that are independent (not correlated) with the same standard deviations of 0.2. The



(a) Dataset-1

(b) Dataset-2

(c) Dataset-3

Figure 3.1: Synthetic datasets shown with ground truth assignment.

cluster centered at $(1, 1)$ has correlation 0.70 between dimensions, while the cluster centered at $(0.5, 1)$ has correlation 0.97 between dimensions. When all three clusters are combined the correlation between the two dimensions is 0.75.

Field	Soil Type	Plantation
CAP	clay, sandy-clay-loam/cl, sandy-loam, sandy-clay-loam	lemon trees
SED	sandy-clay-loam	grapes, fruit trees
UNL	silt-loam, silty-clay-loam	soybean, corn
ALM	sand, loamy-sand,sandy-loam	almond trees
TC1	loamy-sand, sand, sandy-loam>60%	tango citrus
TC2	loamy-sand, sandy-loam: <60% sand, sandy-loam: >65% sand	tango citrus
GR1	loam, clay-loam, sandy-loam, sandy-loam/scl, sandy-clay-loam	grape
GR2	loam, clay-loam, sandy-loam, sandy-loam/scl, sandy-clay-loam	grape
CTR	sandy-loam, sandy-clay-loam, loamy-sand, clay%<50,50-55,>55	citrus trees
RAN	clay-loam, sandy-clay-loam, loam, sandy-clay/scl(/l), sandy-loam	fruit trees

Table 3.1: List of application datasets with soil type and plantation.

- In *Dataset-3*, the cluster centered at $(1, 0)$ has two independent dimensions, while the clusters centered at $(0.75, 1.5)$ and $(0.35, -0.35)$ have correlations of 0.98 and -0.89 respectively. The correlation of the entire set is 0.2.

3.4.2 Application Datasets

Our farm datasets contain measurements of electrical conductivity (EC) of the soil collected using an instrument manufactured by Veris Technologies Inc Ver

Field	**Size**	**EC1**μ	**EC1**σ	**EC2**μ	**EC2**σ	**El**μ	**El**σ
CAP	3232	82.7	30.5	79.7	40.1	82.1	1.4
SED	2675	19.8	9.0	27.2	10.3	328.4	0.7
UNL	5797	20.9	8.9	31.0	14.8	356.5	0.6
ALM	13093	6.3	3.1	6.3	3.1	n.a.	n.a.
TC1	3304	12.2	6.0	7.7	3.9	102.7	1.1
TC2	1208	8.5	3.6	4.9	2.6	102.5	0.9
GR1	2720	21.6	5.8	n.a.	n.a.	n.a.	n.a.
GR2	2715	18.5	5.1	n.a.	n.a.	n.a.	n.a.
CTR	16834	53.1	24.9	120.3	49.2	125.7	1.1
RAN	17039	89.8	36.8	92.5	32.5	237.3	6.4

Table 3.2: Application Dataset Statistics: mean (μ) and st. deviation (σ) for each one of the features used ($EC1, EC2, Elevation$)

(2019). Electrical conductivity is coupled with the GPS coordinates and elevation information for each measurement. Each data file contains longitude, latitude, elevation, EC at 30cm depth (EC1), and EC at 90cm depth (EC2).

We analyze ten farm field datasets from eight different locations in the United States that represent a variety of soil types and management practices. The data sampling is not uniform, the distance between data points is influenced by the type of plant: if it is perennial plant (e.g. trees) data is gathered between rows; empty fields (e.g. soybean or corn) permit narrow row spacing. Sampling is further influenced by the speed at which the sensor was driven. At the speed of 10km/h the sensor produces around 275 readings per hectare.

Most of the datasets are multivariate (containing EC1, EC2, Elevation, Longitude, Latitude) and, when available, we use EC1, EC2 and Elevation for clustering. Some datasets (GR1, and GR2) have only EC1 measurements and for those datasets we compute the clusters based on only one dimension (EC1) and multivariate computation falls back to simple Euclidean distance. Longitude and Latitude fields are only used to visualize the datasets.

Tables 3.1 and 3.2 summarize the information in the datasets that we consider in this study. For each of the datasets, Table 3.1 lists soil varieties and plant types grown on the farm at the time of sampling. Table 3.2 contains dataset statistics: dataset size, mean, and variance for each feature used in the clustering (EC1, EC2, Elevation).

The first example is the *CAP dataset*, which is a 4.85ha lemon farm at California Polytechnic State University, San Louis Obispo, California, USA. We have collected 3,232 data points from this field and its soil consists of sandy clay loam, sandy loam, and clay. The *SED dataset* comes from a 12.1ha field located in the Santa Ynez Valley, California from which we have collected 2,675 data points.

This field mostly consists of sandy clay loam, clay loam, and loam. The *UNL* is a 36.8ha field at University of Nebraska, Lincoln, from which we have 5,823 data points. It is mostly used for corn and soybean and it has silty loam and silty clay loam soil types. The other datasets described in the table are those from private, production farms; the names and locations of which we have been asked to keep anonymous.

3.5 Results

In this section, we evaluate Centaurus k-means cluster quality for multiple k-means variants. We first compare cluster resolution power of the variants using synthetic datasets. We then detail the clustering that Centaurus determines for Veris Ver (2019) electroconductivity (EC) measurements taken from ten farms. From this data, we illustrate both the advantage of including multiple k-means variants in the pool of algorithms that Centaurus implements and the effect of executing multiple randomized trials on the quality of the clustering. We then extend this study to seven more farms from different locations and with various crop types. Finally, we compare Centaurus clusterings to those produced by MZA for both synthetic and Veris EC data.

3.5.1 Synthetic Dataset and K-means Variants

To evaluate the efficacy of Centaurus, we deploy the service and run it on the datasets described in Section 3.4 for each of the k-means variants described in Section 3.2. In this section, we refer to the variants as *Full-Untied*, *Full-Tied*, *Diagonal-Untied*, *Diagonal-Tied*, *Spherical-Untied*, and *Spherical-Tied.*

Variant	**Dataset-1**	**Dataset-2**	**Dataset-3**
Full-Untied	0.0%	3.6%	0.1%
Full-Tied	0.0%	37.6%	57.5%
Diagonal-Untied	0.0%	26.2%	26.0%
Diagonal-Tied	0.0%	34.4%	55.2%
Spherical-Untied	0.0%	27.3%	11.2%
Spherical-Tied	0.0%	34.4%	56.6%

Table 3.3: Percentage error (out of 3,000 points per dataset) for the six k-means variants of Centaurus for the synthetic datasets. Values are the percentage of points incorrectly labeled by the variant (i.e. assigned to the wrong cluster).

For the datasets with known clusters (those that we have generated synthetically) we report classification percentage error, i.e. the percentage of incorrectly classified points out of all the points in the dataset (3,000 data points per dataset in this case). Table 3.3 shows these results for each of the synthetic datasets (Dataset-1, Dataset-2, and Dataset-3) for each of the six k-means variants.

For the results that follow, we parameterize Centaurus with $K = 1, \ldots, 10$ and 100 experiments each with 100 random initial cluster center assignments (for a total of 10,000 k-means algorithm invocations per variant). Centaurus stores the cluster assignments (labels) for each experiment, which is the result with the largest log-likelihood value across initial assignments. This Centaurus instance only considers clustering results when all clusters have at least 30 points, in its computation of BIC and AIC. Finally, as described above, Centaurus reports the result with the highest average BIC score the "best" clustering across every K considered for all variants.

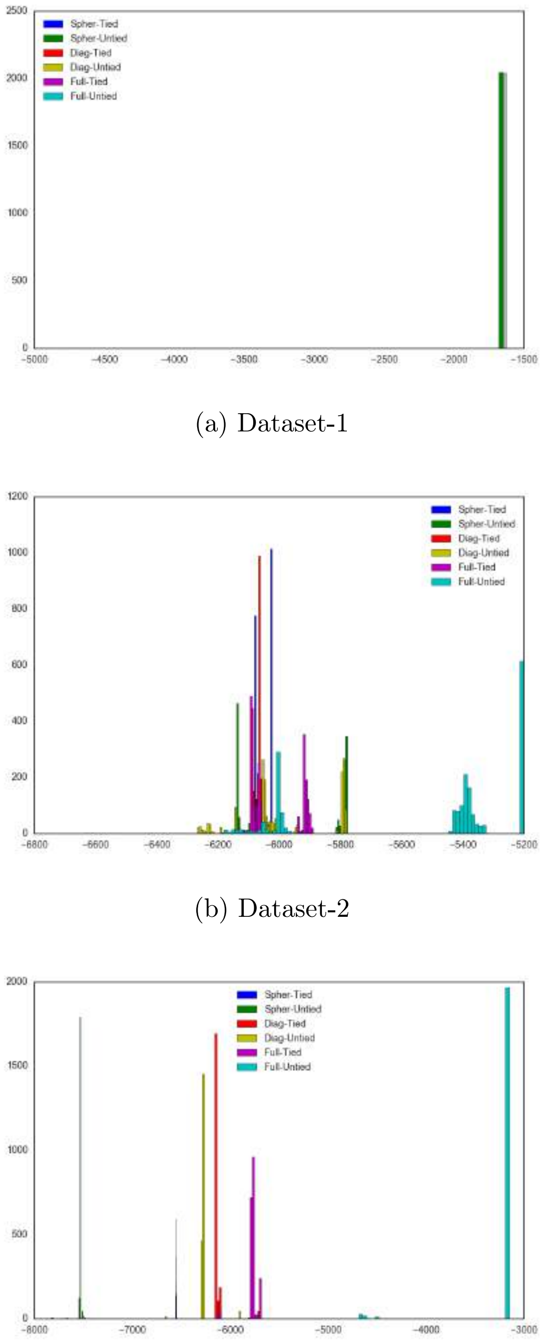
Note that Dataset-1 was generated using a GMM where all dimensions are independent of each other and are identically distributed. Thus the "perfect" classification results (0% error) generated by the Full and Diagonal methods indicate that they correctly disregard any observed sample variance or covariance.

The results for Full-Untied with Dataset-2 and Dataset-3 illustrate Centaurus 's ability to correct for cross-dimensional correlation. The generating GMM in both cases is untied (i.e. each cluster has a distinct covariance matrix). Also, unlike in Dataset-1 where there are three distinct clusters with separated centers, we purposefully placed the cluster centers of Dataset-2 and Dataset-3 near each other and generated distributions that overlap. Doing so poses challenges for k-means clustering and all variants misclassified some points.

To visualize the effect of different k-means variants on BIC score, we perform 2048 single k-means runs for each variant for synthetic datasets described in 3.4. Figure 3.2 shows histograms of the BIC scores for each of three synthetic datasets. We divide the scores among 100 bins. For each dataset we present six histograms, one for each of the k-means variants, represented in different colors, where each variant has a total of 2048 single k-means runs. The X-axis depicts BIC scores from experiments – farther right corresponds to larger BIC and thus higher quality clusterings.

`Dataset-1` consists of well-separated clusters. All six variants perform well making the simpler (i.e. those with fewer parameters to be estimated) variants generate slightly higher BIC scores (as depicted in Figure 3.2a).

However, for datasets where the dimensions are more highly correlated and/or where that correlation differs across clusters, the complex variants (Full Tied and Full Untied) outperform their simpler counterparts in terms of BIC score. `Dataset-2` and `Dataset-3` differ in that for the latter, the cross-dimensional correlation varies by synthetic cluster. Nonetheless, as shown in Figures 3.2b and 3.2c, the Full-Untied variant (which computes a separate co-variance matrix



(a) Dataset-1

(b) Dataset-2

(c) Dataset-3

Figure 3.2: BIC score histograms for synthetic datasets with six k-means variants.

for each cluster) performs best. These experiments, although synthetic, show the importance of considering different variants when employing k-means clustering.

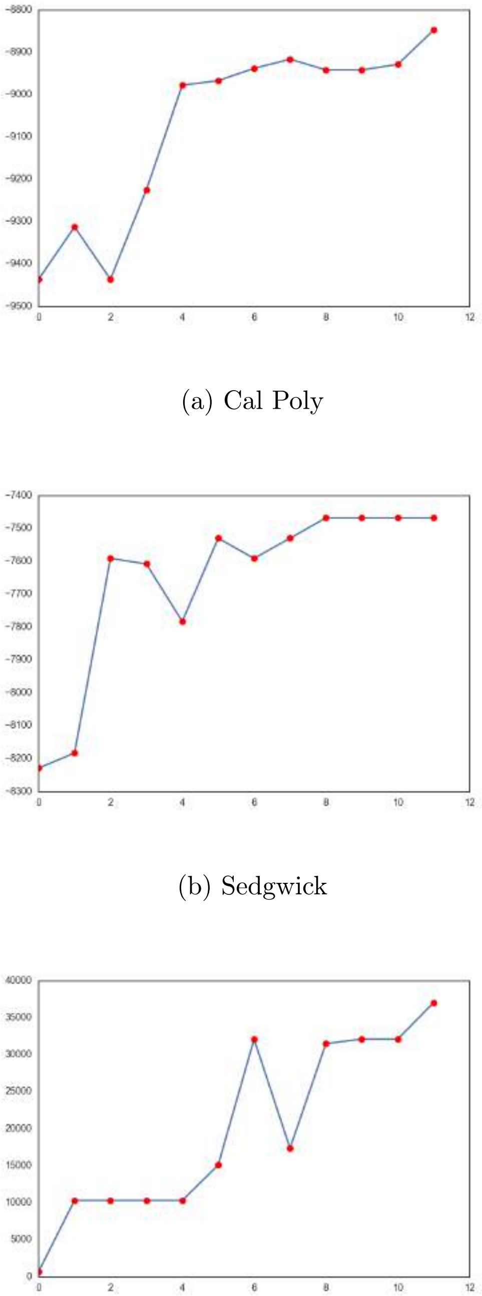
3.5.2 Soil Electrical Conductivity Datasets

In this section, we refer to an `experiment` as 10 repeated clusterings (using k-means++Arthur & Vassilvitskii (2007) to make initial assignments in each repetition) for each number of clusters k between 1 and 10, for each of the six k-means variants we examine in this study. Thus, each individual experiment consists of $10 * 10 * 6 = 600$ individual cluster assignments using k-means.

Centaurus repeats each experiment N times, where $N = 2^i$, for $i = 0, ..., 11$. In this study, we refer to a set of N experiments as `job-N`. Thus `job-N` consists of $N * 600$ individual clusterings. Centaurus filters out any clustering with a cluster having fewer than 30 points (so that any per-cluster statistical estimates are statistically valid). To determine the best clustering from a job, Centaurus computes a BIC score for each clustering in the job and selects the one with the largest score.

Cluster Quality Analysis

To show the effect of using a large sample when determining the "best" clustering, in Figure 3.3 we plot the largest observed BIC score (on the y-axis) versus



(a) Cal Poly

(b) Sedgwick

(c) UNL

Figure 3.3: Largest observed BIC score vs number of experiments (log scale).

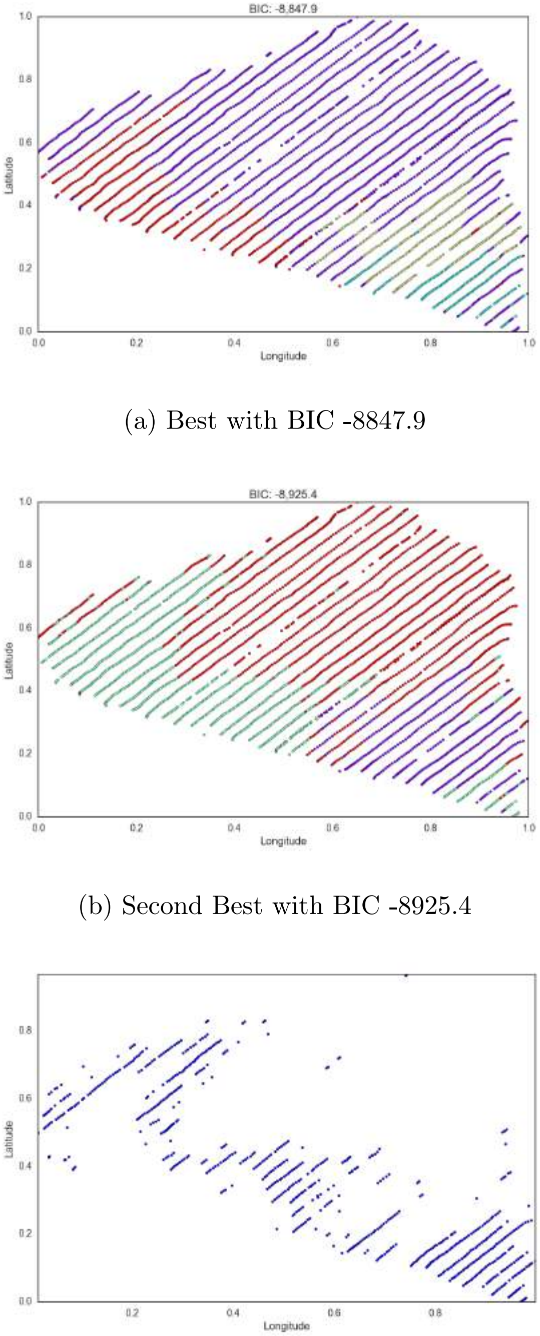
the experiment number N (on the x-axis, which uses a log scale). Figures 3.3a, 3.3b, and 3.3c show EC data from Cal Poly, Sedgwick, and UNL respectively.

As the sample size goes up, the probability of determining a clustering with the "best" BIC score (or, at least a consistently good BIC score) should increase as well. For the Sedgwick data (Figure 3.3b) this effect is clearly visible. Once the number of experiments exceeds $N = 2^8$, there is no further improvement in BIC. However for Cal Poly and UNL, the presence of a higher BIC occurring only at $N = 2^{11}$ indicates that even more repetitions are necessary to identify a consistently "best" clustering. Thus, for these datasets, the best clustering in the "space" of all possible clusterings is rare since it does not occur repeatedly when the sample size is less than 1.23 million ($2^{11} * 600$).

Cluster Specificity

One possibility is that the "best" clustering (the one with the highest BIC score) and the next best are similar. In this case, then, a large exploration of the clustering search space may be unwarranted because the best is not substantially different from the next best (which may be more common and require less computational effort to find).

To investigate this possibility, we consider the two largest jobs from the Cal Poly dataset: the largest job with $N = 2^{11}$ experiments (`job-2048`) and the second

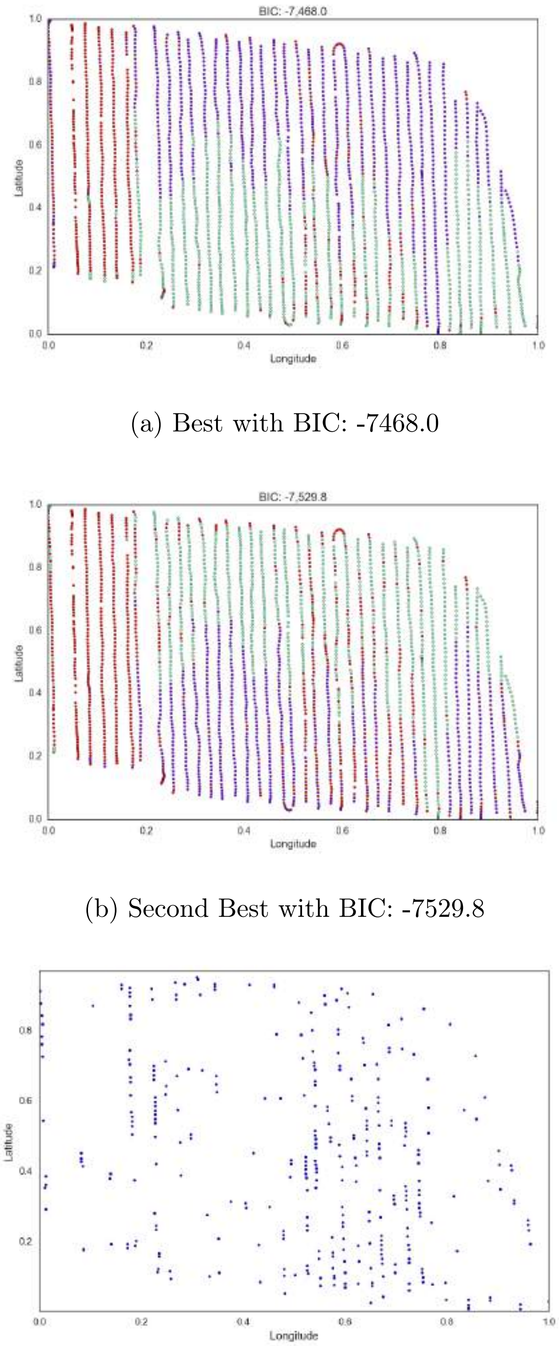


(a) Best with BIC -8847.9

(b) Second Best with BIC -8925.4

(c) Differences

Figure 3.4: Clusterings of Cal Poly Dataset.

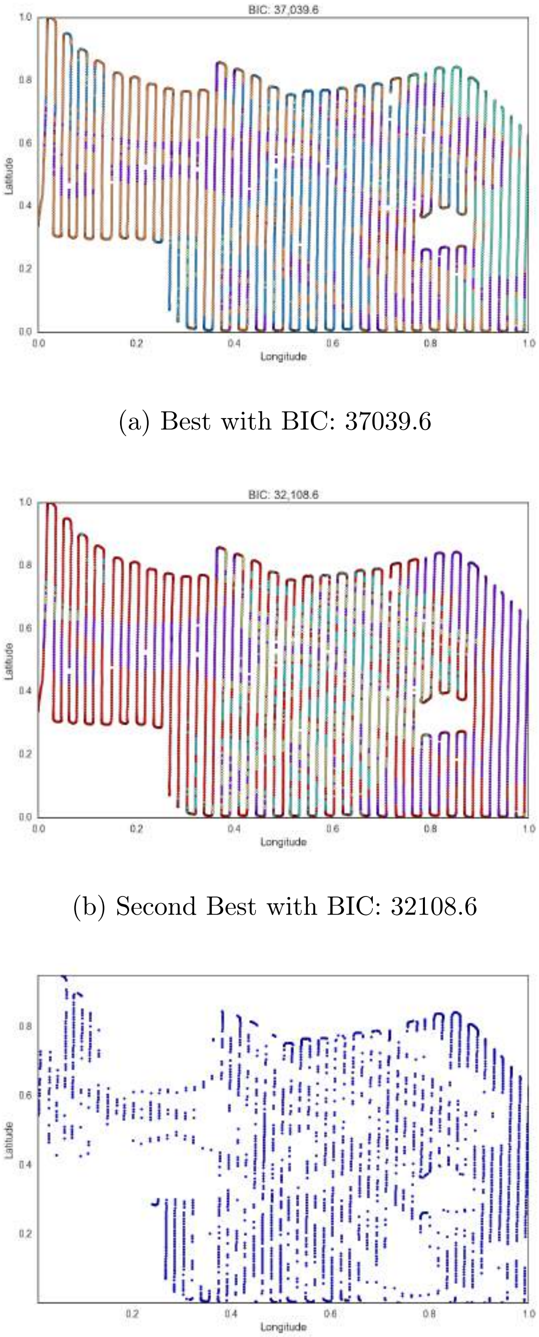


(a) Best with BIC: -7468.0

(b) Second Best with BIC: -7529.8

(c) Differences

Figure 3.5: Clusterings of Sedgwick Dataset.



(a) Best with BIC: 37039.6

(b) Second Best with BIC: 32108.6

(c) Differences

Figure 3.6: Clusterings of UNL dataset.

largest job with $N = 2^{10}$ experiments (`job-1024`). The largest job, `Job-2048`, with twice the number of experiments of `job-1024`, has the best BIC score of -8847.9 (Figure 3.4a). This corresponds to a clustering with four clusters having cardinality of 2188, 531, 308, and 205, respectively. The second best clustering has BIC score of -8925.4 and three clusters with cardinality 1733, 973, and 526, respectively, as shown in (Figure 3.4b).

Figure 3.4c shows the difference between these two clusterings. A specific data point is shown (i.e. is considered "different") if it has a different cluster number assignment (is in a different cluster) when we rank clusters by cardinality. For this data, clearly these clusterings differ. Thus, doubling the number of experiments from 1024 to 2048 allows Centaurus to find a clustering with a better BIC score.

The Sedgwick dataset has more stable outcome in terms of the best BIC score when increasing the number of experiments. Figure-3.3b shows that even with 256 experiments (150K k-means runs), we achieve the same maximum BIC score as with 2048 experiments. The best result has a BIC score of -7468.0 and three clusters with 1111, 996, and 568 elements (as shown in Figure 3.5a). This result is consistent over many repeated jobs with a sufficiently large number of experiments i.e. any job with more than 256 experiments produced this same clustering as the one corresponding to the largest score.

The second best clustering agrees with the best result on the number of clusters ($k = 3$) with cluster cardinalities of 963, 879, and 833, and a BIC score of -7529.8 (as shown in Figure 3.5b). While these clusters do differ, Figure 3.5c shows that the differences are scattered spatially. Thus the best and second best clusterings may not differ in terms of actionable insight.

For the UNL field, the best and second best clusterings (and their respective BIC scores) are shown in Figure 3.6. These are both from `job-2048`. The best clustering has six clusters with cardinalities 2424, 1493, 1138, 561, 111, and 70, respectively. The second best clustering has four clusters with cardinalities 2730, 1615, 838, and 614, respectively. From these features and the differences shown in Figure 3.6c it is clear the best and second best clustering are dissimilar.

