Machine Learning Lens

AWS Well-Architected Framework

Machine Learning Lens: AWS Well-Architected Framework

Table of Contents

Introduction ... i
  Introduction ... 1
Well-Architected Framework pillars ... 2
Well-Architected machine learning lifecycle ... 3
Well-Architected machine learning design principles ... 5
Well-Architected machine learning ... 6
  Business goal identification lifecycle phase ... 6
    Operational Excellence pillar best practices ... 7
    Security pillar best practices ... 8
    Reliability pillar best practices ... 9
    Performance Efficiency pillar best practices ... 10
    Cost Optimization pillar best practices ... 11
  ML problem framing lifecycle phase ... 12
    Operational Excellence best practices ... 13
    Security pillar best practices ... 18
    Reliability pillar best practices ... 19
    Performance Efficiency pillar best practices ... 21
    Cost Optimization pillar best practices ... 24
  Lifecycle architecture diagram ... 27
  Data processing lifecycle phase ... 30
    Data collection ... 31
    Data preparation ... 32
    Operational Excellence pillar – Best Practices ... 34
    Security pillar – Best Practices ... 36
    Reliability pillar – Best Practices ... 40
    Performance Efficiency pillar – Best Practices ... 42
    Cost Optimization pillar – Best Practices ... 43
  Model development lifecycle phase ... 46
    Model training, tuning ... 47
    Model evaluation ... 49
    Operational Excellence pillar – Best Practices ... 49
    Security pillar – Best Practices ... 51
    Reliability pillar – Best Practices ... 54
    Performance Efficiency pillar – Best Practices ... 57
    Cost Optimization pillar – Best Practices ... 61
  Deployment lifecycle phase ... 68
    Operational Excellence pillar – Best Practices ... 70
    Security pillar – Best Practices ... 72
    Reliability pillar – Best Practices ... 73
    Performance Efficiency pillar – Best Practices ... 75
    Cost Optimization pillar – Best Practices ... 76
  Monitoring lifecycle phase ... 77
    Operational Excellence pillar – Best Practices ... 78
    Security pillar – Best Practices ... 80
    Reliability pillar – Best Practices ... 82
    Performance Efficiency pillar – Best Practices ... 84
    Cost Optimization pillar – Best Practices ... 89
Conclusion ... 91
References ... 92
Document history and contributors ... 93
  Contributors ... 93
Best practices arranged by pillar ... 94
  Operational excellence pillar ... 94
  Security pillar ... 94

- Reliability pillar ... 94
- Performance efficiency pillar ... 95
- Cost optimization pillar ... 95

Notices ... 97

AWS glossary ... 98

Machine Learning Lens

Publication date: **October 12, 2021** (*Document history and contributors* (p. 93))

Machine learning (ML) algorithms discover and learn patterns in data, and construct mathematical models to enable predictions on future data. These solutions can revolutionize lives through better diagnosis of diseases, environment protection, products and services transformation, and more.

This whitepaper provides you with a set of established cloud and technology agnostic best practices. You can apply this guidance and architectural principles when designing your ML workloads, or after your workloads have entered production as part of continuous improvement. The paper includes guidance and resources to help you implement these best practices on AWS.

Introduction

The AWS Well-Architected Framework helps you understand the benefits and risks of decisions you make while building workloads on AWS. By using the Framework, you will learn operational and architectural best practices for designing and operating workloads in the cloud. It provides a way to consistently measure your operations and architectures against best practices and identify areas for improvement.

Your ML models depend on the quality of input data to generate accurate results. As data changes with time, monitoring is required to continuously detect, correct, and mitigate issues with accuracy and performance. This may even require you to retrain your model with the latest refined data.

Application workloads rely on step-by-step instructions to solve a problem. ML workloads enable algorithms to learn from data through an iterative and continuous cycle. The ML lens complements and builds upon the Well-Architected Framework to address this difference between these two types of workloads.

This paper is intended for those in a technology role, such as chief technology officers (CTOs), architects, developers, data scientists, and ML engineers. After reading this paper, you will understand the best practices and strategies to use when you design and operate ML workloads on AWS.

Well-Architected Framework pillars

The AWS Well-Architected Framework provides architectural best practices for designing and operating workloads in the cloud. The Framework consists of five pillars.

- **Operational Excellence** — Includes the ability to run, monitor, and gain insights into workloads. It enables delivering business value and improves supporting processes and procedures. Best practice focus areas include: **organization, prepare, operate,** and **evolve**.
- **Security** — Includes the ability to protect information, systems, and assets. It enables delivering business value through risk assessments and mitigation strategies. Best practice focus areas include: **identify and access management, detection, infrastructure protection, data protection,** and **incident response**.
- **Reliability** — Includes the ability of a workload to recover from infrastructure or service disruptions. Ensures a workload performs its intended function correctly and consistently when it's expected to. It enables dynamically acquiring computing resources to meet demand, and mitigating disruptions such as misconfigurations and transient network issues. Best practice focus areas include: **foundations, workload architecture, change management,** and **failure management**.
- **Performance Efficiency** — Focuses on the efficient use of computing resources to meet requirements. It enables maintaining efficiency as demand changes and technologies evolve. Best practice focus areas include: **architecture selection, review, monitoring,** and **trade-offs.**
- **Cost Optimization** — Includes the continual process of refinement and improvement of a system over its entire lifecycle. It enables building and operating cost-aware systems that minimize costs, maximize return on investment, and achieve business outcomes. Best practice focus areas include: **practice cloud financial management, expenditure and usage awareness, cost-effective resources, manage demand and supplying resources,** and **optimize over time.**

While this paper focuses on the details specific to ML workloads, you can refer to the AWS Well-Architected Framework whitepaper for more information on the Framework and its pillars.

Well-Architected machine learning lifecycle

The ML lifecycle is the cyclic iterative process with instructions, and best practices to use across defined phases while developing an ML workload. The ML lifecycle adds clarity and structure for making a machine learning project successful. The end-to-end machine learning lifecycle process illustrated in Figure 1 includes the following phases:

- Business goal identification
- ML problem framing
- Data processing (data collection, data preprocessing, feature engineering)
- Model development (training, tuning, evaluation)
- Model deployment (inference, prediction)
- Model monitoring

The phases of the ML lifecycle are not necessarily sequential in nature and can have feedback loops, a few of which are illustrated in Figure 1, to interrupt the cycle across the lifecycle phases.

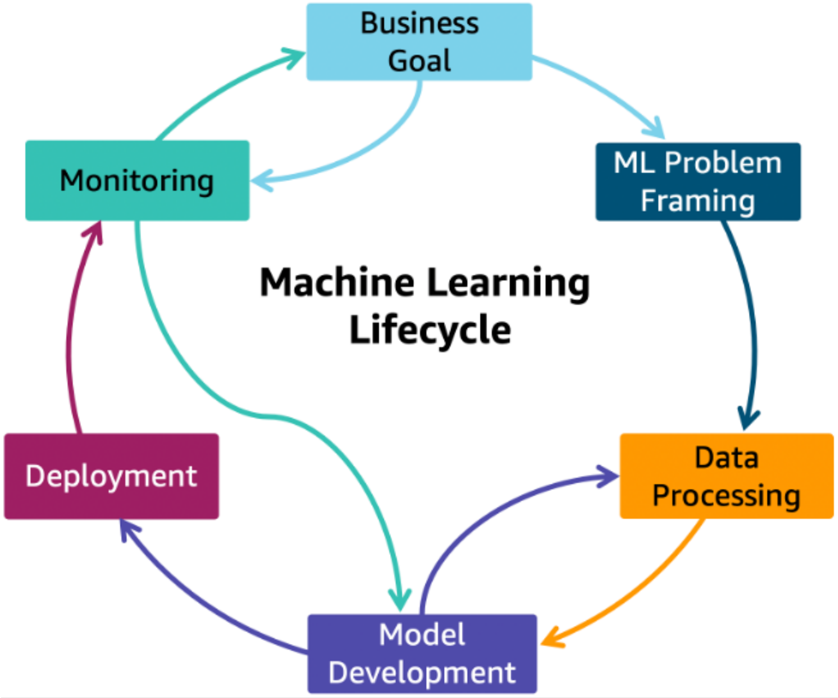

Figure 1: ML lifecycle

The following is a quick introduction to each phase, which will be expanded upon later in this paper.

Business goal

An organization considering ML should have a clear idea of the problem, and the business value to be gained by solving that problem. You must be able to measure business value against specific business objectives and success criteria.

ML problem framing

In this phase, the business problem is framed as a machine learning problem: what is observed and what should be predicted (known as a label or target variable). Determining what to predict and how performance and error metrics must be optimized is a key step in this phase.

Data processing

Training an accurate ML model requires data processing to convert data into a usable format. Data processing steps include collecting data, preparing data, and feature engineering that is the process of creating, transforming, extracting, and selecting variables from data.

Model development

Model development consists of model building, training, tuning, and evaluation. Model building includes creating a CI/CD pipeline that automates the build, train and release to staging and production environments.

Deployment

After a model is trained, tuned, evaluated and validated, you can deploy the model into production. You can then make predictions and inferences against the model.

Monitoring

Model monitoring system ensures your model is maintaining a desired level of performance through early detection and mitigation.

The Well-Architected ML lifecycle, shown in Figure 2, takes the machine learning lifecycle just described, and applies the Well-Architected Framework pillars to each of the lifecycle phases.

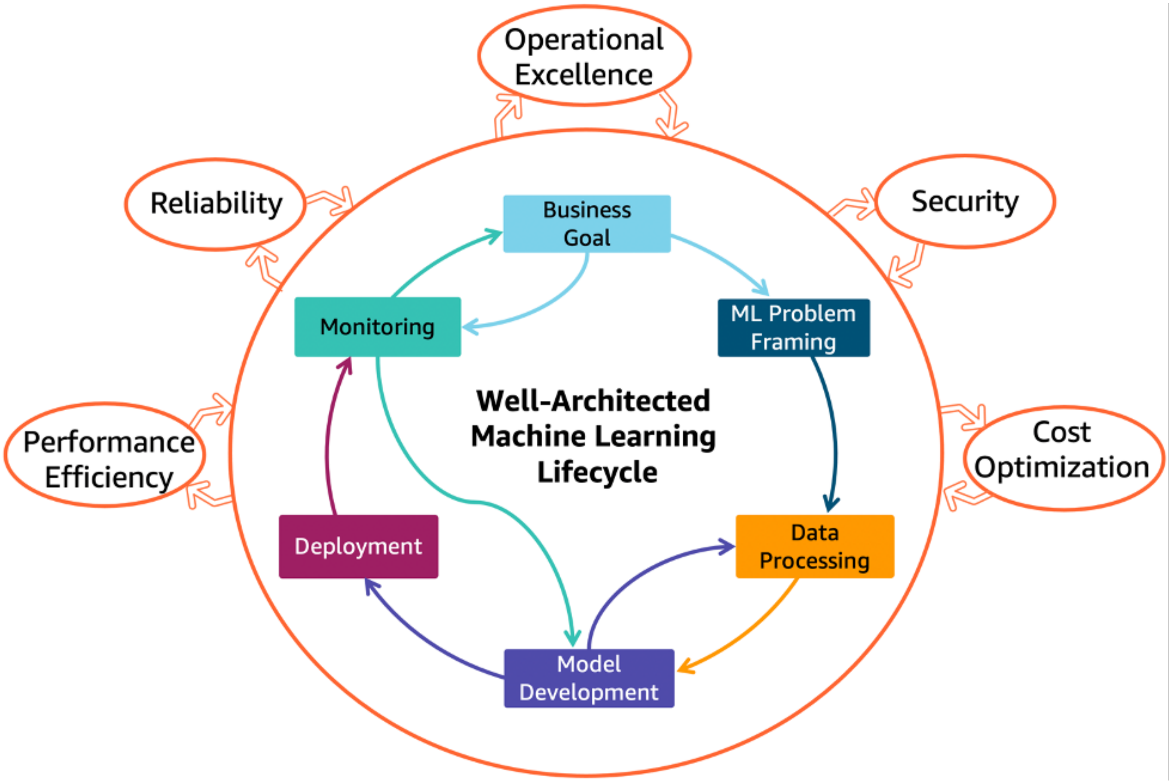

Figure 2: Well-Architected ML lifecycle

Well-Architected machine learning design principles

Well-Architected ML design principles are a set of **considerations** used as the basis for a well-architected ML workload.

Following the Well-Architected Framework guidelines, use these general design principles to facilitate good design in the cloud for ML workloads:

- **Assign ownership** — Apply the right skills and the right number of resources along with accountability and empowerment to increase productivity.
- **Provide protection** — Apply security controls to systems and services hosting model data, algorithms, computation, and endpoints. This ensures secure and uninterrupted operations.
- **Enable resiliency** — Ensure fault tolerance and the recoverability of ML models through version control, traceability, and explainability.
- **Enable reusability** — Use independent modular components that can be shared and reused. This helps enable reliability, improve productivity, and optimize cost.
- **Enable reproducibility** — Use version control across components, such as infrastructure, data, models, and code. Track changes back to a point-in-time release. This approach enables model governance and audit standards.
- **Optimize resources** — Perform trade-off analysis across available resources and configurations to achieve optimal outcome.
- **Reduce cost** — Identify the potentials for reducing cost through automation or optimization, analyzing processes, resources, and operations.
- **Enable automation** — Use technologies, such as pipelining, scripting, and continuous integration (CI), continuous delivery (CD), and continuous training (CT), to increase agility, improve performance, sustain resiliency, and reduce cost.
- **Enable continuous improvement** — Evolve and improve the workload through continuous monitoring, analysis, and learning.

Well-Architected machine learning

This section introduces ML specific Well-Architected best practices. For each of the ML lifecycle phases, Well-Architected best practices are examined across each of the five pillars of operational excellence, security, reliability, performance efficiency, and cost optimization. Best practices for each ML lifecycle phase follow an introductory background on each phase.

The six phases for the ML lifecycle referenced in this paper are illustrated in Figure 3 in a sequence.

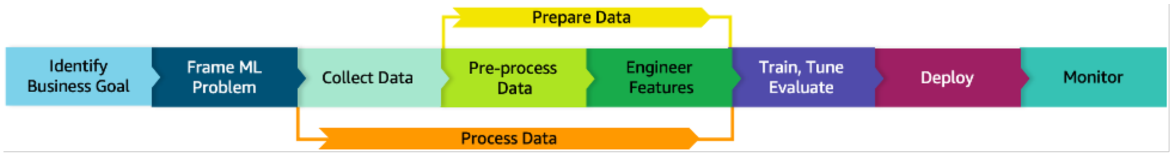

Figure 3: ML Lifecycle phases

The following sections describe the Well-Architected machine learning best practices for each of the lifecycle phases.

> **Note**
> When there is a best practice that applies to multiple pillars or phases, it is described in the pillar or phase where it makes the most impact. A complete list of the ML Lens best practices ordered by pillar instead of by ML lifecycle phase can be found in Best practices arranged by pillar (p. 94).

