A Service Oriented Architecture For Automated Machine Learning At Enterprise-Scale
This thesis presents a solution architecture for productizing machine learning models in an enterprise context and, tracking the model’s performance to gain insights on how and when to retrain the model. There are two challenges which this thesis deals with. First, machine learning models need to be trained regularly to incorporate unseen data to maintain it’s performance. This gives rise to the need of machine learning model management. Second, there is an overhead in deploying machine learning models into production with respect to support and operations. There is scope to reduce the time to production for a machine learning model, thus offering cost-effective solutions. These two challenges are addressed through the introduction of three services under ScienceOps called ModelDeploy, ModelMonitor and DataMonitor. ModelDeploy brings down the time to production for a machine learning model. ModelMonitor and DataMonitor helps gain insights on how and when a model should be retrained. Finally, the time to production for the proposed architecture on two cloud platforms versus a rudimentary approach is evaluated and compared. The monitoring services give insight on the model performance and how the statistics of data change over time.

Keywords: machine learning, model management, machine learning productization, machine learning workflow, machine learning cloud, azure machine learning

Language: English
Acknowledgements

My master’s degree journey has been an amazing experience which I would cherish forever. But it would have not been possible without the support of my family, my friends, my advisors and my colleagues who played a huge role in what I am today.

I would like to thank my supervisor, Prof. Juho Rousu for providing me with the right direction for my thesis, guidance on numerous occasions and constant support at all times.

I would like to express my deepest gratitude to my manager, Ville Suvanto and my advisor, Neelabh Kashyap for giving me an opportunity to be a part of the team & this project, for constantly guiding me and motivating me. My master thesis journey became easier through their constant help and support whenever I needed it. I would also like to thank the entire CGI Advanced Analytics Solutions Data Science team for helping out with brainstorming & discussions for this project.

My friends have played a huge role in my life and I thank them all; Srikanth Gadicherla, Gelin Guo, Sunny Vijay, Sugam Shukla, Ahmed Shehata, Yashasvi Singh Ranawat, Akash Kumar Dhaka, Harshit Agrawal and Andreas Hitz for constantly motivating me to keep on working hard and cheering me up when I started to go off-track.

Finally, I would like to thank Simon Sinek, because of whom I was able identify and pursue what I love. Simon’s talks and speeches are the reason for what I am today; an aspiring, happy and a learner for life doing what he loves.

Espoo, November 16, 2018

Mayank Khandelwal
Contents

1 Introduction .............................................. 7
  1.1 Scope ................................................. 7
  1.2 Contributions ......................................... 8
  1.3 Structure of the Thesis ................................. 8

2 Background ................................................ 10
  2.1 Challenges of Machine Learning in Production ............... 10
    2.1.1 Solutions and Delivery of Models on Cloud ............ 10
    2.1.2 Machine Learning and Big Data in Enterprises .......... 11
    2.1.3 Data Lifecycle Management .......................... 12
  2.2 Machine Learning Model Management ...................... 12
  2.3 Data Monitoring and Model Monitoring .................... 13
    2.3.1 Concept Drift ....................................... 13
    2.3.2 Retraining Models With New Data .................... 15
  2.4 Existing Solutions .................................. 16
    2.4.1 MLflow .............................................. 16
    2.4.2 PipelineAI ........................................ 16
    2.4.3 PredictionIO ........................................ 17
    2.4.4 H2O.ai ............................................. 17
    2.4.5 Azure ML Services .................................. 18
    2.4.6 Pachyderm .......................................... 18
    2.4.7 Google Cloud ML Engine ............................. 19
    2.4.8 Amazon SageMaker .................................. 19
    2.4.9 Evaluation Of Existing Services ...................... 19
  2.5 Infrastructure Services ................................ 20

3 Solution Architecture .................................. 24
  3.1 ModelDeploy Components ............................... 26
    3.1.1 Operationalizing Machine Learning Model ............. 26
    3.1.2 Docker and Flask .................................. 27
    3.1.3 Machine Learning Model Management .................. 27
List of Figures

3.1 ScienceOps Workflow : Azure ..................... 25

4.1 Model pre-operationalization tasks ..................... 32
4.2 Azure Workbench File List .............................. 33
4.3 Train and Pickle ......................................... 34
4.4 Operationalization of the model ......................... 35
4.5 Model Monitor on Power BI .............................. 41
4.6 Data Monitor on Power BI ............................... 41

A.1 Model Registered on Azure ............................... 66
A.2 Manifest Created on Azure Based on Fig. A.1 .......... 66
A.3 Image Created on Azure Based on Fig. A.2 .......... 67
A.4 Web Service Created on Azure Based on Fig. A.3 .... 67
A.5 User Interface for AWS Implementation A.5 ........ 68
Chapter 1

Introduction

With the technological advancements in computational hardware and cloud services over the past few years, developing machine learning solutions has gained immense popularity amongst enterprises. While many enterprises are now actively employing machine learning solutions, it brings along a set of various challenges.

1.1 Scope

There are two primary challenges faced by enterprises investing into machine learning solutions. The first challenge is that most enterprises have huge amounts of historical data as well as incoming (unseen) sets of streaming or batch data. Machine learning models are commonly employed to make predictions and generate actionable insights from data. Machine learning models need to be trained regularly to accommodate new unseen data to maintain optimal performance, which gives birth to the requirement of managing machine learning models in a structured manner. *Machine learning model (lifecycle) management* means managing the performance of models over time; because models can have varying performance over time.

The second challenge for enterprises is that, there is an overhead in terms of support and operations in deploying machine learning models. Deploying in this context means incorporating machine learning models into line-of-business applications to support decision making. For cost-effective solutions, enterprises are in favor of quickly pushing machine learning models into production. Enterprises may wish to migrate machine learning solutions to a different environment in the future, requiring extra resources for migration and setting up the new environment for the machine learning model. Thus, the solution should be portable to allows migrations between service
platforms. It is important to monitor the performance of machine learning models and regularly checking the validity of the predictions after model deployment because this affects the decision making process in an enterprise.

1.2 Contributions

The contribution of this thesis is the concept of ScienceOps which addresses the two challenges mentioned in Section 1.1 and the literature survey mentioned in Chapter 2. ScienceOps is an end-to-end solution architecture to deploy, monitor and manage machine learning models in an enterprise scenario. ScienceOps is an agglomeration of three services aiming at managing, monitoring and deploying machine learning models at enterprise scale. The first service is called ModelDeploy which packages a machine learning model and its functionality to make predictions with the model, into a versioned build artifact and uploads it to a central repository. This triggers the build of a web service which can be used for real-time scoring through a web-based API. The second service is called DataMonitor which exposes a dashboard with summary statistics about data used for training as well as statistics for the incoming data. If the statistics of the incoming data significantly deviate from the statistics obtained during training, the model should be recalibrated. This dashboard also helps in evaluating what portion of the data must be considered for retraining. The third service is called ModelMonitor which exposes a dashboard showing the deployed models and performance measures at the time of training. It also tracks and monitors the predictions generated by the model and their statistics over time.

1.3 Structure of the Thesis

Chapter 1 describes the problem statement with enterprise-level machine learning solutions, the importance of the problem and, the goal and objectives of this thesis. Literature survey can be found in Chapter 2, which describes the challenges of machine learning solutions in production, challenges related to machine learning model management, evaluation of existing solutions and, description of the components used in ScienceOps and the advantages of using them. The workflow of the ScienceOps architecture, the description of the services/tools used and justification for using them are included in Chapter 3. Chapter 4 describes the workflow of the ScienceOps architecture, how it is applied and justification for the usage of different architectural components. This chapter also describes an example of how a
machine learning problem can leverage the ScienceOps architecture to build a solution on Azure. Comparison of ScienceOps workflow versus a rudimentary method, discussion on implementation on Amazon AWS and possible improvements with respect to the ScienceOps architecture to cater to the problem statement are mentioned in Chapter 5. Conclusions and summarization of the work can be found in Chapter 6. Reference code templates and figures related to using the ScienceOps workflow are in the Appendix.
Chapter 2

Background

This chapter is divided into five parts. The first part describes challenges of building and delivering machine learning solutions on the cloud, how challenges related to big data affect the development of machine learning models, and challenges in data lifecycle management. The second part talks about the challenges of model lifecycle management. The third part describes the challenges of up-keeping performance over time and need for monitoring model performance and data statistics. The fourth part describes software packages similar to that of ScienceOps and what features they provide. Background on the technological components used in the ScienceOps architectural workflow on Azure is mentioned in the last part of this chapter.

2.1 Challenges of Machine Learning in Production

2.1.1 Solutions and Delivery of Models on Cloud

Machine learning systems were infeasible in the pre-cloud era for most enterprises with limited processing power and storage on premise. Cloud computing provides scalable and low-cost resources attributing to the adoption of machine learning solutions across enterprises. Cloud computing provides disaster recovery, software updates, version control, collaboration, security, platform access independence and zero capital-expenditure; thus offering itself as a suitable option as compared to on-premise (local) systems which often have limited capability in these aspects [5]. In terms of an enterprise scenario, there are several advantages of adopting and migration to the cloud platform such as lower cost of entry to benefit from compute-intensive business analytics, immediate access to hardware resources with no upfront
capital investments and lowering IT barriers to innovation. [40].

However, there are several challenges for enterprise cloud-based software services both on a technical front as well as on an adaptability front. Uncertainty about security at network, host, application and data levels; high speed internet access, reliability and availability to support 24/7 operations, interoperability and portability of information between private clouds and public clouds, and physical storage of confidential data across borders pose as major technical challenges [6]. For an enterprise, change in role of the IT department, policy compliance, political implications with respect to losing control of some aspects of the services and impact on end-users [36].

The challenge in the service delivery models of cloud computing include accessibility vulnerabilities, virtualization vulnerabilities, web application vulnerabilities such as SQL injection and cross-site scripting, physical access issues, privacy and control issues arising from third parties having physical control of data, issues related to identity and credential management, issues related to data verification, tampering, integrity, confidentiality, data loss and theft, issues related to authentication of the respondent device or devices and IP spoofing [75].

2.1.2 Machine Learning and Big Data in Enterprises

All machine learning solutions are based on the underlying data, and in recent times the data being generated has increased significantly which gives rise to the necessity of understanding big data. There are three key challenges [15] with respect to big data: Volume, Velocity and Variety. The challenge with respect to volume means that there is no standard agreement on the quantification process of big data. Quantification of big data depends on various factors such as the complexity of the data structure and the requirements of target applications. The challenge of velocity means handling the speed with which new data is created (or existing data is updated). The data velocity challenge affects every stack of a data management platform. Both the storage layer and the query processing layer need to be fast and scalable enough to meet the speed of the data generation or updation. The third challenge of variety relates to the fact that data may be generated from various sources in different formats and models [16].