Further, the second best clustering from `job-2048` (shown in Figure 3.6b) is the best clustering in `job-64`, `job-512`, and `job-1024` respectively. As with the Cal Poly data (but not the Sedgwick data), doubling the number of experiments from 1024 to 2048 “exposed” a better and significantly different clustering.

k-means Variants

Unlike the results for the synthetic datasets, the best clustering for the Veris EC datasets is produced by the Full Untied variant for sufficiently large job sizes. This result is somewhat surprising since the Full Untied variant incurs the largest score penalty in the BIC score computation among all of the variants. The score is penalized for the mean, variance, and co-variance estimates from each cluster. The other variants require fewer parameter estimates (and thus have a lower penalty). Related work has also argued for using fewer estimated parameters to produce the best clustering Fridgen et al. (2004) leading to an expectation that a simpler variant (e.g. Full Tied as in Fridgen et al. (2004)) would produce the best clustering, but is not the case for these datasets. Because Centaurus considers all variants, it will find the best clustering even if this effect is not general to all Veris data.

3.5.3 Statistical Clustering of Soil EC Data: Variants, Degeneracy, and Repeated Trials

We next use Centaurus to empirically evaluate and experiment with the EC values from seven additional farm fields with various crops, located across United States. In our empirical evaluation of the use of statistical clustering of the EC data for these farms, we use the following experimental setup for each of the datasets described in the next section:

- *Feature selection*: if available, use three dimensions for clustering ($EC1$, $EC2$, and $Elevation$) and two default dimensions for visualizing ($Longitude$, and $Latitdue$).
- *Variant selection*: with multivariate datasets use all six variants (from *Full-Tied* to *Spher-Untied*); for univariate datasets use only the *Spher-* (*Tied* and *Untied*) variants.
- *Number of experiments (N)*: we vary the number of experiments and name the job based on this variable. E.g. `Job-1024` will have 1024 experiments, and `Job-2048` will have 2048 experiments.
- *Number of initial assignments (M)*: We run 10 randomized cluster center initializations for each set of experiment parameters(*k, N, and a variant type*).

For a particular, Job-N, there are $N * 600$ clusterings (k-means runs). 600 is the product of 10 randomized initial assignments for each of 10 cluster sizes ($k = 1, \cdots, 10$) and 6 k-means variants. Thus *Job-2048* consists of $2048 \times 600 = 1,228,800$ executions of the k-means algorithm to convergence for $1 \leq k \leq 10$ across all 6 variants with 10 different initial assignments per run.

We first evaluate the models produced by different k-means variants and compare their scores. We then investigate the impact of degenerate clusters. Degener-

Farm	**Job-512**		**Job-1024**		**Job-2048**	
	Type	**BIC**	**Type**	**BIC**	**Type**	**BIC**
CAP	F-U	-8928.6	F-U	-8935	F-U	-8918.4
SED	F-U	-7468	F-U	-7468	F-U	-7468
UNL	F-U	32108.6	F-U	32108.6	F-U	45021.5
ALM	F-T	209303	F-T	209303	F-T	214040
TC1	F-U	-8071.7	F-U	-7907.9	F-U	-7853.6
TC2	F-U	-3274.6	D-U	-3197.9	F-U	-3191.2
GR1	S-U	-3655	S-U	-3647.7	S-U	-3647.7
GR2	S-U	-2348.8	S-U	-2329	S-U	-2329.2
CTR	F-U	-45550.7	F-U	-45046.5	F-U	-45282.8
RAN	F-U	-54523.8	F-U	-54438.8	F-U	-54200.1

Table 3.4: Clustering Variants With Best BIC Scores: the best BIC score and the variant that produced it, grouped by the experiment size (512, 1024, and 2048) for each farm

ate clusters are algorithm solutions that are statistically questionable because they include empty clusters, clusters with too few data points to make a meaningful inference, or clusters that share the same cluster center Brimberg & Mladenovic (1999). We empirically analyze how often such degeneracy occurs and present statistics for such solutions for different datasets. We then evaluate the impact of large numbers of runs and analyze the differences between the best scored clustering with the most commonly occurring clustering. Finally, we use core samples as a ground truth analysis to evaluate when soil zones belong to the same or different soil types and visualize differences between the clusterings.

Variants

To evaluate the differences among clustering variants as applied to farm datasets, we present three largest jobs with their best BIC scores and the variant that produced the best score (Table 3.4). For each farm dataset we present results from the three largest experiments: `Job-512`, `Job-1024`, and `Job-2048`.

The results show that for most of the multivariate datasets, the best clusterings came from the Full-Untied variant with a very small number of exceptions: The ALM dataset (Full-Tied) and `Job-1024` for TC1 dataset (Diag-Untied). For the univariate datasets (GR1 and GR2), the only variant possible is spherical since there are no additional dimensions with which to compute the covariance. For these datasets, the Spher-Untied variant performs best.

Farm\K	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
CAP	8171	6410	7169	6788	6422	6218	6162	6144	6144
SED	6258	6779	6428	6212	6140	6123	6116	6114	6111
UNL	9571	8150	7480	6784	6440	6276	6214	6181	6154
ALM	12288	12288	11784	9125	8923	6100	5041	4592	4067
TC1	11516	9341	7558	5589	3912	1488	456	74	21
TC2	9481	5262	2644	2191	1732	1151	207	41	2
GR1	2050	4096	4096	4088	4064	3933	3186	2354	2115
GR2	2049	4091	4087	3493	2532	2295	2125	2063	2049
CTR	11968	11100	8471	7860	7067	6502	6288	6194	6159
RAN	10464	9294	8717	7858	6860	6447	6220	6167	6151

Table 3.5: Numbers of non-degenerate experiments for clusters of size $k = 2, \cdots, 10$ from the total of 12288 for multivariate and 4096 for univariate experiments.

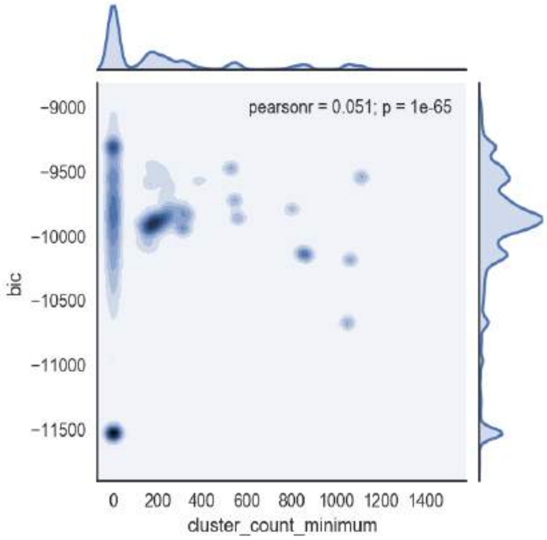


Figure 3.7: Joint distribution of BIC and cluster count minimum for CAP dataset computed from 12288 experiments.

Degeneracy

Degenerate clusters are a surprisingly frequent, yet under-studied, phenomenon when clustering farm datasets. We next investigate the frequency with which degeneracy occurs for different datasets and different number of clusters.

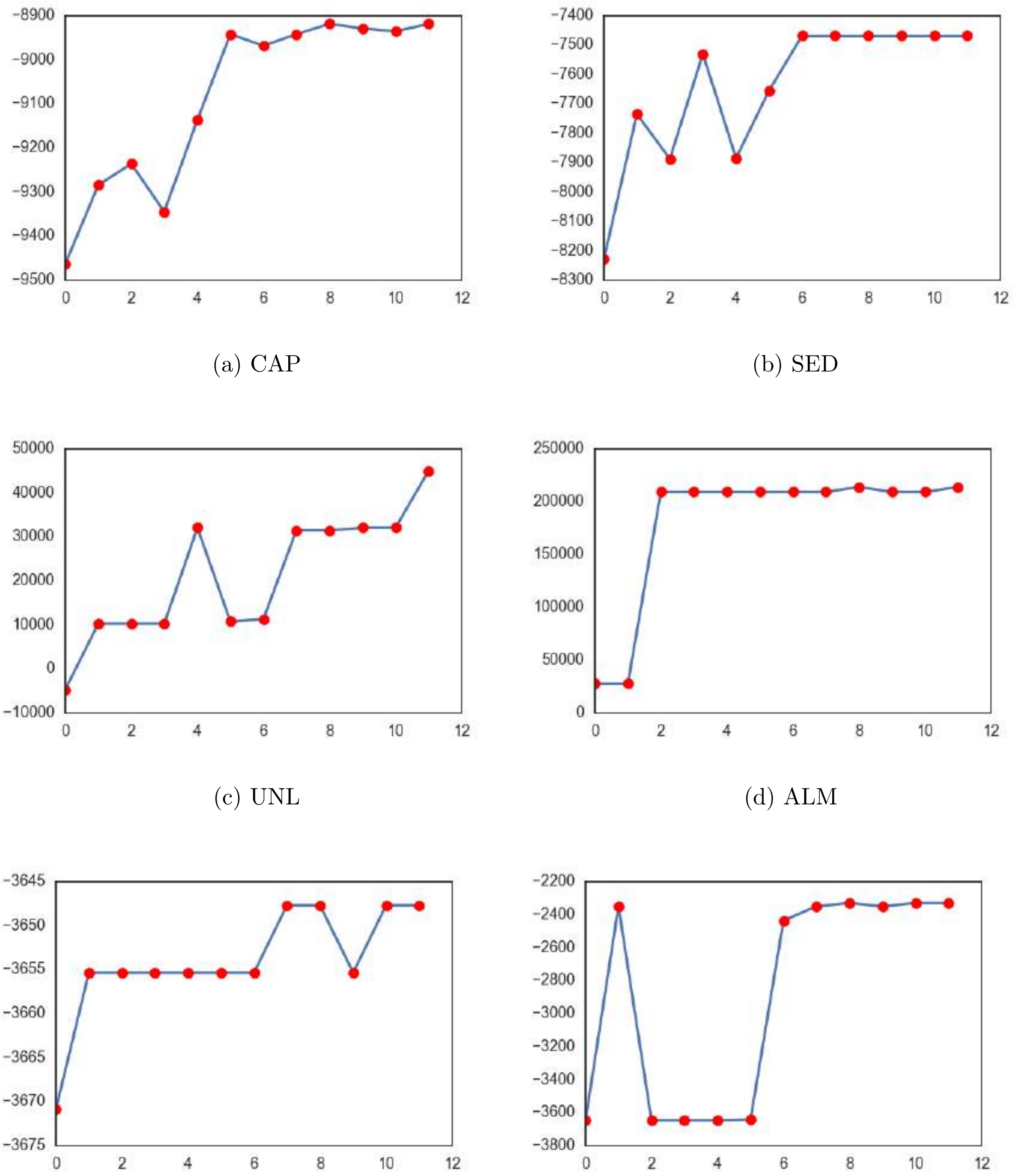
Figure3.7 illustrates the search space for the best BIC score with the estimated joint distribution of BIC scores (y-axis) and the number of elements in the smallest cluster (x-axis). The figure represents a `Job-2048` for CAP dataset, with all six variants and all values of K (1-10). Darker colors on the graph represent higher density regions. Our system uses all of the k values and all of the variants when choosing the model with the highest BIC score. Per-component distributions are available on the sides of the graph.

The graph indicates that the highest BIC scores often come from the clusterings that have one or more almost empty clusters. Particularly for variants that rely on an estimate of co-variance between dimensions, inferences made about the means of these clusters are suspect when their sizes are small. Note that for larger values of k, such clusterings can be common. For example, this particular job had 50961 or 41.5% degenerate and 71916 non-degenerate experiments.

To illustrate this effect more fully, we divide the experiments based on the number of clusters, k, and illustrate how degeneracy behaves with increasing k in Table 3.5. The total number of experiments for multivariate datasets was 12288 for each k and all six variant types. The univariate datasets (GR1 and GR2) had 4096 experiments and include only two (spherical) variants for each k. For each farm, the results show the number of non-degenerate clusterings for each k. In some cases, the number of non-degenerate clusterings decreases as k increases.

Farm	**CAP**	**SED**	**UNL**	**ALM**	**TC1**	**TC2**	**GR1**	**GR2**	**CTR**	**RAN**
%	42	44	39	30	58	72	06	10	32	35

Table 3.6: Percentages of degenerate experiments per field



(a) CAP

(b) SED

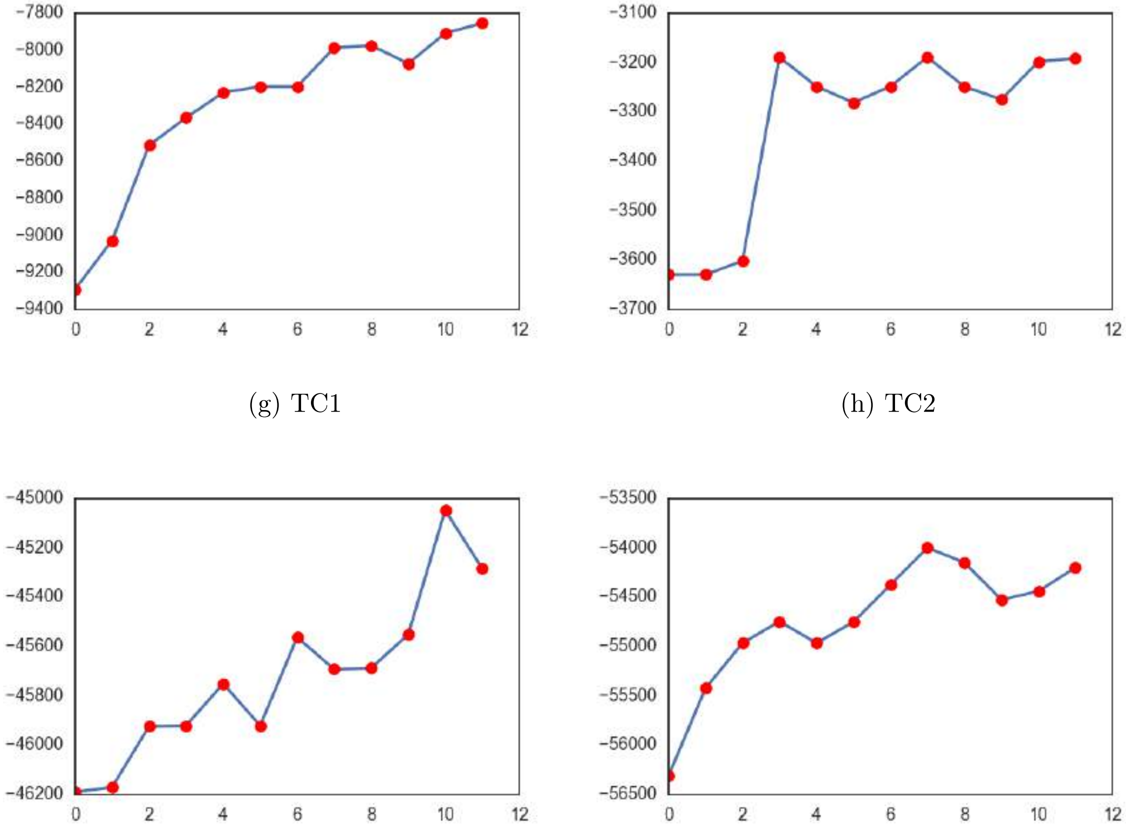
(c) UNL

(d) ALM

(e) GR1

(f) GR2

Figure 3.8: The largest observed BIC score vs the number of experiments on the log scale (from $2^0 = 1$ to $2^{11} = 2048$)



(g) TC1

(h) TC2

(i) CTR

(j) RAN

Figure 3.8: Continued from Previous Page: The largest observed BIC score vs the number of experiments on the log scale (from $2^0 = 1$ to $2^{11} = 2048$) (cont.)

To emphasize the overall degeneracy across all `Job-2048` experiments for each dataset, we summarize the percentages of experiments with fewer than 30 elements in their smallest cluster in Table 3.6. The smallest percentage of degenerate clusters is 6% for the GR1 dataset and the largest percentage was 72% for TC2 dataset. We have chosen 30 as a reasonable rule of thumb for a cluster size from which to make an inference about the mean (centroid) of each cluster in the experiments having three dimensional data.

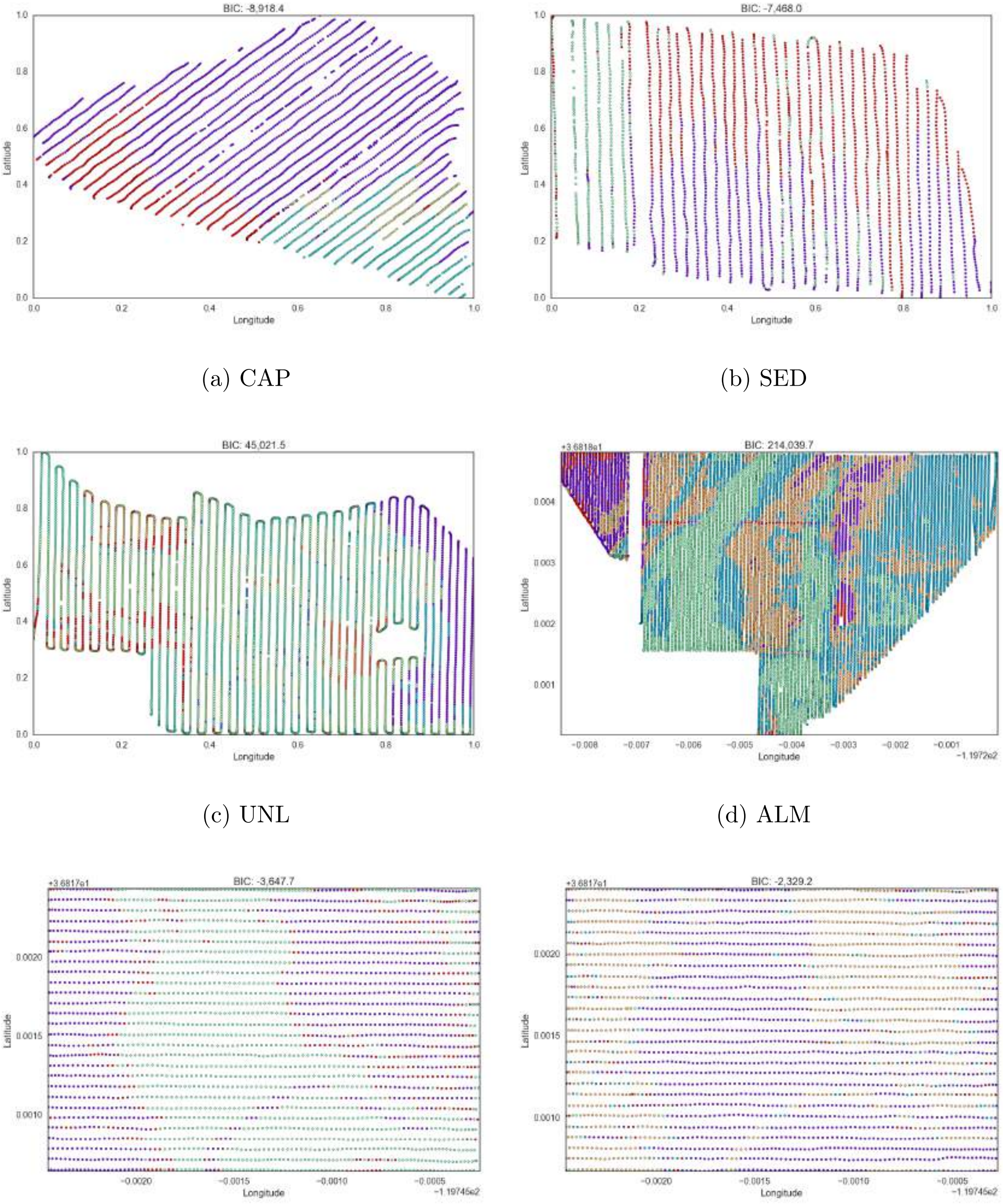
The Effect of Repeated Trials

Because k-means converges to a locally optimum solution for non-convex solution spaces, the choice of initial assignment can effect the clustering it finds. Often, users of k-means will run it once, or a small number of times assuming that the local minimum it finds is "close" to the global minimum. In this subsection, we investigate the validity of this conjecture for soil EC data across farms. Figure 3.8 presents the best BIC scores for different experiment sizes for all of the farm datasets.

In some of the jobs the best BIC score occurs only once amongst all of the experiments while in others the best BIC score is more common among multiple experiments (consistent with the typical use of k-means). Thus a small number of trials is likely to result in the most common clustering rather than the best one.

Farm	Type	BIC	K	Cardinality	F
CAP	B	-8918	4	2103, 500, 473, 156	1
	MC	-10169	2	2169, 1063	1445
SED	B	-7468	3	1111, 996, 568	12
	MC	-9123	2	2220, 455	1720
UNL	B	45021	8	2457,1636,617,313,278,259,151,86	2
	MC	-17655	2	3919, 1878	1172
ALM	B	214039	5	4512, 3611, 3393, 1265, 312	1
	MC	-29460	2	10298, 2795	2048
TC1	B	-7853	8	1205,965,558,299,133,60,48,36	1
	MC	-9474	2	3247, 57	1716
TC2	B	-3191	4	880, 233, 51, 44	1
	MC	-4360	2	989, 219	2029
GR1	B	-3647	3	1304, 1091, 325	4
	MC	-3814	3	1181, 857, 682	1956
GR2	B	-2329	5	1390, 951, 190, 141, 43	1
	MC	-3838	3	1273, 835, 607	2045
CTR	B	-45282	6	13031, 2077, 828, 667, 198, 33	1
	MC	-46188	2	13996, 2838	1536
RAN	B	-54200	5	9337, 3342, 1843, 1489, 1028	1
	MC	-58784	3	15810, 704, 525	1044

Table 3.7: The best (B) and the most common (MC) clusterings comparison for each farm, with their BIC scores, number of clusters (K), cardinality, and result frequency (F).



(a) CAP

(b) SED

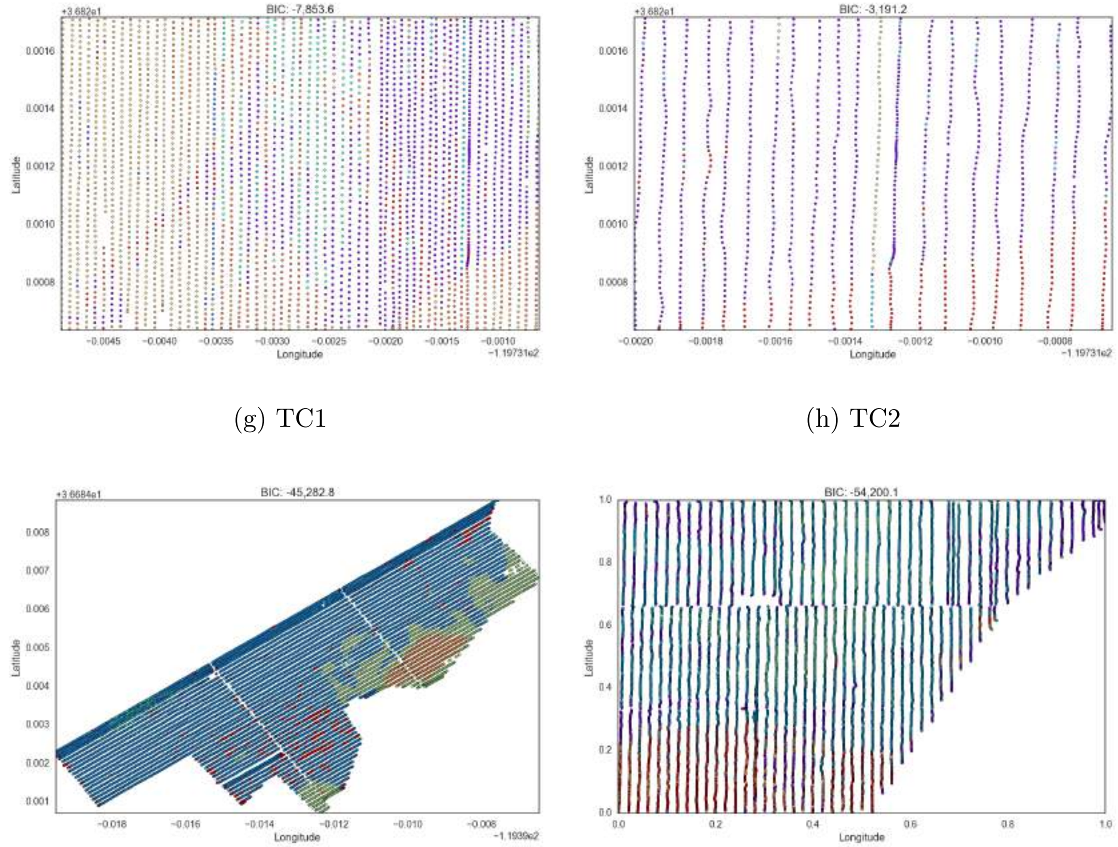
(c) UNL

(d) ALM

(e) GR1

(f) GR2

Figure 3.9: Visualized clusterings with the best BIC score from Table 3.7.



(g) TC1

(h) TC2

(i) CTR

(j) RAN

Figure 3.9: Visualized clusterings with the best BIC score from Table 3.7 (cont.).