Topics

- ML lifecycle phase — Business goal (p. 6)
- ML lifecycle phase — ML problem framing (p. 12)
- ML lifecycle architecture diagram (p. 27)
- ML lifecycle phase - Data processing (p. 30)
- ML lifecycle phase – Model development (p. 46)
- ML lifecycle phase - Deployment (p. 68)
- ML lifecycle phase – Monitoring (p. 77)

ML lifecycle phase — Business goal

Business goal identification is the most important phase. An organization considering ML should have a clear idea of the problem to be solved, and the business value to be gained. You must be able to measure business value against specific business objectives and success criteria. While this holds true for any technical solution, this step is particularly challenging when considering ML solutions because ML is a constantly evolving technology.

After you determine your criteria for success, evaluate your organization's ability to move toward that target. The target should be achievable and provide a clear path to production. Involve all relevant stakeholders from the beginning to align them to this target and any new business processes that will result from this initiative.

Start the review by determining if ML is the appropriate approach for delivering your business goal. Evaluate all of the options that you have available for achieving the goal. Determine how accurate the resulting outcomes would be, while considering the cost and scalability of each approach.

For an ML-based approach to be successful, ensure that enough of relevant, high-quality training data is available to the algorithm. Carefully evaluate the data to make sure that the correct data sources are available and accessible.

Steps in this phase:

- Understand business requirements.
- Form a business question.
- Review a project's ML feasibility and data requirements.
- Evaluate the cost of data acquisition, training, inference, and wrong predictions.
- Review proven or published work in similar domains, if available.
- Determine key performance metrics, including acceptable errors.
- Define the machine learning task based on the business question.
- Identify critical, must have features.
- Design small, focused POCs to validate all of the preceding.
- Evaluate if bringing in external data sources will improve model performance.
- Establish pathways to production.
- Consider new business processes that may come out of this implementation.
- Align relevant stakeholders with this initiative.

Operational Excellence pillar – Best Practices

The operational excellence pillar includes the ability to run and monitor systems to deliver business value and to continually improve supporting processes and procedures. This section includes best practices to consider while identifying the business goal.

Best practices

- MLOE-01: Develop right skills with accountability and empowerment (p. 7)

MLOE-01: Develop right skills with accountability and empowerment

Artificial intelligence (AI) has many different and growing branches, such as machine learning, deep learning, and computer vision. Given the complexity and fast-growing nature of ML technologies, plan to hire specialists with the understanding that additional training will be needed as ML evolves. Keep teams upskilled, engaged and motivated while encouraging accountability and empowerment at all times.

Implementation plan

- **Develop skills** — A key element in any organization's strategy for employee engagement and business growth must be ongoing learning and development. Consider strategies to help you grow your business success through intentional workforce skills development including committing to your learning culture, and incorporating peer connection.
- **Develop accountability and empowerment** — Using a legacy approach to project and skill-based teams alone stands in the way of organizations becoming high-performing agile organizations. Agile organizations are able to innovate quickly, bring new ideas to market faster, and deploy advanced technology solutions in less time. Accountability is needed for these high-performing agile teams. Accountable employees manage their workload according to team objectives. They proactively seek help when they need it and take responsibility when they make mistakes. They work in an environment where they are given the authority to do something. It's the direct opposite of micro-management.

Documents

- AWS Learning Needs Analysis

Blogs

- Want to grow your business? Prioritize ongoing skills development
- Two-Pizza Teams Are Just the Start, Part 1: Accountability and Empowerment Are Key to High-Performing Agile Organizations
- Two-Pizza Teams Are Just the Start, Part 2: Accountability and Empowerment Are Key to High-Performing Agile Organizations

Security pillar – Best Practices

The security pillar encompasses the ability to protect data, systems, and assets to take advantage of cloud technologies to improve your security. This section includes best practices to consider while identifying the business goal.

Best practices

- MLSEC-01: Validate ML software privacy and license terms (p. 8)

MLSEC-01: Validate ML software privacy and license terms

ML libraries and packages handle data processing, model development, training, and hosting. Establish a process to review the privacy and license agreements for all software and ML libraries needed throughout the ML lifecycle. Ensure these agreements comply with your organization's legal, privacy, and security terms and conditions. These terms should not add any limitations on your organization's business plans.

Implementation plan

- **Implement a package mirror for consuming approved packages** — Evaluate the license terms to determine which ML packages are appropriate for your business across the phases of the ML lifecycle. Examples of ML Python packages include: Pandas, PyTorch, Keras, NumPy, Scikit-learn. Once you've determined the set and criteria, build a validation mechanism and automate it where possible. A sample automated mechanism can include a script that runs the download, installation, and package version and dependency checks. Download packages from the internet, only from approved and private repos. Validate what is in the packages downloaded. This will enable importing safely and confirming the validity of packages. Amazon SageMaker notebook instances come with multiple environments already installed. These environments contain Jupyter kernels and Python packages. You can also install your own environments that contain your choice of packages and kernels. SageMaker enables modifying package channel paths to a private repository. Where appropriate, use an internal repository as a proxy for public repositories to minimize the network and time overhead.
- **Bootstrap instances with lifecycle management policies** — Create a lifecycle configuration with a reference to your package repository, and a script to install required packages.
- **Evaluate package integrations that require external lookup services** — Based on your data privacy requirements, opt out of data collection when necessary. Minimize data exposure through trusted relationships. Evaluate the privacy policies as well as the license terms for ML packages that might collect data.
- **Use prebuilt containers** — Start with pre-packaged and verified containers to quickly provide support for commonly used dependencies. For example, AWS Deep Learning Containers contain several deep learning framework libraries and tools including TensorFlow, PyTorch, and Apache MXNet.

Documents

- Amazon Well-Architected Security Pillar for Software Integrity
- AWS Deep Learning Containers
- Installing External Libraries and Kernels on Notebook Instances
- Amazon Well-Architected Security Pillar for Software Integrity
- AWS Deep Learning Containers
- Installing External Libraries and Kernels on Notebook Instances

Blogs

- Private package installation in Amazon SageMaker running in internet-free mode
- Create a hosting VPC for PyPi package mirroring and consumption of approved packages
- Machine Learning Best Practices in Financial Services
- Taming Machine Learning on AWS with MLOps: A Reference Architecture

Videos

- Machine Learning Best Practices in Financial Services

Reliability pillar – Best Practices

The reliability pillar encompasses the ability of a workload to perform its intended function correctly and consistently when it’s expected to. This section includes best practices to consider while identifying the business goal.

Best practices

- MLREL-01: Discuss and agree on the level of model explainability (p. 9)

MLREL-01: Discuss and agree on the level of model explainability

Discuss and agree with the business stakeholders on the acceptable level of model explainability. Use the agreed level as a metric for evaluations and tradeoff analysis across the lifecycle. Explainability can help with understanding the cause of a prediction, auditing and meeting regulatory requirements. It can be useful for building trust ensuring that the model is working as expected.

Implementation plan

- **Understand business requirements** — The adoption of AI systems in regulated domains requires trust. This can be built by providing reliable explanations on the deployed predictions. Model explainability can be particularly important to reliability, safety, and compliance requirements.
- **Agree on an acceptable level of explainability** — Communicate with stakeholders across the project about the level of explainability that is required for the project. Agree to a level that helps you meet business requirements.

Documents

- What Is Fairness and Model Explainability for Machine Learning Predictions?

- Model Explainability

Performance Efficiency pillar – Best Practices

The performance efficiency pillar focuses on the efficient use of computing resources to meet requirements and the maintenance of that efficiency as demand changes and technologies evolve. This section includes best practices to consider while identifying the business goal.

Best practices

- MLPER-01: Determine key performance indicators, including acceptable errors (p. 10)

MLPER-01: Determine key performance indicators, including acceptable errors

Use guidance from business stakeholders to capture key performance indicators (KPIs) relevant to the business use case. The KPIs should be directly linked to business value to guide acceptable model performance. Consider that machine learning inferences are probabilistic and will not provide exact results. Identify a minimum acceptable accuracy and maximum acceptable error in the KPIs. This will enable achieving the required business value and manage the risk of variable results.

Implementation plan

- **Quantify the value of machine learning for the business** — Consider measures of how machine learning and automation will impact the business:
  - How much will machine learning reduce costs?
  - How many more users will be reached by increasing scale?
  - How much more quickly will the business be able to respond to changes such as demand changes or supply disruptions?
  - How many hours of manual effort will be reduced by automating with machine learning?
  - How much will machine learning be able to change user behavior such as reducing churn?
- **Evaluate risks and the tolerance for error** — Quantify the impact of machine learning on the business. Rank order the value of impacts to identify the primary KPIs to optimize with machine learning. Define the cost of error for automated inferences that will be performed by ML models in the use case. Determine the tolerance of the business for error. For example, determine how far off a cost reduction estimate would have to be to negatively impact the business goals. Finally, evaluate risks of machine learning for the business, and whether the benefits of ML solutions are of high enough value to outweigh the risks.

Documents

- Solving Business Problems with Amazon Machine Learning
- Improve Business Outcomes with Machine Learning

Blogs

- Building the Business Case for Machine Learning in the Real World

Videos

- Exploring the Business Use Cases for Amazon Machine Learning

Cost Optimization pillar – Best Practices

The cost optimization pillar includes the continual process of refinement and improvement of a system over its entire lifecycle. From the initial design of your very first proof of concept to the ongoing operation of production workloads, adopting the practices in this document can enable you to build and operate cost-aware systems that achieve business outcomes and minimize costs, thus allowing your business to maximize its return on investment. This section includes best practices to consider while identifying the business goal.

Best practices

- MLCOST-01: Define overall return on investment (ROI) and opportunity cost (p. 11)
- MLCOST-02: Use managed services to reduce total cost of ownership (TCO) (p. 12)

MLCOST-01: Define overall return on investment (ROI) and opportunity cost

Evaluate the opportunity cost of ML for each use case to solve the business problem. Ensure cost effective decisions with respect to long-term resource allocation. Minimize the possible future risks and failures through upfront understanding of the ML development process and its resource requirements. Adopt automation and optimization that can result in reduced cost and improved performance.

Implementation plan

- **Specify the objectives of the ML project** as research or development. A research project is intended to discover the value that could be achieved from an untested ML use case, and the returns will be long-term. A development project is intended for specific production improvements, and is expected to deliver a faster return on investment. Both business management and data scientists should agree on whether the project is research-oriented, or development that applies well-understood methods to a well-known use case.
- **Evaluate and assess** the data pipeline, the ML model, and the expected quality of production inferences to estimate the costs of data and errors.
- **Develop a cost-benefit model**, and reassess as changes occur throughout the project. For example, changes in the external business environment, or addition of expensive data sources, can modify the initial cost/benefit model.
- **Understand, evaluate, and monitor** project risks.
- **Estimate the cost of resources needed** (such as data engineers and data scientists) to maintain a production model.

Documents

- Pricing for Amazon ML

Blogs

- Building the Business Case for Machine Learning in the Real World
- Lowering total cost of ownership for machine learning and increasing productivity with Amazon SageMaker

Videos

- APIs: ROI from artificial intelligence

MLCOST-02: Use managed services to reduce total cost of ownership (TCO)

Evaluate adopting managed services and pay per usage. Using of managed services enables organizations to operate more efficiently with reduced resources and reduced cost.

Implementation plan

- **Use Amazon managed ML services** — Use Amazon SageMaker as a fully managed machine learning service to build, train, and deploy models at scale and at significantly lower costs. The total cost of ownership (TCO) of SageMaker over a three-year period is much lower than other self-managed cloud-based ML options, such as Amazon Elastic Compute Cloud (Amazon EC2) and Amazon Elastic Kubernetes Service (Amazon EKS). Some of the SageMaker technologies include Autopilot, Feature Store, Clarify, Data Wrangler, Debugger, Studio, Training, Monitoring, and Pipelines.

Documents

- Amazon SageMaker Total Cost of Ownership (TCO)

Blogs

- Lowering total cost of ownership for machine learning and increasing productivity with Amazon SageMaker
- Decrease Your Machine Learning Costs with Instance Price Reductions and Savings Plans for Amazon SageMaker
- Amazon SageMaker Continues to Lead the Way in Machine Learning and Announces up to 18% Lower Prices on GPU Instances

Videos

- Optimizing your Machine Learning costs on Amazon SageMaker with Savings Plans (April 2021)

ML lifecycle phase — ML problem framing

In this phase, the business problem is framed as a machine learning problem: what is observed and what should be predicted (known as a label or target variable). Determining what to predict and how performance must be optimized is a key step in ML.

Imagine a scenario where a manufacturing company wants to identify products maximizing profits. Reaching this business goal partially depends on determining the products and the right number to produce. In this scenario, you want to predict the future sales of the product based on past sales. Predicting future sales becomes the problem to solve using ML.

Steps in this phase:

- Define criteria for a successful outcome of the project.
- Establish an observable and quantifiable performance metric for the project, such as accuracy.
- Help ensure business stakeholders understand and agree with the defined performance metrics.
- Formulate the ML question in terms of inputs, desired outputs, and the performance metric to be optimized.

- Evaluate whether ML is the right approach. Some business problems don't need ML, simple business rules can do a much better job. For other business problems, there might not be sufficient data to apply ML as a solution.
- Create a strategy to achieve the data sourcing and data annotation objective.
- Start with a simple model that is easy to interpret, and which makes debugging more manageable.

Operational Excellence pillar – Best Practices

The operational excellence pillar includes the ability to run and monitor systems to deliver business value and to continually improve supporting processes and procedures. This section includes best practices to consider while framing the ML problem.

Best practices

- MLOE-02: Establish ML roles and responsibilities (p. 13)
- MLOE-03: Prepare an ML profile template (p. 14)
- MLOE-04: Establish model improvement strategies (p. 15)
- MLOE-05: Establish a lineage tracker system (p. 16)
- MLOE-06: Establish feedback loops across ML lifecycle phases (p. 17)

MLOE-02: Establish ML roles and responsibilities

Understand the roles, responsibilities, ownership, and required interactions across teams to maximize overall effectiveness. An ML project typically consists of multiple roles, with defined tasks and responsibilities for each. In many cases, the separation of roles and responsibilities is not clear and there is overlap.

Implementation plan

- **Establish cross-functional teams with roles and responsibilities** — Enterprises often struggle getting started with their first enterprise-grade ML platform. This is partly due to ML platform architecture having many components. Complexities around data science, data management, and model and operational governance also contribute to this struggle. Building an enterprise ML platform requires the collaboration of different cross-functional teams. The different personas from the different teams should each have a different role and responsibilities. They all should contribute in the build-out, usage, and operation of an ML platform. Examples of ML technical roles and responsibilities with their definitions include:
  - **Business analyst** — Engages early in the ML lifecycle and translates business requirements into actionable tasks.
  - **Domain expert** — Has valuable functional knowledge and understanding of the environment that the ML problem must be framed in. Helps ML engineers and data scientists with developing and validating assumptions and hypotheses. Engages early in the ML lifecycle and stays in close contact with the engineering owners throughout the evaluation phase.
  - **Data architect** — Defines data management systems and how the data is to be ingested, integrated, consumed.
  - **Data engineer** — Transforms data into a consumable format for machine learning and data science analysis.
  - **AI/ML architect** — Ensures that activities across the ML lifecycle phases are carried out using the right technologies. Designs the architecture for raw data transformation to final model deployment and monitoring.
  - **Data scientist** — Employs machine learning, statistical modeling, and artificial intelligence to derive insights from the data.