With respect to productizing a solution in the big data context, one of the most important challenge is the definition of an analytics structure. It is unclear on how an optimal architecture should be constructed to deal with historic data and realtime data simultaneously. It is also vital to achieve statistically significant result while handling randomness in data because it is easy to get incorrect results with huge dataset and different types of
predictions [26]. Many data mining techniques are not easy to parallelize. Data may be evolving over time, so it is important that the big data mining techniques should be able to detect changes and adapt such as [28]. Claims to accuracy may be misleading - when the number of variables increase, the number of fake correlations increase [69].

When dealing with large quantities of data, storage becomes relevant. There are two approaches to deal with this: either compression of data or sampling of data to choose representative data [27].

Big data mining to extract relevant information is vital for machine learning solutions in order to get a training set which can perform well for generating predictions.

### 2.1.3 Data Lifecycle Management

There are several challenges when dealing with large amounts of data. Collection of data in an effective and time-saving way poses a challenge especially when collecting huge amounts of data in realtime. Transfer of large, unstructured datasets can be challenging because a small deficiency can lead to propagation of issues [39]. It is important to understand the lifecycle of data (quality). As mentioned earlier, data quality is expressed through information such as its uncertainty (spread/distribution), reliability (methods used for measurements/calculations), completeness, age of the data (when it was recorded), the process technology or technological level for which the data is representative for. Thus reliability followed by applicability of the results of a lifecycle assessment depends on the original data quality providing the background for the assessment. Thus data quality management must be integrated as part of the lifecycle management [76].

It is a challenging task to manage and organize diverse and sophisticated datasets during their lifecycle which includes data generation, acquisition, preservation or processing [21, 32]. Easy, efficient and safe access to data sources and repositories enable extraction of value through analytical processes on the data. Thus efficient data management and organization systems are vital for effective data to value generation [67].

### 2.2 Machine Learning Model Management

Performance of machine learning models are highly dependent on the underlying data they are trained on. When the (incoming) data being scored against the model statistically deviates from the data on which the model is trained on, performance of the model worsens, thus rendering the model
invalid. In order to combat the model performance degradation, it is necessary to keep track of the statistics of the model performance. This gives rise to the importance of \textit{machine learning model management}.

In an enterprise context, building a machine learning model is a trial-and-error based iterative process. A machine learning model is built based on a hypothesis about the underlying data, the model is tested, and the hypothesis and model are tuned based on results. The machine learning model process is based on development of tens or hundreds of machine learning models before landing at a model which can be accepted. It is difficult to track a previously built machine learning model and the corresponding insights. Thus, there is a need to remember relevant information about previous models to tune the next set of machine learning models. Lack of model and result persistence can also lead to doubt on the conclusions of a previous experiment leading to re-running of expensive modeling workflows. This iterative, adhoc nature of machine learning model building gives rise to the importance of machine learning model management [72].

\section*{2.3 Data Monitoring and Model Monitoring}

Incoming data can vary in its uncertainty (distribution), reliability (methods used for measurements/calculations), completeness, age of the data (when it was recorded), the process of data extraction or technological level for which the data is representative [77]. Since data plays an important role on how well a model performs and impacts the model management process, it is important to track the deviation of statistical features of the trained data versus incoming data.

However, this data driven approach can have an adverse effect if the data on which models are dependent on are outliers or incorrect values. In cases such as model retraining to account for data (distribution), changes may lead to corruption of the model through \textit{model drift} (or \textit{concept drift}). \textit{Concept drift} means that the feature the model predicts, changes over time due to statistical differences in the underlying data. Often the cause of change is hidden, not known beforehand, making the learning task more complicated[71]. This may cause the model to perform poorly over time.

\subsection*{2.3.1 Concept Drift}

The challenge in handling concept drift is differentiating between true concept drift and noise. Some algorithms may overreact to noise, erroneously interpreting it as concept drift, while others may be highly robust to noise,
There are three primary approaches to handle concept drift. The first approach is called Instance Selection where the goal is to select instances relevant to the current concept. It is possible to handle concept drift based on instance selection and generalizing from a window that moves over recently arrived instances and uses the learnt concepts for prediction only in the immediate future. The window-size can either be kept static or variable depending on the use case. The second approach is called Instance Weighting [37]. In this approach, weighted instances are processed based on machine learning algorithms such as Support Vector Machines. Instances can be weighted according to their age, and their competence with regard to the current concept. However, in general this approach is worse than instance selection approach due to overfitting the data [37]. The third approach is called Ensemble Learning. Ensemble learning maintains a set of concept descriptions, predictions of which are combined using voting or weighted voting, or the most relevant description is selected. There are multiple existing implementations based on this method. A program called STAGGER [65] consists of a set of concept descriptions, which are originally features themselves, and more complicated concept descriptions are then produced iteratively using feature construction. Based on their relevance to the current data, the features are then selected. Conceptual clustering can be used to identify hidden contexts by clustering the data instances assuming that the similarity of context is reflected by the degree to which instances are well classified by the same concept. Based on the identified clusters, a set of models is constructed [34].

An alternate way to combat concept drift is the Monte Carlo simulations approach which can be used to measure the robustness of algorithms with respect to model drift. Monte Carlo simulation is an experimental method which relies on repeated random sampling. Through repeated sampling, data-specific results are eliminated by finding averages across multiple runs thus decreasing the variance in results [55]. Deep learning algorithms (especially in semantic classification problems) can be used to capture, to some extent, the underlying generative factors that explain variations in the input data. This means that these algorithms possess the ability for the learned representations to help in disentangling the underlying factors of variation. Deep learning algorithms can extract features that somewhat disentangle the underlying factors of variation since there could be generic concepts that characterize the results [29]. However, deep learning solely is prone to overfitting so, it is not resistant towards model drift.

There exist statistical approaches to detect concept drift through the use of statistical hypothesis testing. A statistic is computed from the available data which is sensitive to changes between two sets of data (training data
and incoming data). The measured values of the statistic are then compared with the expected value under the null hypothesis that both samples are taken from the same distribution. The p-value obtained can be used as a measure of the strength of the drift. A good statistic should be sensitive to data properties that are likely to change by a large margin between multiple samples. It is insufficient to look at mean or variance based measure because distributions can differ with the same mean or variance range. Rank-based measures such as the Mann-Whitney or the Wald-Wolfowitz statistics are successful in non-parametric drift detection because they are sensitive to higher order moments [24].

2.3.2 Retraining Models With New Data

In the context of managing machine learning solutions in production, there are four key challenges [60] involved in machine learning model management and deciding what type of data needs to be prepared for model retraining.

- The first challenge is understanding the data accounting for adverse model performance and the data to be used for retraining the model. Understanding data includes generating and visualizing features with respect to the data (ex. range, statistical distribution and correlations), outliers, encoding of data, identifying explicit/implicit data dependencies in order to recommend and generate transformations on the data to features based on data characteristics automatically.

- The second challenge is data validation before retraining a new model because data validity affects the quality of the machine learning model. Validation can have varied meaning depending on the context such as ensuring that training data have the expected features, expected values, expected feature correlation and statistically not different from the training data.

- The third challenge is that of data cleaning after validation which involves understanding where the error occurred, the impact of the error, and fixing the error. Data cleaning is important prior to using the data for retraining to avoid spurious cases.

- The fourth challenge is to augment the training and serving new data with new features to improvise on the machine learning model. This is typically achieved by joining the new data source to augment the existing features with new signals or using the same signals with different transformation (example through embeddings for text data). The
challenge here is to find a way to enrich data through additional signals or transformations which can then be fed as an input for retraining a model.

Thus, it is important to keep track of both the model performance as well as the statistics of trained data and incoming (new) data.

2.4 Existing Solutions

In this section, description of existing solutions and their functionalities are described. A summary of the features for the solutions are presented in Table 2.1.

2.4.1 MLflow

MLflow [53] is an open source project which works with any machine learning algorithm, library, language or deployment tool. This is done through the use of REST APIs and simple data formats (model viewed as lambda function). It consists of three components. MLflow Tracking is an API and UI which is used to log parameters, version control, metrics and output files with respect to the machine learning models. MLflow Projects is used to provide a standard format for packaging reusable data science code. MLflow Models is used to packaging machine learning modes in multiple format or flavors through the use of inbuilt tools [19]. MLflow Tracking has code version control (locally/remotely) and recording features for the meta-data for each experiment. Models can be exported to be ready for deployment on multiple platforms such as Azure and AWS Sagemaker. There is an inbuilt feature to track the performance of machine learning models.

2.4.2 PipelineAI

PipelineAI [59] is a realtime enterprise textitAI solution which continuously trains, optimizes and serves machine learning models on realtime streaming data directly into production. Graphical Processing Units (GPUs) and x86 processors are used to host instances of docker which allows frameworks that need to access realtime data. The features include validation and comparison of models, training models with realtime data and optimizing models automatically, productization with a machine learning model, migration between different platforms and an option to view features and predictions in a visual format. However, automatic optimization of models with incoming data
may not always result in better performance, give false predictions and can have an immediate impact on the service provided by the enterprise. This can usually happen when the incoming data being recorded is anomalous because of a fault in the data recording medium. PipelineAI uses containers to deploy the machine learning model locally and Kubernetes to deploy it in production. The evaluation metrics are embedded into PipelineAI’s dashboard which can be viewed locally or online. It provides support for A/B testing and multi-armed bandit in production along with version control and rollback options.

2.4.3 PredictionIO

PredictionIO [14] is an open source machine learning server with an integrated graphical user interface (GUI) to evaluate, compare and deploy scalable algorithms, tune hyper-parameters (manually/automatically) and evaluate the machine learning model training status. There is an API included which can be used for prediction retrieval and data collection. The advantage is that PredictionIO is horizontally scalable with a distributed computing component based on Hadoop. However, feature selection, online evaluation, support for extended or custom algorithms is unavailable [14]. The features include productization of models with customizable templates, respond to dynamic queries in realtime once deployed as a web service, evaluate and tune multiple engine variants systematically; and support machine learning and data processing libraries. The application runtime platform uses a serverless, cloud-agnostic architecture. Models are deployed using Docker containerization.