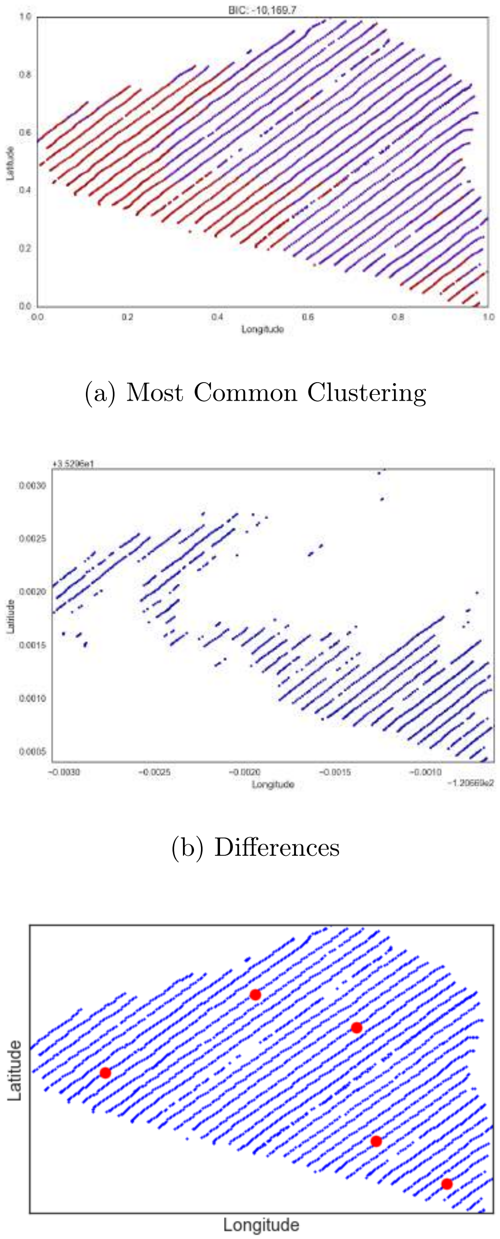
More importantly, an increase in the number of experiments increases the chance of finding the best BIC score. The x-axis for each graph in Figure 3.8 is the power of 2 in the number of experiments. For most of the graphs, k-means finds the best BIC consistently beyond some large number. However for a few of them, it appears that an even greater number of experiments may be necessary before a single consistently large BIC is determined. The graph in Figure 3.3c, for example, seems to indicate that an even larger number of experiments may yield a larger BIC. Thus, for the EC available to our study, it is clear that the best clustering is often rare and thus requires a large number of independent trials to determine.

Even though the best clustering may be rare, it may also be that it differs from the most common clustering by so little as to make the effort (through many repeated trails) required to find it unnecessary or wasteful. Table 3.7 compares the clustering determined by the best BIC scored cluster to the most common clustering for each of the data sets across their largest jobs. We limit the clusterings to those with at least 30 elements in each cluster to prevent degenerate clusterings from clouding the results. For each data set (labeled with a three letter acronym in the first column) we show two rows. The row marked “B” shows the BIC score, the value of k, the cardinality of each of the k clusters, and the number of occurrences of this clustering for the clustering having the best BIC score. The row marked “MC” shows the same information for the most common clustering. Clearly, there is a significant difference between the most common clustering and the best clustering in almost every case (the possible exception being GR1).

Another possibility is that the rare clusterings having the best BIC scores may not correspond to geographically meaningful EC maps. That is, the best BIC may correspond to a statistically meaningful solution that does not provide insight for soil zone management. Figure 3.9 shows the geographic mappings of the best BIC clusterings from Table 3.7. We assign different colors to each EC data point based on the cluster to which it has been assigned and then graph the data points based on latitude and longitude. From the figures, it is clear that the clusterings correspond to feasible zone management maps. That is, points belonging to the same cluster are often adjacent in geographic space indicating a strong EC mapping relationship.

To illustrate in greater detail, consider the CAP dataset results. CAP is a lemon field with soil consisting of clay, sandy-loam, and sandy-clay-loam. Figure 3.8a shows that jobs from `Job-5` on have very stable results with similar BIC scores. Figure3.9a shows the best clustering from `Job-2048` with 4 clusters with cardinality [2103, 500, 473, 156] and a BIC score of -8918.35.

We compare this result with the most common clustering for `Job-2048` that occurred 1445 times with a BIC score of -10169.7 and two clusters having 2169, and



(a) Most Common Clustering

(b) Differences

(c) Core Samples

Figure 3.10: Clusterings of CAP dataset: (a) most common clustering; (b) differences between the most common and the best clustering (Figure 3.9a); (c) core samples that include two sandy-clay-loam samples, clay, sandy-loam, and sandy-clay-loam-cl.

1063 elements respectively (Figure 3.10a). The visual difference between those two clusterings (Figure 3.10b) shows that most of the “disagreement” appears along cluster boundaries.

In addition, we consider how clustering results compare to the soil samples taken at the CAP field. Figure3.10c shows the core samples taken at five different locations and their soil type. Out of five core samples available, the top two in the figure (sandy-clay-loam) belong to the same cluster in both the best and the most common clustering (purple color on the graph). The other three core samples report clay in the lower left corner followed by sandy-loam and sandy-clay-loam-cl. In the best clustering, they all belong to different clusters (red, blue, yellow) while the most common clustering puts all three core samples in the same cluster (red). Thus, the best clustering corresponds more closely to a core-sample analysis than the most common (i.e. most likely determined) clustering.

Note that for each fixed set of parameters (e.g. k), we ran $1,228,800$ different experiments. This is the maximum frequency (“count” column) that can occur for a particular value of k resulting in the best or most common BIC. Table 3.7 shows that the most common clustering usually has fewer clusters (often 2 or 3) while the best clustering provides higher resolution and therefore additional information that a farmer may find useful for management.

The analysis of the other datasets is similar. In each case, the best BIC score is rare, requiring a large number of repeated trials, each with a different initialization to determine. In all but one case (GR1) the best clustering differs substantially from the most frequently occurring clustering. The best clusterings correspond to meaningful EC soil maps, and those maps correctly register with soil core samples.

3.5.4 Comparison with MZA

We next compare the Centaurus performance against "Management Zone Analysis" (MZA Fridgen et al. (2004)) for the Veris EC farm datasets. MZA is a popular methodology with concomitant software for clustering Veris EC data. Results for such clusterings are available from Fridgen et al. (2004), Odeh et al. (1992), Corwin & Lesch (2003).

Management Zone Analyst

MZA requires users to set a real-valued parameter known as the "fuzziness index", which controls the degree of specificity of the algorithm. The authors of Fridgen et al. (2004) use a value of 1.5 in their experiments, and suggest that values between 1.2 and 1.5 are appropriate for clustering soil EC measurements.

For the chosen fuzziness parameter m (for $m > 1.0$) and the maximum number of clusters K, MZA runs a single fuzzy clustering for each k $(2, \ldots, K)$. MZA scores the resulting clusterings using two metrics: Fuzziness Performance Index (FPI) Odeh et al. (1992), and Normalized Classification Entropy (NCE) Odeh et al. (1992), Bezdek (2013). FPI is a measure of the degree of separation between partitions (lower fuzziness means a higher degree of separation) while NCE measures the disorganization of each one of the fuzzy partitions. The authors of Fridgen et al. (2004), Odeh et al. (1992) suggest that the best clustering is the one with the smallest value of k that also has the smallest scores for both metrics among all clusterings. MZA computes the global covariance matrix, and employs either Euclidean, diagonal, or Mahalanobis distance. MZA computes the covariance matrix based on all the data points and uses this same covariance matrix in each iteration.

	Dataset-1	**Dataset-2**	**Dataset-3**
Centaurus	0.0%	3.6%	0.1%
MZA	0.0%	13.8%	11.6%

Table 3.8: Percentage error (out of 3,000 data points per dataset) for Centaurus and MZA on the synthetic datasets for the clustering results in Figure 3.11.

MZA disallows clustering for $k = 1$

Synthetic Datasets

We start by comparing Centaurus against MZA for the synthetic datasets. We use the number that both FPI and NCE scores report for MZA as the optimal number of clusters. We then use the respective cluster assignment (labels) to compute the error rates. Figure 3.11 shows the best assignments produced by Centaurus and MZA and Table 3.8 shows the percentage of incorrectly classified points (out of 3,000 points) in each dataset, for the same assignments.

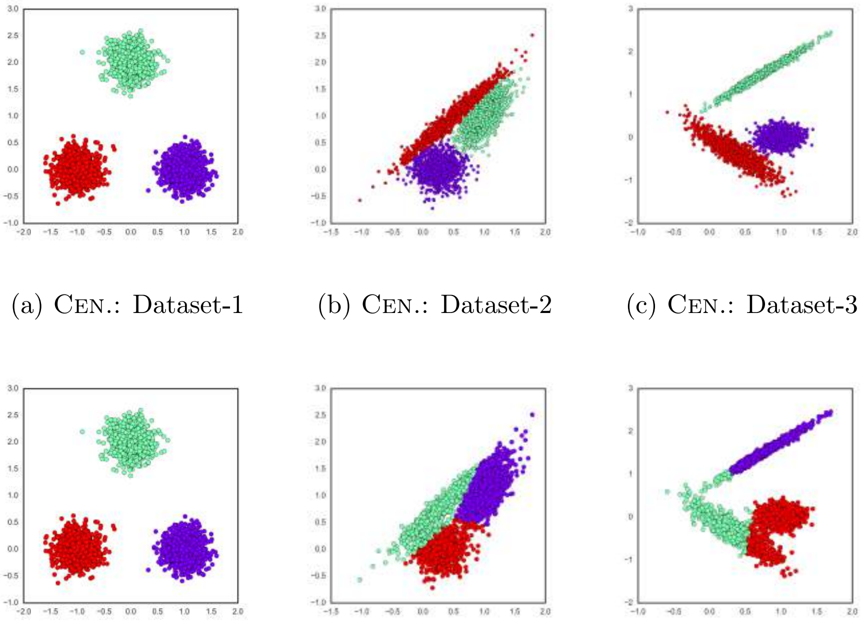
For MZA, the best assignment is achieved by Mahalanobis distance and for Centaurus the best assignment is achieved by Full-Untied. MZA clusters the Dataset-1 correctly and reports $K = 3$ as the ideal number of clusters (as does Centaurus).

For Dataset-2, MZA correctly identifies $K = 3$ but has a higher error rate of 13.8 A possible reason for this is that MZA only considers a single initial assignment of cluster centers, which in this case converges to a local minimum that is different from the global minimum. Centaurus avoids this kind of error by performing several runs (10,000 in this case, specified by $n\_exp \times n\_init$) of k-means algorithm before suggesting the optimal cluster assignment.

Dataset-3 consists of clusters with correlation across features. Centaurus provides better results than MZA for this dataset, achieving a percentage error of only 0.1 A possible reason for this is that MZA employs a global covariance matrix and does not consider Tied and Untied options as Centaurus does, which results in better label assignments.

Another limitation of MZA is that it uses a free variable, called the fuzziness parameter, and multiple scoring techniques. It is challenging (especially for novices) to determine how to set the fuzziness value even though the results are highly sensitive to this value. For the results in this section, we chose the default fuzziness parameter of $m = 1.3$ as suggested by the author Odeh et al. (1992). Furthermore, for the farm datasets, the MZA scoring metrics (NCE and FPI) do not aways agree, providing conflicting recommendation and forcing the user to choose the best clustering.

In combination, these limitations make MZA hard to use as a recommendation service for growers who lack the data science background necessary to interpret its results. Centaurus addresses these limitations by providing high enough number of k-means runs, no free parameters, and more sophisticated ways of computing the covariance matrix in each iteration of its clustering algorithm. It uses a unique scoring method to decide what is a single best clustering that will be presented to a novice user while it provides the diagnostic capabilities that are needed for more advanced users.



(a) Cen.: Dataset-1 (b) Cen.: Dataset-2 (c) Cen.: Dataset-3

(d) MZA: Dataset-1 (e) MZA: Dataset-2 (f) MZA: Dataset-3

Figure 3.11: Centaurus vs. MZA clustering recommendations for the synthetic datasets.

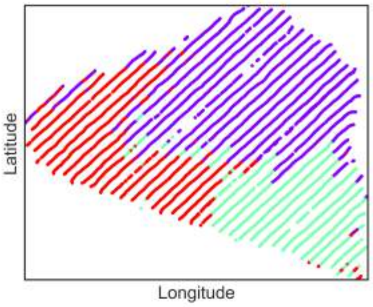


Figure 3.12: Clustering assignment for Cal Poly dataset produced by MZA based on EC2 and elevation.

Cal Poly Veris EC

K	m=1.1		m=1.3		m=1.5		m=2.0	
	FPI	**NCE**	**FPI**	**NCE**	**FPI**	**NCE**	**FPI**	**NCE**
2	0.044	0.016	0.120	**0.044**	0.265	0.093	0.475	**0.162**
3	0.028	**0.013**	0.095	0.048	0.170	0.088	0.361	0.189
4	0.027	0.014	**0.083**	0.046	**0.143**	**0.084**	**0.328**	0.207
5	**0.025**	0.014	0.087	0.053	0.166	0.105	0.370	0.255
6	0.027	0.016	0.098	0.062	0.180	0.122	0.393	0.291
7	0.031	0.019	0.099	0.065	0.173	0.121	0.386	0.304

Sedgwick Veris EC

K	m=1.1		m=1.3		m=1.5		m=2.0	
	FPI	**NCE**	**FPI**	**NCE**	**FPI**	**NCE**	**FPI**	**NCE**
2	**0.018**	**0.006**	**0.063**	**0.023**	**0.126**	**0.047**	0.413	**0.143**
3	0.020	0.010	0.081	0.040	0.145	0.074	**0.315**	0.164
4	0.025	0.013	0.080	0.044	0.140	0.081	0.324	0.201
5	0.025	0.015	0.088	0.053	0.158	0.100	0.356	0.244
6	0.026	0.016	0.091	0.057	0.172	0.116	0.383	0.281
7	0.028	0.017	0.094	0.062	0.167	0.116	0.388	0.299

UNL Veris EC

K	m=1.1		m=1.3		m=1.5		m=2.0	
	FPI	**NCE**	**FPI**	**NCE**	**FPI**	**NCE**	**FPI**	**NCE**
2	0.038	0.014	0.126	0.044	0.201	0.069	0.341	**0.117**
3	0.020	**0.010**	0.068	0.033	0.115	**0.057**	0.233	0.119
4	0.019	0.010	0.059	**0.033**	0.102	0.059	**0.229**	0.142
5	**0.017**	0.010	**0.056**	0.034	0.100	0.063	0.239	0.163
6	0.025	0.015	0.082	0.051	**0.094**	0.062	0.239	0.177
7	0.021	0.013	0.073	0.046	0.136	0.092	0.285	0.212

Table 3.9: MZA results for the farm datasets for different values of k and fuzziness coefficients (m).

Soil electrical Conductivity Datasets

We run MZA for the three farm datasets (Cal Poly, Sedgwick, and UNL) and present results in Table 3.9. For this study, we set $k = 2, \ldots, 7$ and consider fuzziness values of 1.1, 1.3, 1.5, and 2.0. The value of k is given in the first column. The table shows the FPI and NCE scores for each fuzziness value and for each k in the data columns. The lowest (considered the best) score is shown in bold.

The results show that MZA often recommends different clusterings depending upon the scoring metric and fuzziness value used. We first consider scores across values of m and k. In all cases, across datasets, NCE and FPI select $m = 1.1$ as producing the best clustering. This is in contrast to (i) the MZA default ($m = 1.3$), (ii) the values recommended by the authors (1.2-1.5), and (iii) the value for m used in the original MZA study (1.5) Fridgen et al. (2004), which all perform worse. Unfortunately, the best performing cluster size differs between NCE and FPI for both Cal Poly (top table) and UNL (bottom table). For the Cal Poly dataset (top table), NCE reports that the best clustering is ($k = 3$, row 3, column 3). FPI reports that the best clustering is ($k = 5$, row 5, column 2). For Sedgwick (middle table), NCE and FPI agree on ($k = 2$, row 2, columns 2 and 3). For the UNL (bottom table), NCE selects ($k = 3$, row 3, column 3) and FPI selects ($k = 5$, row 5, column 2).

Moreover, FPI and NCE disagree more often than they agree for these datasets. For the Cal Poly dataset (top table) both scores agree only when $m = 1.5$ suggesting that $k = 4$ (row 4, columns 6 and 7) is the best clustering. For other values of m, MZA recommends cluster sizes that range from $k = 2$ to $k = 5$. For Sedgwick (middle table) and $m = 2.0$ (columns 8 and 9), FPI selects $k = 3$ and NCE selects $k = 2$. For UNL, no FPI-NCE pairs agree on the best clustering, with MZA recommending all values of k (except 7) for different m.

Because fine-grained EC measurements (e.g. using soil core samples and lab analysis) are not available for the Cal Poly, Sedgwick, and UNL farm plots, it is not possible to compare the MZA and Centaurus in terms of which produces a more accurate spatial maps from the Veris data. Even with expert interpretation of the conflicting MZA results for Cal Poly and UNL, we do not have access to "ground truth" for the fields. However, it is possible to compare the two methods with the synthetic datasets shown in Figure 3.1.

Note that this evidence suggests Centaurus is more effective for some clustering problems but (again, due to a lack of ground truth) is not conclusive for the empirical data. Instead, from the empirical data we claim that Centaurus is more utilitarian than MZA because disagreement between FPI and NCE differing possible best clusterings based on user-selected values of m, can make MZA results difficult and/or error prone to interpret for non-expert users. MZA recommendations may be useful in providing an overall high level "picture" of the Veris data clustering, but its varying recommendations are challenging to use for making "hard" decisions (e.g. to control irrigation duration) by experts and non-experts alike. In contrast, Centaurus provides both a single "hard" spatial clustering assignment and a way to explain (in terms of maximum likelihood and BIC penalty score) why one clustering should be preferred over another and which one is "best" when ground truth is not available.

In contrast, Centaurus is able to use its variants of k-means, a BIC-based scoring metric, and large state space exploration to determine a single "best" clustering. The only free parameter the user must set is the size of the state space exploration (the default is N=2048 experiments which is 1.23M k-means runs). As the work in this study illustrates, Centaurus can find rare and relatively unique high-quality clusterings when the state space it explores is large.

A large state space (each requiring a separate "run" of the k-means algorithm) of course requires more computational power than MZA. MZA is a stand alone software package that runs on a laptop or desktop computer. In contrast, Centaurus is designed to run as a highly concurrent and scalable cloud service (via a browser) and uses a single processor per k-means run. As such, it automatically harnesses multiple computational resources on behalf of its users. Centaurus can be configured to constrain the number of resources (CPUs) it uses; doing so proportionately increases the time required to complete a job (each independent k-means run takes between 0.3s and 1s in our experiments). For this work, we host Centaurus on two large private cloud systems: Aristotle Ari (2019) and Jetstream Stewart et al. (2015), Towns et al. (2014).

3.6 Related Work

Extensive studies of k-means demonstrate its popularity for data processing and many surveys are available to interested readers Jain et al. (1999), Berkhin (2006). In this section we focus on k-means clustering for multivariate correlated data. We also discuss the application and need for such systems in the context of farm analytics when analyzing soil electrical conductivity.

To integrate k-means into Centaurus, we leverage Murphy's Murphy (1998) work in the domain of Gaussian Mixture Models. This work identifies multiple ways of computing the covariance matrices and using them to determine distances and log likelihood. To the best of our knowledge there is no prior work on using all six variants of cluster covariance computation within a k-means system. We also utilize the k-means++Arthur & Vassilvitskii (2007) work for cluster center initialization.

The research and system that is most closely related to Centaurus, is MZA Fridgen et al. (2004) —a computer program widely used by farmers to identify clusters in soil electro-conductivity (EC) data to aid farm zone identification and to optimize management. MZA uses fuzzy k-means Dunn (1974), Bezdek (2013), computes a global covariance (i.e. one covariance matrix spanning all clusters) and employs either Euclidean Heath et al. (1956), diagonal, or Mahalanobis distance to compute distance between points. MZA computes the covariance matrix once from all data points and uses this same matrix in each iteration. MZA compares clusters using two different scoring metrics: fuzziness performance index (FPI) Odeh et al. (1992) and normalized classification entropy (NCE) Bezdek (2013).

Centaurus attempts to address some of the limitations of MZA (which is only available as desktop software, does not account for poor initial cluster assignments, and places a burden on the user to determine which cluster size, k-means variant, and scoring metric to employ). We also show that although MZA provides multiple scoring metrics (Centaurus provides a single scoring metric) to compare cluster quality, the MZA metrics commonly produce different "recommended" clusterings.

The authors of x-means Pelleg et al. (2000) use Bayesian Information Criterion (BIC) Schwarz (1978) (which Centaurus also employs) as a score for the univariate normal distribution. Our work differs in that we extend the algorithm and scoring to multivariate distributions and account for different ways of covariance matrix computation in the clustering algorithm. We provide six different ways of computing covariance matrix for k-means for multivariate data and examples that illustrate the differences.

Different parallel computational models have been used in other works to speed up the k-means cluster initialization Bahmani et al. (2012), or its overall runtime(Zhao et al. 2009). Our work differs in that we provide not only a scalable system but include k-means variants, flexibility for a user to select any one or all of the variants, as well as a scoring and recommendation system. Finally, Centaurus is pluggable enabling other algorithms to be added and compared.

3.7 Summary

In this chapter, we present Centaurus, a scalable, easy to use, cloud service for clustering multivariate and correlated data. Centaurus simplifies selection of k-means clustering variants, provides a recommendation of the best variant, and enables users to visualize their results in multiple ways. Centaurus leverages cloud resources and services to automatically deploy, scale, and score k-means clustering jobs.

We empirically evaluate Centaurus using synthetically generated and real datasets and compare it to the popular MZA clustering tool. Our results show that Centaurus provides better results than MZA and precludes many of its limitations. In addition, we analyze the sensitivity of k-means clustering to cluster degeneracy, choice of distance metric, variance in results based on the correlation computation techniques, and an analysis of performance of k-means algorithm on farm datasets when the number of experiments increases to up to one million. Our results indicate that for EC soil measurement data, k-means is effective when used in a computationally intensive way (i.e. many repeated trials), using multiple variants, while filtering for non-degenerate solutions.

Finally, Centaurus model selection facilitates decision support for growers by defining management zones boundaries. By visualizing the differences between solutions, growers can decide where to take additional samples in order to obtain more accurate soil maps. We use this system and tools to analyze 10 different field EC datasets and provide statistics and analysis of each.

Chapter 4

Sensor Synthesis

The Internet of Things (IoT) is quickly expanding to include every “thing” from simple Internet-connected objects, to collections of intelligent devices capable of everything from the acquisition, processing, and analysis of data, to data-driven actuation, automation, and control. Since these devices are located “in the wild”, they are typically small, resource-constrained and battery powered. At the same time, low latency requirements of many applications mean that processing and the analysis must be performed near where data is collected. This tension requires new techniques that equip IoT devices with more capabilities.

One way to enable IoT devices to do more is to use integrated sensors to `estimate` the measurements of other sensors, a technique that we call *sensor synthesis*. Since the number of sensors per device is generally bounded by design constraints (e.g. space or power limitations), sensor synthesis makes it possible to free up resources in IoT devices for other sensors, particularly those that are less amenable to synthesis, and to reduce the monetary cost of sensing.

Many, if not most, IoT systems for precision agriculture depend on and integrate measurements of real time, atmospheric temperature. Temperature is used to inform and actuate irrigation scheduling, frost damage mitigation, greenhouse management, plant growth modulation, yield estimation, post-harvest monitoring, crop selection, and disease and pest management, among other farm operations Ghaemi et al. (2009), Stombaugh et al. (1992), Ioslovich et al. (2016), Roberts et al. (2013), Gonzalez-Dugoa et al. (2011). Measuring and predicting temperature accurately is challenging due to variation across farm micro-climates where local temperature can deviate from the surrounding area which is typically measured at mesoscale. Measuring temperature for large number of micro-climates on a farm can be prohibitively expensive with extant weather stations and sensors (which can be hundreds to thousands of dollars).

In this chapter, we explore the use of sensor synthesis to estimate outdoor temperature on farms using the processor (CPU) temperature of simple, inexpensive single board computers (SBCs; e.g. those in the Raspberry Pi family Ras (2018) or micro-controllers such as those in the Arduino family Ard (2019)). Our approach estimates outdoor temperature from the on-board processor temperature sensor that these devices support (and use for system health checks) and which is available via their respective hardware/software interfaces. Such devices cost around $5, are battery or solar powered, and can be packaged in small, inexpensive, weatherproof enclosures, making them practical for use in moderate and large scale geographic deployments.

To investigate how well the processor temperature of these devices can be used to predict outdoor temperature, we have developed an on-farm IoT system in which we place single-board computers *in-situ* throughout the farm. The devices transmit measurements of CPU temperature wirelessly to wall-powered, indoor, edge cloud systems Elias et al. (2017). We first calibrate the device CPU temperature against a co-located, high-quality temperature sensor using linear regression. We then remove the temperature sensor at each remote location.

The edge cloud computes a prediction of outdoor temperature for each device/location for each CPU measurement that it receives from the device. It does so by applying the regression coefficients from the calibration period to the CPU temperature measurement. To account for autocorrelation in the time series, we investigate the use of Single Spectrum Analysis (SSA) Golyandina & Zhigljavsky (2013) to extract a smooth "signal" from the data prior to performing linear regression and compare this approach to non-smoothing. We also evaluate the impact of using different amounts of training data (period over which regression is performed) and calibration durations. Finally, we integrate different outdoor temperature sources that include device-attached sensors (e.g. thermistors), high-end, on-farm weather stations, and remote WeatherUnderground Wea (2019) stations.

We first consider two different configurations. The first is a "limit study" in which we continuously update the regression coefficients using a co-located temperature sensor, to compute a one step ahead (5 minute) prediction. This configuration represents an upper bound on the efficacy of predicting outdoor temperature from processor temperature. Using a second configuration, we consider a practical application of our approach in which the edge cloud estimates the outdoor temperature (at the device) using information from the initial calibration period and the CPU temperature measurements reported by the device every 5 minutes.