- **Data analyst** — Performs descriptive statistics and data visualization to derive insights from the data.
- **ML engineer** — Turns reference implementations of ML models developed by data scientists into production-ready software.
- **MLOps engineer** — Builds and manages automation pipelines to operationalize the ML platform and ML pipelines for fully/partially automated CI/CD pipelines. These pipelines automate building Docker images, model training, and model deployment. MLOps engineers also have a role in overall platform governance such as data / model lineage, as well as infrastructure and model monitoring.
- **IT auditor** — Responsible for analyzing system access activities, identifying anomalies and violations, preparing audit reports for audit findings, and recommending remediations.
- **Model risk manager** — Responsible for ensuring machine learning models meet various external and internal control requirements. These requirements include: model inventory, model explainability, model performance monitoring, and model lifecycle management.
- **Cloud and security engineer** — Responsible for creating, configuring, and managing the cloud accounts, and the resources in the accounts. Works with other security functions, such as identity and access management, to set up the required users, roles, and policies to grant users and services permissions to perform various operations in the cloud accounts. On the governance front, cloud and security engineer implements governance controls such as resource tagging, audit trail, and other preventive and detective controls to meet both internal requirements and external regulations.
- **Software engineer** — Designs, develops, tests, and implements programs and applications.

Documents

- Personas for an ML platform
- AWS MLOps Framework
- Why should you use MLOps?

Blogs

- Architecting Persona-centric Data Platform with on-premises Data Sources

MLOE-03: Prepare an ML profile template

Prepare an ML profile template to capture workload artifacts across ML lifecycle phases. The template will enable evaluating the current maturity status of a workload and plan for improvements accordingly. Artifact examples to capture for deployment phase include: model instance size, model update schedule, and model deployment location. This template should have artifact metrics with thresholds to evaluate and rank the level of maturity. Enable the ML profile template to reflect workload maturity status with snapshots of existing profiles, and alternative target profiles. Provide documentation with rationale for choosing one option over another that meets the business requirements.

Implementation plan

- **Capture ML workload deployment characteristics** — Capture the most impactful deployment characteristics of your ML workload. In this paper, we will highlight the characteristics as a sample profile template on AWS. The collected design and provisioning characteristics will help identify the optimal deployment architecture, including computing and inference instance types and sizes.

  ML workload characteristics are mapped across a spectrum from lower to characteristically higher ranges. Ideally, there should be at least two profile templates generated for each workload characteristic. One ML profile template gives a snapshot of the current workload profile. Another profile template can be instantiated to capture the target or future characteristics of the ML workload.

Documentation should provide the rationale for justifying the characteristic values in the target profile.

Sample design, architecture, and provisioning characteristics include:

- **Model deployment sample characteristics include:**
  - Model size (model.tar.gz) in bytes
  - Number of models deployed per endpoint
  - Instance size (AWS example: r5dn.4x.large)
  - Model/endpoint update schedule (hourly, daily, weekly, monthly, per-event)
  - Model deployment location (on-premises, Amazon EC2, container, serverless, edge)
- **Architectural deployment sample characteristics** for the internal underlying algorithm or neural architecture includes:
  - Inference pipeline architecture (single endpoint, chained endpoints)
  - Neural architecture (single framework (Scikit-learn), multi framework (PyTorch + Scikit-learn + TensorFlow))
  - Containers (SageMaker prebuilt container, bring your own container)
  - Location of the containers and models (on-premises, cloud, hybrid)
- **Traffic pattern deployment sample characteristics include:**
  - Traffic pattern (steady, spiky)
  - Input size (number of bytes)
  - Latency (low, medium, high, batch)
  - Concurrency (single thread, multi-thread)
- **Cold start tolerance characteristics** — Determine and document the tolerance of the various aspects of cold start in milliseconds.
- **Network deployment characteristics** — Check for the applicability of network deployment characteristics including AWS KMS encryption, multi-variant endpoints, network isolation, and third-party Docker repositories.
- **Cost considerations** — Discuss and document the cost considerations for elements, such as Amazon Spot Instances.
- **Determine provisioning matrix** — Critical ML workloads might be vying for resources from cloud providers. For staging and production environments, include a matrix of the expected capacity requirements. This matrix consists of the number of instance types perAWSRegion across training, batch interference, real-time inference, and notebooks.

MLOE-04: Establish model improvement strategies

Plan improvement driver for resources and architecture before ML model development starts. Examples of improvement drivers include: cross-validation, feature engineering, model evaluations, tuning hyperparameters, and ensemble methods.

Implementation plan

- **Use Amazon SageMaker Experiments** — Improvement strategies for ML experimentation follow a sequence from simple to more complex. Begin with minimal data cleaning and the most obvious data. Train a simple classical model using algorithms, such as linear regression or logistic regression, for classification tasks. Iterate by increasing the data processing and model complexity to improve metrics related to business value. Amazon SageMaker Experiments can help to organize multiple tests to compare different configurations and algorithms. A few sample approaches to experiment with include:

  - **Use effective feature selection** — Work with subject matter experts to gain insight into the most significant features that will be related to the target values. Iteratively add more complex features, and remove less important features to improve model accuracy and robustness.
  - **Use deep learning** — For large volume training data, consider deep learning models to find previously unknown features and improve the model accuracy.
  - **Consider ensemble methods** — Ensemble methods can add further improvements to accuracy by combining the best characteristics of various algorithms. However, there is a trade-off with computational performance and maintenance difficulty that should be considered for each specific business use case.

- **Optimize hyperparameters** — Optimize hyperparameters for each algorithm to obtain the top performance before selection of the most appropriate. Amazon SageMaker Hyperparameter Optimization automates this process of selecting the top performance.

Documents

- Manage Machine Learning with Amazon SageMaker Experiments
- Perform Automatic Model Tuning with SageMaker

Blogs

- Developing a business strategy by combining machine learning with sensitivity analysis
- Amazon SageMaker Experiments – Organize, Track And Compare Your Machine Learning Trainings
- Code-free machine learning: AutoML with AutoGluon, Amazon SageMaker, and AWS Lambda

Videos

- Hyperparameter Tuning with Amazon SageMaker's Automatic Model Tuning

Examples

- Ensemble Predictions From Multiple Models

MLOE-05: Establish a lineage tracker system

Maintain a system that tracks changes for each release. These changes include documentation, environment, model, data, code, and infrastructure. Having this system allows you to go back and quickly reproduce a problem on a prior release, allowing rollbacks and reproducibility.

Implementation plan

- **Identify artifacts needed for tracking** — Tracking all the artifacts used for a production model is an essential requirement for reproducing the model to meet regulatory and control requirements. Data and artifacts lineage tracking includes the list of artifacts needed for tracking.
- **Use SageMaker ML Lineage Tracking** — SageMaker ML Lineage Tracking creates and stores information about the steps of an ML workflow from data preparation to model deployment. With the tracking information, you can reproduce the workflow steps, track model and dataset lineage, and establish model governance and audit standards.
- **Use SageMaker Studio** — Use SageMaker Studio to track the lineage of an Amazon SageMaker ML pipeline.

- **Use SageMaker Model Registry** — Use SageMaker Model Registry to catalog models for production, manage model versions, and associate metadata with a model. Model Registry enables lineage tracking.
- **Use SageMaker Pipelines for model building** — With SageMaker Pipelines you can track the history of your data within the pipeline execution. SageMaker ML Lineage Tracking lets you analyze input data, its source, and the outputs generated.

Documents

- SageMaker ML Lineage Tracking
- Data and artifacts lineage tracking
- SageMaker Model Building Pipelines
- Track the Lineage of a SageMaker ML Pipeline
- SageMaker Studio
- SageMaker Model Registry

Blogs

- Using model attributes to track your training runs on Amazon SageMaker

Examples

- Controlling and auditing data exploration activities with SageMaker Studio and AWS Lake Formation

MLOE-06: Establish feedback loops across ML lifecycle phases

Establish a feedback mechanism to share and communicate successful development experiments, analysis of failures, and operational activities. This facilitates continuous improvement on future iterations of the ML workload. Feedback loops allow experimentation with alternative approaches and algorithms until an optimal outcome is achieved. Document your findings to identify key learnings and improve processes over time.

Implementation plan

- **Establish SageMaker Model Monitoring** — Amazon SageMaker Model Monitor continually monitors machine learning models for concept drift. Drifts include changes in data distribution and characteristics over time. SageMaker Model Monitor alerts you if there are any deviations so that you can take remedial action.
  - **Use Amazon CloudWatch** — Configure Amazon CloudWatch to receive notifications if drift in model quality is observed. Monitoring jobs can be scheduled to run at a regular cadence (for example, hourly or daily) and push reports as well as metrics to Amazon CloudWatch and Amazon S3.
  - **Initiate pipelines** — Create a CloudWatch Events rule that alerts on an event emitted by SageMaker Model Monitoring system. The event rule can detect the drifts or anomalies, and start a retraining pipeline.
- **Use Amazon Augmented AI (A2I)** — Check accuracy by having human reviews using tools such as Amazon A2I.

Documents

- Amazon SageMaker Model Monitor
- Creating a CloudWatch Events Rule That Triggers on an Event

- Monitoring Amazon ML with Amazon CloudWatch Metrics

Blogs

- Automated monitoring of your machine learning models with Amazon SageMaker Model Monitor and sending predictions to human review workflows using Amazon A2I
- Automating model retraining and deployment using the AWS Step Functions Data Science SDK for Amazon SageMaker
- Monitoring in-production ML models at large scale using Amazon SageMaker Model Monitor
- Human-in-the-loop review of model explanations with Amazon SageMaker Clarify and Amazon A2I
- Amazon SageMaker Model Monitor now supports new capabilities to maintain model quality in production

Videos

- Easily Implement Human in the Loop into Your Machine Learning Predictions with Amazon A2I

Security pillar – Best Practices

The security pillar encompasses the ability to protect data, systems, and assets to take advantage of cloud technologies to improve your security. This section includes best practices to consider while framing the ML problem.

Best practices

- MLSEC-02: Ensure least privilege access (p. 18)

MLSEC-02: Ensure least privilege access

Protect all resources across the various phases of the ML lifecycle using the principal of least privilege. These resources include: data, algorithms, code, hyperparameters, trained model artifacts, and infrastructure. Provide dedicated network environments with dedicated resources and services to operate any individual project. This approach will help by achieving least privilege access to sensitive data, assets, and services on a project-by-project basis.

Implementation plan

- **Restrict access based on business roles for individuals** — Identify roles that need to explore data to build models, features, and algorithms. Map those roles to access patterns using role-based authentication. This approach can help you achieve least privilege access to sensitive data, assets, and services on a project-by-project basis.
- **Use account separation and AWS Organizations** — Establish tagging and role-based access grants. Understand the workflows of the different user types. Use AWS Service Catalog to create pre-provisioned environments for quick deployment. Tag data and buckets that contain sensitive workloads. Use those tags when granting access to enable granular access patterns to only those that require access.
- **Break out ML workloads by access pattern and structure organizational units** — Delegate specific access to each group, such as administrators or data analysts, as required. Use guard rails and security control policies (SCPs) to enforce best practices for each environment type. Limit infrastructure access to administrators. Verify all sensitive data is accessed through restricted, dedicated, and isolated environments.

Documents

- Amazon SageMaker with Guardrails on AWS
- AWS Service Catalog
- Build a Secure Enterprise Machine Learning Platform on AWS

Blogs

- Building secure Amazon SageMaker access URLs with AWS Service Catalog
- Setting up secure, well-governed machine learning environments on AWS - for detailed guidance on SCP and OU strategies

Videos

- AWS re:Invent 2020: Architectural best practices for machine learning applications
- AWS re:Invent 2020: Secure and compliant machine learning for regulated industries
- AWS re:Inforce 2019: Amazon SageMaker Model Development in a Highly Regulated Environment (SDD315)

Examples

- Build your own Anomaly Detection ML Pipeline
- AWS MLOps Framework

Reliability pillar – Best Practices

The reliability pillar encompasses the ability of a workload to perform its intended function correctly and consistently when it’s expected to. This section includes best practices to consider while framing the ML problem.

Best practices

- MLREL-02: Use APIs to abstract change from model consuming applications (p. 19)
- MLREL-03: Adopt a machine learning microservice strategy (p. 20)

MLREL-02: Use APIs to abstract change from model consuming applications

Use a flexible application and API design to abstract change from model consuming applications. Ensure that changes to an ML model are introduced with minimal or no interruption to existing workload capabilities. Minimize the changes across other downstream applications.

Implementation plan

- **Adopt best practices in use of APIs** — Expose your ML endpoints through APIs so that changes to the model can be introduced without disrupting upstream communications. Document your API in a central repository or documentation site so that any calling services can easily understand your API routes and flags. Ensure that any changes to your API are communicated with any calling services.
- **Deploy a model in Amazon SageMaker** — After you train your model, you can deploy it using Amazon SageMaker to get predictions. To establish a persistent endpoint to get one prediction at a time, use SageMaker hosting services. To get predictions for an entire dataset, use SageMaker batch transform.

- **Use Amazon API Gateway to create APIs** — Amazon API Gateway is a fully managed service that enables developers to create, publish, maintain, monitor, and secure APIs. Using API Gateway, you can create RESTful APIs and WebSocket APIs that enable real-time two-way communication applications. API Gateway supports containerized and serverless workloads, as well as web applications.

Documents

- Deploy a Model in Amazon SageMaker
- What is Amazon API Gateway?
- API Gateway Pattern

Blogs

- Build a serverless frontend for an Amazon SageMaker endpoint
- Creating a machine learning-powered REST API with Amazon API Gateway mapping templates and Amazon SageMaker
- Deploying machine learning models with serverless templates

Videos

- Deploy Your ML Models to Production at Scale with Amazon SageMaker

Examples

- AWS Solutions Constructs aws-apigateway-sagemakerendpoint
- AWS MLOps Framework
- Amazon SageMaker Safe Deployment Pipeline
- Amazon SageMaker Inference Client Application

MLREL-03: Adopt a machine learning microservice strategy

To ensure changes to an ML model are introduced with minimal or no interruption to existing workload, adopt a microservice instead of a monolithic architecture. This approach replaces one large resource with multiple small resources and can reduce the impact of a single failure on the overall workload. This strategy enables distributed development and improves scalability, enabling easier change management.

Implementation plan

- **Adopt a microservice strategy** — Service-oriented architecture (SOA) is the practice of making software components reusable using service interfaces. Instead of building a monolithic application, where all functionality is contained in a single runtime, the application is instead broken into separate components. Microservices extend this by making components that are single-purpose and reusable. When building your architecture, divide components along business boundaries or logical domains. Adopt a philosophy that favors single-purpose applications that can be composed in different ways to deliver different end-user experiences.
- **Use AWS services in developing microservices** — Two popular approaches for developing microservices are using AWS Lambda and Docker containers with AWS Fargate. With AWS Lambda, you can run code for virtually any type of application or backend service with zero administration. You pay only for the compute time you consume, and there is no charge when your code is not running. A common approach to reduce operational efforts for deployment is using a container-based deployment. AWS Fargate is a container management service that allows you to run serverless

containers so you don't have to worry about provisioning, configuring, and scaling clusters of virtual machines to run containers.