2.4.4 H2O.ai

H2O.ai [13] is an open source platform which allows deployment of AI and deep learning problems to solve complex problems which can be easily interfaced with languages such as R, Python, Scala and Java to create complete analytical workflows. It uses in-memory compression to handle billions of rows in-memory even with small clusters. It can run in standalone mode, on Hadoop, or within a Spark Cluster. It includes common machine learning algorithms such as generalized linear modeling, K-Means clustering and Word2Vec along with implementation for algorithms at scale, such as distributed random forest, gradient boosting, and deep learning. The features include algorithms developed for distributed computing, automating the machine learning workflow which includes automatic training and tuning of many models within a user-specified time limit and deploy POJOs and MO-
JOs to deploy models [33]. However, there is no model management nor is there any data or model monitor included in this service.

2.4.5 Azure ML Services

Azure ML Services [52] is a web service which provides a machine learning model authoring environment which enables creation and publishing of machine learning models. It includes functionality for collaboration, versioning, visual workflows, external language support, push-button operationalization, monetization and service tiers [70]. The features include automated machine learning (to select the best algorithms) and hyper-parameter tuning, version control for experiments, manage and monitor models using the image and model registry, upgrade models through Azure-integrated CI/CD and containerization. The model management is taken care through Azure’s own model management service.

2.4.6 Pachyderm

Pachyderm [57] is an open source workflow system and data management framework which overcomes challenges such as data size, reproducibility of results by enabling a reliable way to run processing stages in any computational environment, providing a well defined way to orchestrate those processing stages; and providing a data management layer that tracks data as it moves through the processing pipeline. This is achieved by creating a data pipelining and data versioning layer on top of projects from the container ecosystem, having Kubernetes as the backbone for container orchestration [56]. There are six key features. The first feature is Reproducibility, which means the ability to consistently reconstruct any previous state of the data and analysis. The second feature is Data Provenance, the ability to track any result all the way back to its raw input data, including all analysis, code, and intermediate results. The third feature is Collaboration, with other developers. The fourth feature is Incrementality, which means that results are synchronized with input data and no redundant processing is performed. The fifth feature is Autonomy, in terms of selecting toolchain and deployment strategies. The sixth feature is called Infrastructure Agnostic, which means the ability to deploy on any existing infrastructure. Pachyderm lacks the ability to visually monitor the data and model in production.
2.4.7 Google Cloud ML Engine

Google Cloud ML Engine [30] is used to train machine learning models at scale and then use the machine learning model to make predictions about new data. It provides services to train machine learning models, evaluate model accuracy, tune hyperparameters, deploy the model, send prediction requests to the machine learning model, monitor predictions on an ongoing basis and, manage machine learning models and model versions. The model management is implemented using model resources in Google Cloud ML Engine in the form of logical containers for individual implementations of models. However, it lacks a visual way to evaluate the machine learning model.

2.4.8 Amazon SageMaker

Amazon SageMaker [3] is a fully managed platform that enables developers and data scientists to quickly and easily build, train, and deploy machine learning models at any scale. It allows selection and optimization of the algorithm and framework for the application. It includes hosted Jupyter notebooks that makes it easy to explore and visualize training data. Through the Amazon SageMaker it is possible to train machine learning models with a single click with all the infrastructure managed automatically with the option to scale train models at petabyte scale with auto-tuning of parameters. Finally it makes it easy to deploy in production so predictions can be generated.

2.4.9 Evaluation Of Existing Services

All existing solutions provide the ability to deploy models and most are able to manage models, as shown in Table 2.1. However, not all solutions can be deployed on any platform (platform-agnostic). Most existing solutions do not provide the ability to monitor incoming data (statistics) and the performance of the model. A visual feedback is relevant in business critical use cases where domain-knowledge is required to detect anomalous entries and careful selection of training data is needed for retraining a model. Compared to the other existing solutions, ScienceOps lacks the ability to automatically retrain the model by analyzing anomalies and selecting the new training data. While this is possible to integrate to the ScienceOps solution, it is out of the scope for this thesis. The manual updation of models (retraining) means a new machine learning model can effectively replace the old model while keeping the other architectural components and connections intact.
CHAPTER 2. BACKGROUND

Table 2.1: Comparison : Existing Services

<table>
<thead>
<tr>
<th>Solution</th>
<th>Model Deploy</th>
<th>Monitor Models (Visual)</th>
<th>Monitor Incoming Data (Visual)</th>
<th>Model Management</th>
<th>Update Models (Retraining)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLflow</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>PipelineAI</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Automatic</td>
</tr>
<tr>
<td>PredictionIO</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>H2O.ai</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Automatic</td>
</tr>
<tr>
<td>Azure ML Services</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Automatic</td>
</tr>
<tr>
<td>Pachyderm</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Manual</td>
</tr>
<tr>
<td>Google Cloud</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>ML Engine</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Manual</td>
</tr>
<tr>
<td>Amazon SageMaker</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Manual</td>
</tr>
<tr>
<td>ScienceOps</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Manual</td>
</tr>
</tbody>
</table>

2.5 Infrastructure Services

Microsoft Azure [2] is an agglomeration of various inter-operable cloud computing services managed by Microsoft. Azure leverages the cloud computing concept in order to enable and instantly provision building, testing, deploying and managing (web) applications/services through Microsoft’s data centers. It provides Software as a Service (SaaS), Platform as a Service (PaaS) as well as Infrastructure as a Service (IaaS).

Azure IaaS enables quick set up of development environments, web application interfaces, storage/backup/restore solutions and high-performance scalable computational environments. Azure PaaS [11] offers servers, storage, networking and database management systems which alleviates the time and resource spent on configuring and setting up the the OS configuration. Azure SaaS provides a complete software solution and enables hosted applications, development tools and applications such as Power BI to be easily deployed and used.

Azure Machine Learning Workbench (Workbench) [48] is a visual AI powered data wrangling, experimentation and lifecycle management tool. This cross-platform Azure based client tool provisions a central graphical user interface (GUI) for script version control and enabling script creation. The interactive data preparation tools simplify data cleaning, transformation and provides the ability of running scripts on Spark or a local/remote Docker container; thus easing portability to multiple platforms. The ability to package and deploy a machine learning model to Docker, Spark (HDInsight), Microsoft Machine Learning Server or SQL Server allows it to be used as a development base.

Docker [42] is an open source program which enables virtualization at
CHAPTER 2. BACKGROUND

an operating system level, called as containerization implemented through containers. Containers can be viewed as lightweight virtual machines which allow setting up a computational environment including configurations, execution dependencies and data files within an image. An image is a file, which consists of multiple layers used to execute the code inside a Docker container. The computational environment is defined by infrastructure configuration and commands stored in a Docker script. The images can then be distributed and run on any compatible platform. Since the docker images share the kernel with the underlying machine, the image sizes are small with a high performance [18].

Today, modern applications are built from existing components and are dependent on other services. Through the use of Docker (and the containerization concept in general), the problem of conflicting dependencies, missing dependencies and platform differences are resolved. Docker Images are created by including specific dependencies as per the use case, which then are used to create runtime containers which ensure the exact same execution environment.

Flask [31] is a lightweight micro-framework for Python based on Werkzeug, Jinja 2 and Click enabling building of web applications while supporting data sources.

Azure Machine Learning Model Management [46] provides the ability to manage, package and deploy machine-learning models and workflows as REST APIs. This service is useful for enterprise level solutions because of its inbuilt ability to track models in production, provide model versioning (through registry of model versions), creation of (Linux-based) docker containers with the model with prediction API and capture model insights in AppInsights for visual analysis. Through Azure Machine Learning Model Management, images can directly be deployed on developer machine, organizational servers or IoT edge devices, offering multiple choices for enterprise level solutions.

Kubernetes [9] is an open source cluster manager for Docker containers, essentially decoupling application containers from the system details on which they run on. Kubernetes schedules containers to use raw resources. Decoupling simplifies the development lifecycle to cater to abstract resources like memory and cores. The real power of (Docker) containers stems from the implementation of distributed systems where each group of containers has a unique IP address that is reachable from any other group of containers in the same cluster.

Microsoft Azure Container Service (ACS) [7] is a Container Service Platform which uses Kubernetes, Docker Swarm or DC/OS orchestration tools. Once a cluster is deployed (with containers), Kubernetes can be used for
CHAPTER 2. BACKGROUND

orchestration operations, for example: list container instances in the cluster, running containers and view status of containers. There is a master control plane, a cluster state storage system and container instance(s) in the form of node agent(s). Developers can deploy containerized applications either through the user interface or through providing YAML/JSON definitions which include image name and resource allocations. Docker Swarm is an alternate orchestration tool used for deploying containerized applications across pool of Docker hosts or in ACS container-instances. Datacenter Operating System (DC/OS) is yet another alternative for managing and deploying containerized applications. DC/OS on ACS includes two natively implemented orchestrators called Marathon and Metronome. Marathon manages scheduling and execution of containerized applications, along with long running jobs. It provides a user interface through which developers can spin up a new container on the ACS-DC/OS cluster. Metronome manages batch jobs (short in nature) and configure the container in JSON format. Azure Kubernetes Service (AKS) takes this concept further where enterprises do not have to worry about patching, upgradation, scaling, and managing the clusters [45].

Azure Container Registry (ACR) is a container repository which enables building, storing and managing images for container deployments using DC/OS, Docker Swarm and Kubernetes. The docker registry and deployments are maintained within the same data center, which enables ACR to significantly reduce network latency and additional cross-platform costs in an enterprise scenario.

Azure Blob Storage [35] is an object storage solution for the cloud which is optimized for storing huge amount of unstructured data such as text/binary data. It enables storing of files for distributed access and streaming content and serving files directly to the browser. Data in blob can be accessed easily through an Azure Storage Account. Data is organized and stored within containers, which provides a logical grouping and the level of sharing can be defined.

Apache Spark [20] is an open source cluster computing framework which is used for big data processing. Spark maintains MapReduce's linear scalability and fault tolerance and has several APIs available in Python, Java, Scala along with core data abstraction, distributed dataframe with support for interactive queries, streaming, graphic processing and machine learning [66]. The Azure inbuilt implementation has been used for ScienceOps on Azure to process and generate statistics on the incoming data.

Azure Databricks [51] is an Azure Apache Spark based analytics platform which includes the entire open-source Apache Spark cluster technologies and capabilities, thus providing a unified analytics engine for large-scale data pro-
cessing. Apache Spark in Azure HDInsight is the Microsoft’s implementation of Apache Hadoop in the cloud.

*Azure SQL Database* [49] is a database-as-a-service (DBaaS) based on the latest stable version of Microsoft SQL Server Database Engine. *Azure SQL Database* is chosen for *ScienceOps* because it runs on the latest stable version of SQL and patched with OS integrated in the *Azure* platform. It has compatibility with various external services such as *PowerBI*, and is useful for visualization related queries from external services.