Next, because sensor synthesis is based on computed estimates rather than actual measurement, it introduces the possibility of additional error beyond measurement error. To address this, we examine how a larger ensemble of measurements improves the accuracy of "synthetic" temperature measurement while, at the same time, not requiring the use of powerful computational resources.

Reducing the prediction error is not only academically interesting, rather, precision has a direct impact on the cost and efficiency of what has become known as precision agriculture or precision farming. In precision agriculture, farmers use technology to increase the efficiency of farming techniques increasing crop yields and reducing costs. Having more precise temperate data reduces the cost of frost prevention (by avoiding the unnecessary use of frost mitigation systems (e.g. fans)) and prevents excessive resource use without negatively impacting crop production. Consequently, we believe that our approach can contribute to improved farming outcomes, enable water and energy savings, and help reduce carbon emissions, by providing high-quality data to data-driven, IoT-based agricultural applications.

We then extend our approach to use a combination of processor temperatures from multiple devices and outdoor temperature from high-quality, remote weather stations to train a multiple linear regression model. We use this model to estimate the future outdoor temperature at a particular device location that is not part of the model. We also investigate the efficacy of computationally simple smoothing techniques (based on sliding window reductions) to reduce noise.

We also investigate how well our approach performs when the processors on the devices experience load. Load may affect processor temperature and thus negatively impact the accuracy of our outdoor temperature estimates. To do so, we develop techniques that successfully deal with the perturbations caused by load variability, which is an important requirement to make our sensor synthesis practical in the field (and which went uninvestigated in prior work).

Finally, to evaluate the practical effectiveness of this extension, we deploy multiple Raspberry Pi Zero devices in an agricultural setting where citrus trees are grown. To compare the values of our synthesized sensors with measured temperature values, we equip the devices with temperature sensors, which we use to establish ground truth. We evaluate different combinations of explanatory variables with and without smoothing, and with and without a computational load on the processor, as part of our multiple linear regression models. Our results show that using this approach, we can reduce the mean absolute prediction error (MAE) and that it is robust to processor load.

We first overview the basic framework in 4.1 and then detail the extension for using multiple linear regression in 4.3. We empirically evaluate these advances in 4.2 and 4.4, present related work in 4.5, and summarize our contributions in 4.6.

4.1 Approach

In this Chapter, we investigate the relationship between processor temperature (henceforth simply referred to as CPU temperature) embedded in single-board computers, and the atmospheric temperature that surrounds them. Our goal is to place these computers *in-situ* in agricultural settings for use as thermometers. By doing so, we can leverage their measurements to actuate and control a wide

In linear regression, an explanatory variable is an independent variable that is used to predict a value. In our context, the independent variables are the CPU temperatures and weather station temperature (gathered from a weather station that is in the area of the SBCs but not necessarily co-located), which we use in the model to predict the synthesized sensor. Explanatory variables are also called predictors in the literature. Since we use multiple regression, we use more than one predictor in our synthesis.

range of IoT-based farm operations, while driving down the cost of implementing such solutions at scale.

Examples of such farm operations include irrigation scheduling and frost damage mitigation strategies. For automatic irrigation scheduling, real-time temperature measurements are used to compute localized estimates of evapotranspiration (ET), which indicates the amount of water that has been lost (since the last irrigation) and that must be replaced via irrigation. Both under and over watering can decrease productivity, destroy crops, and degrade soil health. Irrigation scheduling is the most common form of IoT and data-driven decision support system on farms and is especially important for managing farms in drought stricken regions.

The terms “frost” or “freeze” are used by the public to describe a meteorological event that causes freezing injury to crops and other plants, when the air temperature falls below the tolerance level of the specific plant Levitt et al. (1980). The ability to predict the onset of frost, its duration, and the specific locations where frost will occur is of tremendous value to the agricultural industry. In the USA, there are more economic losses to frost damage than to any other weather-related phenomenon White & Haas (1975). Active frost protection strategies include application of water, use of wind engine-driven machines and heaters, and/or some combination of these methods, all of which are extremely labor intensive and costly for growers. If the onset or duration of frost is mis-predicted, the cost of any mitigation strategies applied is lost. Alternatively, incorrectly predicting that a freeze will not occur to save these costs can devastate a crop. For this reason, current practice is conservative, passing any unnecessary mitigation costs on to the consumer in exchange for a low risk of crop loss.

In both operations, accurately measuring and predicting temperature in real-time is required. However, temperature is not uniform and can vary widely across a farm, requiring that operations account for very localized differences to obtain measurable outcomes. Micro-climates can occur in large numbers due to topographic differences, surrounding structures, ground cover, plant maturity, and nearby bodies of water. Measuring temperature across vast numbers of micro-climates is costly and labor intensive given the price of high quality sensors and complexity of sensor management (data extraction, advanced analytics, connection inferences, and prediction). Many IoT vendors provide managed services to reduce this complexity for growers, but these services are expensive, require that data be transmitted off-farm to cloud based applications via cellular, and impose a recurring subscription fee on farmers in order to view their data. As a result, IoT advances have not achieved wide spread uptake in the agriculture, despite their potential.

As part of the UCSB SmartFarm effort Krintz et al. (2016), we have investigated ways of reducing cost and complexity of temperature-based IoT solutions, while maintaining accuracy and robustness. SmartFarm implements a low cost, on-farm edge cloud comprised of multiple Intel Next Unit of Computation (NUC) machines Int (2019*a*). Using open source cloud software (AppScale Krintz (2013) and Eucalyptus Nurmi et al. (2009)), we design the edge clouds to be self-managing and to perform a wide range of data analytics on farm data, thereby precluding the need to transmit data off-farm and keeping cost, complexity, and latency low Krintz et al. (2016), Elias et al. (2017).

We use SmartFarm and single-board computers to provide accurate, real time estimates of micro-climate temperature across a farm. To do so, we place battery or solar powered devices *in-situ* in various settings and configurations within inexpensive enclosures. The devices transmit their CPU temperature wirelessly (via 802.11 or Zigbee) to an on-farm edge cloud every 5 minutes. As ground-truth, we consider co-located (device-attached) DHT digital sensors (thermistors Ada (2018)), high-end, on-farm weather stations, and WeatherUnderground remote weather service Wea (2019), which farmers commonly use to estimate temperature.

Figure 4.1 shows a two week time series trace (starting May 10^{th}, 2018) of CPU temperature (Pi Zero CPU) from a Raspberry Pi Zero, the outdoor temperature from an attached digital DHT22 temperature sensor (DHT Temp), and the outdoor temperature from a nearby WeatherUnderground (WU) station (WU Temp).

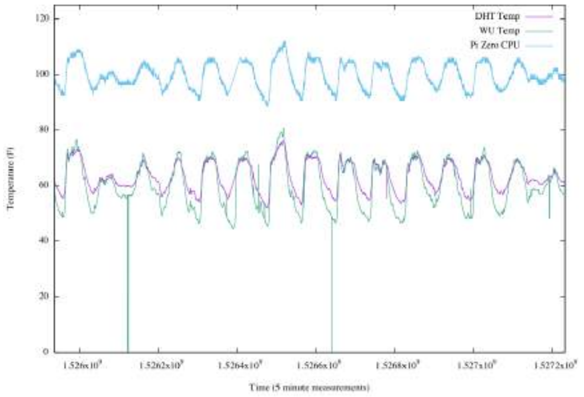


Figure 4.1: Two week time series trace of outdoor (device-attached DHT sensor and a nearby (WU) station) and 5-minute CPU temperature data in Fahrenheit from a Pi Zero single board computer (Pi1 in the Results section)

WU measures outdoor temperature at 10 meters and the Pi Zero is at a 1 meter altitude. The Pi Zero is in a plastic enclosure with a small, covered hole from which the DHT wires exit; the DHT sensor is outdoors and hanging freely. The device is located outdoors under constant shade in Goleta, CA. We refer to this device as Pi1 in later sections of the paper. The average CPU temperature on the Pi Zero during this period is 99.71 °F with a standard deviation of 4.69. The mean and standard deviation for the DHT sensor and WU station are 61.93 (5.79) and 60.20 (8.35), respectively. DHT and WU temperature is similar but WU exhibits data dropout (0 values), more variance, and more extreme temperatures.

From this graph, there appears to be a correlation between CPU temperature and both outdoor temperature measures for this location. The CPU values exhibit

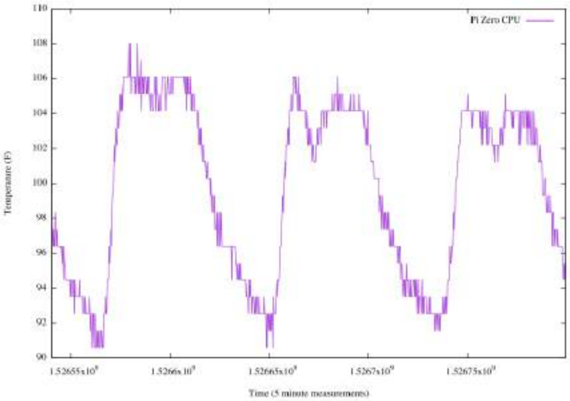


Figure 4.2: Two day time series sub-trace from Fig. 4.1 of 5-minute Pi Zero CPU temperature ($^\circ F$)

small oscillations or noise (making the curve appear darker). A sub-portion (2 days starting May 17^{th} at midnight) of the CPU data alone is shown in Figure 4.2 using a different scale. We note that there are some discrepancies in the shape of different curves. We observe similar relationships using other types of devices, locations, and sources for ground-truth (e.g. DHT or WU) temperature measurements. We next investigate how accurately we can predict outdoor temperature (of these different sources) using CPU temperature of these devices.

4.1.1 Predicting Air Temperature from CPU Temperature

The data in Figure 4.1 is typical of the outdoor SmartFarm installations we have deployed suggesting that linear regression would be an effective way to predict outdoor temperature from CPU temperature. Because each single-board computer is running a multi-user operating system (Linux in this study), however, the CPU temperature exhibits fluctuations that we do not observe in the outdoor temperature. Further, because these fluctuations are caused by programs that are running on the computer, they are autocorrelated in time.

To account for this autocorrelated "noise" in the CPU temperature series, we apply Single Spectrum Analysis (SSA) Golyandina & Zhigljavsky (2013) to the CPU series before performing regression. SSA decomposes an autocorrelated time series into "basis time series" which are analogous to principle components Abdi & Williams (2010), Wold et al. (1987). By summing the most significant basis series (based on a clustering of the series by eigenvalues), SSA can extract a smooth "signal" from a noisy time series. To do so, SSA requires the number of lags over which autocorrelation is significant to be supplied as a parameter.

To investigate the accuracy with which it is possible to predict outdoor temperature, our system runs multiple smoothing passes, each with a successively larger number of lags up to 12 (1 hour). During daylight and nighttime hours, outdoor temperature can be autocorrelated for several hours, but during the early morning (diurnal heating) or early evening (diurnal cooling) the significant autocorrelation duration is significantly less. For each lag we compute the coefficient of determination (R^2) for a regression covering a previous window of time and choose the number of lags that generates the highest R^2 value. We refer to this window as the *training window (TW)*. Typically (but not always) the best R^2 value is for 6 lags indicating that the significant autocorrelation in the CPU temperature series covers about 30 minutes.

The method recomputes both the smoothed series and the regression coefficients every time a new outdoor measurement is generated (every 5 minutes in this study). Thus the approach is a "piecewise" linear regression approach where the data is re-smoothed using the "best" number of lags (based on R^2 value) before each regression.

When a new CPU value arrives, we use the regression coefficients to compute a prediction of outdoor temperature. Prior to applying smoothed regression coefficients, we append the new CPU value to the training window (and remove its head, effectively sliding the window right). We compute the prediction using the smoothed CPU value (last value of the smoothed training window). We then compare this value to the actual outdoor measurement to compute the absolute difference and square difference as the error.

To summarize, the steps of our algorithm are as follows.

1. Match the temperature and CPU series using the nearest timestamps

2. Divide the matched series into a training window (TW) and test window (TE)

3. If SSA is used, smooth the CPU series using different smoothing parameters.

4. Compute the regression coefficients, i.e. y-intercept and slope, for each to model the linear relationship between temperature (the dependent variable) and CPU (the explanatory variable) in TW

5. Extract the best parameterization for each smoothing technique using largest coefficient of determination (R^2)

6. For each matched pair of measurements in the TE, append the pair of measurements to TW and remove the first pair in TW, effectively sliding the training window right

7. Predict outdoor temperature by applying the regression coefficients to the latest CPU value (smoothed or non-smoothed), and compute and record the error (difference from actual, matched outdoor temperature); for the smoothed case, we smooth across the updated TW.

8. Repeat starting at Step 3 above and end when there are no more new measurement pairs in the test window (TE)

We refer to this configuration as a "limit study" because we believe that it provides us with an upper bound on the efficacy of our approach. However, it requires that the device and temperature sensor be co-located so that we can continuously update the regression coefficients.

We therefore consider a second configuration that does not continuously update the coefficients using the most recent temperature data. We refer this configuration as a "practical application" of our approach. For this configuration, we co-locate a temperature sensor with each device for a short, fixed period of time, which we refer to as the *calibration period.* We then remove the temperature sensor (and use it to calibrate other *in-situ* devices as needed). We apply the regression coefficients from the calibration period (which do not change) to CPU measurements reported by the device to predict the outdoor temperature at the device.

For the calibrated results, we use the algorithm above with minor modifications. The remote device transmits only its CPU measurement values via low-power radio to the edge cloud every 5 minutes. The edge cloud keeps a CPU history from the device for the same duration as the calibration period. It smooths these values if necessary and chooses the best-performing smoothing parameterization (e.g. lags) using R^2. The edge cloud then computes a prediction using the last CPU value it received (smoothed or non-smoothed). For the results in this paper, we compare this prediction against that from a co-located temperature sensor. However, we only use data from this co-located sensor to compute the prediction error after the devices have been "separated".

4.2 Empirical Evaluation

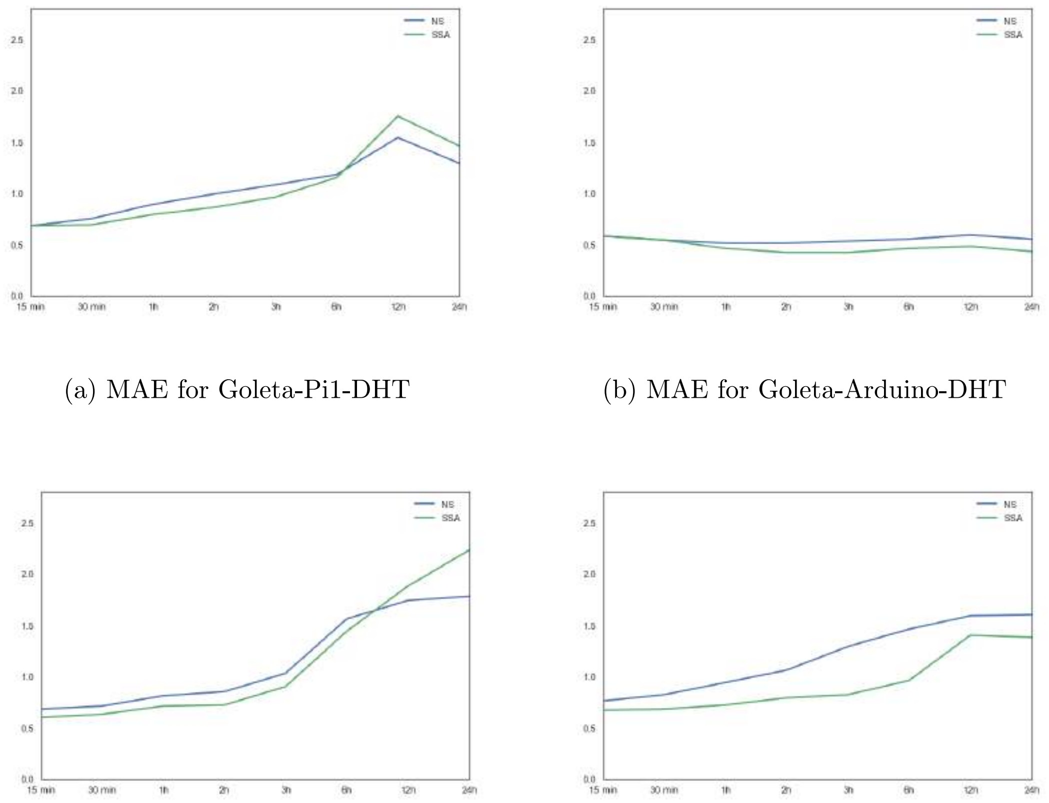
4.2.1 Devices and Data Sets

The devices we consider as temperature sensors in this study include the Raspberry Pi Zero Version 1.3 with a 1GHz ARMv7 processor and 512MB of RAM and the Raspberry Pi 3 Model B with a 1.2GHz ARMv8 processor and 1GB of RAM. Each Pi is equipped with 32GB of storage. We also evaluate an Arduino Uno with a ATmega328P processor with 2KB of data memory and 32KB of program memory, and an Intel Next Unit of Computation (NUC) with 8 Intel Core i7 processors (each 2.6GHz), 32GB of memory, and 1TB of SSD storage. The devices cost $5, $35, $22, and $1619 for the Pi Zero, Pi3, Uno, and NUC, respectively.

The Pi devices read their CPU temperature via a "thermal zone" which reports temperature in Celsius. The Uno reads the internal analog to digital converter using the 8th channel of the micro-controller (currently without the noise reduction feature). The NUC reads its CPU via the `sensors` utility. We convert all values to degrees Fahrenheit for this study.

The locations include a residential backyard in Goleta, CA, an experimental citrus farm at the Lindcove Research and Extension Center (LREC) in Exeter, CA, and an experimental almond farm on the campus of the California State University, in Fresno, CA. There are multiple Pi Zero devices at the Goleta location (referred to as Pi1, Pi2, Pi4, and Arduino, prefixed with "Goleta-" in the results section), a Pi Zero (LREC-PiZ) and Pi 3 (LREC-Pi3) at LREC, and a NUC at Fresno State (Fresno-NUC). All devices are in shaded, weather proof enclosures outdoors; the NUC is in a tin shed housing a powered irrigation pump next to the almond orchard. Each location is very different in terms of its vegetation and topography. LREC is located in the foot hills of the Sierra mountains; the Fresno State farm is flat and in the central valley of California; and the Goleta residence is near the ocean.

We measure atmospheric temperature (ground truth measurements used for calibration and empirical evaluation of accuracy) using device-attached temperature (AM2302 DHT22 Ada (2018)) sensors which we refer to as DHT for Goleta devices, a high end weather station at LREC called the Flux tower, and the nearest WeatherUnderground (WU) station (station 30) in Fresno. We also consider a WeatherUnderground station (station 8) for Goleta devices.



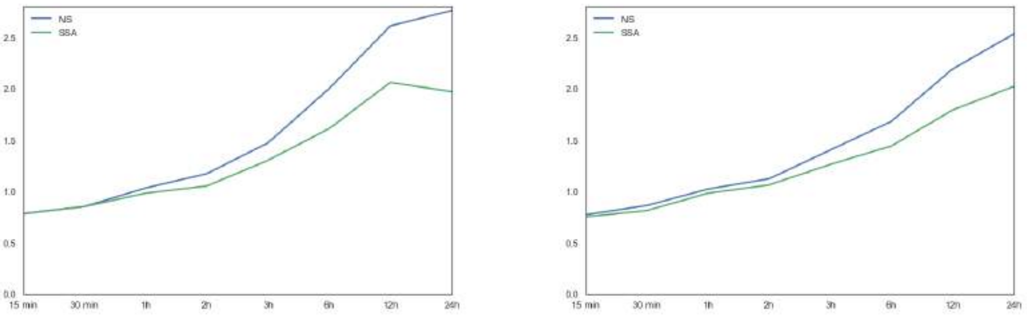
(a) MAE for Goleta-Pi1-DHT

(b) MAE for Goleta-Arduino-DHT

(c) MAE for Goleta-Pi1-WU

(d) MAE for Fresno-NUC-WU

Figure 4.3: Mean Absolute Error in degrees Fahrenheit for predictions of outdoor from CPU temperature of different devices, locations, and sources of ground-truth temperature (DHT= high quality temperature sensor; WU=WeatherUnderground; LREC=high-end on-farm weather station) for two methods: Non-Smoothing (NS) and Single Spectrum Analysis (SSA).



(e) MAE for LREC-Pi3

(f) MAE for LREC-PiZ

Figure 4.3: Continued from Previous Page: Mean Absolute Error in degrees Fahrenheit for predictions of outdoor from CPU temperature of different devices, locations, and sources of ground-truth temperature (DHT= high quality temperature sensor; WU=WeatherUnderground; LREC=high-end on-farm weather station) for two methods: Non-Smoothing (NS) and Single Spectrum Analysis (SSA).

4.2.2 Regression, Prediction Error, and Training Window

We begin by examining the effect of smoothing on each regression as part of a “limit study”. To do so, we compare SSA and no smoothing over a number of different training window sizes. As described previously, we use the regression coefficients for the number of lags for SSA that results in the highest R^2 value. We detail the effect of using smoothing and training window size to enhance regression on temperature *prediction.* At time step t we predict the outdoor temperature at time step $t + 1$ (5 minutes later). Since an application may need the temperature at an arbitrary moment in time (and not on a precise 5-minute periodicity), this prediction error serves as an upper bound on the error that an application which is not time synchronized with the measurement system might experience. We then compare different sources for predictions (locally attached DHT vs Internet-accessible WeatherUnderground) and we conclude with results showing the application of our approach in a practical IoT setting.

4.2.3 Prediction Error

In Figure 4.3 we show the Mean Absolute Error (MAE) for the one-step ahead prediction as a function of history size . Each graph compares the effect on

Most typically, the prediction error is presented as the Mean Square Error (MSE) or the Root Mean Square Error (RMSE) in an error analysis. While these statistics offer insights into the distributional properties of the errors, our experience with professional agricultural personnel has led us to concentrate on the MAE as a practical error metric since it can be interpreted as prediction accuracy of different smoothing methods for the different locations and devices for a prediction period of 3 days. The x-axis is the training window size; the y-axis shows errors in $^\circ F$.

From the graphs in Figure 4.3, we see that SSA improves prediction accuracy compared to the absence of smoothing (NS). In this study, 15 minutes corresponds to 3 measurements. When the temperature is slowly changing (e.g. the CPU temperature does not change over a 15 minute period) regression becomes numerically unstable (i.e. the co-variance matrix has values that are nearly zero on the diagonal). Compared to Fresno or Goleta, for example, the CPU temperatures at LREC is more stable since the devices are sited near a large irrigation reservoir. SSA smooths the previous 3 measurements more than the other methods, occasionally generating regression coefficients that are very large or numerically infinite (i.e. NaN) as a result of trying to invert the co-variance matrix. Our system detects this condition and disables smoothing when it leads to a failed regression.

Also note that the errors are relatively small. All of the locations we have tested are located in California and during the prediction periods, the temperature varied from the mid 40s to the mid 80s $^\circ F$. In each case, the MAE error is under $1^\circ F$ for a TW of 1 hour or less. The Arduino Uno (Goleta-Arduino-DHT) produces the lowest error and the error does not grow with window size. We believe this is

how far “off” the measurements are “on the average.” We omitted the RMSE results in favor of brevity.

	Pi1		Pi2		Pi4	
h	NS	SSA	NS	SSA	NS	SSA
1	5.5	4.5	6.4	6.5	10.4	14.6
4	5.5	4.5	2.6	2.4	1.9	1.6
8	2.7	2.6	1.5	1.5	1.4	1.4
12	2.2	2.2	1.5	1.6	1.5	1.6
24	1.4	1.4	1.4	1.4	1.5	1.5
48	1.5	1.6	1.4	1.4	1.3	1.3
72	1.3	1.3	1.4	1.4	1.3	1.4
96	1.3	1.4	1.4	1.4	1.3	1.4
168	4.8	4.8	1.4	1.4	1.3	1.3
336	4.8	4.8	1.4	1.4	1.3	1.3

Table 4.1: Mean Absolute Error ($^{\circ}F$) with No Smoothing (NS) and SSA for different calibration periods in hours (h).

due to the very consistent and slowly changing temperature of the location during the prediction period (i.e. it is nearer to the ocean than the other Goleta devices and the 3 day prediction period is in May vs March for the other locations). The accuracy of our approach is similar regardless of location (e.g. Goleta, Lindcove (LREC), or Fresno) and source of ground truth temperature measurement (DHT, Flux tower (LREC), or WU) for a TW of 1 hour or less.

4.2.4 Practical Application

The data and analysis presented in the previous subsection show the minimum error that is possible. That is, they verify that it is possible to predict the outdoor temperature from the internal CPU temperature sensor with a high degree of accuracy in a variety of meteorological settings. To be practically useful, however, the technique must be able to predict outdoor temperature without the presence of an outdoor thermometer (i.e. from CPU temperature alone). That is, our goal is to investigate whether we can use the CPU temperature sensor (which will be present by virtue of the need for a controller) as a replacement for a localized outdoor thermometer.

Specifically, in a practical application of this technique, with no outdoor thermometer, it is not possible to perform a regression at each time step using the current outdoor temperature reading. Instead, our approach is to generate a regression coefficients from a *calibration period* that we then use over a later prediction period. We site single-board computers in each location with an attached DHT outdoor temperature sensor for a fixed, continuous calibration period. Then we remove the DHT sensor (so it can be used for another calibration) and estimate outdoor temperature from the computer's CPU temperature using the regression coefficients we computed at the end of the calibration period. Table 4.1 shows the Mean Absolute Errors for different calibration periods and three Pi Zero devices.