Documents

- Implementing Microservices on AWS
- Microservices on AWS
- Break a Monolith Application into Microservices
- AWS Lambda Documentation
- What is AWS Fargate?

Blogs

- Deploying Python Flask microservices to AWS using open-source tools
- Deploying machine learning models as serverless APIs
- Integrating machine learning models into your Java-based microservices
- Adopting machine learning in your microservices with DJL (Deep Java Library) and Spring Boot
- Building, deploying, and operating containerized applications with AWS Fargate

Videos

- Breaking the Monolith Using AWS Container Services
- AWS New York Summit 2019: Migrating Monolithic Applications with the Strangler Pattern (FSV303)

Examples

- Run a Serverless "Hello, World" with AWS Lambda

Performance Efficiency pillar – Best Practices

The performance efficiency pillar focuses on the efficient use of computing resources to meet requirements and the maintenance of that efficiency as demand changes and technologies evolve. This section includes best practices to consider while framing the ML problem.

Best practices

- MLPER-02: Understand and manage the available services and resources (p. 21)
- MLPER-03: Review fairness and explainability (p. 22)
- MLPER-04: Define relevant evaluation metrics (p. 23)

MLPER-02: Understand and manage the available services and resources

Identify the relevant services and configuration options for the workload. Understand how to achieve optimal performance. Select proper infrastructure that meet the workload requirements.

Implementation plan

- **Learn about AWS AI services** — Determine whether AWS managed AI services are applicable to the business use case. Understand how managed AWS AI services can relieve the burden of training and maintaining an ML pipeline. Use Amazon SageMaker to develop in the cloud and understand the roles and responsibilities needed to maintain the ML workload. Consider combining managed AI services with custom ML models built on Amazon SageMaker. This will enable balancing the tradeoffs between ML workload management, and solutions specificity for the business use case.

Documents

- Overview of Amazon Web Services: Machine Learning
- Machine Learning on AWS
- Architecture Best Practices for Machine Learning

Blogs

- Amazon SageMaker – Accelerating Machine Learning
- Amazon SageMaker Studio: The First Fully Integrated Development Environment For Machine Learning

Videos

- Bring Your Own Custom ML Models with Amazon SageMaker
- An Overview of AI and Machine Learning Services From AWS

Examples

- Example Notebooks: Use Your Own Algorithm or Model
- Amazon SageMaker Examples – end_to_end

MLPER-03: Review fairness and explainability

Consider fairness and explainability during each stage of the ML lifecycle. Compile a list of questions to review for each stage including:

- Problem framing — Is an algorithm an ethical solution to the problem?
- Data management — Is the training data representative of different groups? Are there biases in labels or features? Does the data need to be modified to mitigate bias?
- Training and evaluation — Do fairness constraints need to be included in the objective function? Does changing the number of models to train needed to mitigate bias? Has the model been evaluated using relevant fairness metrics?
- Deployment — Is the model deployed on a population for which it was not trained or evaluated?
- Monitoring — Are there unequal effects across users?

Implementation plan

- **Use Amazon SageMaker Clarify** — Understand model characteristics, debug predictions, and explain how ML models make predictions with Amazon SageMaker Clarify. Amazon SageMaker Clarify uses a model-agnostic feature attribution approach that includes an efficient implementation of SHAP (Shapley Additive Explanations). SageMaker Clarify enables you to:
  - Understand the compliance requirements for fairness and explainability.
  - Determine whether training data is biased in its classes or population segments, particularly protected groups.
  - Develop a strategy for monitoring for bias in data when the model is in production.

- Consider the trade-offs between model complexity and explainability, and select simpler models if explainability is required.

Documents

- What Is Fairness and Model Explainability for Machine Learning Predictions?
- Amazon SageMaker Clarify: Detect bias in ML models and understand model predictions
- Feature Attributions that Use Shapley Values
- Amazon SageMaker Clarify: Machine Learning Bias Detection and Explainability in the Cloud

Blogs

- ML model explainability with Amazon SageMaker Clarify and the SKLearn pre-built container
- Explaining Amazon SageMaker Autopilot models with SHAP

Videos

- Machine learning and society: Bias, fairness, and explainability
- How Clarify helps machine learning developers detect unintended bias
- Interpretability and explainability in machine learning

Examples

- Fairness and Explainability with SageMaker Clarify

MLPER-04: Define relevant evaluation metrics

To validate and monitor model performance, establish numerical metrics that directly relate to the KPIs. These KPIs are established in the business goal identification phase. Model accuracy might be an adequate metric for general model performance. Simple numerical metrics could be misleading in business applications depending on the use case. Evaluate whether the performance metrics accurately reflect the business' tolerance for the error. For instance, false positives might lead to excessive maintenance costs in predictive maintenance use cases. Numerical metrics such as precision and recall would help differentiate the business requirements and be closer aligned to business value. Consider developing custom metrics that tune the model directly for the business objectives. Examples of standard metrics for ML models include:

- ***Classification***
  - Confusion matrix (precision, recall, accuracy, F1 score)
  - Receiver operating characteristic (ROC)-area under curve (AUC)
  - Logarithmic loss (log-loss)
- ***Regression***
  - Root mean square error (RMSE)
  - Mean absolute percentage error (MAPE)

Implementation plan

- **Optimize business-related metrics** — Identify performance metrics relevant to use-case and model type. Implement the metric as a loss function or use the loss function included in Amazon SageMaker. Use Amazon SageMaker Experiments to evaluate the metrics with consideration to the business use case to maximize business value. Track model and concept drift in real time with Amazon SageMaker Model Monitor to estimate errors.

- Calculate the maximum probability of error that will be required for the ML model to produce results considering the tolerance set by the business.
- Select and train ML models on the available data to make prediction within the probability bounds. Organize tests on different models with Amazon SageMaker Experiments.

Documents

- Monitor and Analyze Training Jobs Using Metrics
- Manage Machine Learning with Amazon SageMaker Experiments

Blogs

- Training models with unequal economic error costs using Amazon SageMaker
- Amazon SageMaker Experiments – Organize, Track And Compare Your Machine Learning Trainings

Videos

- Organize, Track, and Evaluate ML Training Runs With Amazon SageMaker Experiments

Examples

- Scikit-Learn Data Processing and Model Evaluation

Cost Optimization pillar – Best Practices

The cost optimization pillar includes the continual process of refinement and improvement of a system over its entire lifecycle. From the initial design of your very first proof of concept to the ongoing operation of production workloads, adopting the practices in this document can enable you to build and operate cost-aware systems that achieve business outcomes and minimize costs, thus allowing your business to maximize its return on investment. This section includes best practices to consider while framing the ML problem.

Best practices

- MLCOST-03: Identify if machine learning is the right solution (p. 24)
- MLCOST-04: Enable data and compute proximity (p. 25)
- MLCOST-05: Select optimal algorithms (p. 25)
- MLCOST-06: Tradeoff analysis on custom versus pre-trained models (p. 26)
- MLCOST-07: Enable debugging and logging (p. 27)

MLCOST-03: Identify if machine learning is the right solution

Evaluate if there are alternatives, such as a simple rule-based approach, that could do a better job than ML. Reasons for not selecting ML as a solution include: slower prediction time, lack of explainability, and insufficient data or features.

Implementation plan

- **Use Amazon SageMaker Autopilot and SageMaker Clarify to validate that ML is the right solution**

- **Baseline the solution** by reviewing how the problem is solved today. If a rules-based solution is available, then use it as a baseline.
- **Build a machine learning model** using SageMaker or SageMaker Autopilot and compare the metrics of this solution against the baseline.
- **Use SageMaker Clarify** to explain the model that you have built using SageMaker or Autopilot.
- **Identify** if the ML model is performing better than your existing solution or a rules-based approach before investing on an ML-based solution.

Documents

- Amazon SageMaker
- Amazon SageMaker Autopilot
- SageMaker Clarify

MLCOST-04: Enable data and compute proximity

Ensure that the Region used for training and developing models is the same as the data. This helps minimize the time and cost of transferring data to the computation environment.

Implementation plan

- **Keep data and compute resources in close proximity** — Amazon Elastic Computing Cloud (EC2) is hosted in multiple locations world-wide. These locations are composed of Regions, Availability Zones, Local Zones, AWS Outposts, and Wavelength Zones. Each *Region* is a separate geographic area. If you are launching a compute cluster, you should launch the cluster in close proximity to your data to get the best performance. Say, your Amazon S3 bucket is in the US West (Oregon) Region, you should launch your cluster in the US West (Oregon) Region to avoid Cross-Region data transfer fees.

Documents

- Regions and Zones

Blogs

- Overview of Data Transfer Costs for Common Architectures

Videos

- AWS re:Invent 2018: Architecture Patterns for Multi-Region Active-Active Applications

MLCOST-05: Select optimal algorithms

Identify the basic machine learning paradigm that addresses your ML problem type. Basic machine learning paradigms include: supervised learning, unsupervised learning, and reinforcement learning. Identify the acceptable level of tradeoff between explainability and accuracy per business requirements. Run prototypes and experiments to explore high performing algorithms. Select the optimal cost-efficient algorithms that meet all the business requirements. Improved runtime performance of a tuned algorithm within business requirements, is one step towards optimizing the cost of ML.

Implementation plan

- **Adopt optimal practices**

  - Start with simple algorithms, such as regression, and work towards more complex algorithms, such as deep learning, to compare the accuracy of the models. Optimize hyperparameters to determine which algorithm yields the best metrics for the business use case.
  - When selecting the optimal algorithm, run trade-off analysis between data constraints, computational performance, and maintenance efforts. For example, deep learning networks might produce more accurate results, but require more data than tree-based methods. Deep learning methods are also more difficult to maintain.
- **Use AWS services**
  - Use Amazon SageMaker with a suite of built-in algorithms to train and deploy machine learning models. AWS provides optimized versions of frameworks, such as TensorFlow, Chainer, Keras, and Theano. These frameworks include optimizations for high-performance training across Amazon EC2 instance families.
  - Use Amazon SageMaker Experiments to keep track of the models during testing.
  - Use Amazon SageMaker Autopilot to select algorithms automatically.

Documents

- Choose an Algorithm
- Manage Machine Learning with Amazon SageMaker Experiments
- Automate model development with Amazon SageMaker Autopilot

Blogs

- Optimizing costs for machine learning with Amazon SageMaker
- Amazon SageMaker Experiments – Organize, Track And Compare Your Machine Learning Trainings
- Amazon SageMaker Autopilot – Automatically Create High-Quality Machine Learning Models With Full Control And Visibility
- Streamline modeling with Amazon SageMaker Studio and the Amazon Experiments SDK

Videos

- Accelerate Machine Learning Projects with Hundreds of Algorithms and Models in AWS Marketplace
- Organize, Track, and Evaluate ML Training Runs With Amazon SageMaker Experiments

MLCOST-06: Tradeoff analysis on custom versus pre-trained models

Optimize the cost through tradeoff analysis based on custom versus pre-trained models. This tradeoff analysis should keep the security and performance efficiency in perspective and within the acceptable thresholds.

Implementation plan

- **Use Amazon SageMaker built in algorithms and AWS Marketplace** — Amazon SageMaker provides a suite of built-in algorithms to help data scientists and machine learning practitioners get started on training and deploying machine learning models. Pre-trained ML models are ready-to-use models that can be quickly deployed on Amazon SageMaker. By pre-training the ML models for you, solutions in the AWS Marketplace take care of the heavy lifting, helping you deliver AI- and ML-powered features faster and at a lower cost. Evaluate the cost of your data scientists' time and other resource requirements to develop your own custom model vs. bringing a pre-trained model and deploying it on SageMaker for inferencing. The advantage of a custom model is the flexibility to fine-tune it to match the needs of your business use case. A pre-trained model can be difficult to modify and you might have to use it as is.

Documents

- Pre-trained machine learning models available in AWS Marketplace
- Use Hugging Face with Amazon SageMaker

Blogs

- Bring your own pre-trained MXNet or TensorFlow models into Amazon SageMaker
- How Startups Deploy Pretrained Models on Amazon SageMaker

MLCOST-07: Enable debugging and logging

Ensure that there are sufficient logs and metrics recorded to capture the runtime and resource consumption. The collected logs and metrics can be analyzed to identify the areas for improvement. Monitor compute and data storage consumption. Instrument the machine learning code, and use debugging tools to capture metrics at runtime.

Implementation plan

- **Use Amazon SageMaker Debugger** — Amazon SageMaker Debugger captures the state of a training job at periodic intervals. It provides visibility into the ML training process by monitoring, recording, and analyzing data with the ability to perform interactive exploration of data captured during training. The debugger has an alerting capability for errors detected during training. For example, it can automatically detect and alert you to commonly occurring errors, such as gradient values getting too large or too small.
- **Use Amazon CloudWatch** — Logs generated during training by Amazon SageMaker are logged to Amazon CloudWatch Logs. Use an AWS KMS key to encrypt log data ingested by Amazon CloudWatch Logs.

Documents

- Logging and Monitoring
- Logging Amazon ML API Calls with AWS CloudTrail
- Amazon SageMaker Debugger

Blogs

- Build Your Own Log Analytics Solution on AWS

ML lifecycle architecture diagram

Figure 4 lists the ML lifecycle phases with data processing phase expanded into data collection and data preparation phases. These phases will be discussed in more detail in this section.

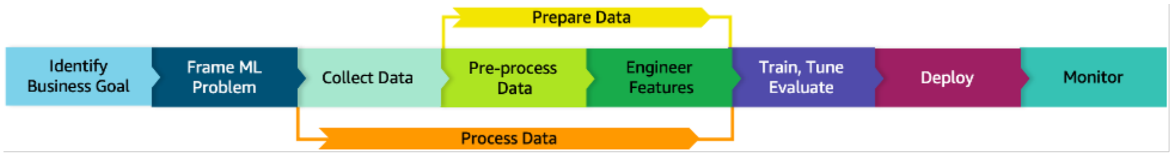

Figure 4: ML lifecycle with data processing sub-phases included

Figure 5 illustrates the ML lifecycle phases beyond problem framing phase in a zoomed-in version. The architecture diagrams in these figures show detail with expanded components that enable best practices that will be discussed in this paper. The data processing phase expands into data collection and data preparation. Data preparation expands into data preprocessing and feature engineering. Model development includes training, tuning, and evaluation. Deploy phase includes the staging environment for model validation for security and robustness. Monitoring is key in timely detection and mitigation of drifts. Feedback loops across the ML lifecycle phases are key enablers for monitoring. Feature stores (online/offline) provide consistent and reusable features across model development and deployment phases. The model registry enables the version control and lineage tracking for model and data components. This figure also emphasizes on the lineage tracking and its components that are discussed in this section in more detail.