*Power BI* is an enterprise business analytics tool used to deliver visually interactive and informative insights. Due to its ability to connect to several hundred data sources and perform data filtering, it is a suitable tool for enterprise solutions.
Chapter 3

Solution Architecture

The goal of this chapter is to define ScienceOps’s architecture and architectural workflow when used on the Azure platform. The ModelDeploy service packages a machine learning model and its functionality to make predictions with the model into a versioned build artifact and uploads to a central repository which can be used to create a web service. The DataMonitor exposes a dashboard with historical summary statistics about data modeling to understand when a model should be retrained. The ModelMonitor exposes a dashboard showing the deployed models, the performance measures and statistics of predictions over time.

In this chapter, the solution architecture of ScienceOps is presented. This architecture consists of three service blocks, called ModelDeploy, ModelMonitor and DataMonitor. The ModelDeploy block is responsible for operationalizing a machine learning model. This block enables reduction in the time to production for a model to optimize costs in enterprises. Models and the files used for prediction are containerized before being deployed which allows portability of the solution to other platforms. The ModelMonitor block is responsible for visually monitoring the performance of a machine learning model. This service allows to track the performance of the model as new data is scored against the model. The DataMonitor block is responsible for visually monitoring the statistics of the incoming data versus the training data. This service allows to track the statistics of the incoming data versus historical data used for training. If the statistics deviate, a domain-knowledge expert can flag the new data as incorrect or flag it to be used for retraining the machine learning model, thus maintaining optimal performance of the model. Through the visualization services (ModelMonitor/DataMonitor), challenges in terms of understanding the data as listed in Section 2.1.3 can be evaluated and mitigated as per the use case.
ScienceOps is an aggregation of various interconnected services requiring IaaS, PaaS and SaaS resources (when used on a cloud platform). When the ScienceOps architecture is used with respect to a platform to build a solution, the constituent services building the architectural blocks get replaced with the constituent services available on that platform. For example: a SaaS tool available on Azure may have a similar service bundled differently on Amazon or Google Cloud platform. The ScienceOps architecture is realized on a platform through workflows, which means the way individual services on a particular platform are built and connected to build the three services mentioned above. In general, the functionality and the workflow remain similar irrespective of the platform. Using the ScienceOps architecture with respect to the cloud platform has its advantages and disadvantages as listed in Section 2.1.1 and helps mitigate the challenges with respect to resource constraints such as volume, velocity and storage as listed in Section 2.1.2.

ScienceOps workflow as part of this thesis is revolved around Microsoft Azure. Thus, the components described in this solution architecture are with respect to the services available on Azure. It is possible to use the same workflow on other similar cloud providers such as Amazon Web Services and Google Cloud Platform using similar services to that of Azure. Implementation on the Amazon platform is briefly discussed in Chapter 5.

Figure 3.1: ScienceOps Workflow: Azure
ScienceOps consists of three services (ModelDeploy, ModelMonitor and DataMonitor), and a web interface which would allow the end-user to score data against the machine learning model. The first step of the ScienceOps architecture is building the ModelDeploy service which includes training the machine learning model and operationalization of the model through the use of containers. This is followed by a bundled ModelMonitor and DataMonitor service which includes processing and logging of the model/data statistics over time, which can then be visualized. A web interface allows a client/user to score new data against the deployed model through an API call. Fig. 3.1 demonstrates the ScienceOps solution architectural workflow on Microsoft Azure.

ScienceOps workflow for Azure uses several IaaS, PaaS and SaaS resources and services from Azure. Below are the components in the solution architecture described and explained in the context of ScienceOps. The description is laid out in the same order as Fig 3.1.

### 3.1 ModelDeploy Components

In the ModelDeploy architecture solution, a machine learning model (pickle file), scoring script for predictions and schema file to define the format in which data will be accepted for predictions; is created. These three files are packaged inside a Docker Container to spin-up a web service. The task of containerization to spin up a web service as part of building the service is constant irrespective of the platform. The three constituent (micro) services which build up the ModelDeploy service block are described below.

#### 3.1.1 Operationalizing Machine Learning Model

The first constituent service required to build the ModelDeploy service should enable development of a model, creation of compute clusters and a web service exposing an API through which the client node can score new data against. In the case of Azure, this is performed by a bundled service called Azure Machine Learning Workbench (Workbench) and Azure Command Line Interface (Azure CLI).

ScienceOps workflow on Azure uses Workbench for importing data sources (post feature extraction), data preparation, running the model training script, build a scoring script and schema file and finally operationalizing it by deploying a web-service through Azure CLI. Azure CLI is a command-line tool used to manage Azure resources. Thus, Azure Machine Learning Workbench serves as the crux of building a machine learning web service.
3.1.2 Docker and Flask

*ScienceOps* workflow, irrespective of the platform, revolves around the use of containers. *ScienceOps* leverages *Docker* to ensure portability with respect to deploying the model on any platform. *ScienceOps* uses *Flask* to build a web application user-interface where the user can enter a query to retrieve the prediction result. The Flask application is containerized using *Docker* making it portable across platforms without dealing with the underlying infrastructure. Additionally, the *HTTPS* port of the docker container is exposed to the user-node to display the user-interface of the web application.

3.1.3 Machine Learning Model Management

The second constituent service required to build the *ModelDeploy* solution should provide the ability to perform version control for tracking models, keeping track of *Docker* images and deploy web service on a container service cluster. In *Azure*, this is performed by *Azure Machine Learning Model Management*. Data with respect to each trained and deployed model is tracked, which allows the mitigation of challenges listed in Section 2.2.

In *ScienceOps*, when productizing the machine learning model through the *Azure CLI* in the *Azure Machine Learning Workbench*, the model management registers the model (pickle) created in the Workbench, creates a manifest, creates a *Docker image* and finally deploys the web service. Version control functionality in the *Azure Machine Learning Model Management* helps maintain and restore models through a simple user-interface. The user can send a query to the machine learning web service via an *API* call which can be delivered via command-line or web-interface; and receive a response with the prediction. In order to deploy a model through model management, there are four steps:

1. Registering the machine learning model in the *Model Management* in order to tag and describe it.

2. Create a manifest which includes the machine learning model and the dependencies allowing it to run as a service. This manifest describes the conda [4] dependencies to be executed at runtime in the execution environment (Python/PySpark), the scoring script and the schema file which will validate incoming unseen data.

3. Create a *Docker* image to install system dependencies, thus allowing a uniform execution environment across different platforms. This docker image is then used to create one or more container(s) running that service.
4. Create a web service by hosting the service on the running container and exposing an API.

3.1.4 Container Service

The third constituent service required to build the ModelDeploy solution should provide the ability to create, deploy and manage virtual machine clusters where the containers will run. This task is performed by Azure Container Service (ACS) in Azure.

Azure Container Service enables creation, configuration, deployment and management of virtual machine clusters and are pre-configured to run containerized applications. Since ACS leverages Docker images, it enables multi-platform portability. ACS is where the machine learning model is deployed in the context of ScienceOps workflow on Azure. The image created during the ScienceOps model operationalization is stored and managed in the Container Registry and Docker Image Storage. For enterprise solutions, ACS is useful because of its ability to orchestrate and scale containers using Kubernetes and Docker Swarm; as well as exposing APIs such as REST APIs, which are complemented with authentication, load balancing, automatic scale-out and encryption services. ScienceOps uses Kubernetes to manage the cluster of deployed containers on the Azure platform.

ScienceOps employs ACS because Windows platform containers are not yet supported on AKS; and to take advantage of granular controls and configurations tweaked as per the use case (for example configuring the head node of a compute cluster). The Docker containers are deployed into the Azure Cloud Services Kubernetes cluster. Application Insights is used to monitor the live web application (deployed model) in order to detect performance issues, help diagnose web service issues and see usage statistics of the web service.

3.2 ModelMonitor and DataMonitor Components

The ModelMonitor and DataMonitor services consists of four constituent (micro) services which are described in the subsections below. The general workflow for these two service blocks in the architectural solution is that the user queries the web service for scoring new data. The input as well as prediction result is stored and processed to generate statistics. The statistics
of the data is logged into a relational database. This database is used for generating visualizations.

### 3.2.1 Blob Storage

The first constituent service should be able to store the incoming data as well as predictions generated from the model. In Azure, this is handled by Blob Storage. All the incoming (unseen data), the prediction results and model meta-data in ScienceOps is transferred and stored in the (Block) Blob storage. Blob storage is a good choice for unstructured data where the format of data is variable and structure is not consistent, as compared to traditional (SQL) databases where a proper data structure and format is required [22].

For example, a new model built on more/less features than the original model will change the number of data-points being recorded. In a relational database, the schema would have to be altered each time there is a change in the number of data-points, unlike Blob, which is resistant to such changes. Blobs are organized and reside within a container, and can belong to three categories [50]:

1. Block Blobs: To store text/binary data up to 4.7 TeraBytes. Typically sufficient for files found in enterprises.

2. Append Blobs: Similar to Block Blobs, but are optimized for append operations.

3. Page Blobs: To store random access files up to 8 TeraBytes. Commonly used to store Virtual Hard Disks (VHDs).

### 3.2.2 Azure Databricks

The second constituent service is responsible for processing the stored incoming data and the predictions to generate statistics. In Azure, this is handled by Azure Databricks. Azure Databricks is chosen for ScienceOps because batch/streaming data can be processed at a high rate (as compared to single thread Python scripts) using Spark and can be integrated with data store services such as Azure SQL Data Warehouse and Azure Blob Storage. This service is especially useful when the data to be processed is in the form of stream by easily scaling up or down the service based on the volume of data.

Azure Databricks, in the ScienceOps context, processes both the predicted as well as user-input (unseen data) stored in the Blob storage to generate data and model related statistics. This processing is vital in order to monitor the incoming data quality as well as the performance of the model (prediction
results). The statistics are also important to understand when the model needs to retrained.

### 3.2.3 Azure SQL Database (DB)

The third constituent service should be able to store the calculated statistics in a tabular (relational) manner. Relational method is preferred in order to carry out querying tasks such as finding the statistics between two time periods or finding statistics for a particular feature. On the Azure platform, this is handled by *Azure SQL Database (Azure DB)*. If done using flat files stored on *Blob*, querying will be slow due to absence of indexing of the data.