Pi1, Pi2, and Pi4 are all Raspberry Pi Zero single-board computers with externally attached DHT temperature sensors. All three were located in the same outdoor setting in Goleta, California. We chose a random date between January 1^{st}, 2018 and May 15^{th}, 2018 in each case to use as the start of the test/prediction period. We installed the Arduino too late to include in this study, but we plan to include it once we collect sufficient data. In each experiment, we use a trace of the DHT external measurements and the corresponding CPU measurements over a fixed calibration period (shown in column 1, measured in hours) to compute a set of regression coefficients. We then use the coefficients to predict the DHT measurements from the CPU measurements (without re-regressing) for the next two weeks following the calibration period. Columns 2 through 7 show the Mean Absolute Error (MAE) during the measurement period immediately following calibration without smoothing and with SSA for the calibration regression. Thus, this table shows the errors when one set of regression coefficients is used to predict the next two weeks of outdoor temperature (as a function of calibration period).

While SSA improves the errors in the piecewise regression case (cf Figure 4.3), it is less effective when one set of coefficients must be used over a long period of time when a sufficiently long calibration period is available. Note that for some short calibration periods, SSA can improve accuracy, but only when there is sufficient variation to maintain numerical stability in the regression. Further,

	Pi1		Pi2		Pi4	
h	NS	SSA	NS	SSA	NS	SSA
24	1.1	1.4	2.5	2.1	1.3	1.2
48	1.2	1.6	0.7	0.9	0.9	1.3
72	1.1	1.2	0.7	1.0	1.3	0.9
96	1.1	1.2	0.8	1.0	0.9	1.0
168	4.1	4.1	0.7	0.7	0.8	0.8
336	4.1	4.1	0.7	0.7	0.8	0.8

Table 4.2: Mean Absolute Error in degrees Fahrenheit with No Smoothing (NS) and SSA for different calibration periods (measured in hours (h)) using data from noon to 3 PM.

	Pi1		Pi2		Pi4	
h	NS	SSA	NS	SSA	NS	SSA
24	1.1	1.4	1.4	1.4	1.3	1.4
48	1.2	1.4	1.4	1.4	1.3	1.3
72	1.1	1.2	1.5	1.4	1.3	1.3
96	1.3	1.4	1.5	1.4	1.3	1.3
168	4.1	4.1	1.5	1.5	1.4	1.4
336	4.1	4.1	1.6	1.6	1.4	1.3

Table 4.3: Mean Absolute Error in degrees Fahrenheit with No Smoothing (NS) and SSA for different calibration periods (measured in hours (h)) using data from 10 PM to 7 AM.

the calibration period should include at least one full diurnal cycle to be effective. Finally, the minimum error is consistently 1.3°F or 1.4°F.

Finally, not all time periods during a diurnal cycle may be needed for certain applications. As part of SmartFarm, for example, we are developing a new algorithm for computing localized evapotranspiration (ET) Penman (1948). ET is an often-used metric for computing crop water stress or water requirements and it is typically based on meteorological measurements that cover large areas (e.g. a county or zip code). ET computations rely, in part, on outdoor temperature measured during "solar max" – typically between noon and 3 PM in North America. Similarly, frost prevention using wind machines mixes warm air aloft (e.g. at 10 meters) with colder air that has settled near the ground during the nighttime hours (e.g. between 10:00 PM and 7:00 AM). Thus, it may be that it is possible to obtain more accurate measurements by including only those hours that are of interest during a diurnal cycle.

Tables 4.2 and 4.3 show the MAE for non-smoothed and SSA calibration using only data gathered from noon to 3 PM and from 10 PM to 7 AM respectively. We show only results for calibration periods of at least 24 hours since the calibration period must span at least one diurnal cycle. In most cases (particularly for the solar max predictions) the best prediction (lowest MAE) improves when we use only the periods of interest for the regression. However, the improvements are small in absolute terms (often $0.1°F$). We have yet to determine whether the additional accuracy is necessary for either localized ET calculation or frost prevention. Doing so is the subject of the on-going SmartFarm work that is leveraging this technique.

4.3 Employing Multiple Linear Regression Models

In the prior sections, we use univariate linear regression to estimate the outdoor temperature based on the CPU temperature of a single co-located SBC. In two of the experiments, (non-smoothed and smoothed with the single spectrum analysis Golyandina & Zhigljavsky (2013)), the model does not perform as well when the training window is smaller than 6h ($4.5 - 14.6°F$) or larger than one week ($1.3 - 4.8°F$). The reason for this is that because the technique uses computed estimates rather than actual measurement, it also introduces additional error beyond measurement error.

We next investitgate novel ways of reducing this error, explore the efficacy of alternative smoothing techniques, and evaluate the impact of processor load on prediction. To reduce this error, we consider processor temperature measurements from multiple SBCs (deployed in other on-farm microclimates), and outdoor temperature from a remote weather station, as possible predictors. We use the term processor and CPU interchangeably throughout.

4.3.1 Deployment and Datasets

We deploy four Raspberry Pi (RPi) Zero Ras (2018) devices (named Pi1, Pi2, Pi3, and Pi4) equipped with temperature sensors, at different locations (microclimates) in an agricultural setting (citrus trees). We place a pair of RPis within 3 feet of each other, in two different trees, spaced 10 feet apart. Pi1 and Pi2 monitor tree #1 and Pi3 and Pi4 monitor tree #2. Each device is housed in an inexpensive plastic enclosure and has an on-board processor temperature sensor that is part of its hardware/software interface.

The devices read their processor temperature sensor value every 5 minutes and can process, store, or wirelessly transmit their measurements. We label the measurements CPU-1, CPU-2, CPU-3, and CPU-4, for the CPUs of Pi1 through Pi4, respectively. The RPi devices then transmit the measurements to an on-farm computer for aggregation and analysis.

Each RPi is additionally equipped with an AM2302 DHT22 digital temperature and humidity sensor Ada (2018), which we use to measure ground truth. The devices read and transmit these values every 5 minutes (labeled DHT-1, DHT-2, DHT-3, and DHT-4, with temperature value DHT-{i} representing the temperature measured by the DHT22 sensor attached to the Pi{i}) along with their CPU temperature readings to a remote analysis system. We only use this DHT22 data as ground truth (to compute prediction error), i.e., it is not used as part of modeling or prediction.

Finally, we also consider the use of freely available, high-end weather station data from the Internet weather service WeatherUnderground Wea (2019). The closest weather station is 2640 feet (800m) away from our field deployment. We collect the temperature reported by the WeatherUnderground station closest to the deployment site every five minutes (labeled WU-T). We align the measurements (CPU, DHT22, and WU) using the nearest timestamp. If there is data dropout, i.e, if one of the three temperature values is missing, we skip all measurements for that five-minute interval.

4.3.2 Linear Regression Models

We model the outdoor temperature that surrounds a single RPi, using one or more predictors. Predictors can include the CPU temperature of RPi itself, the CPU temperature of neighboring RPis, and the outdoor temperature reported by a high-quality, remote weather station. We estimate model parameters $\theta \in \mathbf{R}^n$ by minimizing the residual sum of squares:

$$RSS(\theta) = (y - X\theta)^T(y - X\theta)$$

where $y_i \in \mathbf{R}, i \in \{1, \ldots, N\}$ represents the ground truth outdoor temperature and $X \in \mathbf{R}^{N \times n}$ represents the entire training set, where each row $x_i \in \mathbf{R}^n$ represents the values that predictors take, and n is the number of predictors.

To evaluate this approach, we analyze models with testing windows of size one hour to two weeks, which correspond to 12 and 4032 data points respectively. To measure error, we compute the mean absolute error (MAE) (versus R-squared) because of its direct utility in our IoT agriculture applications. In particular, we are interested in using the models to make predictions and not in their explanatory power. We compute MAE as the average absolute distance between estimated temperatures and their corresponding ground truth values.

Finally, we evaluate the efficacy of smoothing the training data prior to performing regression. We investigate rolling mean, minimum, and median smoothing methods. In our experiments, rolling mean produces the smallest error for the datasets we investigate. We thus report results using only this smoothing technique, for brevity. To implement rolling mean, we use a window of size w and replace each element with the mean value of the previous w elements including the current element. More formally, we replace X in the RSS equation with S

where

$$s_{ij} = \begin{cases} \sum_{l=j-w}^{j} \frac{x_{il}}{w} & j >= w \\ \\ \sum_{l=o}^{j} \frac{x_{il}}{j} & j < w \end{cases}$$

For all the experiments presented in Section 4.4 we use a window size $w = 6$, which corresponds to 30 minutes.

4.4 Evaluation of the Efficacy of Using Multiple Linear Regression Models for Sensor Synthesis

In our experiments, we use four RPi-based, single board computers (SBCs) deployed outdoors as described in Section 4.3.1. We denote the processor temperature measurements from each as CPU-1, CPU-2, CPU-3, CPU-4. We refer to the outdoor temperature measurements from a nearby WeatherUnderground stations as WU-T.

The goal of this evaluation is to illustrate the degree to which it is possible to make an accurate prediction of outdoor temperature based on a combination of CPU temperature measurements and temperature measurements from the WeatherUnderground station. In this study, “ground truth” – the true outdoor temperature – comes from DHT22 sensors connected externally to each RPi. We do not use the measurements from the DHT22 sensors in any prediction. However,

Device	CPU-1	CPU-2	CPU-3	CPU-4	DHT-1	DHT-2	DHT-3	DHT-4	WU-T
CPU-1	0.00	4.78	7.15	3.20	**29.23**	30.12	30.78	29.84	32.26
CPU-2	-	0.00	4.07	3.40	24.51	25.37	26.06	25.12	27.55
CPU-3	-	-	0.00	4.86	23.09	23.99	24.68	23.71	26.16
CPU-4	-	-	-	0.00	27.87	28.76	29.45	28.50	30.95
DHT-1	-	-	-	-	0.00	2.07	2.60	2.15	3.61
DHT-2	-	-	-	-	-	0.00	1.32	1.23	4.31
DHT-3	-	-	-	-	-	-	0.00	1.00	3.75
DHT-4	-	-	-	-	-	-	-	0.00	4.01
WU-T	-	-	-	-	-	-	-	-	0.00

Table 4.4: Average absolute difference in temperature measurements among CPU and DHT22 sensors from four RPi's (Pi1, Pi2, Pi3, and Pi4) measured during the 72 hours period on August 25th, 26th, and 27th, 2018.

we use them to determine the mean absolute error (MAE) between a prediction based on CPU and WU-T values and ground truth as established by the DHT value and thereby determine our prediction accuracy. Our RPis are equipped with a 1GHz ARMv7 processor, 512MB memory, 32GB of SSD storage, and Wifi communication. All the temperature readings in the experiments are reported in degrees Fahrenheit.

4.4.1 Experimental Results

As a baseline, the upper triangle of the matrix in Table 4.4 shows the average difference in temperature, pairwise, between all pairs of temperature measurement traces we include in our study. Thus, for example, the average difference in temperature between CPU-1 and DHT-1 (the DHT connected directly to the RPi hosting CPU-1) is given in row 2, column 6 of the table as $29.23°F$ marked in bold in the table (assuming the header and row labels are row 1 and column 1 respectively). This data spans 72 hours beginning August 27th, 2018 and includes 864 temperature measurements gathered at 5-minute intervals.

Overall, this baseline illustration shows that

- CPU and external DHT measurements differ by approximately $30°F$;
- average differences among DHT22 sensors (ground truth) vary from $1°F$ to $2.6°F$ (despite their proximity); and
- the differences in local temperature from the one reported by the nearby weather station vary from $3.61°F$ to $4.31°F$.

Since the matrix of comparisons is symmetric, we only show values in the upper triangle.

For frost prevention, the application is attempting to determine when a small difference in temperature between warm air aloft and colder air near the ground will result in frost avoidance if the air is mixed. Specifically, large wind machines move the warm air downwards to raise the temperature enough near the ground to prevent frost from forming. The temperature differences are on the order of a few degrees Fahrenheit putting a premium on accurate measurement. The baseline in Table 4.4 shows the errors that result when each temperature sensor is used directly to predict another. That is, it is the “worst case” prediction in the sense that it includes no prediction mechanism – only the raw data.

In order to provide a more accurate prediction of local temperature based solely on the devices’ CPU temperatures and the nearby weather station, we combine multiple linear regression with smoothing. We hypothesize that the relationship between outdoor temperature and nearby CPU temperatures measured at the same time is linear. Further, particularly if one or more of the CPUs are loaded, we use one-dimensional smoothing of the CPU temperature series to improve the “signal” from the CPU temperature sensor.

For the regressions, the explanatory variables are a subset of CPU and a weather station temperature (CPU-1, CPU-2, CPU-3, CPU-4, WU-T), as indicated at the top of each results tables. Also, when smoothing is performed, we indicate this in the table header.

In each case, we separate the experimental period under study into a “training” period followed immediately by a “testing” period. The regression coefficients are

	DHT-1				**DHT-3**			
	Original		**Smoothed**		**Original**		**Smoothed**	
TE	**CPU-1**	***All***	**CPU-1**	***All***	**CPU-3**	***All***	**CPU-3**	***All***
1	0.55	*0.39*	**0.38**	*0.40*	**0.32**	***0.32***	**0.37**	*0.21*
3	**0.45**	*0.34*	0.38	*0.33*	0.50	*0.32*	0.47	***0.20***
6	0.46	***0.32***	0.41	***0.28***	0.78	*0.41*	0.83	*0.28*
12	0.48	*0.46*	0.44	*0.43*	0.70	*0.48*	0.74	*0.37*
24	0.55	*0.43*	0.55	*0.44*	0.95	*0.57*	0.99	*0.46*
48	0.62	*0.47*	0.62	*0.46*	1.04	*0.63*	1.04	*0.51*
72	0.70	*0.49*	0.70	*0.49*	1.28	*0.69*	1.21	*0.55*
96	0.75	*0.52*	0.78	*0.53*	1.36	*0.72*	1.31	*0.62*
168	**0.85**	*0.72*	**0.92**	***0.69***	**1.68**	*0.83*	1.64	*0.80*
336	0.79	***0.81***	0.77	*0.66*	1.54	***1.26***	**1.56**	***1.24***

Table 4.5: MAE for different sets of smoothed and non-smoothed explanatory variables and lengths of Test Window (TE) when predicting DHT-1 and DHT-3 temperature based on a 72h train window and a test start day on Aug. 25th.

computed only from data in the training period. We then use the coefficients for the entire duration of the testing period.

Table 4.5 shows the MAE between the temperature that our method predicts and the outdoor temperature for two “ground truth” sensors – DHT-1 and DHT-3 – using two separate subsets of explanatory variables for each. On the lefthand side of the table, we show the MAE (both with and without smoothing) when predicting DHT-1 using CPU-1 alone (a univariate regression) and also when using all CPUs and WU-T (a multiple linear regression, denoted as *All*). On the righthand side of the table, we show the same results for DHT-3 using CPU-3 in the univariate case. The experiment (testing period) start date is Aug. 25th. For all experiments, we use a training window of 72 hours (864 readings). As mentioned in section 4.3.2, we use MAE as our measure of accuracy since it captures the “distance” between the predicted temperature and the DHT-measured temperature. It is this distance that concerns farmers who are deciding on whether to trust their crops to the methodology.

Note that columns CPU-1 and CPU-3 under the Original column show values corresponding to results based on univariate linear regression. Note also that we highlight the minimum and maximum MAE in each column using boldface type.

When predicting DHT-1, we observe that errors from univariate regression using only the CPU temperature from Pi1 (CPU-1) are in the range from $0.45°F$

to $0.85°F$. MAE for multiple linear regression with CPU temperatures from all four devices and a nearby weather station data range from $0.32°F$ to $0.81°F$. When predicting DHT-3 from its Pi3's CPU sensor deployed in a similar manner we observe MAE values between $0.32°F$ to $1.68°F$ (listed in the left DHT-3 subtable as CPU-3 column). MAE decreases to a range from $0.32°F$ to $1.26°F$ when we introduce multiple linear regression (*All* column). Note that even though the setup is similar (the same set of devices and outdoor conditions), the readings are influenced by other environmental factors (tree coverage, sun exposure, etc.).

We find that multiple linear regression which includes CPU and nearby weather station temperatures as its predictors reduces prediction error. For DHT-1, the minimum error decreases from $0.45°F$ (minimum error in *CPU-1* column) to $0.32°F$ (minimum error in *All* column) while the maximum error decreases from $0.85°F$ (maximum error in *CPU-1* column) to $0.81°F$ (maximum error in *All* column). For DHT-3 the minimum error is $0.32°F$ for both columns (*CPU-3* and *All*) while the maximum error decreases from $1.68°F$ for *CPU-3* to $1.26°F$ for *All*. If we compare errors per test window length, we note that for DHT-1 all errors but for the 2 weeks test window were reduced (where $0.79 < 0.81$) and for DHT-3 all errors but for 1h test window were reduced (1h row had the same error of $0.32°F$ in both columns).

These results indicate that it is possible to make predictions with an average absolute error of under 1°F that require infrequent model refitting (e.g. once per several days) using a combination of CPU and weather station data. Indeed, the accuracy of DHT22 sensors is approximately 0.5°F. Thus this methodology is approaching the limit of accuracy that is possible using DHT22 sensors as ground truth. Under 1°F is acceptable for frost prevention where current manual methods use measurements in the 3°F range.

For the smoothing results in Table 4.5, each value (except the first 6) in the training period is replaced by the average of the 6 preceding it in the period (i.e. we use a *sliding window average* to smooth the data in the training period). When comparing the *All* column from *Original* and *Smoothed* columns, we observe that the smoothing decreases the mean absolute error (MAE) from the range of 0.32°F to 0.81°F (original) to the range of 0.28°F to 0.69°F (smoothed). Similarly, for DHT-3 prediction, the MAE goes from the range 0.32°F to 1.26°F (original) to the range 0.20°F to 1.24°F (smoothed).

4.4.2 Computational Load: the Effect of Smoothing and Multiple Linear Regression

CPU temperatures are correlated with the CPU load Moore et al. (2005), Haywood et al. (2015) and while the CPUs are idle for much of the time in our setting temporary computational load at the time of temperature recording may influence the prediction error (e.g. if the CPU were performing encryption as part of transmitting the data over the network). We next analyze the effect of the CPU load on the temperature prediction error.

Out of the four devices that we consider, we keep Pi2 and Pi4 unloaded and add hourly jobs to Pi1 and Pi3, which increase the CPU load by encrypting and copying a 1GB file on Pi1 and a 512MB file on Pi3. Figure 4.4 illustrates CPU temperature measurements from Pi1 with hourly spikes due to the load. The load testing for Pi1 and Pi3 started mid September and we use September 20th as a test start date. Note that Pi2 and Pi4 have no artificial load and are kept idle for comparison. We observe that, compared to the August test, all four Pi's show smaller errors on average, however, we omit these averages for brevity.

Table 4.6 shows the MAE for predicting DHT-1 and DHT-3 based on different sets of explanatory variables (listed on the top of the table) for different duration of the test window (TE), while both Pi1 and Pi3 are loaded. For predicting DHT-1 based on CPU-1, we observe MAE in the range of $0.71°F$ to $0.85°F$ and for the DHT-3 of $0.65°F$ to $0.78°F$. The effect of the CPU load is more pronounced in univariate prediction. Moreover, this effect is mitigated when we include nearby devices' CPU temperature measurements. Including nearby devices in the DHT-1

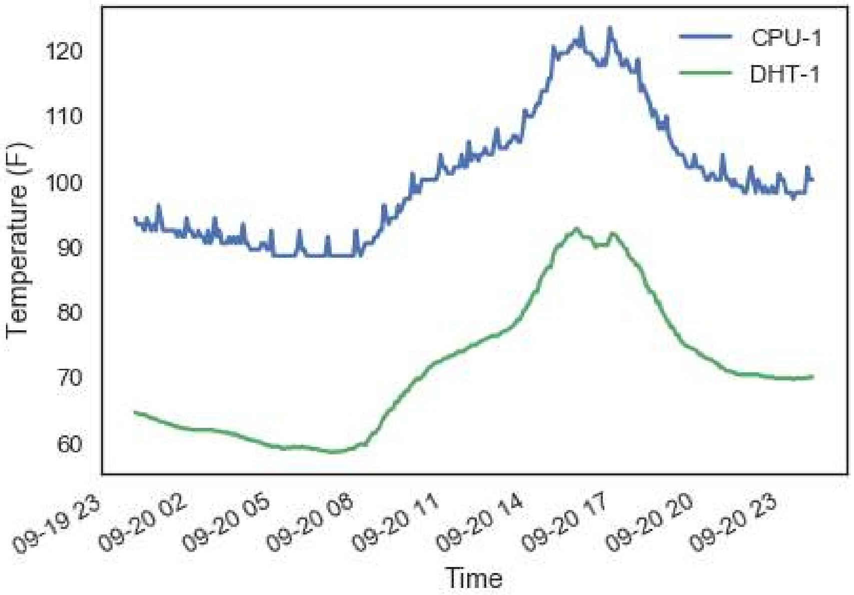


Figure 4.4: CPU-1 temperature under load and DHT-1 temperature in $^{\circ}F$.

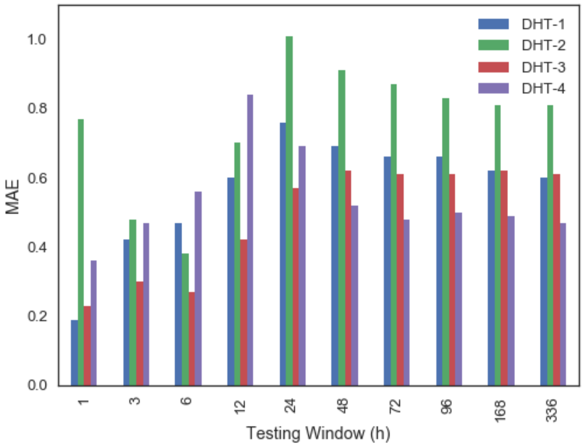
	DHT-1				**DHT-3**			
	Original		**Smoothed**		**Original**		**Smoothed**	
TE	**CPU-1**	***All***	**CPU-1**	***All***	**CPU-3**	***All***	**CPU-3**	***All***
1	0.85	*0.54*	**0.19**	*0.46*	0.75	***0.39***	**0.23**	***0.32***
3	**0.71**	***0.49***	0.42	***0.36***	**0.78**	*0.53*	0.30	*0.34*
6	0.73	*0.55*	0.47	*0.37*	0.70	*0.44*	0.27	*0.34*
12	0.73	*0.53*	0.60	*0.42*	0.74	*0.53*	0.42	*0.50*
24	0.85	*0.57*	**0.76**	***0.54***	0.70	*0.52*	0.57	*0.49*
48	0.84	***0.58***	0.69	*0.51*	0.67	*0.50*	0.62	*0.48*
72	0.82	*0.55*	0.66	*0.50*	0.66	*0.50*	0.61	*0.48*
96	0.80	*0.54*	0.66	*0.53*	0.66	*0.52*	0.61	*0.49*
168	0.80	*0.53*	0.62	*0.51*	0.66	***0.53***	**0.62**	*0.50*
336	**0.85**	*0.53*	0.60	*0.51*	**0.65**	*0.51*	0.61	***0.50***

Table 4.6: Prediction error when CPU-1 and CPU-3 experience periodic load. The data shows MAE for different sets of smoothed and non-smoothed explanatory variables and lengths of Test Window (TE) when predicting outdoor temperature for DHT-1 and DHT-3 based on a train window of 72h and with a test start day on Sep 20th.

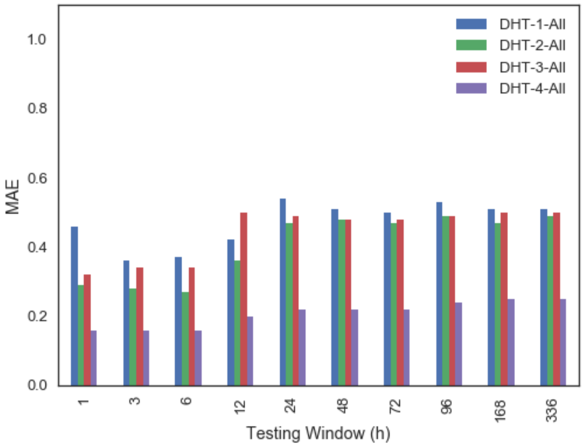
prediction (*All*) results in MAE in the range of 0.49°F to 0.58°F for DHT-1 and in the range of 0.39°F to 0.53°F for DHT-3.

Similar to the results for the unloaded experiments, when the CPUs are loaded we also observe improvement in prediction error when we apply smoothing, as shown in Table 4.6. The two columns show MAE for DHT-1 and DHT-3 temperature prediction with the same smoothing technique explained earlier (rolling mean with a window size of 30 minutes or 6 readings). Note that this type of smoothing is computationally simple enough to be performed on each device (rather than as a remote computation requiring a more powerful computational resource (used in past work)).

We observe that for any length of test window the error when all the predictors are used (*All* column) is smaller than when any single predictor counterpart is used: CPU-1 for DHT-1, and CPU-3 for DHT-3. With smoothing, the prediction MAE decreases from the range of 0.71°F to 0.85°F to the range of 0.36°F to 0.54°F for DHT-1, and from the range 0.65°F to 0.78°F to a range of 0.32°F to 0.50°F for DHT-3. While not strictly lower or higher, these results are similar (in terms of accuracy) to the results for the unloaded case. We conclude that, using a combination of multivariate regression and smoothing, it is possible to obtain high degrees of prediction accuracy (relative to measurement error) regardless of whether the CPU is loaded or not.