The cloud agnostic architecture diagrams in this paper provide high-level best practices with the following assumptions:

- All presented concepts are cloud and technology agnostic.
- Solid black lines are indicative of process flow.
- Dashed color lines are indicative of input and output flow.
- Architecture diagram components are color coded for ease of communication across this document.

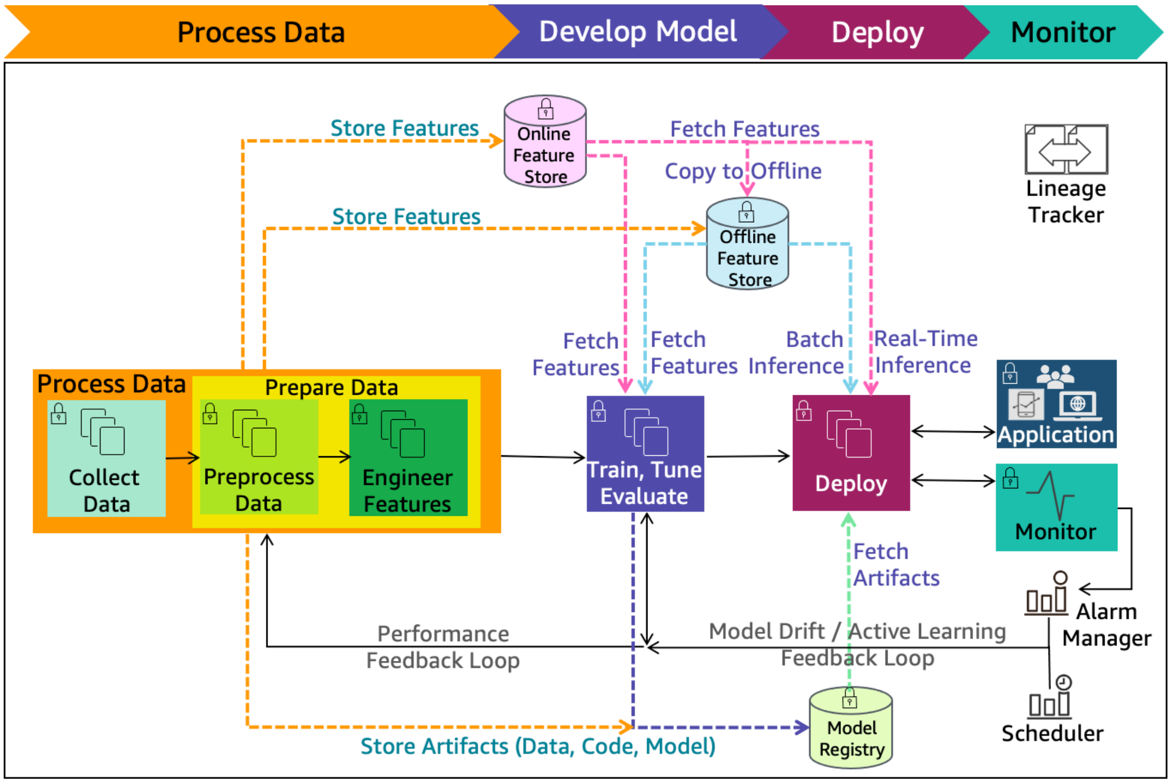

Figure 5: ML lifecycle with detailed phases and expanded components

ML lifecycle as shown in Figure 5 includes the following components:

- **Online/Offline feature store** — Feature store reduces duplication and rerun of feature engineering code across teams and projects. Online store with low-latency retrieval capabilities is ideal for real-time inference. Offline store should maintain a history of feature values and is suited for training and batch scoring.

- **Model registry** — Model registry is a repository for storing ML model artifacts including trained model and related metadata (data, code, model). It enables lineage for ML models as it can act as a version control system.
- **Performance feedback loop** — Automates model performance evaluation tasks initiated from the model development to data processing phase.
- **Model drift feedback loop** — Automates model update re-training tasks initiated from the production deployment to data processing phase.
- **Alarm manager** — Alarm manager receives the alerts from the model monitoring system. It then runs actions by publishing notifications to services that can deliver alerts to target applications to handle them. The model update re-training pipeline is one such target application.
- **Scheduler** — A scheduler can initiate a re-training at business defined intervals.

- **Lineage tracker** — The machine learning lineage tracking enables reproducible machine learning experiences. It enables re-creating the ML environment at a specific point-in-time, reflecting the versions of all resources and environments at that time.

  The ML lineage tracker collects references to traceable data, model and infrastructure resource changes. It consists of the following components:

  - System architecture (infrastructure as code to address environment drift)
  - Data (metadata, values, and features)
  - Model (algorithm, features, parameters, and hyperparameters)
  - Code (implementation, modeling, and pipeline)

  The lineage tracker collects changed references through alternative iterations of ML lifecycle phases. Alternative algorithms and feature lists are evaluated as experiments for final production deployment.

  Figure 6 includes machine learning components and their information that the lineage tracker collects across different releases. The collected information enables going back to a specific point-in-time release and recreate it.

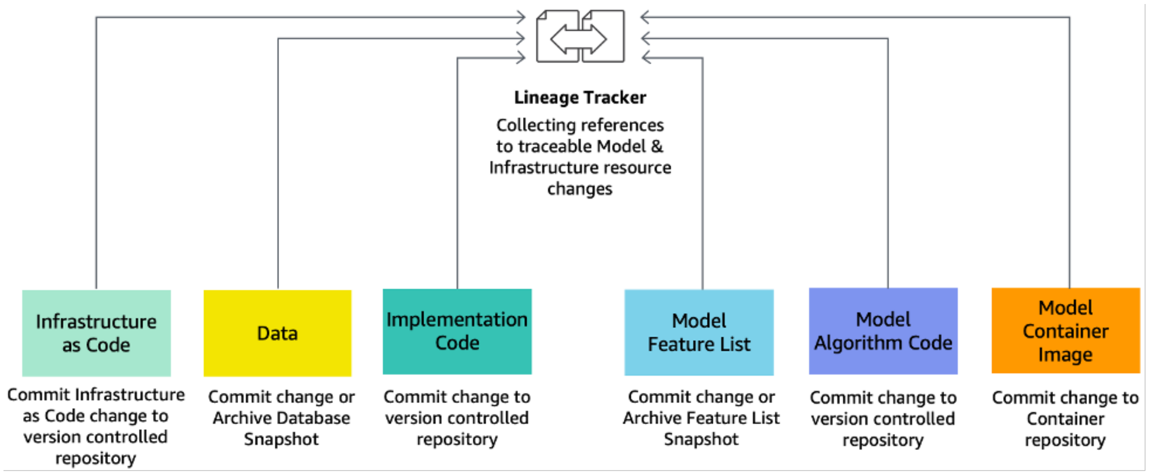
Lineage Tracker
Collecting references to traceable Model & Infrastructure resource changes
Infrastructure as Code
Commit Infrastructure as Code change to version controlled repository
Data
Commit change or Archive Database Snapshot
Implementation Code
Commit change to version controlled repository
Model Feature List
Commit change or Archive Feature List Snapshot
Model Algorithm Code
Commit change to version controlled repository
Model Container Image
Commit change to Container repository

Figure 6: Lineage tracker

Lineage tracker components include:

- **Infrastructure as code (IaC)** —Modeling, provisioning, and managing cloud computing resources (compute, storage, network, and application services) can be automated using infrastructure as code. Cloud computing takes advantage of virtualization to enable the on-demand provisioning of resources. IaC eliminates configuration drift through automation, while increasing the speed and agility of infrastructure deployments. IaC code changes are committed to version-controlled repository.

- **Data** — Store data schemes and metadata in version control systems. Store the data in a storage media like a data lake. The location or link to the data can be in a configuration file and stored in code version control media.
- **Implementation code** — Changes to any implementation code at any point-in-time can be stored in version control media.
- **Model feature list** — Feature store technology referenced in the scenario architecture diagrams stores features as well as their versions for any point-in-time changes.
- **Model algorithm code** — Changes to any model algorithm code at any point-in-time can be stored in version control media.
- **Model container image** — Versions of model container images for any point-in-time changes can be stored in container repositories managed by container registry.

ML lifecycle phase - Data processing

In ML workloads, the data (inputs and corresponding desired output) serves important functions including:

- Defining the goal of the system: the output representation and the relationship of each output to each input, by means of input/output pairs.
- Training the algorithm that associates inputs to outputs.
- Measuring the performance of the trained model, and evaluating whether the performance target was met.
- Building baselines to monitor the performance of the models deployed to production.

As shown in Figure 7, data processing in this paper includes data collection and data preparation. Data preparation includes data preprocessing and feature engineering that will be covered in more detail in this paper. Data wrangling is data preparation during the interactive data analysis and model development. Data Visualization can help with exploratory data analysis (EDA). EDA can help with understanding data, sanity checks, and validating the quality of the data.

The same sequence of data processing steps that you apply to the training data is also applied to the inference requests.

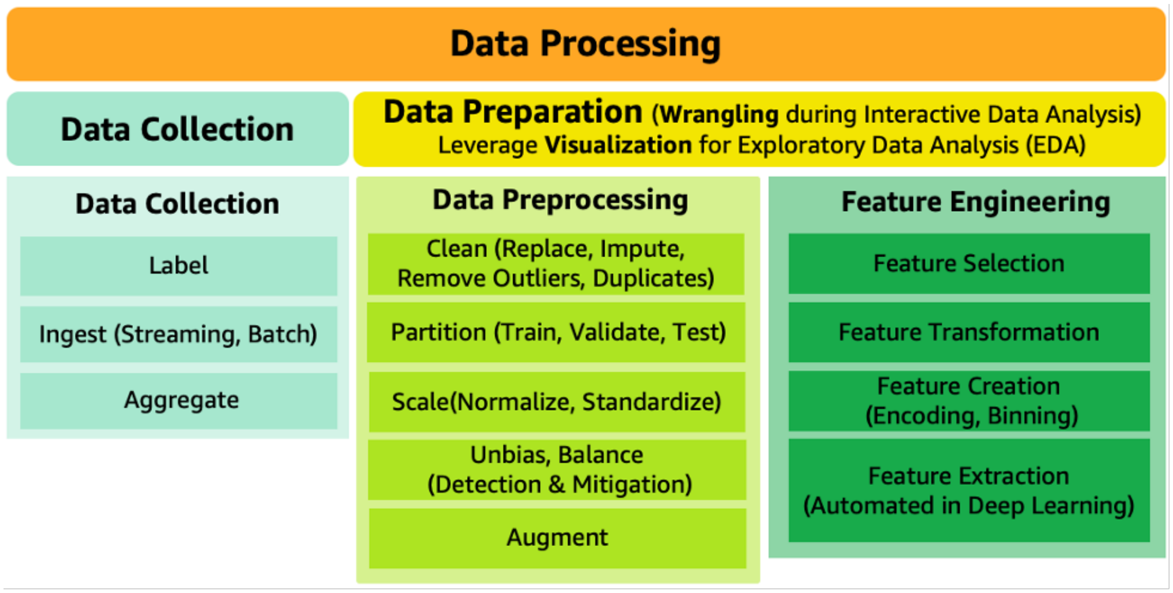
Data Processing
Data Collection
Data Preparation (Wrangling during Interactive Data Analysis)
Leverage Visualization for Exploratory Data Analysis (EDA)
Data Collection
Label
Ingest (Streaming, Batch)
Aggregate
Data Preprocessing
Clean (Replace, Impute, Remove Outliers, Duplicates)
Partition (Train, Validate, Test)
Scale(Normalize, Standardize)
Unbias, Balance (Detection & Mitigation)
Augment
Feature Engineering
Feature Selection
Feature Transformation
Feature Creation (Encoding, Binning)
Feature Extraction (Automated in Deep Learning)

Figure 7: Data processing components

Best practices

- Data collection (p. 31)
- Data preparation (p. 32)
- Operational Excellence pillar – Best Practices (p. 34)
- Security pillar – Best Practices (p. 36)
- Reliability pillar – Best Practices (p. 40)
- Performance Efficiency pillar – Best Practices (p. 42)
- Cost Optimization pillar – Best Practices (p. 43)

Data collection

One of the first steps in the ML lifecycle is to identify what data is needed. Then evaluate the various means available for collecting that data to train your model.

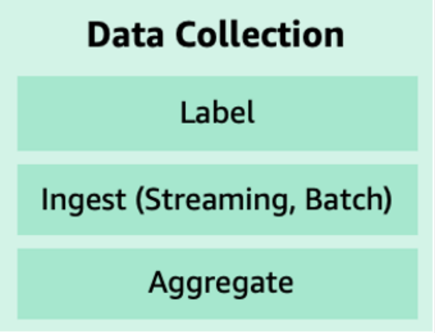

Figure 8: Data collection main components

The following are among the activities in the data collection phase shown in Figure 8:

- **Label** — Data for which you already know the target answer is called labeled data. If labels are missing, then some effort is required to label it (manual or automated).

- **Ingest & Aggregate** — Data collection includes ingesting and aggregating data from multiple data sources.

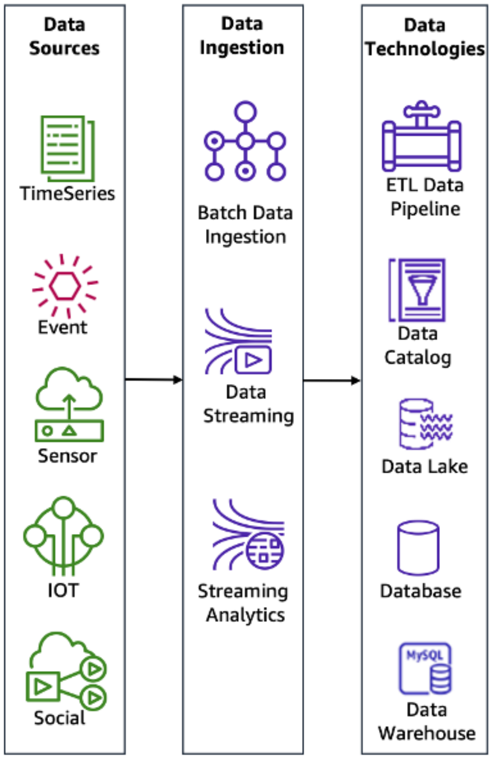

Figure 9: Data sources, ingestion, technologies

Some components of the ingest and aggregate can be seen in Figure 9:

- **Data sources** — Data sources include time-series, events, sensors, IoT devices, and social networks, depending on the nature of the use case.
- **Data ingestion** — Data ingestion processes and technologies capture and store data on storage media. Data ingestion can occur in real-time using streaming technologies or historical mode using batch technologies.
- **Data technologies** — Data storage technologies vary from transactional (SQL) databases, to data lakes and data warehouses. ETL pipeline technology automates and orchestrates the data movement and transformations across cloud services and resources. A data lake technology enables storing and analyzing structured and unstructured data.

Data preparation

ML models are only as good as the data that is used to train them. Ensure that suitable training data is available and is optimized for learning and generalization. Data preparation includes data preprocessing and feature engineering.