Statistics of the data processing in *Azure Databricks* is stored in the *Azure DB*. For every use case, the calculated statistical metrics are defined as per the use case (example: mean, median, mode and variance). Since the format in which these metrics are stored is relatively constant, a relational database is suitable. Metrics for each feature is stored as a tuple, making it resistant to changes in the number of features being recorded from the model. For example, the tuple can consist of the machine learning container image ID, model ID, the statistical metric name (mean and median) and the value of the metric. If any statistical metric is added or removed, the number of tuples added in any new iteration is increased/decreased.

### 3.2.4 Power BI

The fourth constituent service should be able to present the statistic results in a visual manner for a developer or domain-expert to evaluate. In Azure, *PowerBI* acts as a dashboard visualization tool depicting the performance of the model over time; and the statistics of the user-input (unseen) data over time. While it is possible to develop a custom dashboard, *Power BI* complements its services with filtering/slicing tools and visual customizability. Through visualization of the statistics, concept drift can be tracked and evaluation of the training data for model retraining can be done to counter the challenges listed in Section 2.2.
Chapter 4

ScienceOps Workflow and Building a Solution

This chapter describes the setup of the Azure workflow of ScienceOps in order to build ModelDeploy, ModelMonitor and DataMonitor services; and an example of how a machine learning task can leverage this workflow to build a solution. The setup of architectural components and the example are described together to showcase how the workflow is used to solve the task. The example described in this thesis is the classification task of the Iris dataset from the UCI Machine Learning Repository [23]. In order to leverage the ScienceOps workflow to build a solution, code reference templates have been provided for the tasks as part of the Appendix. These templates are filled with use case specific scripts.

In order to build the architectural components as part of the workflow, contributor access to a Resource Group on the Azure portal is required which allows code development and architectural resource creation. The workflow follows the order of the sections below:

4.1 ModelDeploy

In this section, we set up resources according to the ScienceOps workflow to build the ModelDeploy service and demonstrate an example of how a machine learning model is deployed using Azure CLI.

The first resource needed to build the workflow is a local installation of Azure Workbench (with CLI) on the development node which acts as the toolkit to develop the model. Further, a local installation of Docker on the same node should be present if the testing of the scripts is done locally in a container, although this step is optional and does not relate to the workflow.
Building of the classification task solution starts with the *Workbench*. The developer performs data preparation (data cleaning and feature extraction) on the dataset. The developer then segregates the processed clean features and the corresponding labels into a training, validation and test set. Fig. 4.1 depicts the workflow before model operationalization in the *Workbench*.

**Figure 4.1: Model pre-operationalization tasks**

Below is a description of how the *Workbench* is set up and a brief description of the data preparation step.

### 4.1.1 Project Setup on Workbench

The project is setup on *Workbench* and the following files are created as shown in Fig. 4.2:

- **aml_config**: Local/Docker Configuration Files
  - **conda_dependencies.yml**: Dependencies required for running the training script.
  - **docker.runconfig**: Add environment variables, Framework (Python) and Path for *conda_dependencies.yml*.
  - **docker.compute**: Type of implementation (local-docker) and Base Docker Image.
• score.py and train.py: Contains the scoring script and training script

Next, import data: Parquet, Excel, CSV or database files and create a data preparation package. This opens up a view for exploratory data analysis and transformation of the data. For this implementation, data stored in Microsoft Excel is selected.

Once the data is prepared, it goes through the data preparation step for transformation of data, if required; for example: data type transformation, replacing null values, appending data and renaming columns. After transformation, a Data Preparation Package is generated which records the transformations made to the data and is maintained in JSON format. Additionally, a Data Access Code File can be generated with a script template which defines the Azure logger and runs the data preparation package to import data using the Data Preparation Package.

### 4.1.2 Training the model

In this step, we are not required to provision any resources with respect to the ScienceOps workflow.

The training script consists of the typical training scripts used in Machine Learning, specific to the use case. However, there are two optional additions to the training script which enhances the functionality of the script as shown in A.1:
CHAPTER 4. SCIENCEOPS WORKFLOW AND BUILDING A SOLUTION

- **Azure Machine Learning Loggers**: The metrics stored are tied to its corresponding running instance for future analysis.

- **Run the data preparation package** to tune the data according to the correct configuration.

Once the training script is ready, run it in local mode (to iron out any errors) followed by execution in docker mode. Optionally arguments can be supplied in the GUI to emulate command line arguments. The local/docker selection on the *Workbench* and the way to run it is shown in Fig. 4.3.

![Figure 4.3: Train and Pickle](image)

The developer performs model training and saves the model in the form of a *pickle* file. This process is called *pickling*. *Pickling* is essentially serializing and de-serializing a Python object structure which means conversion into a byte stream or vice-versa. This pickle is then stored on the local disk for reusability purposes. An alternate approach to running the training scripts is through *Azure CLI*. A.3 shows how to execute the training script locally, on a local docker or on a remote docker environment along with hints on potential modifications required. In case of remote docker environment, first a compute target is attached with a name, IP address and credentials followed by the preparation step of the compute target which creates a docker image in the remote virtual machine. The remote server must allow connections through the firewall and must be enabled separately.

### 4.1.3 Scoring and Schema

In this step, we are not required to provision any resources with respect to the ScienceOps workflow. In order to obtain prediction results from the web service, it is important to have a prediction script which evaluates and delivers results. In order to use the web service for scoring data, it is vital to have a defined schema. The schema defines the format of the data to be used for the web service. A.2 defines the scoring and scheme generation script. The output of this script is a *JSON* file containing the schema. In the A.2 code, first the *Pandas Dataframe* [41] is populated with the data and corresponding column names. Then provide a test value as the input. For generating the schema, we use *generate_schema* method from *azureml* library which takes the following arguments:
• Prediction Function (run method) which returns the prediction result in JSON format.

• Test input in the form of Pandas Dataframe.

• The file path for the schema output (to be saved).

Once the pickle file, scoring script and the schema file is created, the minimum file requirements to deploy the realtime web service is complete.

4.1.4 Operationalize using Azure CLI

In this step, according to the ScienceOps workflow, we provision the Azure Machine Learning Model Management resource, a compute cluster, container registry, storage resource where docker image is stored and a storage resource to store incoming data/predictions. In addition to this, an Application Insights resource is created which helps track the service diagnostics. In order to provision a web service for the classification task, we first register the machine learning model on Azure. From the registration, a manifest is created containing details of the model, scoring function, the schema by which an API can call the service and the dependencies which the scoring script requires to run. From this manifest a docker image is built. An Azure container service with a cluster is spun up and a web service is deployed on this cluster. The workflow is depicted in Fig. 4.4.

![Figure 4.4: Operationalization of the model](image)

For the classification task, in order to create the realtime web service, we first set up the ACS cluster. This can be achieved through a series of commands as shown in A.4.

1. Setup a cluster by providing the name of the deployment environment name, location of deployment and the resource group name.
2. Create a model management account by providing the name of the model management account, location of deployment and the resource group name.

3. Set the model management account as the default model management account to be used.

4. Create Realtime Web Service by providing :
   - Scoring script
   - Machine learning model (Pickle File)
   - The schema JSON file (optional)
   - Name of the web application
   - Execution environment (Python and PySpark)
   - Flag to enable/disable model data collection (user inputs/predictions into blob storage)
   - Conda Dependencies File (conda_dependencies.yml relative path)

Next, a web service build process is initiated. The creation of the web-service can be broken down into smaller units instead of a single command. However since this offers no significant advantage (except for testing), it is only briefly described as follows :

- Register the model in the Azure Machine Learning Model Management by providing the pickle file. The output would be a model ID as shown in Fig. A.1.

- Create a manifest using the model ID, scoring script, schema file and conda dependencies file. The output would be a manifest ID as shown in Fig. A.2.

- The image is created on the basis of the manifest ID, resulting into an image ID; and the web service is created on the basis of the image ID. The created image and web service is shown in Fig. A.3 and Fig. A.4 respectively.

- The image is pushed into ACS as depicted in Fig. 3.1.

A user-input request can be queried to the deployed web service through Python using A.6 :

- Parsing the input data into a JSON object.
• Defining the URL of the web service.

• Header indicating the type of data to expect (JSON) along with the API key (retrieved from the web service settings on Azure).

4.1.5 Blob Storage

In this step of the workflow, a storage account is provisioned to store the incoming data and the predictions. Any user data entered into the web service or any prediction result from the web service is captured and stored in Azure Blob Storage as shown in Fig 3.1. The data is by default segregated on the basis of the date. All data from the same date in the same context is stored in one file. While it is possible to manually access data files from the GUI, it is often easier to computationally retrieve files for processing from programming languages such as Python.

For the classification task, in order to access Blob storage from an external programming language, such as Python, the Storage Account Name and Storage Access Key is required. They access key is generated through a single command line as depicted in A.5.

In order to create, download or access blob data (for later analysis) in ScienceOps, the account name and the account key is used as shown in A.7.

This completes the ModelDeploy service of the ScienceOps workflow.

4.2 DataMonitor and ModelMonitor

In this section, we set up resources according to the ScienceOps workflow to build the DataMonitor and ModelMonitor service and demonstrate an example of how the data metrics and the model performance can be tracked.

4.2.1 Azure Databricks (PySpark) and Azure SQL

In this step of the workflow, data stored in Blob needs to be processed to generate statistics out of it. These statistics are then stored in a relational database. The visualizing tool then reads the statistics from the relational database and presents with plots for a domain-expert to analyze.

In the case of the classification task, in order to process large amounts of data stored in Azure Blob Storage, the data is imported into Databricks for processing. A new cluster is created from the Azure Databricks portal setting the desired spark configuration, environment variables, python and spark version. Databricks can automatically scale the number of computational nodes.
CHAPTER 4. SCIENCEOPS WORKFLOW AND BUILDING A SOLUTION

depending on the workload between the minimum and maximum number of nodes defined.

The Azure SQL Database is created through the Azure Portal using a Graphical Wizard. The exact configuration of the underlying schema is specific to each use case, and cannot be generalized.

First a spark session is created by using the PySpark python library by defining the application name and the configuration for Windows Azure Storage Blob (WASB). The PySpark script imports data through the WASB path referring to the data stored in the blob. Statistics for each feature is computed and appended into the Pandas dataframe. Statistics are always calculated for the same model management account, model ID and web service name (data from the same model). The default computed statistics in the ScienceOps implementation include:

- **environment**: The model management account name.
- **web_service_name**: The name of the web service which generated the data.
- **model_id**: Uniquely identify the model which generated the data in the blob.
- **prediction_date**: Timestamp (which can also be used to define statistical time-based buckets).
- **feature_name**: Name of the feature for which the statistics is being computed.
- **mean, median, standard_deviation, minimum and maximum**

Additional statistics can be added based on specific use cases. Once the pandas dataframe consisting of statistics is created, it is converted to a spark dataframe. The spark dataframe is then written in append mode into the table in the SQL Database through Java Database Connectivity (JDBC) connection. The implementation template is shown in A.8 which creates the statistics.