(a) Single CPU per DHT



(b) All CPUs + WU

Figure 4.5: MAE when predicting DHT-1, DHT-2, DHT-3, and DHT-4 temperature based on a 72h Train Window for different Test Window sizes (TE) and sets of smoothed explanatory variables (single CPU in the left and all variables of the right). Test start date is Sep. 20th. Pi1 and Pi3 experience additional periodic load, Pi2 and Pi4 do not.

To account for the possibility that the specific timeframe may have influenced the results (i.e. outdoor conditions may have been more dynamic in late August than late September), we show comparative results for the September timeframe for loaded and unloaded experiments in Figure 4.5. The data shown in this figure is taken during the same period as the results shown in Table 4.6. That is, we use the 72-hour period ending on September 20th, 2018 as a training period and the remaining time as a test period (ranging from 1h to 2 weeks). The bars in the figure corresponding to CPU-1 and CPU-3 show the same data as in Table 4.6 from the *Smoothed All* columns. For comparison, we show data for two other CPUs – CPU-2 and CPU-4 – taken at the same time, again using smoothing and all explanatory variables in each regression (i.e. *Smoothed All*).

Figure 4.5a shows the comparison when only the CPU directly attached to the DHT is used as a single explanatory variable (i.e. the "nearest" CPU). In Figure 4.5b, we show the the results when all explanatory variables are used to predict each DHT.

In Figure 4.5b, the maximum MAE observed in any experiment does not exceed $0.54°F$ across all CPUs, DHTs, and load patterns. These results indicate that the methodology is robust with respect to typical loads that the CPUs may experience in our IoT setting. Comparing Figure 4.5a to Figure 4.5b shows that multivariate regression improves accuracy across all DHTs and load patterns.

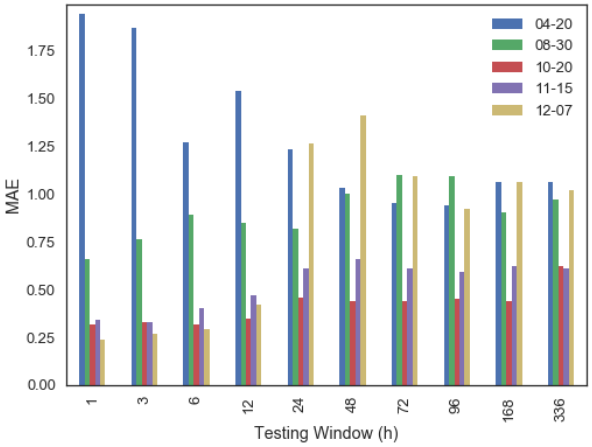


Figure 4.6: Comparison of MAE when predicting DHT-1 values for five different dates from April 20th to Dec. 7th.

4.4.3 Effects of Seasons and Precipitation

In addition to the two dates in August and September, we observed very similar error rates when testing during different seasons (Summer, Fall, and Winter). This is illustrated in Figure 4.6 where we predict DHT-1 temperature for different days from April to December. April 20th (04-20) has a higher error because Pi3 and Pi4 were not yet deployed and thus their CPU values were not available as features. December 7th had variable weather conditions with alternating rainy and sunny days, which may have contributed to a somewhat higher MAE. However, even so, the MAE for most of the days it was less than 1.25° F.

We also tested the accuracy of the model when there were changes in precipitation. From a time series perspective, precipitation could constitute a change-point in each temperature series (due to the sudden onset of evaporative cooling effects). Table 4.7 shows the comparison of errors when training and testing periods had different levels of precipitation. For each column, the training period was 3 days and the test periods listed go from 1h to 3 days. In the first column, both training and testing days were without any precipitation (this data is the same data that is represented graphically in Figure 4.5b as DHT-1-ALL). In the second column, we show the effects of training using rainy days to predict the temperatures during sunny days. December 4th, 5th, and 6th were rainy days with 2.54, 1.27, and 1.27 inches of rain respectively followed by three days without precipitation that were used for testing the model. In the third column, we show results for training during sunny days followed by prediction during rainy periods. January 2nd, 3rd, and 4th were days without precipitation followed by three days with 1.29, 1.06, and 1.0 inches of precipitation respectively.

The results show that the model trained only on three rainy days had errors slightly higher than when tested on sunny days, while the model trained on sunny days behaved similarly to the models we discussed before, even when tested on rainy days. Part of our future work is to expand test cases to more variable weather conditions (e.g., including changes in wind, solar radiance, etc.). However

TE	**Sep. 20th**	**Dec. 7th**	**Jan. 5th**
1	0.46	0.24	0.22
3	0.36	0.27	0.40
6	0.37	0.29	0.57
12	0.42	0.42	0.76
24	0.54	1.26	0.56
48	0.51	1.41	0.54
72	0.50	1.09	0.46

Table 4.7: MAE for models trained and tested during dry periods (Sep. 20), training during a rainy period and testing during a dry period (Dec. 7th) and training during a dry period and tested during a rainy period (Jan. 5th). Models are trained on 3 days to predict DHT-1 temperature based on all five explanatory variables using smoothing.

these results indicate that the prediction errors are robust to what are essentially "shocks" to the temperature time series in the explanatory weather data (WU-T) and the predicted variables (DHT values). Because the CPUs were in sealed containers (and the DHT sensors were exposed to the atmosphere) the effects of precipitation on the CPU series is less pronounced. Still, the errors are largely unaffected by precipitation.

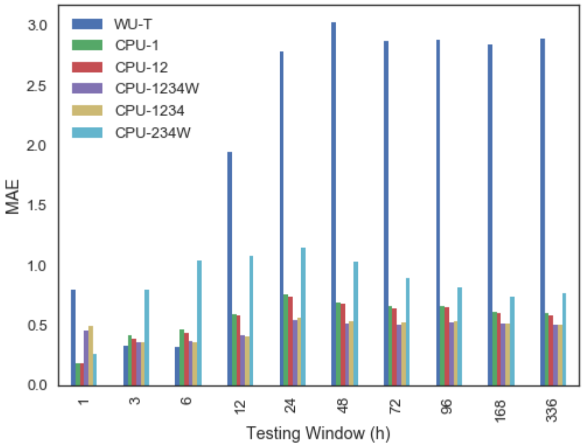


Figure 4.7: Comparison of MAE when predicting DHT-1 values for different sets of features for Sep. 20th.

Figure 4.7 illustrates the errors when predicting DHT-1 temperature with different subsets of explanatory variables. We observe that if we only rely on the nearby weather station (which is approximately 800m from the nearest DHT) the error (WU-T) is much higher ($2 - 3°F$) than for a subset that includes at least one of the CPU temperatures ($< 1.15°F$). Farmers, today, often use only a weather station temperature reading when implementing manual frost prevention practices. Often, though, the weather station they choose to use for the outdoor temperature is even farther away from the target growing block than the station we use in this study.

Notice, also, that when the CPU that is directly connected to the DHT is not included (denoted CPU-234W in the figure), the errors are higher than when it is included (all other bars in the figure except for W). Thus, as one might expect, proximity plays a role in determining the error. However using only the attached CPU (CPU-1 in the figure which is necessarily physically closest to DHT-1) generates a higher MAE than all CPUs and the weather station (denoted CPU-1234W in the figure). Indeed, the best performing model is this one that uses all four CPU temperatures and WU-T measurements as explanatory variables, yielding an MAE $< 0.5°F$ across all time frames. Thus using the nearest CPU improves accuracy, but using only the nearest CPU does not yield the most accurate prediction. Finally, while the weather station data does not generate an accurate prediction by itself, including it does improve the accuracy (slightly) over leaving it out.

In summary, our methodology is capable of automatically synthesizing a "virtual" temperature sensor from a set of CPU measurements and externally available weather data. By including all of the available temperature time series, it automatically "tunes" itself to generate the most accurate predictions even when one of the explanatory variables (WU-T in Figure 4.7) is, by itself, a poor predictor. These predictions are durable (lasting up to 2 weeks without refitting the regression coefficients), with errors often at the threshold of measurement error (for DHT sensors), on average, and relatively insensitive to seasonal and meteorological effects, as well as typical CPU loads in the frost-prevention setting where we have deployed it as part of an IoT system.

4.5 Related Work

Accurate micro-climate modeling is essential for agriculture operations such as irrigation scheduling and frost protection Ghaemi et al. (2009), Stombaugh et al. (1992), Ioslovich et al. (2016), Roberts et al. (2013), Gonzalez-Dugoa et al. (2011). We investigate the use of simple, low cost, single board computers to estimate air temperature for use in these applications. Although such devices are increasingly integrated into IoT solutions for agriculture Nikolidakis et al. (2015), Krintz et al. (2016), Vasisht et al. (2017)(e.g. providing alerts, irrigation control, communication of sensor data Zheleva et al. (2017), Ojha et al. (2017), Karimi et al. (2018), Foughali et al. (2018), Jawad et al. (2017), Golubovic et al. (2016), Beckwith et al. (2004)), there are no studies of which we are aware that use the devices themselves as thermometers.

To enable this, we estimate outdoor temperature from CPU temperature linear regression Hastie et al. (2009) of temperature time series. Others have shown that doing so is useful for other applications and analysesGuestrin et al. (2004), Xie et al. (2017), Lane et al. (2016), Yao et al. (2017). Our work is complementary to these and is unique in that it combines SSA (noise reduction) with regression to improve prediction accuracy. As in other work, we leverage edge computing to facilitate low latency response and actuation for IoT systems Alturki et al. (2017), Feng et al. (2017).

4.6 Summary

In this chapter, we investigate an alternative, low cost way of measuring and predicting outdoor temperature using inexpensive, single board computers as temperature sensors. Our approach uses linear regression to model the relationship between outdoor temperature and device CPU temperature at each device and employs SSA to account for autocorrelation in the time series. We calibrate each in-situ device using a high quality temperature sensor for a fixed duration of time; we use the regression coefficients from the calibration period (which do not change) to predict outdoor temperature from CPU temperature thereafter, using different devices and ground truth temperature sensor/stations (e.g. on-farm weather station, thermistor, WeatherUnderground station). We empirically evaluate the approach using different amounts of training data, calibration durations, and locations. Our results show that this approach can generate average errors of $1.5°F$ or lower in real-world precision agricultural deployments.

We then present an extension that uses multiple linear regression using nearby SBC processors and weather stations. We use these models to predict microclimate temperatures, which can be used (if sufficiently accurate) in agricultural settings to guide irrigation, frost control, and other IoT applications. We deploy our system in a citrus grove and perform an extensive empirical study using the devices and methodology. In addition, we consider the impact of loaded and unloaded processors as well as alternative smoothing techniques. We train our models for up to three days and evaluate their accuracy for a duration of up to two weeks. We find that our approach enables a prediction error that is less than $1.5°F$.

Chapter 5

Hypatia

With the prior chapters, we have contributed new methods for clustering correlated, multidimensional data and for synthesizing virtual sensors using the data produced from combinations of other sensors. We next unify these advances into a scalable, open source, end-to-end system called HYPATIA. We design HYPATIA to permit multiple analytics algorithms to be "plugged in" and to simplify the implementation and deployment of a wide range of data science applications. Specifically, HYPATIA is a distributed system that automatically deploys data analytics jobs across different cloud-like systems. Our goal with HYPATIA is to provide low latency, reliable, and actionable analytics, machine learning model selection, error analysis, data visualization, and scheduling, in a unified scalable system.

To enable this, HYPATIA places this functionality "near" (in terms of network transfer latency) the sensing devices that generate data, at the *edge* of the network. It then automates the process of distributing the application execution across different computational tiers: “edge clouds” and public/private cloud systems. Hypatia does so to reduce the response latency of applications so that data-driven decisions can be made by people and devices at the edge more quickly. Such edge decision making is important for a wide range of application domains including agriculture, smart cities, and home automation where decisions, actuation, and control are all local and make use of information from the surrounding environment. Hypatia automatically deploys and scales tasks on-demand both locally (at the edge) and remotely – if/when there are insufficient resources at the edge.

Hypatia presents users with an easy-to-use interface that it makes available via any web browser. Users can choose the algorithms they need for data analysis and prediction and select the dataset they are interested in. Hypatia iterates through the list of available parameters, and multiple training and scoring models for each parameter set. It then selects those with the best score. Such model selection can be used to provide data-driven decision support for users as well as to actuate and control digital and physical systems (e.g. other software services, trigger alerts, adapt sensing, irrigate crops, etc.). In this chapter, we focus on Hypatia support for clustering and regression.

The Hypatia scheduler automates distributed deployment across edge and cloud systems to minimize time to completion (i.e. end-to-end application execution time). It uses the computational and communication requirements of model training, testing, and inference, to make placement decisions for independent jobs that comprise a workload. For data intensive workloads, Hypatia prioritizes use of the (local) edge cloud. For compute intensive jobs (e.g. those with small input/output data sets), Hypatia prioritizes public/private (remote) cloud use.

In summary, with Hypatia, we design and develop a new, scalable, end-to-end distributed system that executes data analytics services across the edge, private, and public cloud resources such that analysis latency is minimized. To enable this, Hypatia combines

- data ingress services for sensing devices, web APIs (e.g. weather stations and other public datasets), files uploaded by users, and database tables,
- data analysis services for common machine learning tasks (classification, regression, k-means clustering, etc.),
- automatic model selection and scoring,
- dataset and result visualization, and
- a workload scheduler that minimizes the time to completion.

In the sections that follow, we present the design and implementation of Hypatia and the analytics services it supports. We then evaluate Hypatia in a multi-tier setting using real applications and workloads and show that it reduces time to completion for a wide range of workloads and deployment configurations. Finally, we overview recent work related to Hypatia and summarize our contributions.

5.1 Hypatia

Hypatia is an online platform for distributed cloud services that implement common data analytics utilities. It takes advantage of cloud-based, large-scale distributed computation, provides automatic scaling (where computational resources are added or removed on-demand), and implements data management and user interfaces in support of visualization and browser-based user interaction. Hypatia currently supports two key building blocks for popular statistical analysis and machine learning applications: clustering and linear regression.

For clustering, Hypatia implements the different variants of k-means clustering (described in Chapter 3). The variants include different distance computations (Euclidean and Mahalanobis), input data scaling (e.g. whether or not to scale each dimension to have zero mean and unit standard deviation), and the six combinations of covariance matrices. Hypatia runs the configuration for successive values of K ranging from 1 to a user-assigned large number, max_k. For each clustering, Hypatia computes a pair of scores based on both the Bayesian Information Criterion (BIC) Schwarz (1978) and the Akaike Information Criterion (AIC) Akaike (1974). Hypatia allows the user to change the number of independent, randomly seeded runs to account for statistical variation. Finally, it provides ways for the user to graph and visualize both two-dimensional "slices" of all clusterings as well as the relative BIC and AIC scores. It uses these scores to provide decision support for user – e.g. presenting the user with the "best" clustering across all variants.

For linear regression, Hypatia implements different approaches for analyzing correlated, multidimensional data (described in Chapter 4). Since we focus on synthesizing new sensors, we are looking for most important inputs from other sensors that can be used to accurately estimate a synthesized measurement. Hypatia allows users to decide on the number of input variables and which ones to use. They also can specify the start time of the test, duration of the training and testing periods, the scoring metric to use (mean absolute error, mean square error, R2 score, etc.). Users also choose whether or not to smooth the input data (prior to training) using different techniques (e.g. sliding window, mean, max, median, etc.). Finally, to predict outdoor temperature, users can select nearby single board computers and/or weather stations (e.g. on-site, WeatherUnderground, CIMIS, etc.).

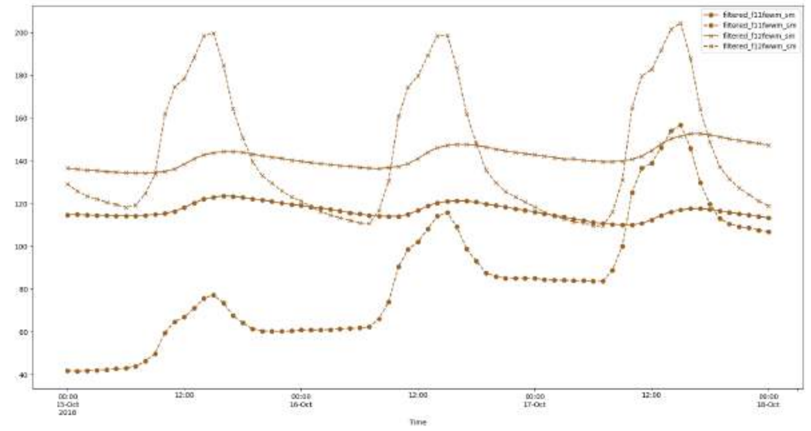


Figure 5.1: Hypatia visualization tool.

Once user makes these choices or accepts/modifies the defaults, Hypatia create an experiment with as many tasks as there are parameter choices. Each task produces a linear regression model with coefficients for each input variable and a score that can be used for model selection. As is done for clustering, Hypatia scores the various parameterizations using the scoring metric to provide decision support (i.e. to identify the best parameterization) to users. The user can then use the visualization tools to verify the similarity between input variables and estimated sensor measurements.

5.1.1 Design and Implementation

Hypatia is unique in that it is extensible – different data analytics algorithms can be "plugged in" easily, and automatically deployed with and compared against others. User can also extend the platform with both scoring (model selection) and visualization tools. Visualization is particularly important when some of the sensors are faulty and unreliable, or some of the smoothing or filtering techniques do not produce the desired outcome. Figure 5.1 shows such an example where visualization is used to show growers how soil moisture responds to precipitation and temperature on east, and west sides of a tree in an almond grove at different depths of 1 foot and 2 feet (where most of the tree's root system is). Being able to understand how significant each parameter is to soil moisture provides decision support that can be used to guide irrigation and harvest.

To implement Hypatia, we have developed a user-facing web service and distributed, cloud-enabled backend. Users upload their datasets to the web service frontend as files in a common, simple format: as comma-separated values (CSV files, easily exported from Excel spreadsheets). The user interface (UI) also enables users to modify the various algorithms and their parameters, or accept the defaults.

Hypatia considers each parameterization that the user chooses (including the default) as a "job". Each job consists of multiple tasks (experiment runs) that

Hypatia deploys. Users can also use the service to check the status of a job or to view the report and results for a job (when completed). The status page provides an overview of all the tasks for a job showing a progress bar for the percentage of tasks completed and a table showing task parameters and outcomes.

Hypatia uses a report page to provide its recommendation for both analysis building blocks, clustering and regression. For clustering, the recommendation consists of the number of clusters and the k-means variant that produces the best BIC score. This page also shows the cluster assignments, spatial plots using longitude and latitude (if included in the original data set), BIC and AIC scores plots. Hypatia also provides cluster labels in CSV files that the user can download. For regression, the report page consists of a map of error analysis for each model grouped by their parameters. User can quickly navigate to the model with the smallest error.

The software architecture of Hypatia is shown in Figure 5.2. We implement Hypatia using Python v3.6 and integrate a number of open source software packages and cloud services. At the edge, Hypatia uses a small private cloud that runs Eucalyptus software v4.4Nurmi et al. (2009), Ari (2019). The public cloud is Amazon Web Services (AWS) Elastic Compute Cloud (EC2). Hypatia integrates virtual servers from these two cloud systems with different capabilities (memory size and compute resources), which we describe in our empirical methodology. The architecture consists of five primary components (depicted in the figure):

1. *Frontend*: We couple the Python Flask Pyt (2019) (v0.12.1) web framework with Gunicorn Gun (2019) (v19.7.1) web server and NGINX NGI (2019) (v1.4.6) reverse proxy server to provide a robust application hosting service.

2. *Backend Worker*: We use Python Celery Cel (2019) (v4.0.2), a distributed computation framework, to perform analysis computation tasks asynchronously and at scale Lunacek et al. (2013). Hypatia is able to leverage autoscaling groups in Eucalyptus and AWS to automatically grow and shrink the number of workers performing the computation according to the demand for each job.

3. *Backend Queue*: We use RabbitMQ Rab (2019) (v3.2.4) message broker to send information about each job from the Frontend to the Workers. This enables to Frontend to quickly off-load its work to the Queue where it is sent systematically to the Workers as they become available.

4. *Backend Database*: We use a Postgres Pos (2019) database to store parameters and results for jobs and tasks. Different statistics about the tasks are later used to inform the scheduling.

5. *Backend File Store*: We use the a cloud based bucket/object storage (Walrus in Eucalyptus or S3 in AWS) using the AWS S3 interface.

6. *Network Profiler*: Implemented as a two dimensional lookup table of iPerf bandwidth data for different data sizes and numbers of connections. This table is used to bootstrap Hypatia until it has enough history data to perform its predictions using it.

7. *Scheduler*: Implemented in Python, the scheduler uses the status of remote and local queues and task statistics from the database (DB), to compute the best split for remote and local tasks. If task types are seen for the first time, the scheduler uses the information provided by the network profiler.

Other packages that Hypatia leverages include Numpy Walt et al. (2011) (v1.12.1), Pandas McKinney et al. (2010) (v0.19.2), SciKit-Learn Pedregosa et al. (2011) (v0.18.1), and SciPy Jones et al. (2001–) (v0.19.0) for data processing. Hypatia uses Matplotlib Hunter (2007) (v2.0.1) and Seaborn Sea (2019) (v0.7.1) to provide data visualization and plots.

5.2 Hypatia Scheduling

Hypatia is deployed on an edge cloud and a private/public cloud if available. We assume that the edge cloud has limited resources and is located near where

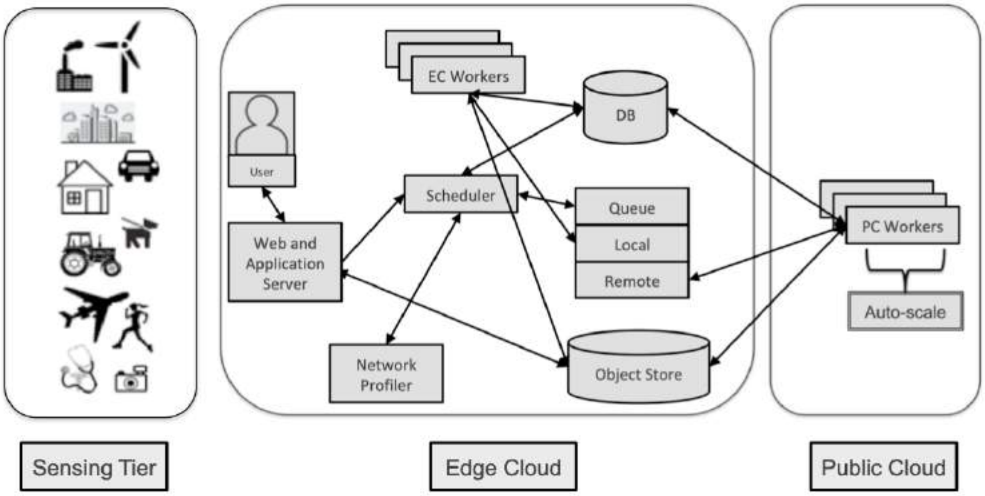


Figure 5.2: Hypatia system architecture with sensing, edge cloud, and public cloud tiers.

data is produced by sensors. The public cloud provides vast resources and is located across a long-haul network with varying performance (bandwidth and latency) and perhaps intermittent connectivity. We use N_{EC} to denote the number of machines available in the edge cloud (EC) and (N_{PC}) to denote the number of machines available in the public cloud where $N_{EC} << N_{PC}$.

Users submit multiple jobs to the edge system (via a web browser interface). Each job (J) describes the datasets (D) to be used for training, testing, and inference or analysis. In some jobs we can assume that the entire dataset is needed while in others we can assume that data can be split and tasks within the job can operate on different parts of the dataset in parallel. Each job has n

tasks (T). In the numerous jobs that we have evaluated over the course of this dissertation, we have observed that for the applications we have studied, n can range tens of tasks to millions of tasks.

We consider tasks from the same job has having the same "type". To estimate the time each task will take to complete the data transfer (to the public cloud) and computation, we compute an average per job i as t_i, across past tasks of the same type. Each task fetches its dataset upon invocation. Local tasks fetch from the file system or co-located database, remote tasks fetch (i.e. receive) their data over the network between EC and PC.

Hypatia estimates the input/output dataset transfer time to the remote (PC) in two ways. We use the first when Hypatia has no history on the job type in the database, i.e., when the job is run for the first time. knowledge of job's tasks is available, Hypatia uses the job's input and output dataset sizes and the number of concurrent connections, to estimate transfer time using the iPerf lookup table with values representing time in seconds needed to transfer data from edge to remote cloud for a dataset of a size given in kilobytes and a number of concurrent connections. The lookup table provides fast access but may introduce error because the data in the table is a "snapshot" in time.

Table 5.1 shows a snapshot of a Hypatia lookup table. Files of different sizes, listed on the left, are sent over the network with a variable number of concurrent connections, listed across the top. The data is produced using iPerf to measure the network performance between the EC and PC (using file transfer), when Hypatia is first started. Each of the numbers in the table is an average over 10 profiling runs.

For job types that Hypatia has seen (i.e. has statistics on in its database for at least one mixed job with both local and remote tasks), Hypatia uses the average time measured across past tasks of the same `type` to estimate the transfer time. This computation takes longer than a simple table lookup but uses recent history to makes predictions (which are hopefully more accurate than for static iPerf measurements).