A key aspect to understanding data is to identify patterns. These patterns are often not evident with data in tables. Exploratory data analysis (EDA) with visualization tools can help quickly gain a deeper understanding of data. Prepare data using wrangler tools for interactive data analysis and model building. The no-code/low-code, automation, and visual capabilities improve the productivity and reduces the cost for interactive analysis.

Data preprocessing

Data preprocessing puts data into the right shape and quality for training. There are many data preprocessing strategies including: data cleaning, balancing, replacing, imputing, partitioning, scaling, augmenting and unbiasing.

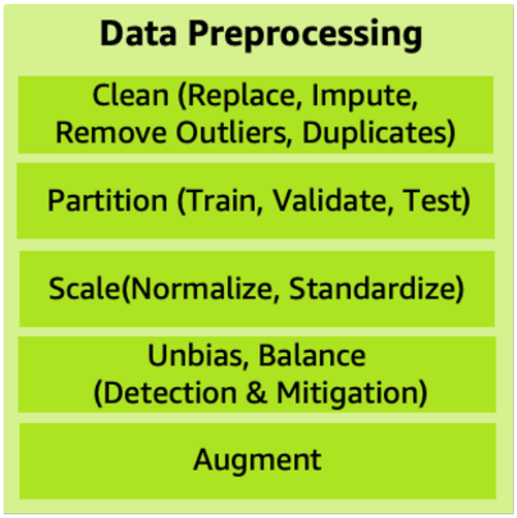
Data Preprocessing
Clean (Replace, Impute, Remove Outliers, Duplicates)
Partition (Train, Validate, Test)
Scale(Normalize, Standardize)
Unbias, Balance (Detection & Mitigation)
Augment

Figure 10: Data processing main components

Data preprocessing strategies listed in Figure 10 can be expanded as following:

- **Clean (replace, impute, remove outliers and duplicates)** — Remove outliers and duplicates, replace inaccurate or irrelevant data, and correct missing data using imputation techniques that will minimize bias as part of data cleaning.
- **Partition** — To prevent ML models from overfitting and evaluate trained model accurately, randomly split data into train, validate, and test sets. Care is needed to avoid data leakage. Data leakage happens when information from hold-out test dataset leaks into the training data. One way to avoid data leakage is to remove duplicates before splitting of the data.
- **Scale (normalize, standardize)** — Having features on a similar scale close to normally distributed will ensure that each feature is equally important. This will make it easier for most ML algorithms including K-Means, KNN, PCA, gradient descent. Normalized numeric features will have values into a range of [0,1]. Standardized numeric features will have a mean of 0 and standard deviation of 1. Standardization better handles the outliers.
- **Unbias, balance (detection & mitigation)** — Detecting and mitigating bias will help avoiding inaccurate model results. Biases are imbalances in the accuracy of predictions across different groups, such as age or income bracket. Biases can come from the data or algorithm used to train your model.
- **Augment** — Data augmentation increases the amount of data artificially by synthesizing new data from existing data. Data augmentation can help regularize and reduce overfitting.

Feature engineering

After exploring and gaining understanding of your data through visualizations and analysis, it’s time for feature engineering. Every unique attribute of the data is considered a feature. For example, when designing a solution for predicting customer churn, you start with the customer data that has been collected over time. The customer data captures features (also known as attributes), such as customer location, age, income level, and recent purchases.

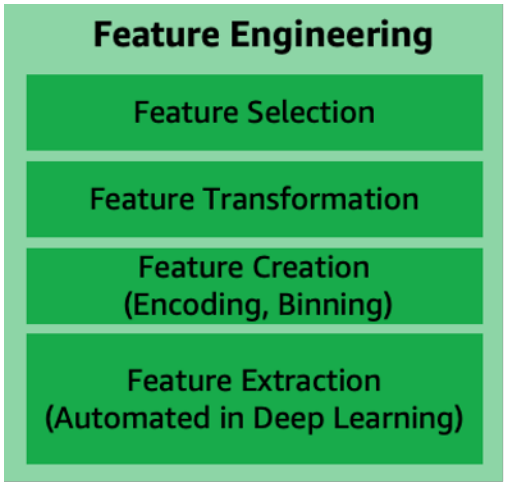

Figure 11: Feature engineering main components

Feature engineering is a process to select and transform variables when creating a predictive model using machine learning or statistical modeling. Feature engineering typically includes feature creation, feature transformation, feature extraction, and feature selection as listed in Figure 11. With deep learning, the feature engineering is automated as part of the algorithm learning.

- **Feature creation** is creating new features from existing data to help with better predictions. Examples of feature creation techniques include: one-hot-encoding, binning, splitting, and calculated features.
- **Feature transformation and imputation** manage replacing missing features or features that are not valid. Some techniques include: forming Cartesian products of features, non-linear transformations (such as binning numeric variables into categories), and creating domain-specific features.
- **Feature extraction** involves reducing the amount of data to be processed using dimensionality reduction techniques. These techniques include: PCA, ICA, and LDA. This reduces the amount of memory and computing power required, while still accurately maintaining original data characteristics.
- **Feature selection** is the process of selecting subset of extracted features. This subset is relevant and contributes to minimizing the error rate of a trained model. Feature importance score and correlation matrix can be factors in selecting the most relevant features for model training.

Operational Excellence pillar – Best Practices

The operational excellence pillar includes the ability to run and monitor systems to deliver business value and to continually improve supporting processes and procedures. This section includes best practices to consider while processing data.

Best practices

- MLOE-07: Profile data to improve quality (p. 34)
- MLOE-08: Document data processing (p. 35)

MLOE-07: Profile data to improve quality

Profile data to use data characteristics like distribution, descriptive statistics, data types, and data patterns. Review source data for content and quality. Filter out or correct any data not passing the reviews. This will contribute to quality improvement.

Implementation plan

- **Use Amazon SageMaker Data Wrangler** — Import, prepare, transform, visualize, and analyze data with SageMaker Data Wrangler. You can integrate Data Wrangler into your ML workflows to simplify and streamline data pre-processing and feature engineering with little to no coding. You can also add your own Python scripts and transformations to customize your data preparation workflow. Import data from Amazon S3, Amazon Redshift, or other data sources, and then query the data using Amazon Athena. Use Data Wrangler to create sophisticated machine learning data preparation workflows with built-in and custom data transformations and analysis features. These features include feature target leakage and quick modeling.
- **Create an automatic data profile and a reporting system** — Use AWS Glue Crawler to crawl the data sources and create a data schema. Use AWSAWS Glue Data Catalog to list all the tables and schemas. Use Amazon Athena for serverless SQL querying to constantly profile data and then use Amazon QuickSight dashboards for visualization of the data.

Documents

- Data Wrangler – Getting Started

Blogs

- Introducing Amazon SageMaker Data Wrangler, a Visual Interface to Prepare Data for Machine Learning
- Exploratory data analysis, feature engineering, and operationalizing your data flow into your ML pipeline with Amazon SageMaker Data Wrangler
- Prepare data for predicting credit risk using Amazon SageMaker Data Wrangler and Amazon SageMaker Clarify
- Prepare data from Snowflake for machine learning with Amazon SageMaker Data Wrangler
- Develop and deploy ML models using Amazon SageMaker Data Wrangler and Amazon SageMaker Autopilot
- Build an automatic data profiling and reporting solution with Amazon EMR, AWS Glue, and Amazon QuickSight

MLOE-08: Document data processing

Document and version control any data processing-related findings, processes, and improvement to enable easier future referencing and reuse. Sample topics to document with respect to data processing include: sources of data, time of data collection, processes, scripts used for data collection, structure of data, types of transformation, and manipulations conducted on data.

Implementation plan

- **Associate notebook instances with Git repositories** — To analyze data, document its processing, and evaluate ML models on Amazon SageMaker, you can use Amazon SageMaker Processing. SageMaker Processing can be used for feature engineering, data validation, model evaluation, and model interpretation. SageMaker notebook instances can be associated with Git repositories. This enables saving notebooks in a source control environment that persists after stopping or deleting the notebook instance. The notebooks will hold the data processing coding and its documentation. You can associate one default repository and up to three additional repositories with a notebook instance. The repositories can be hosted in AWS CodeCommit, GitHub, or on any other Git server.

Documents

- Amazon SageMaker – Process Data
- Associate Git Repositories with SageMaker Notebook Instances

Blogs

- Amazon SageMaker notebooks now support Git integration for increased persistence, collaboration, and reproducibility

Examples

- Amazon SageMaker processing

Security pillar – Best Practices

The security pillar encompasses the ability to protect data, systems, and assets to take advantage of cloud technologies to improve your security. This section includes best practices to consider while processing data.

Best practices

- MLSEC-03: Secure data and modeling environment (p. 36)
- MLSEC-04: Protect sensitive data privacy (p. 38)
- MLSEC-05: Enforce data lineage (p. 39)
- MLSEC-06: Keep only relevant data (p. 39)

MLSEC-03: Secure data and modeling environment

Secure any system or environment that hosts data or enables model development. Store training data on secured storage and repositories. Run data preparation model development in a secure cloud. Tightly control access to the destination compute instances as data moves from the data repositories to the instances. Encrypt data in transit to, and at rest on the compute and storage infrastructure.

Implementation plan

- **Build a secure analysis environment** — During the data preparation and feature engineering phases, there are multiple options for secure data exploration on AWS. Data can be explored in an Amazon SageMaker managed notebook environment, or in an Amazon EMR notebook. You can also use managed services, such as Amazon Athena and AWS Glue, to explore the data without moving the data out of your data lake. A combination of the two approaches can also be used. Use an Amazon SageMaker Jupyter notebook instance to explore, visualize, and feature engineer a small subset of data. Scale up the feature engineering using a managed ETL service, such as Amazon EMR or AWS Glue.

  Create dedicated AWS IAM and AWS KMS resources for each project, limiting the scope of impact of credentials and keys. Create a private S3 bucket and enable version control for the data and intellectual property (IP) created. In AWS, a centralized data lake is implemented using AWS Lake Formation on Amazon S3. Securing and monitoring a data lake on Amazon S3 is achieved using a combination of various services and capabilities to encrypt data in transit and at rest and monitor access including granular AWS IAM policies, S3 bucket policies, S3 Access Logs, Amazon CloudWatch, and AWS CloudTrail. Building Big Data Storage Solutions (Data Lakes) for Maximum Flexibility discusses using these various capabilities to build a secure data lake.

- **Use Secrets Manager to protect credentials** — Secrets Manager enables you to replace hardcoded secrets in your code, such as credentials, with an API call to Secrets Manager to decrypt and retrieve the secret programmatically. With AWS Secrets Manager you can store your credentials, and then grant permissions to your SageMaker IAM role to access Secrets Manager from your notebook.
- **Automate managing configuration** — Use lifecycle configurations as shell scripts to manage Jupyter notebook instances. They can run when the notebook instance is first created, or every time it starts. They enable you to install custom tools, packages, or monitoring. Lifecycle configurations can be changed and reused across multiple notebook instances. You can make a change once and apply the updated configuration by restarting the managed notebook instances. This gives IT, operations, and security teams the control they need, while supporting the needs of your developers and data scientists. Use AWS CloudFormation infrastructure as code, as well as AWS Service Catalog to simplify configuration for end users.
- **Create private, isolated, network environments** — Use Amazon Virtual Private Cloud (Amazon VPC) to enable connectivity to only the services and users you need. Deploy the Amazon SageMaker notebook instance in an Amazon VPC to enable network level controls to limit communication to the hosted notebook. Additionally, network calls into and out of the notebook instance can be captured in VPC Flow Logs to enable additional visibility and control at the network level. By deploying the notebook in your VPC, you will also be able to query data sources and systems accessible from within your VPC, such as relational databases in Amazon RDS or Amazon Redshift data warehouses. Using IAM, you can further restrict access to the web-based UI of the notebook instance so that it can only be accessed from within your VPC.

  Use AWS PrivateLink to privately connect your SageMaker notebook instance VPC with supported AWS services. This ensures secure communication between your notebook instance and Amazon S3 within the AWS network. Use AWS KMS to encrypt data on the EBS volumes attached to SageMaker notebook instances.

- **Restrict access** — The Jupyter notebook server provides web-based access to the underlying operating system on an EC2 instance. This gives you the ability to install additional software packages or Jupyter kernels to customize your environment. The access is granted by default to a user with access to the root or super user on the operating system, giving them total control of the underlying EC2 instance. This access should be restricted to remove the user's ability to assume root permissions but still give them control over their local user's environment.
- **Secure ML algorithms** — Amazon SageMaker uses container technology to train and host algorithms and models. This enables Amazon SageMaker and other ML partners to package algorithms and models as containers that you can then use as part of your ML project. When creating your own containers, publish them to a private container registry hosted on Amazon Elastic Container Repository (Amazon ECR). Encrypt containers that are hosted on Amazon ECR at rest using AWS KMS.
- **Enforce code best practices** — Use secure git repositories through fully managed AWS CodeCommit for storing code.

Documents

- Amazon Sagemaker Workshop - Using Secure Environments
- What is AWS CodeCommit?
- Security in Amazon SageMaker

Blogs

- 7 ways to improve security of your machine learning
- Building secure machine learning environments with Amazon SageMaker
- Setting up secure, well-governed machine learning environments on AWS
- Private package installation in Amazon SageMaker running internet-free mode

- Secure Deployment of Amazon SageMaker resource

Videos

- AWS re:Invent 2019: Security for ML environments w/ Amazon SageMaker, featuring Vanguard (AIM327-R1)
- AWS re:Invent 2020: Security best practices the AWS Well-Architected way

Examples

- Secure Data Science Reference Architecture

MLSEC-04: Protect sensitive data privacy

Protect sensitive data used in training against unintended disclosure. Identify and classify the sensitive data. Handle the sensitive data using strategies including: removing, masking, tokenizing, and principal component analysis (PCA). Document best governance practices for future reuse and references.

Implementation plan

- **Use automated mechanisms to classify data where possible** — Use systems such as Amazon Macie to classify and identify data locations and potentially sensitive data.
- **Use tagging** - Tag resources and models that are made from sensitive elements to quickly differentiate between resources requiring protection and those that do not.
- **Encrypt sensitive data** — Encrypt sensitive data using services such as AWS KMS, the AWS Encryption SDK, or client-side encryption.
- **Reduce data sensitivity** — Evaluate and identify data for anonymization or de-identification to reduce sensitivity.

Documents

- Running sensitive data discovery jobs in Amazon Macie
- Categorizing your storage using tags
- AWS Key Management service best practices
- Getting started with the AWS Encryption SDK

Blogs

- 7 ways to improve security of your machine learning workflows
- Macie for Data Classification
- Building a Serverless Tokenization Solution to Mask Sensitive Data

Examples

- Amazon SageMaker Solution for Privacy in Natural Language Processing
- How Amazon is advancing privacy-aware data processing

MLSEC-05: Enforce data lineage

Monitor and track data origins and transformations over time. Strictly control who can access the data and what they can do with it. Perform preventative controls, auditing, and monitoring to demonstrate how data has been controlled during its lifetime. Implement integrity checks against training data to detect any unexpected deviances caused by loss, corruption, or manipulation. Data lineage enables visibility and tracing data processing errors back to the root cause.