4.2.2 Power BI

According to the ScienceOps workflow, the visualization tool is used to present the plots for data statistics and model monitoring. In the visualization tool, there are two views:
Model Monitor

Data Monitor

Power BI acts as the visualization tool for the classification task. The data is queried from the Azure SQL Server. There are two types of connections from Power BI to a data source:

- Import
- Direct Query

The Import type of connection which will retrieve data from the data source locally. The data is not automatically retrieved from the server, which would result into manual importing of data through Power BI each time data refresh is required. Enterprise datasets can be huge and often change quickly; therefore Import connection type is seldom used. This connection is however useful when the data is not updated frequently, there is no good internet connection (thus locally storage data preferred) and when there are heavy manipulations to be done on the data.

Direct Query connection type is the default connection type used by the ScienceOps implementation. This requires a connection with the data source because data is not loaded locally into Power BI - it simply retrieves new query results as the user interacts with the visuals. Data is queried at run-time, thus it is a more practical and common solution for huge datasets which change quickly. However, this requires a good internet connection (commonly available at enterprise premises) and the data can be refreshed with not less than 15 minutes frequency. According to the ScienceOps workflow, statistics are calculated on a daily basis by default, thus the periodicity of the data refresh does not inhibit functionality.

The ground-truth (correct) label to queries cannot be known when a user input is provided. Therefore, in order to determine the performance of the service, it becomes crucial to investigate the metrics of both incoming as well as predicted data to track statistical deviations. If the type of input data is similar across multiple days, then the predictions should be similar.

Data monitor checks how the data varies across different days, which aids in selecting the type of data for retraining a new model. This helps tackle the problem of model drifting, where updated models are deployed with faulty data included in the training set. Model monitor plays a crucial role when the correct labels are known over the course of time. Then the correct labels can be visually compared against the predicted labels to see the performance of the model. For instance, in a stock prediction use case forecasting the next day’s stocks, the correct labels would be available after one day.
CHAPTER 4. SCIENCEOPS WORKFLOW AND BUILDING A SOLUTION

4.2.3 Model Monitor Service
The model monitor visualizes the statistics of the predicted data. The exact statistics depend on the use case. By default, a count-ratio statistic which displays the ratio of predictions over time, is included as shown in Fig 4.5. The visuals created depend on the use case. For instance, in a text classification task, a histogram could be useful if there are less classes; whereas a word cloud could be more insightful if there are a higher number of classes.

4.2.4 Data Monitor
Data Monitor provides an interface to view statistics of the data being queried by the user. The statistics which are viewed by default are mean, median, standard deviation, minimum and maximum. Additional metrics can be added by computing more statistics in Azure Databricks and then pushed to the Azure SQL Server. In an enterprise scenario, there can easily be hundreds or even thousands of features in a dataset. Visually analyzing all the features in a single plot would yield no visual cues for an analyst, nor would having one graph per feature bring out the statistics which shows a major change in behaviour. Thus, ScienceOps, by default, retrieves data where the Coefficient of Variation is higher than a certain threshold. This criteria is used by default because it is used to implicate the level of variability for a given population without any dependence on the observation’s absolute value. Other metrics can be employed as per use case such as rate of change in the moving average. Features are then plotted on graphs on Power BI where each graph represents a statistic (example : mean and median). This is shown in Fig 4.6.

Features may have a very varied level of statistic; for example: three features with a mean within the range of 0 to 3 and another feature with a mean ranging above 1000. On a graph, the feature with the very high mean range can easily over-shadow the rest of the data. Power BI can use filtering to view one or more features and the graph scaling is adjusted accordingly to easily analyze from the graph.

4.2.5 User Interface For Input
The user can enter data for prediction via a web interface. This data is pre-processed and scored against the machine learning model through an API request. The pre-processing is performed using the same technique applied on training data. The user then receives the predictions back on the user-interface.
CHAPTER 4. SCIENCEOPS WORKFLOW AND BUILDING A SOLUTION

Figure 4.5: Model Monitor on Power BI

Figure 4.6: Data Monitor on Power BI
Chapter 5

Discussion

5.1 Evaluation

The ScienceOps architecture was successfully deployed and tested with the Iris dataset from the UCI Machine Learning Repository. The success was measured based on the creation of the web service, verification through API calls, ensuring model data storage (input/output), verify the stored data being processed and being stored in a tabular format and correctly displayed on the visualization tool. A summary of the evaluation can be found in Table 5.1.

5.1.1 Azure Workflow Evaluation

First the ModelDeploy service was created. ModelDeploy service creation took 160 minutes. This is primarily due to the time Azure takes to provision resources. Files are uploaded at this stage (model pickle file, scoring script and schema file) - if the model pickle file is huge, it can take more time. In our testing phase, the pickle file creation took 2 minutes and uploaded in a matter of a few seconds.

- Installation of Machine Learning Workbench took 20 minutes.
- Creation of training script, scoring file and schema file took 45 minutes.
- Development of a machine learning model took 5 minutes.
- Provisioning machine learning model management account, storage account, container registry, container services and deploying the web service took 70 minutes.
• Development of a simple user interface where a user can enter data in a text field to score against the model, took 20 minutes.

The next step was to create the ModelMonitor and DataMonitor service. Testing the API calls to the machine learning web application, creation of database, the database connection test and creation of a Databricks instance took 50 minutes. Setting up PowerBI, setting up a connection with the database and adding basic visualizations took 45 minutes. The ModelMonitor and DataMonitor service was ready in 95 minutes.

In total, the entire setup starting from machine learning model to a functioning machine learning service took 4.25 hours. Service Level Agreements (SLAs) are taken care by the cloud platform service provider (in this case Microsoft).

5.1.2 Rudimentary Workflow Evaluation

The rudimentary workflow evaluation was conducted on a Linux virtual machine deployed on Microsoft Azure. First the ModelDeploy service was created, similar to the ScienceOps workflow evaluation.

• Creation of training script, scoring file and schema file took 45 minutes.

• The configuration of a version control service took 90 minutes. This involved setting up repositories on Azure Repositories [43], installation of Git LFS (Git Large File Storage) [10] and configuration. The reason Git LFS was chosen is because it replaces large files (such as large pickle files) inside Git with text pointers and storing the file contents on a remote server.

• In order to operationalize the model, a container is built which took 20 minutes.

However the stand-alone container required additional configurations to be able to meet high loads with secure communication. The configurations took 90 minutes and included:

• Port 443 access through the firewall (for HTTPS connections)

• Generation of self-signed certificates for server verification through OpenSSL [73]
CHAPTER 5. DISCUSSION

• Production web server to be chosen to be Gunicorn [17]. The advantage of using Gunicorn is that it will run enough aiohttp [1] processes in order to utilize all available CPU cores, inbuilt security features and configurability of the web server.

NGINX [62], which is a web server which can be used for load balancing and reverse proxy, has not been used for simplicity purposes. In an ideal implementation, NGINX should be included for enterprise solutions [54] and its configuration will also contribute to time to production. Development of a simple user interface where a user can enter data in a text field to score against the model, took 20 minutes. The ModelDeploy service was ready in 270 minutes.

The next step was to create the ModelMonitor and DataMonitor service. Testing the API calls to the machine learning web application and development of a script to store the input data for the model and the predicted data took 20 minutes. It takes 40 minutes to setup a SQL Database where the statistics of the stored data will be inferred by the visualization tool. Setting up a Spark environment takes 90 minutes with several modifications required in configuration files.

A custom visualization webpage was built using D3.js [78] in 120 minutes. The statistics were updated every 3000 milliseconds (configurable). More frequent update rates would cause a high load on the server and decrease in the performance. In contrast, PowerBI can update the visualizations in increments of 15 minutes (minimum). So a custom visualization webpage proved to be better for close to realtime visualizations. The ModelMonitor and DataMonitor service was ready in 270 minutes.

In total, the entire setup starting from machine learning model to a functioning machine learning service took 9 hours. The advantage of this approach is the close to realtime monitoring services, however this can be combated by attaching a different visualization setup to the ScienceOps implementation. The disadvantage of this approach is that it takes more than double the time to productize a machine learning model with deployment and monitoring services. A lot of configurations are required to ensure secure communication between different data processing and storing related services. This setup cannot be easily scaled-up or scaled-down in terms of compute power and storage capacity. Furthermore, this setup will not perform well with multiple requests being handled simultaneously because of the lack of a compute cluster. Overcoming all these disadvantages contributes to overhead and further increases time to production.

The DataMonitor service shows the statistics of the features, as shown in Fig 4.6. In the Iris dataset example, if the mean or variance starts to
change, the new data must be evaluated to verify non-existence of anomalies
and used to retrain the machine learning model.

The ModelMonitor service shows the prediction of classes and how they
are distributed, as shown in Fig 4.5. In the Iris dataset example, if the ratios
start changing, the model needs to be retrained. Once labels for the new data
is received, the ModelMonitor service can help track accuracy levels.

5.1.3 Amazon Workflow Evaluation

An independent experiment by a developer was conducted to use the ScienceOps
architectural workflow on Amazon AWS.

The idea of ScienceOps is to be portable so that enterprises can use
the service with the choice of their cloud platform. Currently there major
cloud platform providers include: Amazon AWS (Amazon Web Services),
Microsoft Azure and Google AppEngine [38]. The scope of this thesis has
been limited to Azure. However, similar implementations are possible on
other cloud platforms. Let us consider the case of Amazon Web Services.
The steps of creating a machine learning model are largely independent from
cloud services, unless specific cloud services are explicitly used.

First, the ScienceOps workflow on AWS is described followed by the
evaluation of the time taken to productize a machine learning model.