Hypatia launches a set of virtual machine (VM) instance types to the EC and PC, the number of each is specified by the user in the job description. Instance type names map to the amount of memory and compute resources that each provides. We deploy one Hypatia worker to each processor. Thus, we view each compute resource (VM instance) in terms of the number of workers it can support. The Hypatia queue is placed on the local instance and has two queues: `local` and `remote`. Based on the load split ratio, Hypatia places tasks in the `local` or `remote` queues. Workers then pull tasks from their assigned queue for execution on a first-come-first-served basis.

size	**scale**	**1**	**2**	**4**	**8**	**16**	**32**	**64**	**128**
1	10^{-5}	12	15	14	13	18	14	20	15
10	10^{-5}	37	22	19	20	19	18	22	21
100	10^{-3}	117	118	117	118	119	124	150	184
500	10^{-2}	87	79	82	75	81	100	155	238
$1 * 10^3$	10^0	1.36	1.40	1.39	1.35	1.37	1.87	3.07	5.50
$1 * 10^4$	10^0	5.23	5.47	6.03	6.97	10.39	19.99	30.93	51.3
$1 * 10^6$	10^1	2.25	2.67	4.11	7.74	9.77	18.27	30.59	18.75

Table 5.1: Network speed (time in seconds) lookup table for the given data size in kilobytes and number of concurrent connections.

5.3 Problem Formulation

Given a Job J with n tasks and D dataset, d_i is the amount of data that must be transferred if the job is to use the PC. The scheduling plan uses this value, the network bandwidth between the EC and PC (V), the number of concurrent connections required to saturate the link, the number of available workers in each cloud, and the current state of the queues. The scheduler computes the ratio of the number of tasks that will execute locally and remotely such that time to completion for the job (all tasks) is minimized.

Hypatia estimates the time to completion (T_i) for EC (local) tasks as the average time to complete past tasks of a similar type (t_i), and the state of the EC queue, which is expressed as the lag caused by unfinished tasks in the EC queue (L_{EC}). The estimate for PC tasks includes a time estimate for data transfer. In addition, Hypatia uses the lag in the PC queue instead of the EC queue for this estimate.

Table 5.2 summarizes our terminology. Hypatia attempts to balance the workload between the EC and PC so that they finish at the same time. It thus attempts to satisfy the following two conditions:

$$T_{EC} + T_{PC} = n,$$

Name	**Description**
J_i	i-th job
T_i	a task from the i-th job
n	number of tasks in J_i
t_i	average time it takes to execute a task T_i
d_i	average time it takes to transfer data needed for a task T_i
T_{EC}	number of tasks from J_i to be sent to EC queue
T_{PC}	number of tasks from J_i to be sent to PC queue
W_{EC}	number of workers available in EC
W_{PC}	number of workers available in PC
L_{EC}	lag in EC queue
L_{PC}	lag in PC queue

Table 5.2: List of terms used for the scheduling algorithm.

where T_{EC} and T_{PC} are the numbers of tasks sent to EC and PC workers, respectively.

$$\frac{T_{EC} * t_i + L_{EC}}{N_{EC}} \sim \frac{T_{PC} * (t_i + d_i) + L_{PC}}{N_{PC}},$$

5.4 Empirical Evaluation

To evaluate the efficacy of the Hypatia scheduler, we execute multiple workloads across multi-tier deployments and measure the time to completion per job. For the experiments, we consider one edge instance and two sets of public cloud instances. On the edge, we use an m3.xlarge instance with 4CPUs and 2GiB memory. In the public cloud, our first instance set consists of a single “free tier” t2.medium instance type with 2 CPUs and 4GiB of memory. Our second public cloud set consist of an m5ad.xlarge instance type with 4 CPUs and 16GiB of memory. Our experiments consider deployments with 2, 4, 8, and 12 CPUs in the public cloud.

Our workloads consist of two machine learning applications that we developed in the previous chapters of this dissertation: *linear regression* and *k-means clustering*. For each experiment, we evaluate the performance of the mixed load (one that uses both EC and PC workers) against executing all of the jobs on the edge cloud (local workers only) or all of the jobs in the public cloud (PC workers only). We refer to Hypatia deployments as *mixed* because Hypatia will schedule job tasks on both the edge cloud (EC) and the public cloud (PC) when doing so reduces time to completion.

	LinReg								**K-means**							
	Uni				**Var**				**Uni**				**Var**			
	L		**R**		**L**		**R**		**L**		**R**		**L**		**R**	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
T	0.85	0.23	2.42	0.23	0.47	0.24	1.62	0.31	0.06	0.01	0.37	0.03	0.06	0.02	0.39	0.03
P	0.03	0.01	0.13	0.03	0.02	0.01	0.123	0.01	0.19	0.04	0.31	0.05	2.86	5.09	4.36	8.13

Table 5.3: Task statistics (mean and standard deviation of time needed in seconds) for different task types and worker locations for data transfer (T) and processing (P) time.

Since our jobs consist of tasks with different parameters, their time to transfer data and to perform the computation differs across tasks. To empirically evaluate this effect, for each set of machine learning models, we use two sets of jobs: `uniform` containing tasks having the same parameters, and `variable` containing tasks having different parameters (the latter therefore is likely to have significantly different execution and data transfer times). The statistics for each machine learning algorithm, for the variable or uniform sets, for EC and PC workers are listed in Table 5.3. We present mean and standard deviation for the time it takes to transfer the data needed for the task (row T) and to process the task (row P).

5.4.1 Load Balancing

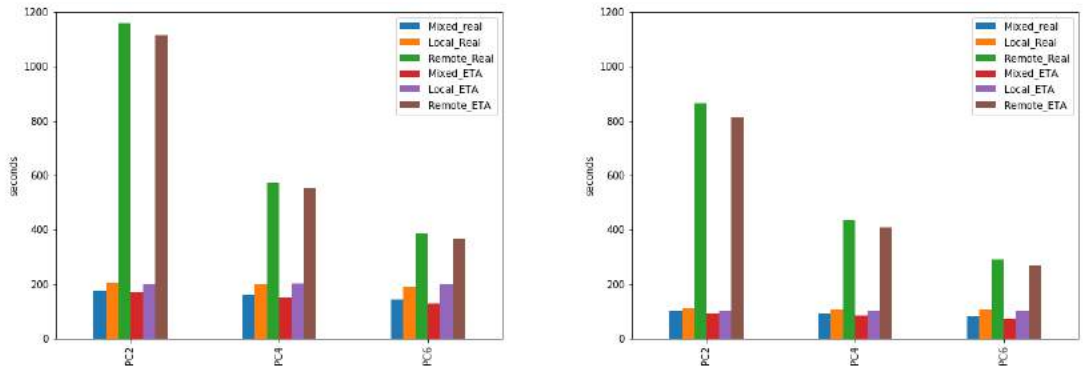
We first evaluate how well the Hypatia scheduler balances the load between EC and PC workers. As mentioned above, we compare time to completion of a mixed job (one that utilizes both EC and PC workers) with time to completion of jobs that are executed using EC workers only (EC-Only) or PC workers only (PC-Only).

Linear Regression Workload

The first experiment uses jobs that have 900 tasks, where each task is given a set of parameters that it then uses to compute the coefficients of a linear regression model based on two time series. The length of the time series defines the dataset size, and thus the computation time per task.

Jobs are either `uniform` or `variable` task sets as defined above. The uniform linear regression tasks take as input a one month time series of 5-minute interval measurements (float values). The variable job tasks take as input time series that vary between one day and one month worth of 5-minute measurements. For this experiment, the EC has 4 workers, and the PC has 2, 4, or 6 workers.

Figure 5.3 (a) shows the average time in seconds it took to complete a job with 900 `uniform` tasks using 4 EC workers and either 2 (PC2 bar in the figure), 4 (PC4), or 6 PC workers (PC6), on average across 5 jobs. For each set of results, the



(a) Uniform Load

(b) Variable Load

Figure 5.3: Average time for a linear regression job with 4 EC and 2, 4, and 6 PC workers with AWS free tier PC instances.

first three bars show the average time in seconds that it takes to complete a task using mixed (EC and PC), local (EC only), and remote (PC only) deployments, respectively. The second three bars per set show the average time in seconds that Hypatia estimates that each deployment should have taken, which we discuss later in this chapter.

In every case, the mixed Hypatia workload performs best for time to completion for the workload. For uniform jobs, the mixed load using 2 remote workers (PC2) finishes in 176 seconds on average, while EC-only takes 207s and PC-only takes 1160s on average, respectively. With 6 PC workers, the mixed workload takes 144s, EC-only takes 199s, and PC-only takes 387s on average. The results also show that as the number of PC workers increases, Hypatia is able to accurately split jobs between the two resource sets and time to completion decreases: 176s for PC2, 160s for PC4, and 144s for PC6 on average.

Data for linear regression experiments is stored in the database running on a separate instance within the local edge cloud (EC). This makes data transfer from a EC worker (on a separate instance within the same cloud) much faster than the transfer from the PC worker. The average data transfer time for EC workers is 0.85s and for PC workers is 2.42s. Processing time is 0.03s for EC and 0.13s for PC workers on average.

For the variable jobs in Figure 5.3 (b), the mixed workload takes an average of 102s with 2 remote workers, 91s with 4 remote workers, and 80s with 6 remote workers. Even though we expect EC workload to be the same across all 3 experiments in this figure (PC2, PC4, and PC6) since local workers don't change, we still see some variation with workloads taking on average 110s in PC2, 105s in PC4 and 105s in PC6. The total time of the workload is significantly less than for uniform workloads because many jobs employ much smaller input data sets sizes (e.g. 1 day vs 1 month worth of 5-minute data). The average data transfer time is 0.47s for EC and 1.6s for PC workers, while computation takes 0.02s for EC and 0.12s for PC workers, respectively.

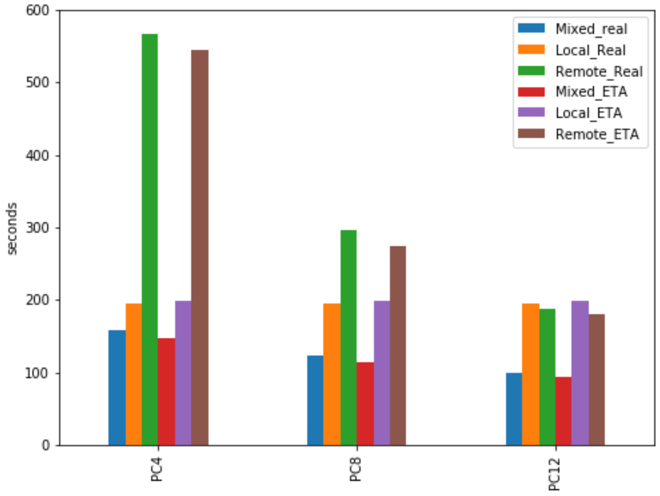
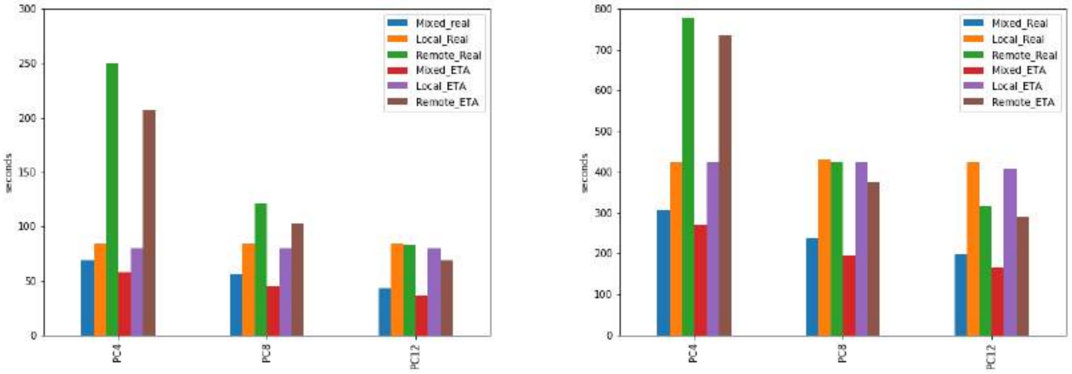


Figure 5.4: Average time for a linear regression job with 4 EC and 4, 8, and 12 PC workers with AWS m5 instances.

We see that for this particular job type, the PC workers take more time to fetch and process tasks. To investigate this further, we next change the instance type from the free tier t2.medium to use m5ad.xlarge instead – which we note is 4.4 times more expensive monetarily.

Figure 5.4 shows results for the linear regression application with uniform jobs (one month input time series per task) and with remote (PC) instances having 4, 8, and 12 workers, respectively. Only using 12 PC workers, can we observe remote experiments outperforming EC-only workloads. Similarly, with the increased number of workers the scheduler picked a correct split to minimize



(a) Uniform Load

(b) Variable Load

Figure 5.5: Average time for a K-means job with 4 EC and 4, 8, and 12 PC workers with AWS m5 instances.

the time to completion, which was 158s, 122s, 100s, for 4, 8 and 12 PC workers, respectively. Local only experiments took on average 195s, while remote took 566s, 296s, and 187s respectively.

K-means Workload

Figure 5.5 shows the results for the second application we consider – K-means clustering. Again, we evaluate uniform and variable workloads. The variable workloads employ six different options for computing the correlation in the dataset and performs the clustering for ten (K=10) different cluster set sizes. Uniform uses a single cluster size (K=3) and one way of computing the correlation. In both cases, we start each K-means algorithm with a random initialization for the cluster centers. Doing so introduces variability among tasks in terms of their computation time.

The mean computation time of the uniform workload is 0.19s for EC and 0.31s for PC workers, with standard deviations of 0.04s and 0.05s respectively. For the variable workload, the mean computation time for EC workers was 2.86s with standard deviation of 5.09s, while PC workers have a mean of 4.36s and standard deviation of 8.13s.

Like for the linear regression application, Hypatia is able to deploy the K-means workload (mixed workload) to achieve the shortest time to completion on average. As the number of PC workers increases, Hypatia adapts to employ the extra computational power by using the high-overhead communication link sparingly. The average time to completion is 306s, 238s, and 198s for mixed variable workload for 4, 8, and 12 PC workers, respectively. For EC-only variable workload we see 423s, 430s, and 424s for three different repeats of the same experiment. PC-only workloads took 778s with PC4, 423s with PC8, and 316s with PC12.

For uniform mixed workloads, the average time to completion is 69s, 56s, and 43s for 4, 8, and 12 PC workers, respectively. EC-only workload took 84s on average while PC-only workloads took 250s, 121s, and 82s on average.

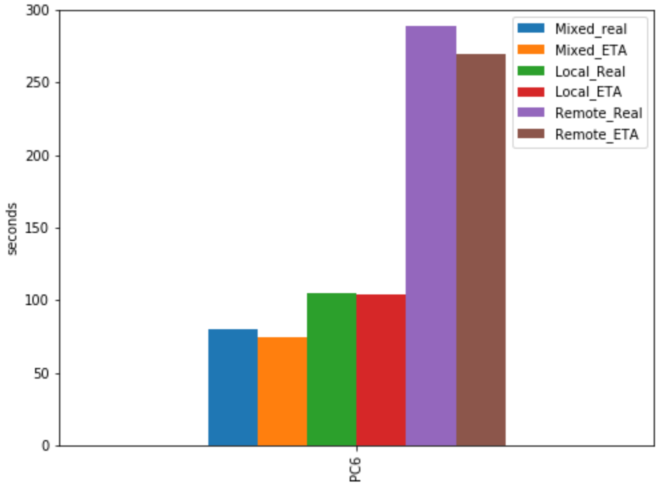


Figure 5.6: Average time for a linear regression job compared to its estimates with 4 EC and 6 PC workers (zoomed in section of Figure 5.3)

With more remote workers (e.g. 12) we observe that PC-only outperforms EC-only. These results reflect the importance of considering both data transfer and computation time when deciding when to use remote, public/private cloud resources in multi-tier settings. Moreover, they show that Hypatia is able to adapt to different resource configurations to achieve the best time to completion for different machine learning applications automatically.

5.4.2 Prediction Error

We next evaluate Hypatia prediction error. Prediction error in this setting is the difference between the Hypatia estimate of time to completion and the actual time to completion. In the previous figures, the second set of three bars show the Hypatia estimates. To understand this error, we zoom in on Figure 5.6 using Figure 5.3. In this latter figure, we show the estimated time to completion to the right of the observed average time per job. In this example, there is a prediction error of 14s for the mixed, 8s for the EC-only, and 18s for the PC-only jobs, on average. This corresponds to an average percentage error of 10%, 4%, and 5%, respectively.

Across all experiments, we observe errors that range from -4.32% to +20.12%. The errors are higher for the K-means applications. This is expected due to the random initial cluster center placement for each job (and thus variance is computation time). For the highest error, we have K-means uniform and variable experiments with 20.12% and 17.47% errors in time to completion, which correspond to 11s and 42s in absolute terms. The mean error over all experiments is 7.92 seconds with a standard deviation of 6.2s.

5.5 Related work

Distributed systems scheduling is a heavily studied area. It has played a key role in grid and high performance computing, cloud, multi-cloud, and more recently in edge computing Litzkow et al. (1988), Yoo et al. (2003)Kingsbury (1986). With the growth of scale and size of data, new resource managers have been developed to better utilize cloud resources Vavilapalli et al. (2013), Hindman et al. (2011), Boutin et al. (2014), Schwarzkopf et al. (2013). Many other schedulers have been developed that are tailored to machine learning workloads Moritz et al. (2018), Crankshaw et al. (2017), Li et al. (2018). HYPATIA is similar to these recent works in that its goal is to provide automatic scheduling of data analytics workloads. However, it differs from these systems in that it considers heterogeneous systems in sensor-edge-cloud (i.e. IoT settings), which have different amounts of resources and that may be intermittently connected.

With advancements in mobile device technology and their increased connectivity and computation resources, more resource-demanding applications are being pushed to the edge of the network. Those applications have motivated development of new techniques to offload computation to a nearby computing resources. A significant body of recent research have developed such technologies for mobile devices, phones and tablets, video streaming, and online games Satyanarayanan

(2013), Satyanarayanan et al. (2009), Chen & Hao (2018), Liu et al. (2016). Those applications require compute power beyond what the devices can provide. The offloading systems assume continuous connectivity between devices and cloud. Offloading can be performed at multiple levels, e.g. utilizing virtualization and offloading the tasks that can be shared across mobile devices Chun et al. (2011), Scoca et al. (2018), Kosta et al. (2012).

The difference between our work and past work is that we do not assume reliable, fast connectivity to the public cloud. We think of "edge computing" as done locally (e.g. on farms, on ships, in cars, etc.) where resources are limited and communication is intermittent and costly. HYPATIA automatically adapts to resource availability (local and remote) and provides robustness and fault resiliency via its co-location with sensors (data producers). While other schedulers focus on mobile devices and application responsiveness, we focus on the link between edge and remote public/private clouds and the application domain of machine learning training, testing and inference. Our scheduler design is motivated by our previous work Elias et al. (2017) on using edge resources for image classification in remote settings – in our case at the Sedgwick Natural Reserve. We utilize public cloud resources only if/when it is available and only if doing so will minimize the execution delay.

5.6 Summary

In this chapter, we present an end-to-end system for scalable deployment of data analytics and machine learning applications across multi-tier IoT deployments. Specifically, we present Hypatia which provides automatic deployment, scaling, scoring, and visualization of data science applications. We develop a new scheduler for this edge-cloud setting that attempts to deploy such applications at the edge near the sensor systems that produce the data being operated on in the analyses (or visualized). Doing so reduces the latency (i.e. response time) of these applications so that they can actuate and control digital and physical systems in the local environment in near real time. The Hypatia scheduler does so by performing as much of the analysis at the edge and then offloading any load which the local edge cloud cannot handle to a remote public or private cloud. We show that by doing so, Hypatia is able to adapt to resource availability and reduce the time to completion of applications versus executing every locally or everything in the cloud. We show this with an extensive evaluation of the analytics applications from the previous chapters (for classification and clustering) using different types of workloads and deployment configurations.

Chapter 6

Conclusions, Impact, and Future Work

With this dissertation, we present Hypatia – a scalable system for distributed, data-driven IoT applications. Hypatia ingresses data from disparate sensors and systems (including humans), and provides a wide range of analytics, visualization, and recommendation services with which to process this data and extract actionable insights. With a few examples of commonly used machine learning algorithms, like clustering and regression, we provide abstractions that make it easy to plug in different algorithms that are of interest to agronomists and other specialists who work with datasets that can benefit from locality. Hypatia integrates an intelligent scheduler that automatically splits analytics applications and workloads across edge, private, and public cloud systems to minimize the time to completion, while accounting for the cost of data transfer and remote computation.

We use Hypatia to investigate K-means clustering consider (i) different methods for computing correlation, (ii) using large numbers of trials (repeated randomized runs), and (iii) cluster degeneracy. Hypatia automatically deploys clustering experiments to account for all of these challenges. It then scores the clustering results using Bayesian Information Criterion (BIC) to provide users with recommendations as to the "best" clustering. We validate the system using synthesized data sets with known clusters for validation and use it to analyze scale measurements electrical conductivity (EC) of soil from a large number of farms. We compare our approach to the state of the art in clustering for EC data and show that our work significantly outperforms it. We also show that the system is easy to use by experts and novices and provides a wide range of visualization options for analysts.

We next extend the system with support for data ingress from sensors and develop a new approach for "virtualizing" sensors (called sensor synthesis) to extend their capability. Specifically, we show that it is possible to estimate outdoor temperature accurately from the processor (CPU) temperature of simple, low-cost, single board computers (SBCs). We present a novel approach that uses multiple linear regression to combine the CPU temperature from nearby SBCs and remote weather stations, to estimate the temperature at outdoor locations that do not have temperature sensors. We use the sensor data to train and test multiple regression models. We investigate the efficacy of using different smoothing techniques and we account for computational load of SBCs at the time of measurement and data collection. We find that our approach enables a prediction error that is less than 1.5 degrees Fahrenheit, while past work results in errors of 1–14 degrees Fahrenheit for similar datasets. We integrate sensor synthesis into Hypatia and use it to facilitate automatic and scalable model selection, as well as visualization for different data sets and recommendations.

Finally, we developed a new approach to distributed scheduling for analytics applications in IoT settings: sensor-edge-cloud deployments. Our scheduler takes advantage of remote (cloud) resources when available, while fully utilizing local edge systems, as it optimizes for time to completion for applications and workloads. The scheduler uses remote resources only if doing so reduces the latency of providing actionable insights locally. The scheduler uses histories of both computation and communication time the applications, which it uses to construct a job placement schedule that minimizes application response latency (i.e. time to completion). Hypatia then uses this schedule to automatically deploy workloads across edge systems and cloud computing systems. We empirically evaluate Hypatia using both clustering and regression services and show that its able to achieve better end-to-end performance than using the edge or cloud alone.

The result is the first end-to-end system that fully utilizes edge computing resources as it serves the needs of precision agriculture. It does so by accounting for resource constraints at the edge, the lack of or intermittent connectivity to the public cloud, and the expense of transmitting the data to/from remote cloud systems. Moreover, the system is open source and integrates a wide range of analytics, scoring methods, and visualization tools, which can be easily extended with new and emerging techniques. By doing so, we enable others to easily build upon, extend, reproduce, and compare to our work in the future.

Moving forward, we hope to encourage adoption of Hypatia by growers, farm consultants, and data analysts interested in taking advantage of the locality of edge systems to provide low latency analytics. Given the existing infrastructure, we plan to add new sensors, develop more synthesized, sensors, and to integrate additional analytics and scoring methods. Specifically, we plan to extend Hypatia with support for image classification and to use analytics accelerators (GPU/TPU systems) at both the edge and in the cloud when available. Other future work includes investigating new data sources and machine learning algorithms that inform a more refined scheduling algorithm that can take advantage of even more granular resources. In addition, Hypatia error analysis can benefit from additional abstractions that account error propagation, which has the potential for making the results and recommendation more informative and trustworthy.

Bibliography

Abdi, H. & Williams, L. (2010), ‘Principal component analysis’, *Wiley Interdisciplinary Reviews: Computational Statistics* **2**(4).

Ada (2018), ‘Adafruit AM2302 Wired DHT22 Temperature and Humidity Sensor’. [Online; accessed 01-Dec-2019] `https://www.adafruit.com/product/393`.

Adamchuk, V. I., Hummel, J., Morgan, M. & Upadhyaya, S. (2004), ‘On-the-go soil sensors for precision agriculture’, *Computers and electronics in agriculture* **44**(1), 71–91.

Aggelopooulou, K., Castrignanò, A., Gemtos, T. & De Benedetto, D. (2013), ‘Delineation of management zones in an apple orchard in greece using a multivariate approach’, *Computers and electronics in agriculture* **90**, 119–130.

Akaike, H. (1974), ‘A new look at the statistical model identification’, *IEEE transactions on automatic control* **19**(6), 716–723.

Alturki, B., Reiff-Marganiec, S. & Perera, C. (2017), A hybrid approach for data analytics for internet of things, *in* ‘Int. Conf. on the Internet of Things’.

ArcGIS (2019), ‘ArcGIS Online’. [Online; accessed 01-Dec-2019] `https://www.arcgis.com`.

Ard (2019), ‘Arduino’, `https://www.arduino.cc`. [Online; accessed 01-Dec-2019].

Ari (2019), ‘Aristotle Cloud Federation’. `https://federatedcloud.org` [Online; accessed 01-Dec-2019].

Arthur, D. & Vassilvitskii, S. (2007), k-means++: The advantages of careful seeding, *in* ‘Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms’, Society for Industrial and Applied Mathematics, pp. 1027–1035.

AWS IoT Core (2019), 'AWS IoT Core', https://aws.amazon.com/iot-core/. [Online; accessed 01-Dec-2019].

AWS IoT Greengrass (2019), 'AWS IoT Greengrass', https://aws.amazon.com/greengrass/. [Online; accessed 01-Dec-2019].

Azure IoT Edge (2019), 'Azure IoT Edge', https://azure.microsoft.com/en-us/services/iot-edge/. [Online; accessed 01-Dec-2019].