Implementation plan

- **Track records for any update** — Create and store information about the steps of a ML workflow from data preparation to model deployment using Amazon SageMaker ML Lineage Tracker. With the tracking information you can:
  - Reproduce the workflow steps, track model and dataset lineage, and establish model governance and audit standards.
  - Consider origin data to be the source of truth.
  - Ingest and process into derived datasets and retain mappings throughout the process. Iterate from the end result back to the original data element it came from.
  - Apply these concepts not just to data, but also the code, models, pipelines, and infrastructure. Validate that you can trace and audit any activity, against data, pipeline actions, or machine learning models.

Documents

- Amazon SageMaker ML Lineage Tracking

Blogs

- Using model attributes to track your training runs on Amazon SageMaker

Videos

- AWS re:Invent 2019: Creating a data-driven, cloud-native ecosystem at BMW Group (AUT306)
- AWS re:Invent 2020: Nationwide’s journey to a governed data lake on AWS

Examples

- LAB 04: DevOps WorkFlow

MLSEC-06: Keep only relevant data

Preserve data across computing environments (such as development and staging) and only store the data that has business need to reduce data exposure risks. Implement mechanisms to enforce a lifecycle management process across the data. Decide when to remove data automatically based on age.

Implementation plan

- **Establish a data lifecycle plan** — Understand usage patterns and requirements for debugging and operational tasks. Establish a data lifecycle plan to reduce data sprawl over time.
- **Design for privacy** — Remove sensitive elements that are not needed for the ML workflow. Detect and redact personally identifiable information (PII), while maintaining data usability. Determine what

features and data elements are required to solve the business problem, and valuable to keep for iterations in the future.

Documents

- Reference Guide: Extract More Value from your Data

Blogs

- Building a data analytics practice across the data lifecycle
- Detecting and redacting PII using Amazon Comprehend
- Now available in Amazon Transcribe: Automatic Redaction of Personally Identifiable Information
- Machine learning models that act on encrypted data
- Redacting sensitive information with user-defined functions in Amazon Athena

Videos

- AWS re:Invent 2020: Privacy-preserving machine learning
- AWS re:Invent 2019: Best practices for Amazon S3

Examples

- Field Notes: Redacting Personal Data from Connected Cars Using Amazon Rekognition
- How to Create a Modern CPG Data Architecture with Data Mesh

Reliability pillar – Best Practices

The reliability pillar encompasses the ability of a workload to perform its intended function correctly and consistently when it’s expected to. This section includes best practices to consider while processing data.

Best practices

- MLREL-04: Use a data catalog (p. 40)
- MLREL-05: Use a data pipeline (p. 41)
- MLREL-06: Automate managing data changes (p. 42)

MLREL-04: Use a data catalog

Process data across multiple data stores using data catalog technology. An advanced data catalog service can enable ETL process integration. This approach enables more reliability and efficiency.

Implementation plan

- **Use AWS Glue Data Catalog** — The AWS Glue Data Catalog provides a way to track the data assets that have been loaded into your ML workload. Data catalogs not only track the assets but also describe how they are transformed as they are loaded into your data lake and data warehouse. AWS Glue is a fully managed ETL (extract, transform, and load) service. It enables simple and cost-effective approach to categorize your data, clean it, enrich it, and move it reliably between various data stores and data streams. AWS Glue consists of a central metadata repository known as the AWS AWS Glue Data Catalog. It also has an ETL engine that automatically generates Python or Scala code. With a flexible scheduler, AWS Glue handles dependency resolution, job monitoring, and retries.

Documents

- Data Cataloging
- Populating the AWSAWS Glue Data Catalog

Blogs

- Moving from notebooks to automated ML pipelines using Amazon SageMaker and AWS Glue
- How Genworth built a serverless ML pipeline on AWS using Amazon SageMaker and AWS Glue

Videos

- Getting Started with AWSAWS Glue Data Catalog
- AWS re:Invent 2018: How Bill.com Uses Amazon SageMaker & AWS Glue to Enable Machine Learning - STP10
- AWS re:Invent 2018: Build and Govern Your Data Lakes with AWS Glue (ANT309)

Examples

- Explaining Credit Decisions with Amazon SageMaker
- How to build an end-to-end Machine Learning pipeline using AWS Glue, Amazon S3, Amazon SageMaker and Amazon Athena.

MLREL-05: Use a data pipeline

Automates processing and movement and transformation of data between different compute and storage services. It enables data processing that is fault tolerant, repeatable, and highly available.

Implementation plan

- **Use Amazon SageMaker Data Wrangler and Pipelines** — SageMaker Data Wrangler simplifies the preparation of machine learning data. It enables data selection, cleansing, exploration, and visualization, using a single visual interface. After you've created a workflow, export it to SageMaker Pipelines to automate model deployment and management. Data pipelines provide an automated way to move and transform data in your ML workload. Manually moving and transforming data can lead to errors and inconsistencies. Use AWS Data Pipeline to save time and decrease errors.

Documents

- Prepare ML Data with Amazon SageMaker Data Wrangler
- Amazon SageMaker Model Building Pipelines

Blogs

- Building, automating, managing, and scaling ML workflows using Amazon SageMaker Pipelines
- Develop and deploy ML models using Amazon SageMaker Data Wrangler and Amazon SageMaker Autopilot

Videos

- AWS re:Invent 2020: Accelerate data preparation with Amazon SageMaker Data Wrangler

- Amazon SageMaker Data Wrangler Deep Dive Demo

Examples

- Amazon SageMaker Data Wrangler and Feature Store
- SageMaker Pipelines

MLREL-06: Automate managing data changes

Automate managing changes to training data using version control technology. This will enable reproducibility to re-create the exact version of a model in the event of a failure.

Implementation plan

- **Use AWS MLOps Framework** — AWS MLOps Framework provides a standard interface for managing ML pipelines for Amazon Machine Learning services and third-party services. The solution's template allows you to upload your trained models (also referred to as *bring your own model*). It configures the orchestration of the pipeline, and monitors the pipeline's operations. This solution increases agility and efficiency by allowing repeating of successful processes at scale. One of the key components of MLOps pipeline in SageMaker is Model Registry. SageMaker Model Registry tracks the model versions and respective artifacts, including the lineage and metadata.

Documents

- AWS MLOps Framework
- Register and Deploy Models with Model Registry

Blogs

- Amazon SageMaker Pipelines Brings DevOps Capabilities to your Machine Learning Projects

Videos

- Solving with AWS Solutions: AWS MLOps Framework

Examples

- Amazon SageMaker secure MLOps

Performance Efficiency pillar – Best Practices

The performance efficiency pillar focuses on the efficient use of computing resources to meet requirements and the maintenance of that efficiency as demand changes and technologies evolve. This section includes best practices to consider while processing data.

Best practices

- MLPER-05: Use a data lake house architecture (p. 43)

MLPER-05: Use a data lake house architecture

Get the best insights from exponentially growing data using lake house data management architecture. It enables easy movement of data between data lake and purpose-built stores including: data warehouse, relational databases, non-relational databases, ML and big data processing, and log analytics. A data lake provides a single place to run analytics across mixed data structures collected from disparate sources. Purpose-built analytics services provide the speed required for specific use cases like real-time dashboards and log analytics.

Implementation plan

- **Unify data governance and access** — Integrate a data lake, a data warehouse, and purpose-built stores. This will enable unified governance and easy data movement. With a Lake House architecture on AWS, you can store data in a data lake and use data services around it. Use AWS Lake Formation to build a scalable and secure data lake. Build a high-speed analytic layer with purpose-built services, such as Amazon Redshift, Amazon Kinesis, and Amazon Athena. Integrate data across services and data stores with AWS Glue. Apply governance policies to manage security, access control, and audit trails across all the data stores using AWS IAM.

Documents

- Data Lake on AWS
- AWS Lake Formation
- Derive Insights from AWS Lake House

Blogs

- Build a Lake House Architecture on AWS
- Moving from notebooks to automated ML pipelines using Amazon SageMaker and AWS Glue
- Data preprocessing for machine learning on Amazon EMR made easy with AWS Glue DataBrew

Videos

- Build and Govern Your Data Lakes with AWS Glue
- The lake house approach to data warehousing with Amazon Redshift

Examples

- Predictive Data Science with Amazon SageMaker and a Data Lake on AWS

Cost Optimization pillar – Best Practices

The cost optimization pillar includes the continual process of refinement and improvement of a system over its entire lifecycle. From the initial design of your very first proof of concept to the ongoing operation of production workloads, adopting the practices in this document can enable you to build and operate cost-aware systems that achieve business outcomes and minimize costs, thus allowing your business to maximize its return on investment. This section includes best practices to consider while processing data.

Best practices

- MLCOST-08: Use managed data labeling (p. 44)

- MLCOST-09: Use data wrangler tools for interactive analysis (p. 44)
- MLCOST-10: Enable feature reusability (p. 45)
- MLCOST-11: Establish data bias detection and mitigation (p. 46)

MLCOST-08: Use managed data labeling

Choose a managed labeling tool that provides automation and access to cost-effective labeling workforce. It should also provide flexibility to choose variable number of labelers for a given input. The tool should have a user interface, and learn to label data by itself over time.

Implementation plan

- **Use Amazon SageMaker Ground Truth** — To train a machine learning model, you need a large, high-quality, labeled dataset. Amazon SageMaker Ground Truth helps you build high-quality training datasets for your ML models. With Ground Truth, you can use ML along with workers from Amazon Mechanical Turk, a vendor company that you choose, or an internal, private workforce to a labeled dataset. You can use the labeled dataset output from Ground Truth to train your own models. You can also use the output as a training dataset for an Amazon SageMaker model.

Documents

- Use Amazon SageMaker Ground Truth to Label Data

Blogs

- Real-time data labeling pipeline for ML workflows using Amazon SageMaker Ground Truth
- Implementing a custom labeling GUI with built-in processing logic with Amazon SageMaker Ground Truth

Examples

- Bring your own model for SageMaker labeling workflows with active learning
- SageMaker Ground Truth recipe

MLCOST-09: Use data wrangler tools for interactive analysis

Prepare data through wrangler tools for interactive data analysis and model building. The no-code/low-code, automation, and visual capabilities improve the productivity and reduce the cost for interactive analysis.

Implementation plan

- **Use Amazon SageMaker Data Wrangler** — Amazon SageMaker Data Wrangler, a feature of SageMaker Studio, provides an end-to-end solution to import, prepare, transform, select features, and analyze data. You can integrate a Data Wrangler data flow into your ML workflows to simplify and streamline data pre-processing and feature engineering using little to no coding. You can also add your own Python scripts and transformations to customize workflows.

Documents

- Amazon SageMaker Data Wrangler

Blogs

- Introducing Amazon SageMaker Data Wrangler, a Visual Interface to Prepare Data for Machine Learning
- Develop and deploy ML models using Amazon SageMaker Data Wrangler and Amazon SageMaker Autopilot

Videos

- Amazon SageMaker Data Wrangler Deep Dive Demo

Examples

- Prepare ML Data with Amazon SageMaker Data Wrangler

MLCOST-10: Enable feature reusability

Reduce duplication and rerun of feature engineering code across teams and projects using a feature storage. The store should have online and offline storage, and data encryption capabilities. Online store with low-latency retrieval capabilities is ideal for real-time inference. Offline store should maintain a history of feature values and is suited for training and batch scoring.

Implementation plan

- **Use Amazon SageMaker Feature Store** — Amazon SageMaker Feature Store is a fully managed, purpose-built repository to store, update, retrieve, and share ML features. Feature Store makes it easy for data scientists, machine learning engineers, and general practitioners to create, share, and manage features for ML development. Feature Store accelerates this process by reducing repetitive data processing and curation work required to convert raw data into features for training an ML algorithm.

  You can use Feature Store in the following modes:

  - Online — Features are read with low latency reads (milliseconds) and used for high throughput predictions. This mode requires a feature group to be stored in an online store.
  - Offline — Large streams of data are fed to an offline store, which is used for training and batch inference. This mode requires a feature group to be stored in an offline store. The offline store uses your S3 bucket for storage and can also fetch data using Amazon Athena queries.
  - Online and offline — This includes both online and offline modes.

Documents

- Create, Store, and Share Features with Amazon SageMaker Feature Store

Blogs

- Using Amazon SageMaker Feature Store with streaming feature aggregation

Videos

- Amazon SageMaker Feature Store Deep Dive Demo

Examples

- Using Amazon SageMaker Feature Store with streaming feature aggregation

MLCOST-11: Establish data bias detection and mitigation

Detect and mitigate bias to avoid inaccurate model results. Establish bias detection methodologies at data preparation stage before training starts. Also, monitor, detect, and mitigate bias after the model is in production. Establish feedback loops to track the drift over time and initiate a re-training.

Implementation plan

- **Use Amazon SageMaker Clarify** — Amazon SageMaker Clarify helps improve your machine learning models by detecting potential bias and helping explain how these models make predictions. The fairness and explainability functionality provided by SageMaker Clarify takes a step towards enabling you to build trustworthy and understandable ML models. Clarify helps you with the following tasks:
  - Measure biases that can occur during each stage of the ML lifecycle. These stages include data collection, model training, model tuning, and model monitoring.
  - Generate model governance reports targeting risk and compliance teams and external regulators.
  - Provide explanations of the data, models, and monitoring used to assess predictions.

Documents

- Run SageMaker Clarify Processing Jobs for Bias Analysis and Explainability

Blogs

- Amazon SageMaker Clarify Detects Bias and Increases the Transparency of Machine Learning Models

Videos

- Introducing Amazon SageMaker Clarify, part 1 - Bias detection - AWS re:Invent 2020
- Introducing Amazon SageMaker Clarify, part 2 - Model explainability - AWS re:Invent 2020

Examples

- SageMaker Clarify

ML lifecycle phase – Model development

Model development consists of model building, training, tuning, and evaluation.

Topics

- Model training, tuning (p. 47)
- Model evaluation (p. 49)
- Operational Excellence pillar – Best Practices (p. 49)
- Security pillar – Best Practices (p. 51)
- Reliability pillar – Best Practices (p. 54)
- Performance Efficiency pillar – Best Practices (p. 57)

- Cost Optimization pillar – Best Practices (p. 61)

Model training, tuning

In this phase, you select a machine learning algorithm that is appropriate for your problem and then train the ML model. For this phase, you provide the algorithm with the training data, set an objective metric for the ML model to optimize on, and set the hyperparameters to optimize the training process.