The workflow starts with the development of the machine learning model,
as described in the rudimentary workflow. Once the scoring script, schema
file and machine learning model is created, developers can upload it to a
central repository which enables the use of version control through AWS
CodeCommit. AWS CodeCommit is a managed source control service that
hosts private repositories [25]. Once the files are uploaded AWS CodeBuild is
triggered which is a fully built manager on AWS which builds the code in the
cloud service. The AWS CodeBuild process follows CI-CD (continuous inte-
gration and continuous delivery) practices, which means that once the test
is executed and passed, it is compiled and the code is released. This helps
speed up the release process [63] in enterprises. Next the solution is con-
tainerized through Amazon Elastic Container Service (ECS). Amazon ECS
enables provisioning of resources composed of (docker) container-instance
clusters and deploy containerized applications [8]. In this context, Amazon
ECS performs, the equivalent tasks to that of Azure Container Service - the
exact working of both services is slightly different, but both services are used
for the same purpose of containerization. The statistics of the data is stored
in Amazon DynamoDB. DynamoDB is a proprietary scalable non-relational
database service [68] - the reason for choosing a non-relational database over
a relational database is to ease the variable nature of data metrics (fea-
Table 5.1: Time To Production: ScienceOps vs. Rudimentary Approach

<table>
<thead>
<tr>
<th>ScienceOps Task</th>
<th>ScienceOps Workflow (minutes)</th>
<th>Equivalent Rudimentary Approach Task on Linux VM</th>
<th>Rudimentary Workflow (minutes)</th>
<th>Amazon Task</th>
<th>ScienceOps Workflow (minutes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Machine Learning WorkBench Installation</td>
<td>20</td>
<td>Local Python Installation</td>
<td>0</td>
<td>Local Python Installation (Spyder)</td>
<td>0</td>
</tr>
<tr>
<td>Training Script</td>
<td>20</td>
<td>Training Script</td>
<td>20</td>
<td>Training Script</td>
<td>40</td>
</tr>
<tr>
<td>Scoring File</td>
<td>15</td>
<td>Scoring File</td>
<td>15</td>
<td>Scoring File</td>
<td>25</td>
</tr>
<tr>
<td>Schema File</td>
<td>10</td>
<td>Schema File</td>
<td>10</td>
<td>Schema File</td>
<td>25</td>
</tr>
<tr>
<td>Model Development</td>
<td>5</td>
<td>Model Development</td>
<td>5</td>
<td>Model Development</td>
<td>5</td>
</tr>
<tr>
<td>Storage/Model Management Service Creation</td>
<td>20</td>
<td>Use Git Large File Storage (LFS)</td>
<td>90</td>
<td>AWS CodeCommit AWS CodeBuild (Version Control)</td>
<td>70</td>
</tr>
<tr>
<td>Container Service</td>
<td>25</td>
<td>Single Container</td>
<td>20</td>
<td>Container Service (ECS)</td>
<td>20</td>
</tr>
<tr>
<td>Operationalizing Web Service on a Compute Cluster</td>
<td>25</td>
<td>Opening Up 443 Port SSL Enable (HTTPS) Production Server: Gunicorn Just One Node (No NGINX)</td>
<td>90</td>
<td>Operationalizing Web Service on a Compute Cluster (Multiple Compute Nodes)</td>
<td>20</td>
</tr>
<tr>
<td>Web Application Usage/Diagnostics Statistics</td>
<td>0</td>
<td>Web Application Usage/Diagnostics Statistics</td>
<td>N/A</td>
<td>Web Application Usage/Diagnostics Statistics</td>
<td>0</td>
</tr>
<tr>
<td>API Testing Client URL (cURL)</td>
<td>10</td>
<td>API Testing (cURL)</td>
<td>10</td>
<td>API Testing (cURL)</td>
<td>10</td>
</tr>
<tr>
<td>Store Incoming Data/Predictions Configuration</td>
<td>0</td>
<td>Store Incoming Data/Predictions Configuration (Local Storage)</td>
<td>10</td>
<td>Store Incoming Data/Predictions Configuration (DynamoDB)</td>
<td>5</td>
</tr>
<tr>
<td>Connect Azure SQL Database</td>
<td>20</td>
<td>Connect Local SQL Database</td>
<td>40</td>
<td>Connect DynamoDB</td>
<td>10</td>
</tr>
<tr>
<td>Azure Databricks Setup</td>
<td>20</td>
<td>Spark Setup (Local Configuration)</td>
<td>90</td>
<td>Python Script Setup (No Spark)</td>
<td>10</td>
</tr>
<tr>
<td>Visualization Setup (PowerBI)</td>
<td>45</td>
<td>Visualization Setup (Developed Using D3.js)</td>
<td>120</td>
<td>Visualization Setup (Developed Using D3.js)</td>
<td>70</td>
</tr>
<tr>
<td>Basic Client User Interface</td>
<td>20</td>
<td>Basic Client User Interface</td>
<td>20</td>
<td>Basic Client User Interface</td>
<td>20</td>
</tr>
</tbody>
</table>

TIME TO PRODUCTION | 255 | TIME TO PRODUCTION | 540 | TIME TO PRODUCTION | 330 |
CHAPTER 5. DISCUSSION

... features) over time. As part of this thesis in the Azure implementation, a relational database has been used. However, in the next iteration, CosmosDB, a schema-less database, will be used [58]. The equivalent of Azure Functions, in AWS, called AWS Lambda is used for triggering retraining of a model with new training data and updation of the web service with the new model. AWS Lambda allows implementing microservice architectures without the need of managing servers [74].

In order to use the ScienceOps workflow on AWS, first the ModelDeploy service was created. The tool for development of the machine learning model was Spyder [61] since there is no alternative to Azure’s Workbench. It took 90 minutes to create the training script, the scoring file and schema file. This took more time than local development in the Azure workflow because there are no code templates available on the basis of which the scripts can be written. For the Iris (5 KiloBytes) dataset model development, it took 5 minutes. It took 110 minutes to set up all the services from scratch on AWS which includes AWS CodeCommit to provide the version control functionality, AWS CodeBuild to build the code on AWS and Amazon ECS which is the equivalent service of Azure Container Service to deploy the web service. The local development time could be avoided if we use the same docker container (with a few modifications to log processes on AWS), however we wanted to estimate building the solution from scratch.

Development of a user interface where a user can enter data in a text field and select from a drop-down list to score against the model, took 90 minutes. The user interface took more time because on the AWS implementation, there was a provision to retrain a model using AWS Lambda from the user interface itself based on the date filters as well as the dashboard for ModelMonitor and DataMonitor were integrated in the interface as shown in Fig A.5. Fig A.5 is an intermediate user interface and was polished later on with additional plots. The ModelDeploy service was ready in 295 minutes.

The next step was to create the ModelMonitor and DataMonitor service. Testing the API calls to the machine learning web application, creation of a non relational database (DynamoDB), the database connection test and creation of a processing script (no Spark alternative used in this experiment intentionally) took 25 minutes. Setting up a connection of the dashboard with the database for the data to be plotted took 10 minutes. The ModelMonitor and DataMonitor service was ready in 35 minutes.

In total, the entire setup starting from machine learning model to a functioning machine learning service took 5.5 hours. This is more than the time taken on Azure because in this experiment we had a custom dashboard built, retraining using AWS Lambda and scripts written from scratch. Service Level
Agreements (SLAs) are taken care by the cloud platform service provider (in this case AWS). A similar setup on other cloud providers such as Google Cloud Platform (GCP) will take approximately the same time; though the entire workflow has not been tested on GCP at the moment, but through estimation of using the equivalent services separately, the time to production is very similar. Through the use of containerization, a level of portability is enabled and can be deployed on any compute cluster. Challenges listed in Section 1.1 are addressed by providing a service to reduce the time taken to productize a machine learning solution, enabling portability through the use of Docker, providing model management and version control and enabling services to keep track of machine learning model performance and data statistics.

5.2 Future Work

In the future, it is worth exploring the reduction of overheads by switching from Azure Container Service (ACS) to the latest (as of 2018) Azure Kubernetes Service (AKS). AKS[64] makes it easier to manage and operate the Kubernetes environment while maintaining portability. The advantages are automatic upgrades, self-healing, simple user experience and easy scaling. The idea is to get the benefit of open source Kubernetes without the complexity and operational overhead [47]. Furthermore, AKS manages the head-node of the compute cluster on which the containers are deployed automatically, unlike ACS.

One of the research challenges is to understand when the machine learning model should be trained. This means, for example, what characteristics should the machine learning model statistics exhibit - difference in variance, mean, medians in the input and output data; what thresholds of performance - if the performance of model goes below some threshold; evaluating the possibility of performance dip of the model being temporary because of anomalous data; how much and what type of data to be taken for retraining. In the future, different options as listed above could be integrated to at least have a semi-automated machine learning retraining solution. Semi-automated, in this context means with some intervention by a human to evaluate the new data to be trained and tuning of parameters.
Chapter 6

Conclusion

In this thesis, we reasoned out the necessity for a solution like *ScienceOps* in the form of a problem statement, how it works and how it solves the problem. The thesis describes the challenges of machine learning model management, portability, the advantages and disadvantages of cloud services, the need for quick productization of machine learning models and auto-retraining of machine learning models over time. Next, the background study is conducted which includes studies on the challenges and solutions to enterprise cloud-based software services, challenges in lifecycle management in the machine learning context, the challenges in service delivery models, importance of managing machine learning models in enterprise workflows, description of related services to that of *ScienceOps* and a description of the *Azure* technological components used in *ScienceOps*. The architectural workflow consists of three services. *ModelDeploy* service is responsible for training the machine learning model and operationalization of the model through the use of containers by creating a web service. *ModelMonitor* and *DataMonitor* service involves processing and logging of the model/data statistics periodically over time, which can then be visualized. A web interface allows a client/user to score new data against the deployed model through an *API* call.

The different *Azure* services used and the reason for using them was described; which included *Machine Learning Workbench*, *Model Management*, *Container Registry*, *Container Service*, *Blob Storage*, *Databricks*, *SQL Database*, *Azure Functions* and *PowerBI*; followed by *Docker* and *Kubernetes*. The technique for building a machine learning model from *Machine Learning Workbench* and deploying as a web service was described. Once a web service had been deployed, the input/output was stored on *Blob*, processed through *Azure Databricks* and visualized through *PowerBI*. The time to production on *Azure/AWS* versus a rudimentary method was compared. A brief description of the rudimentary workflow and *ScienceOps* workflow on
Azure/AWS has been described. It was concluded that the time to production is faster using the ScienceOps architecture. Statistics visualized from DataMonitor and ModelMonitor are subject to the interpretation of the use case and the decision-maker, thus cannot be uniformly evaluated.
Bibliography


BIBLIOGRAPHY


on Human-In-the-Loop Data Analytics (New York, NY, USA, 2016), HILDA ’16, ACM, pp. 14:1–14:3.