Azure IoT Hub (2019), 'Azure IoT Hub', https://azure.microsoft.com/en-us/services/iot-hub/. [Online; accessed 01-Dec-2019].

Bahmani, B., Moseley, B., Vattani, A., Kumar, R. & Vassilvitskii, S. (2012), 'Scalable k-means++', *Proceedings of the VLDB Endowment* **5**(7), 622–633.

Beckwith, R., Teibel, D. & Bowen, P. (2004), Report from the field: results from an agricultural wireless sensor network, *in* 'Local Computer Networks'.

Bell, J., Butler, C. & Thompson, J. (1995), Soil-terrain modeling for site-specific agricultural management, *in* 'Site specific management for agricultural systems. Proc. conference, Minneapolis, 1995'.

Berkhin, P. (2006), A survey of clustering data mining techniques, *in* 'Grouping multidimensional data', Springer, pp. 25–71.

Bezdek, J. C. (2013), *Pattern recognition with fuzzy objective function algorithms*, Springer Science & Business Media.

Bishop, C. M. (2006), 'Pattern recognition', *Machine Learning* **128**, 1–58.

Bonomi, F., Milito, R., Zhu, J. & Addepalli, S. (2012), Fog computing and its role in the internet of things, *in* 'Proceedings of the first edition of the MCC workshop on Mobile cloud computing', ACM, pp. 13–16.

Bosch IoT Suite (2019), 'Bosch IoT Suite', https://www.bosch-iot-suite.com/. [Online; accessed 01-Dec-2019].

Boutin, E., Ekanayake, J., Lin, W., Shi, B., Zhou, J., Qian, Z., Wu, M. & Zhou, L. (2014), Apollo: scalable and coordinated scheduling for cloud-scale computing, *in* '11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)', pp. 285–300.

Brimberg, J. & Mladenovic, N. (1999), 'Degeneracy in the multi-source weber problem', *Mathematical Programming* **85**(1), 213–220.

Cel (2019), 'Celery'. "http://www.celeryproject.org/" [Online; accessed 01-Dec-2019].

Cerioli, A. (2005), 'K-means cluster analysis and mahalanobis metrics: a problematic match or an overlooked opportunity', *Statistica Applicata* **17**(1).

Chen, M. & Hao, Y. (2018), 'Task offloading for mobile edge computing in software defined ultra-dense network', *IEEE Journal on Selected Areas in Communications* **36**(3), 587–597.

Chun, B.-G., Ihm, S., Maniatis, P., Naik, M. & Patti, A. (2011), Clonecloud: elastic execution between mobile device and cloud, *in* 'Proceedings of the sixth conference on Computer systems', ACM, pp. 301–314.

Climate Corporation (A Monsanto Company) (2014). `http://www.climate.com/` [Online; accessed 01-Dec-2019].

Cloud IoT Core (2019), 'Cloud IoT Core', https://cloud.google.com/iot-core/. [Online; accessed 01-Dec-2019].

Committee on Assessing Crop Yield: Site-Specific Farming, Information Systems, and Research Opportunities, Board on Agriculture, National Research Council (1997), *Precision Agriculture in the 21st Century: Geospatial and Information Technologies in Crop Management*, National Academy Press, Washington, D.C.

Corwin, D. & Lesch, S. (2003), 'Application of soil electrical conductivity to precision agriculture', *Agronomy Journal* **95**(3), 455–471.

Crankshaw, D., Wang, X., Zhou, G., Franklin, M. J., Gonzalez, J. E. & Stoica, I. (2017), Clipper: A low-latency online prediction serving system, *in* '14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)', pp. 613–627.

Duda, R. O., Hart, P. E., Stork, D. G. et al. (1973), *Pattern classification*, Vol. 2, Wiley New York.

Dunn, J. C. (1974), 'Well-separated clusters and optimal fuzzy partitions', *Journal of cybernetics* **4**(1), 95–104.

Edge TPU (2019), 'Edge TPU', https://cloud.google.com/edge-tpu/. [Online; accessed 01-Dec-2019].

Elias, A. R., Golubovic, N., Krintz, C. & Wolski, R. (2017), Where's the Bear? – Automating Wildlife Image Processing Using IoT and Edge Cloud Systems, *in* 'Internet-of-Things Design and Implementation (IoTDI), 2017 IEEE/ACM Second International Conference on', IEEE, pp. 247–258.

et al, C. J. (2003), 'Site-specific management zones based on soil electrical conductivity in a semiarid cropping system', *Agronomy journal* **95**(2).

Federation, A. F. B. (2014), 'Ponder These Nine Before you Sign: Data Privacy Expectation Guide'. `http://www.kfb.org/Assets/uploads/images/DataPrivacy.pdf` [Online; accessed 01-Dec-2019].

Feng, L., Kortoçi, P. & Liu, Y. (2017), A multi-tier data reduction mechanism for iot sensors, *in* 'Intl Conf on the Internet of Things', p. 6.

Floyer, D. (2015), 'The Vital Role of Edge Computing inthe Internet of Things'. "http://wikibon.com/the-vital-role-of-edge-computing-in-the-internet-of-things/" [Online; accessed 01-Dec-2019].

Fortes, R., Millán, S., Prieto, M. & Campillo, C. (2015), 'A methodology based on apparent electrical conductivity and guided soil samples to improve irrigation zoning', *Precision agriculture* **16**(4), 441–454.

Foughali, K., Fathallah, K. & Frihida, A. (2018), 'Using cloud iot for disease prevention in precision agriculture', *Procedia Comput. Sci* **130**, 575–582.

Foukas, X., Patounas, G., Elmokashfi, A. & Marina, M. K. (2017), 'Network slicing in 5g: Survey and challenges', *IEEE Communications Magazine* **55**(5), 94–100.

Fraisse, C. W., Sudduth, K. A. & Kitchen, N. R. (2001), 'Delineation of Site-Specific Management Zones by Unsupervised Classification of Topographic Attributes and Soil Electrical Conductivity', *American Society of Agricultural and Biological Engineers* **44**(1).

Fridgen, J. J., Kitchen, N. R., Sudduth, K. A., Drummond, S. T., Wiebold, W. J. & Fraisse, C. W. (2004), 'Management zone analyst (mza)', *Agronomy Journal* **96**(1), 100–108.

General Electric IoT (2019), 'General Electric IoT', https://www.ge.com/digital/iiot-platform. [Online; accessed 01-Dec-2019].

Ghaemi, A. A., Rafiee, M. R. & Sepaskhah, A. R. (2009), 'Tree-temperature monitoring for frost protection of orchards in semi-arid regions using sprinkler irrigation', *Agricultural Sciences in China* **8**(1), 98–107.

Gili, A., Álvarez, C., Bagnato, R. & Noellemeyer, E. (2017), 'Comparison of three methods for delineating management zones for site-specific crop management', *Computers and Electronics in Agriculture* **139**, 213–223.

Golubovic, N., Gill, A., Krintz, C. & Wolski, R. (2017), Centaurus: A cloud service for k-means clustering, *in* 'IEEE DataCom'.

Golubovic, N., Krintz, C., Wolski, R., Lafia, S., Hervey, T. & Kuhn, W. (2016), Extracting Spatial Information from Social Media in Support of Agricultural Management Decisions, *in* 'ACM SIGSPATIAL Workshop on Geographic Information Retrieval'.

Golubovic, N., Krintz, C., Wolski, R., Sethuramasamyraja, B. & Liu, B. (2018), 'A Scalable System for Executing and Scoring K-Means Clustering Techniques and Its Impact on Applications in Agriculture', *International Journal of Big Data Intelligence* .

Golubovic, N., Wolski, R., Krintz, C. & Mock, M. (2019), Improving the Accuracy of Outdoor Temperature Prediction by IoT Devices, *in* 'IEEE Conference on IoT'.

Golyandina, N. & Zhigljavsky, A. (2013), *Singular Spectrum Analysis for time series*, Springer Science & Business Media.

Gonzalez-Dugoa, V., Zarco-Tejadaa, P., Bernia, J., Suareza, L., Goldhamerb, D. & Fereres, E. (2011), Almond tree canopy temperature reveals intra-crown variability that is water stress-dependent, Technical report, Agricultural and Forest Meteorology.

gThrive (2014). `http://www.gthrive.com/` [Online; accessed 01-Dec-2019].

Guestrin, C., Bodik, P., Thibaux, R., Paskin, M. & Madden, S. (2004), Distributed regression: an efficient framework for modeling sensor network data, *in* 'Intl Symp on Information processing in sensor networks'.

Gun (2019), 'Gunicorn'. "http://gunicorn.org/" [Online; accessed 01-Dec-2019].

Hastie, T., Tibshirani, R. & Friedman, J. (2009), 'The elements of statistical learning new york', *NY: Springer* .

Haywood, A. M., Sherbeck, J., Phelan, P., Varsamopoulos, G. & Gupta, S. K. (2015), 'The relationship among cpu utilization, temperature, and thermal power for waste heat utilization', *Energy Conversion and Management* **95**, 297–303.

Heath, T. L. et al. (1956), *The thirteen books of Euclid's Elements*, Courier Corporation.

Hindman, B., Konwinski, A., Zaharia, M., Ghodsi, A., Joseph, A. D., Katz, R. H., Shenker, S. & Stoica, I. (2011), Mesos: A platform for fine-grained resource sharing in the data center., *in* 'NSDI'.

Hunter, J. (2007), 'Matplotlib: A 2d graphics environment. computing in science& engineering 9, 90–95 (2007)'.

Int (2019*a*), 'Intel NUC'. `http://www.intel.com/content/www/us/en/nuc/overview.html` [Online; accessed 01-Dec-2019].

Int (2019*b*), 'Internet of Things - Cisco', `http://www.cisco.com/c/en/us/solutions/internet-of-things/overview.html`. [Online; accessed 01-Dec-2019].

Ioslovich, I., Sylaios, G., Plauborg, F. & Battilani, A. (2016), 'Optimal model-based deficit irrigation scheduling using aquacrop: A simulation study with cotton, potato and tomato', *Agricultural Water Management* **163**.

Jain, A. K., Murty, M. N. & Flynn, P. J. (1999), 'Data clustering: a review', *ACM computing surveys (CSUR)* **31**(3), 264–323.

Jawad, H., Nordin, R., Gharghan, S., Jawad, A. & Ismail, M. (2017), 'Energy-efficient wireless sensor networks for precision agriculture: A review', *Sensors* **17**(8), 1781.

Jaynes, D., Colvin, T. & Ambuel, J. (1995), Yield mapping by electromagnetic induction, *in* 'Site-specific management for agricultural systems'.

Jones, E., Oliphant, T., Peterson, P. et al. (2001–), 'SciPy: Open source scientific tools for Python'. [Online; accessed 01-Dec-2019].
URL: *http://www.scipy.org/*

Karimi, N., Arabhosseini, A., Karimi, M. & Kianmehr, M. H. (2018), 'Web-based monitoring system using wireless sensor networks for traditional vineyards and grape drying buildings', *Computers and Electronics in Agriculture* **144**, 269–283.

Kingsbury, B. (1986), The network queueing system, Technical Report NASA-CR-17743, NASA.

Kitchen, N., Sudduth, S. & Drummond, S. (2006), An evaluation of methods for determining site-specific management zones, *in* 'North Central Extension Industry Soil Fertility Conference Proceedings'.

Kosta, S., Aucinas, A., Hui, P., Mortier, R. & Zhang, X. (2012), Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading, *in* '2012 Proceedings IEEE Infocom', IEEE, pp. 945–953.

Krintz, C. (2013), The appscale cloud platform: Enabling portable, scalable web application deployment, *in* 'Internet Computing', IEEE.

Krintz, C., Wolski, R., Golubovic, N. & Bakir, F. (2018*a*), Estimating Outdoor Temperature from CPU Temperature for IoT Applications in Agriculture, *in* 'International Conference on the Internet of Things'.

Krintz, C., Wolski, R., Golubovic, N. & Bakir, F. (2018*b*), Estimating outdoor temperature from cpu temperature for iot applications in agriculture, *in* 'International Conference on the Internet of Things'.

Krintz, C., Wolski, R., Golubovic, N., Lampel, B., Kulkarni, V., Sethuramasamyraja, B., Roberts, B. & Liu, B. (2016), SmartFarm: Improving Agriculture Sustainability Using Modern Information Technology, *in* 'KDD Workshop on Data Science for Food, Energy, and Water'.

Lane, N. D., Bhattacharya, S., Georgiev, P., Forlivesi, C., Jiao, L., Qendro, L. & Kawsar, F. (2016), Deepx: A software accelerator for low-power deep learning inference on mobile devices, *in* 'Information Processing in Sensor Networks (IPSN)'.

Levitt, J. et al. (1980), *Responses of Plants to Environmental Stress, Volume 1: Chilling, Freezing, and High Temperature Stresses.*, Academic Press.

Li, T., Zhong, J., Liu, J., Wu, W. & Zhang, C. (2018), 'Ease. ml: Towards multi-tenant resource sharing for machine learning workloads', *VLDB Endowment* **11**(5), 607–620.

Linde, Y., Buzo, A. & Gray, R. (1980), 'An algorithm for vector quantizer design', *IEEE Transactions on communications* **28**(1), 84–95.

Litzkow, M. J., Livny, M. & Mutka, M. W. (1988), Condor-a hunter of idle workstations, *in* 'International Conference on Distributed Computing Systems'.

Liu, J., Mao, Y., Zhang, J. & Letaief, K. B. (2016), Delay-optimal computation task scheduling for mobile-edge computing systems, *in* '2016 IEEE International Symposium on Information Theory (ISIT)', IEEE, pp. 1451–1455.

Lloyd, S. (1982), 'Least squares quantization in pcm', *IEEE transactions on information theory* **28**(2), 129–137.

Lunacek, M., Braden, J. & Hauser, T. (2013), The scaling of many-task computing approaches in python on cluster supercomputers, *in* 'IEEE Cluster Computing', pp. 1–8.

Lund, E., Christy, C. & Drummond, P. (1999), 'Practical applications of soil electrical conductivity mapping', *Precision agriculture* **99**, 771–779.

Mahalanobis, P. C. (1936), 'On the generalized distance in statistics', *Proceedings of the National Institute of Sciences (Calcutta)* **2**, 49–55.

Mausbach, M. J., Lytle, D. J. & Spivey, L. D. (1993), Application of soil survey information to soil specific farming, *in* 'Proceedings of Soil Specific Crop Management'.

McKinney, W. et al. (2010), Data structures for statistical computing in python, *in* 'Proceedings of the 9th Python in Science Conference', Vol. 445, van der Voort S, Millman J, pp. 51–56.

Molin, J. P. & Castro, C. N. D. (2008), 'Establishing management zones using soil electrical conductivity and other soil properties by the fuzzy clustering technique', *Agricultural Engineering* **65**(6).

Mon (2019), 'Mongodb'. "https://www.mongodb.com/" [Online; 9-Jun-2017].

Moore, J. D., Chase, J. S., Ranganathan, P. & Sharma, R. K. (2005), Making scheduling cool: Temperature-aware workload placement in data centers, *in* 'USENIX Annual Techical Conference'.

Moral, F., Terrón, J. & Da Silva, J. M. (2010), 'Delineation of management zones using mobile measurements of soil apparent electrical conductivity and multivariate geostatistical techniques', *Soil and Tillage Research* **106**(2), 335–343.

Moritz, P., Nishihara, R., Wang, S., Tumanov, A., Liaw, R., Liang, E., Elibol, M., Yang, Z., Paul, W., Jordan, M. I. et al. (2018), Ray: A distributed framework for emerging {AI} applications, *in* '13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)', pp. 561–577.

Mulla, D., Bhatti, A., Hammond, M. & Benson, J. (1992), 'A comparison of winter wheat yield and quality under uniform versus spatially variable fertilizer management', *Agriculture, ecosystems & environment* **38**(4), 301–311.

Murphy, K. P. (1998), 'Fitting a conditional linear gaussian distribution'.

Murphy, K. P. (2012), *Machine learning: a probabilistic perspective*, MIT press.

MyAgCentral (2014), 'MyAgCentral Launches Cloud-Based Solution For UAV Imagery'. `http://www.myagcentral.com/` `http://www.precisionag.com/equipment/myagcentral-launches-cloud-based-solution-for-uav-imagery/` [Online; accessed 01-Dec-2019].

NGI (2019), 'Nginx'. "https://nginx.org/" [Online; accessed 01-Dec-2019].

Nikolidakis, S. A., Kandris, D., Vergados, D. D. & Douligeris, C. (2015), 'Energy efficient automated control of irrigation in agriculture by using wireless sensor networks', *Computers and Electronics in Agriculture* **113**, 154–163.

Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L. & Zagorodnov, D. (2009), The eucalyptus open-source cloud-computing system, *in* 'Cluster Computing and the Grid, 2009. CCGRID'09. 9th IEEE/ACM International Symposium on', IEEE, pp. 124–131.

Odeh, I., Chittleborough, D. & McBratney, A. (1992), 'Soil pattern recognition with fuzzy-c-means: application to classification and soil-landform interrelationships', *Soil Science Society of America Journal* **56**(2), 505–516.

Ojha, T., Misra, S. & Raghuwanshi, N. S. (2017), 'Sensing-cloud: Leveraging the benefits for agricultural applications', *Computers and electronics in agriculture* **135**, 96–107.

OnFarm (2019), 'OnFarm: SaaS-based Data Visualization Product', `www.onfarm.com`. [Online; accessed 01-Dec-2019].

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V. et al. (2011), 'Scikit-learn: Machine learning in python', *Journal of Machine Learning Research* **12**(Oct), 2825–2830.

Peeters, A., Zude, M., Kathner, J., Unlu, M., Kanber, R., Hetzroni, A., Gebbers, R. & Ben-Gal, A. (2015), 'Getiss hot-and cold-spot statistics as a basis for multivariate spatial clustering of orchard tree data', *Computers and Electronics in Agriculture* **111**, 140–150.

Pelleg, D., Moore, A. W. et al. (2000), X-means: Extending k-means with efficient estimation of the number of clusters., *in* 'ICML', Vol. 1, pp. 727–734.

Penman, H. L. (1948), 'Natural evaporation from open water, bare soil and grass', *Proc. R. Soc. Lond. A* **193**(1032), 120–145.

Pos (2019), 'PostgreSQL opensource relational database'. "https://www.postgresql.org" [Online; accessed 01-Dec-2019].

PowWow (2019), 'Pow Wow Energy: SaaS-based Smart Leak Detection', `https://www.powwowenergy.com/learnMore/`. [Online; accessed 01-Dec-2019].

Pyt (2019), 'Python flask'. "http://flask.pocoo.org/" [Online; accessed 01-Dec-2019].

Rab (2019), 'Rabbitmq'. "https://www.rabbitmq.com/" [Online; 9-Jun-2017].

Ras (2018), 'Raspberry Pi'. [Online; accessed 01-Dec-2019] `www.raspberrypi.org`.

Rhoades, J., Manteghi, N., Shouse, P. & Alves, W. (1989), 'Soil electrical conductivity and soil salinity: New formulations and calibrations', *Soil Science Society of America Journal* **53**(2).

Roberts, B., Fritschi, F., Horwath, W. & Bardhan, S. (2013), 'Nitrogen Mineralization potential as influenced by microbial biomass, cotton residues and temperature', *Plant Nutrition* .

Rodrigues, T. (2013), 'What happens when your cloud provider goes out of business?'. `http://www.techrepublic.com/blog/the-enterprise-cloud/what-happens-when-your-cloud-provider-goes-out-of-business/` [Online; accessed 01-Dec-2019].

Russo, J. (2013), 'Data Privacy, Ownership In Precision Agriculture'. `http://www.precisionag.com/opinion/joe-russo/data-privacy-ownership-in-precision-agriculture/` [Online; accessed 01-Dec-2019].

Satyanarayanan, M. (2013), Cloudlets: at the leading edge of cloud-mobile convergence, *in* 'Proceedings of the 9th international ACM Sigsoft conference on Quality of software architectures', ACM, pp. 1–2.

Satyanarayanan, M., Bahl, V., Caceres, R. & Davies, N. (2009), 'The case for vm-based cloudlets in mobile computing', *IEEE pervasive Computing* .

Schwarz, G. (1978), 'Estimating the dimension of a model', *Annals of Statistics* **6**(2).

Schwarzkopf, M., Konwinski, A., Abd-El-Malek, M. & Wilkes, J. (2013), Omega: Flexible, scalable schedulers for large compute clusters, *in* 'ACM European Conference on Computer Systems'.

Scoca, V., Aral, A., Brandic, I., De Nicola, R. & Uriarte, R. B. (2018), Scheduling latency-sensitive applications in edge computing., *in* 'Closer', pp. 158–168.

Sea (2019), 'Seaborn: statistical data visualization'. "http://seaborn.pydata.org/index.html" [Online; accessed 01-Dec-2019].

Stewart, C., Cockerill, T., Foster, I., Hancock, D., Merchant, N., Skidmore, E., Stanzione, D., Taylor, J., Tuecke, S., Turner, G., Vaughn, M., & Gaffney, N. (2015), Jetstream: a self-provisioned, scalable science and engineering cloud environment, *in* 'XSEDE Conference: Scientific Advancements Enabled by Enhanced Cyberinfrastructure'. ACM: 2792774 [Online] `http://dx.doi.org/10.1145/2792745.2792774`.

Stombaugh, T. S., Heinemann, P., Morrow, C. & Goulart, B. (1992), 'Automation of a pulsed irrigation system for frost protection of strawberries', *Applied Engineering in Agriculture* **8**(5), 597–602.

Sudduth, K. (1997), Spatial modeling of crop yield using soil and topographic data, *in* 'Precision agriculture'97: papers presented at the first European Conference on Precision Agriculture, Warwick University Conference Centre, UK, 7-10 September 1997', Oxford; Herndon, VA: BIOS Scientific Pub., c1997.

Sudduth, K., Kitchen, N., Wiebold, W., Batchelor, W., Bollero, G., Bullock, D., Clay, D., Palm, H., Pierce, F., Schuler, R. et al. (2005), 'Relating apparent electrical conductivity to soil properties across the north-central USA', *Computers and Electronics in Agriculture* **46**(1), 263–283.

Towns, J., Cockerill, T., Dahan, M., Foster, I., Gaither, K., Grimshaw, A., Hazlewood, V., Lathrop, S., Lifka, D., Peterson, G. D., Roskies, R., Scott, J. R. & Wilkins-Diehr, N. (2014), 'Xsede: Accelerating scientific discovery', *Computing in Science and Engineering* **16**(5). Sept.-Oct.

USDA (2017), Task Force on Agriculture and Rural Prosperity Report, Technical report, USDA.

Vasisht, D., Kapetanovic, Z., Won, J., Jin, X., Chandra, R., Sinha, S. N., Kapoor, A., Sudarshan, M. & Stratman, S. (2017), Farmbeats: An iot platform for data-driven agriculture., *in* 'NSDI', pp. 515–529.

Vavilapalli, V. K., Murthy, A. C., Douglas, C., Agarwal, S., Konar, M., Evans, R., Graves, T., Lowe, J., Shah, H., Seth, S. et al. (2013), Apache hadoop yarn: Yet another resource negotiator, *in* 'Proceedings of the 4th annual Symposium on Cloud Computing', ACM, p. 5.

Ver (2019), 'Veris tech. inc.'. "http://www.veris.com/" [Online; 2-Jun-2017].

Verbelen, T., Simoens, P., De Turck, F. & Dhoedt, B. (2012), Cloudlets: bringing the cloud to the mobile user, *in* 'ACM workshop on Mobile cloud computing and services', ACM.

Vogt, W. (2014), 'Climate Corp, Monsanto Lay Out New Data Privacy Policies'. `http://farmindustrynews.com/precision-farming/climate-corp-monsanto-lay-out-new-data-privacy-policies` [Online; accessed 01-Dec-2019].

Walt, S. v. d., Colbert, S. C. & Varoquaux, G. (2011), 'The numpy array: a structure for efficient numerical computation', *Computing in Science & Engineering* **13**(2), 22–30.

WatrHub (2014). `http://www.watrhub.com/` [Online; accessed 01-Dec-2019].

Wea (2019), 'WeatherUnderground'. "http://www.weatherunderground.com/" [Online; accessed 01-Dec-2019].

White, G. & Haas, J. (1975), Assessment of Research on Natural Hazards, Technical report, MIT Press.

Wold, S., Esbensen, K. & Geladi, P. (1987), 'Principal component analysis', *Chemometrics and intelligent laboratory systems* **2**(1-3), 37–52.

Wollenhaupt, N., Mulla, D. & Crawford, C. (1997), 'Soil sampling and interpolation techniques for mapping spatial variability of soil properties', *The state of site-specific management for agriculture* .

Xie, C., Tank, A., Greaves-Tunnell, A. & Fox, E. (2017), 'A unified framework for long range and cold start forecasting of seasonal profiles in time series', *stat* **1050**, 23.

Yao, S., Hu, S., Zhao, Y., Zhang, A. & Abdelzaher, T. (2017), Deepsense: A unified deep learning framework for time-series mobile sensing data processing, *in* 'WWW'.

Yoo, A. B., Jette, M. A. & Grondona, M. (2003), Slurm: Simple linux utility for resource management, *in* 'Workshop on Job Scheduling Strategies for Parallel Processing', pp. 44–60.

Zhao, W., Ma, H. & He, Q. (2009), Parallel k-means clustering based on mapreduce, *in* 'IEEE International Conference on Cloud Computing', Springer, pp. 674–679.

Zheleva, M., Bogdanov, P., Zois, D.-S., Xiong, W., Chandra, R. & Kimball, M. (2017), Smallholder agriculture in the information age: Limits and opportunities, *in* 'Workshop on Computing Within Limits'.