Appendix A

Code Implementations

```python
# import all packages <>
from azureml.dataprep import package

# Use the Azure Machine Learning data collector to log various metrics
from azureml.logging import get_azureml_logger
logger = get_azureml_logger()

# arg0 : dprep path (assume name is iris.dprep)
# arg1: zero-based index of which data flow in the package to execute - if specified data flow references other data flows/sources, they are executed as well.
# arg2: return spark dataframe or pandas dataframe
pkg = package.run('iris.dprep', dataflow_idx=0, spark=False)

X = pkg[['feature_1', 'feature_2']].values
Y = pkg['label'].values

# Add Machine Learning Model Code

# Pickle The Model
```

Code Listing A.1: Training & Pickling
APPENDIX A. CODE IMPLEMENTATIONS

```python
# Put try-catch(ImportError) since ModelDataCollector is only supported in Docker Mode
from azureml.datacollector import ModelDataCollector
from azureml.api.schema.dataTypes import DataTypes
from azureml.api.schema.sampleDefinition import SampleDefinition
from azureml.api.realtime.services import generate_schema
from azureml.assets import get_local_path

def init():
    global inputs_dc, prediction_dc, model
    model = joblib.load('model.pkl') # sklearn
    inputs_dc = ModelDataCollector("model.pkl", identifier="inputs")
    prediction_dc = ModelDataCollector("model.pkl", identifier="prediction")

def run(input_df):
    import json
    inputs_dc.collect(input_df)
    pred = model.predict(input_df)
    prediction_dc.collect(pred)
    return json.dumps(str(pred))

def main():
    from azureml.api.schema.dataTypes import DataTypes
    from azureml.api.schema.sampleDefinition import SampleDefinition
    from azureml.api.realtime.services import generate_schema
    df = pandas.DataFrame(data=[[feature_value_1, feature_value_2]], columns=['feature_1', 'feature_2'])
    os.environ['AML_MODEL_DC_DEBUG'] = 'true' # Debug mode to view output in stdout
    inputs = {"input_df": SampleDefinition(DataTypes.PANDAS, df)}
    generate_schema(run_func=run, inputs=inputs, filepath='./outputs/service_schema.json') # Generate schema
    if __name__ == "__main__":
        main()
```

Code Listing A.2: Scoring 1& Schema Generation
APPENDIX A. CODE IMPLEMENTATIONS

```bash
az login
az account list --o table #List subscriptions
az account set --s <your-subscription-id> #Set subscription ID
az ml experiment submit --c local \training.py #Local Compute
   Environment
az ml experiment submit --c docker python \training.py #Docker
   Environment

## Running in Remote Docker Container
az ml computetarget attach remotedocker --name <compute target>
   --address <IP> --username <username> --password <password>

## Running in Remote Docker Container
az ml experiment prepare --c <compute target> #Change aml_config
   /<compute target>.runconfig from 'Pyspark' --> 'Python'

# If Connection Problems (Tested Azure Ubuntu Server 2016), Try:
sudo apt-get install openssh-server
sudo ufw enable
sudo ufw allow ssh
sudo ufw reload
sudo iptables --I INPUT --p tcp --dport 22 -j ACCEPT
sudo iptables --I OUTPUT --p tcp --dport 22 -j ACCEPT
sudo iptables-save

### Changing aml_config files ===
# local.runconfig / docker-python.runconfig ===
#UseSampling: true
#PrepareEnvironment: true

### conda_dependencies.yml ===
# scikit-learn (under dependencies)
# azureml-model-management-sdk (under pip)
```

Code Listing A.3: Training & Pickling Using Azure CLI
APPENDIX A. CODE IMPLEMENTATIONS

Code Listing A.4: Running in Cluster

```bash
###Running in Cluster
az ml env setup --cluster -n <new deployment environment name> 
                   --location westeurope --g <existing resource group name>
az ml env show -n <deployment environment name> --g <existing 
                   resource group name> #Check Status till Succeeded
az ml account modelmanagement create --location <e.g. eastus2> -- 
         n <new model management account name> --g <existing resource 
         group name> --sku-name S1
az ml account modelmanagement set -n <youracctname> --g < 
                  yourresourcegroupname>
az ml env set -n <deployment environment name> --g <existing 
                  resource group name>
az ml env show
az ml service create realtime -f score_iris.py --model-file 
               model.pkl --s service_schema.json -n irisapp --r python -- 
               collect-model-data true --c aml_config\conda_dependencies.yml
az ml service list realtime -o table #Check Deployed Service
az ml service usage realtime -i <web service id>
az ml service keys realtime -i <web service id>
```
APPENDIX A. CODE IMPLEMENTATIONS

Code Listing A.5: Manage Storage

```bash
### ACCESS STORAGE FROM OUTSIDE
az storage account keys list \
  --account-name mystorageaccount \
  --resource-group myResourceGroup \
  --output table ###Get Key

export AZURE_STORAGE_ACCOUNT="mystorageaccountname"
export AZURE_STORAGE_ACCESS_KEY="myStorageAccountKey"
```

Code Listing A.5: Manage Storage
```python
import requests
import json
data = "{
  "input_df": {
    "feature1": value1,
    "feature2": value2
  }
}"
body = str.encode(json.dumps(data))
url = 'http://<service_ip_address>:80/api/v1/service/<service_name>/score'
api_key = 'your_service_key'
headers = {'Content-Type': 'application/json', 'Authorization': ('Bearer' + api_key)}

resp = requests.post(url, data, headers=headers)
resp.text # Use print to display the result here

 Code Listing A.6: Run Web Service
```
from azure.storage.blob import BlockBlobService, PublicAccess

block_blob_service = BlockBlobService(account_name=<>,
                                       account_key=<>)
full_path_to_file = os.path.join(local_path, local_file_name)  
# local_path example : C:\\Users

# Create Blob
block_blob_service.create_blob_from_path(container_name,
                                          local_file_name, os.path.join(local_path, local_file_name))

# List of all Blobs
generator = block_blob_service.list_blobs(container_name)

# Downloading Blob
block_blob_service.get_blob_to_path(container_name, 'Blob URL to CSV', full_path_to_file2)

Code Listing A.7: Access Blob Storage
```python
import pyspark
from scipy import stats
import pandas as pd
from pyspark.sql.types import DoubleType

"""
Add to Azure Databricks Spark Config :
spark.hadoop.fs.azure.account.key.$<name$>.core.windows.net $<key$>$
"""
path_train = "wasb://modeldata@abc.blob.core.windows.net/..."
path_test = "wasb://modeldata@abc.blob.core.windows.net/..."
account_key = #Account Key

spark = (pyspark.sql.SparkSession.builder
    .master("local")
    .appName("X")
    .config("fs.azure.account.key.abc.blob.core.windows.net",
            account_key).getOrCreate())

data_train=spark.read.csv(path_train) #Read Training Data
data_test=spark.read.csv(path_test)

for col_name in data_train.columns: #Same for test
data_train = data_train.withColumn(col_name, data_train[col_name].cast(DoubleType()))

def function_name(args): #Do Computation
    return var

for col_name in data_train.columns:
    result.append(function_name(data_train.select(col_name)...))

jdbcUrl = "jdbc:sqlserver://<xxx>.database.windows.net:1433;\
<database>=<yyy>;user=<name>@<zzz>;
<password>=<xyz>; encrypt=true;\ntrustServerCertificate=false;\nhostnameInCertificate=*.database.windows.net;\nloginTimeout=30;"

#SQL DB READ
read_query = "(select * from <dbo>.<table_name>)<alias>"
df = spark.read.jdbc(url=jdbcUrl, table=read_query) #then use display(df)

#SQL DB WRITE
df = spark.createDataFrame(pd.DataFrame(result).transpose())
df.write.mode("append").jdbc(url=jdbcUrl, table="table_name")
```

Code Listing A.8: Configure Spark on Azure Databricks
### Model "model.pkl"

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>57</td>
</tr>
<tr>
<td>Id</td>
<td>8c875a7c14da49e3bfd66cb</td>
</tr>
<tr>
<td>Date created</td>
<td>5/7/2018, 2:16:03 PM</td>
</tr>
<tr>
<td>Location</td>
<td><a href="http://mlcrpstorage.blob.core.wir">http://mlcrpstorage.blob.core.wir</a></td>
</tr>
</tbody>
</table>

![Figure A.1: Model Registered on Azure](image)

### Manifest "mayankmanifest"

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>7</td>
</tr>
<tr>
<td>Id</td>
<td>d1d8f767-b671-4633-dbcd</td>
</tr>
<tr>
<td>Date created</td>
<td>5/7/2018, 2:16:53 PM</td>
</tr>
<tr>
<td>Runtime type</td>
<td>Python</td>
</tr>
<tr>
<td>Other assets</td>
<td>driver</td>
</tr>
<tr>
<td></td>
<td>ee0d9704-9d76-4ff1-816e</td>
</tr>
<tr>
<td></td>
<td>8637717b-90f0-4ab3-9e3f</td>
</tr>
</tbody>
</table>

![Figure A.2: Manifest Created on Azure Based on Fig. A.1](image)
### Image “irisimage”

<table>
<thead>
<tr>
<th>Description</th>
<th>6724a1ee-5a71-4123-8fcd-22a04e508f7d</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>6724a1ee-5a71-4123-8fcd-22a04e508f7d</td>
</tr>
<tr>
<td>Date created</td>
<td>5/7/2018, 2:17:43 PM</td>
</tr>
<tr>
<td>Version</td>
<td>9</td>
</tr>
<tr>
<td>Location</td>
<td>mcrpcac5-9abe-4d9c-9ee6-ea97a6c740cc azurecr.io/irisimagi</td>
</tr>
<tr>
<td>Environment</td>
<td>acscluster</td>
</tr>
<tr>
<td>Image type</td>
<td>Docker</td>
</tr>
<tr>
<td>Creation state</td>
<td>Succeeded</td>
</tr>
</tbody>
</table>

**Figure A.3:** Image Created on Azure Based on Fig. A.2

---

### Service “irisapplication”

<table>
<thead>
<tr>
<th>Service id</th>
<th>irisapplication.acscluster-westeurope</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creation date</td>
<td>5/4/2018, 4:40:56 PM</td>
</tr>
<tr>
<td>Last updated</td>
<td>5/7/2018, 2:19:38 PM</td>
</tr>
<tr>
<td>State</td>
<td>Succeeded</td>
</tr>
<tr>
<td>Environment</td>
<td>acscluster</td>
</tr>
<tr>
<td>URL</td>
<td><a href="http://166516/api/v1/service/irisap">http://166516/api/v1/service/irisap</a></td>
</tr>
</tbody>
</table>

**Figure A.4:** Web Service Created on Azure Based on Fig. A.3
Figure A.5: User Interface for AWS Implementation A.5