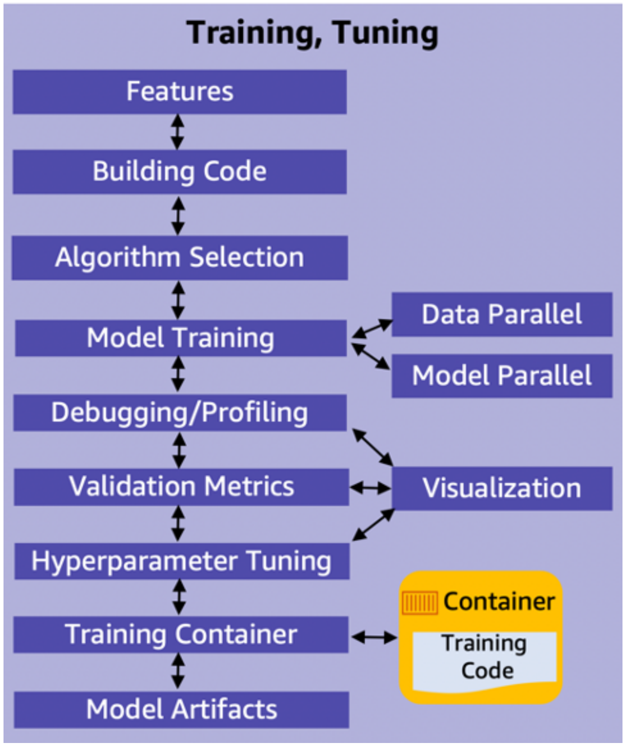

Figure 12: Training and tuning main components

Model training, tuning, and evaluation require prepared data and engineered features. The following are among the activities in this stage, as listed in Figure 12:

- **Features** – Features are selected as part of the data processing after a bias strategy is implemented.

- **Building code** – Model development includes building the algorithm and its supporting code. Building code process should support version control, and continuous build, test, and integration through a pipeline.
- **Algorithm selection** – Selecting the right algorithm involves running many experiments with parameter tunings across available options. Factors to consider when evaluating each option can include accuracy, explainability, training/prediction time, memory requirements.
- **Model training** (data training parallel, model training parallel) – The process of training an ML model involves providing an ML algorithm with training data to learn from. Distributed training enables splitting large models and training datasets across computing instances to reduce runtime to fraction of it takes to do manually. Model parallelism and data parallelism are techniques to achieve distributed training.
  - **Model parallelism** is the process of splitting a model up between multiple devices or nodes.
  - **Data parallelism** is the process of splitting the training set in mini-batches evenly distributed across nodes. Thus, each node only trains the model on a fraction of the total dataset.

- **Debugging/profiling** – A machine learning training job can have problems including: system bottlenecks, overfitting, saturated activation functions, and vanishing gradients. These problems can compromise model performance. A debugger provides visibility into the ML training process through monitoring, recording, and analyzing data. It captures the state of a training job at periodic intervals.
- **Validation metrics** – Typically, a training algorithm computes several metrics, such as loss and prediction accuracy. These metrics determine if the model is learning and generalizing well for making predictions on unseen data. Metrics reported by the algorithm depend on the business problem and the ML technique used. For example, a *confusion matrix* measures a classification model, and RMSE measures a regression model.
- **Hyperparameter tuning** – Settings that can be tuned to control the behavior of the ML algorithm are referred to as *hyperparameters*. The number and type of hyperparameters in ML algorithms are specific to each model. Examples of commonly used hyperparameters include: learning rate, number of epochs, hidden layers, hidden units, and activation functions. Hyperparameter tuning, or optimization, is the process of choosing the optimal hyperparameters for a learning algorithm.
- **Training code container** – Create container images with your training code and its entire dependency stack. This will enable training machine learning algorithms and deploy models quickly and reliably at any scale.
- **Model artifacts** – Model artifacts are the output that results from training a model. They typically consist of trained parameters, a model definition that describes how to compute inferences, and other metadata.
- **Visualization** – Enables exploring and understanding data during metrics validation, debugging, profiling, and hyperparameter tuning.

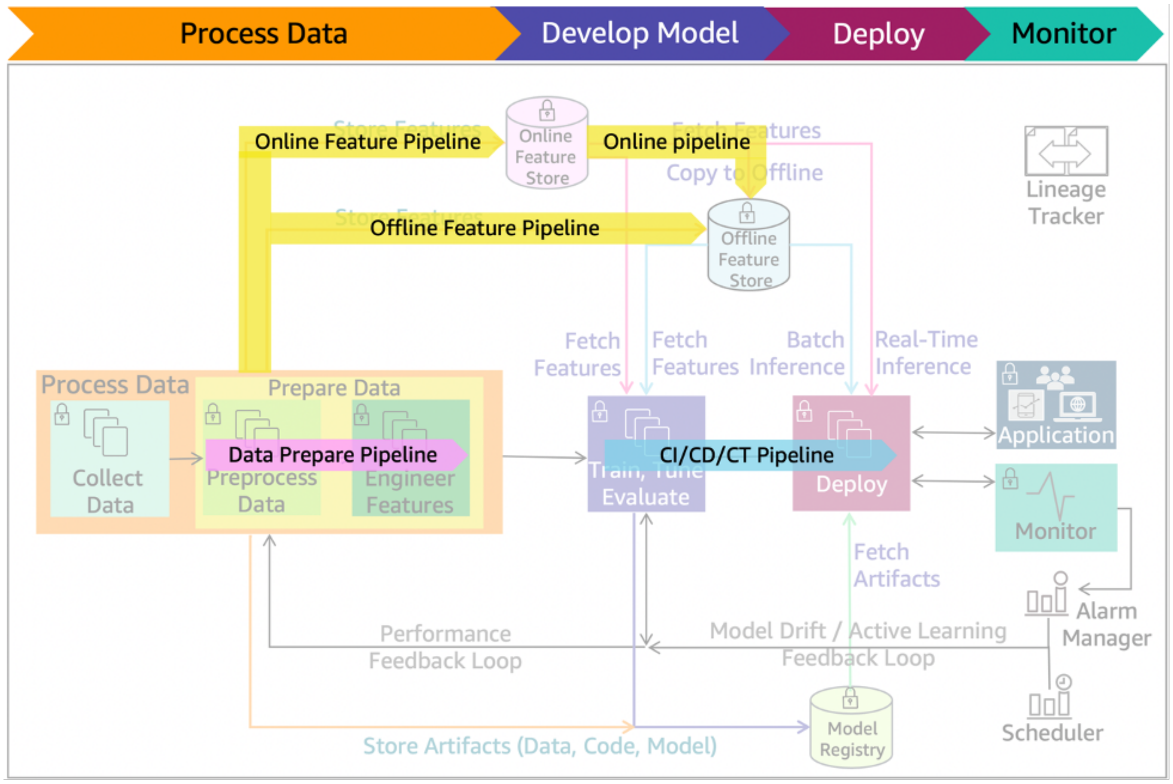

Figure 13: ML lifecycle with pre-production pipelines

Figure 13 shows the pre-production pipelines. The data prepare pipeline automates data preparation tasks. The feature pipeline automates the storing, fetching, and copying of the features into and from online/offline store. The CI/CD/CT pipeline automates the build, train, and release to staging and production environments.

Model evaluation

After the model has been trained, evaluate it for its performance and accuracy. You might want to generate multiple models using different methods and evaluate the effectiveness of each model. For example, apply different business rules, and then apply various measures to determine each model's suitability. You also might evaluate whether your model must be more sensitive than specific, or more specific than sensitive. For multiclass models, determine error rates for each class separately.

You can evaluate your model using historical data (offline evaluation) or live data (online evaluation). In offline evaluation, the trained model is evaluated with a portion of the dataset that has been set aside as a *holdout set*. This holdout data is never used for model training or validation—it's only used to evaluate errors in the final model. The holdout data annotations must have high accuracy for the evaluation to make sense. Allocate additional resources to verify the accuracy of the holdout data.

Based on the evaluation results, you might fine-tune the data, the algorithm, or both. When you fine-tune the data, you apply the concepts of data cleansing, preparation, and feature engineering.

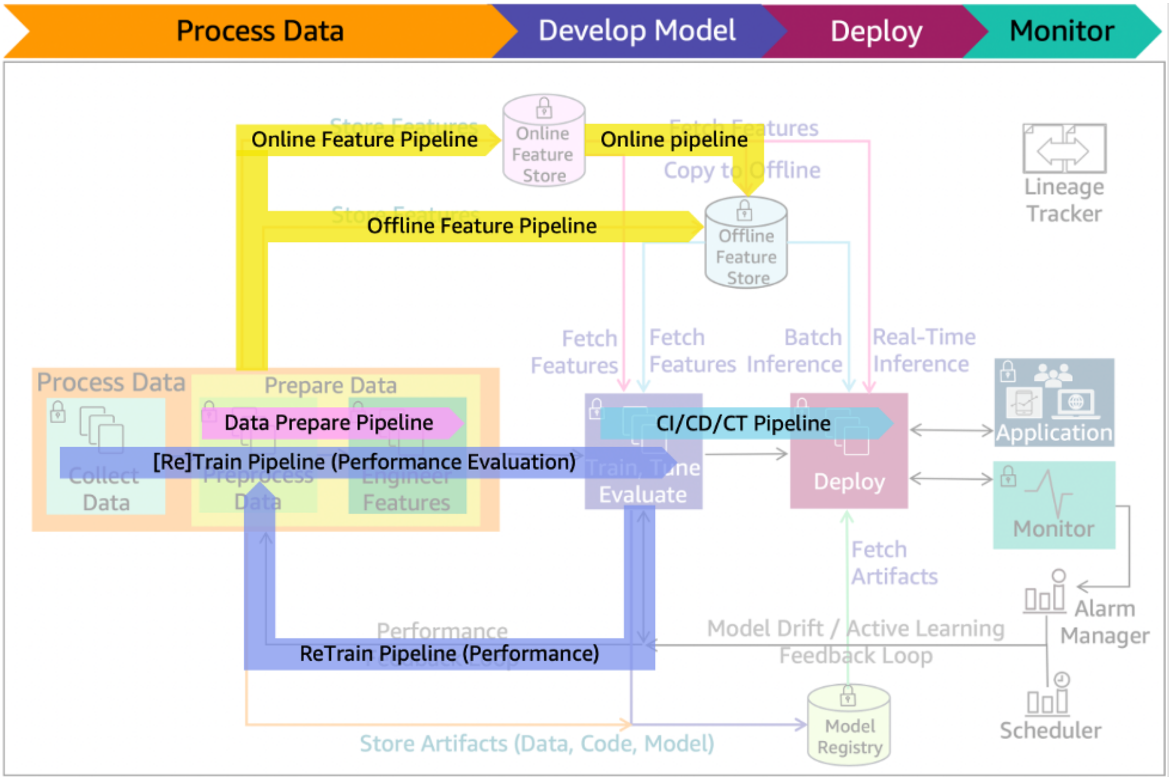

Figure 14: ML lifecycle with performance evaluation pipeline added

Figure 14 includes the model performance evaluation, data prepare and CI/CD/CT pipelines that fine-tune data and/or algorithm, re-train and evaluate model results.

Operational Excellence pillar – Best Practices

The operational excellence pillar includes the ability to run and monitor systems to deliver business value and to continually improve supporting processes and procedures. This section includes best practices to consider while developing models that includes training, tuning, and model performance evaluation.

Best practices

- MLOE-09: Automate operations through IaC and CaC (p. 50)
- MLOE-10: Establish scalable patterns to access approved public libraries (p. 50)

MLOE-09: Automate operations through IaC and CaC

Automate ML workload operations using infrastructure as code (IaC) and configuration as code (CaC). This ensures consistency across staging, and production deployment environments. Enable traceability and version control across infrastructures and configurations.

Implementation plan

- **Use AWS CloudFormation** — AWS CloudFormation enables you to create and provision AWS deployments predictably and repeatedly. AWS CloudFormation enables you to use a template file to create and delete a collection of resources together as a single unit (a stack). You can manage and provision stacks across multiple AWS accounts and AWS Regions.
- **Use AWS Cloud Development Kit (CDK)** — Use AWS Cloud Development Kit (CDK) (AWS CDK) as a software development framework for defining cloud infrastructure in code and provisioning it through AWS CloudFormation. Use the AWS CDK to define your cloud resources in a familiar programming language.

Documents

- Implement infrastructure as code
- Infrastructure as Code
- AWS CloudFormation
- AWS Cloud Development Kit (CDK)

Blogs

- AWS CloudFormation – Create Your AWS Stack From a Recipe
- Automate Amazon SageMaker Studio setup using AWS SDK
- Secure multi-account model deployment with Amazon SageMaker: Part 1
- Secure multi-account model deployment with Amazon SageMaker: Part 2

MLOE-10: Establish scalable patterns to access approved public libraries

Establish scalable patterns for data scientists to access approved public libraries by creating separate kernels for common ML frameworks. Examples of such common ML frameworks include: Tensorflow, PyTorch, Scikit-learn, and Keras. This allows using internal repositories to give access to public libraries and create separate kernels for common ML frameworks.

Implementation plan

- **Use container technology** — Use AWS Deep Learning Containers or alternatively bring custom containers and store them in Amazon Elastic Container Registry (Amazon ECR). Using containers, you can train machine learning algorithms and deploy models quickly and reliably at any scale.
- **Use artifact repository** — Set up AWS CodeArtifact to be used as a central internal artifact repository. This will enable pulling artifacts from internal repositories and reusing them.

Documents

- Train a Deep Learning model with AWS Deep Learning Containers on Amazon EC2
- AWS Deep Learning Containers
- What is AWS CodeArtifact?

Blogs

- Private package installation in Amazon SageMaker running in internet-free mode
- Bringing your own custom container image to Amazon SageMaker Studio notebooks
- Integrating Jenkins with AWS CodeArtifact to publish and consume Python artifacts

Security pillar – Best Practices

The security pillar encompasses the ability to protect data, systems, and assets to take advantage of cloud technologies to improve your security. This section includes best practices to consider during model development that includes model training, tuning, and model performance evaluation.

Best practices

- MLSEC-07: Detect transfer learning risk (p. 51)
- MLSEC-08: Secure governed ML environment (p. 51)
- MLSEC-09: Secure inter-node cluster communications (p. 52)
- MLSEC-10: Protect against data poisoning threats (p. 53)

MLSEC-07: Detect transfer learning risk

Monitor and ensure the inherited prediction weights from a transferred model lead to the correct results. This helps minimize the risk of weak learning and incorrect outputs using pre-trained models.

Implementation plan

- **Leverage Amazon SageMaker Debugger** — Transfer learning is a machine learning technique where a model pre-trained on one task is fine-tuned on a new task. When using the transfer learning approach, use Amazon SageMaker Debugger to detect hidden problems that might have serious consequences. Examine model predictions to see what mistakes were made. Validate the robustness of your model, and consider how much of this robustness is from the inherited capabilities. Validate input and preprocesses to the model for realistic expectations.

Blogs

- When does transfer learning work?
- Detecting hidden but non-trivial problems in transfer learning models using Amazon SageMaker Debugger

MLSEC-08: Secure governed ML environment

Protect ML operations environments using managed services with best practices including: Detective and preventive guardrails, monitoring, security and incident management. Explore data in a managed and secure notebook environment. Centrally manage the configuration of Jupyter environments and enable self-service provisioning for the users.

Implementation plan

- **Break out ML workloads** by organizational unit access patterns. This will enable delegating required access to each group, such as administrators or data analysts.
- **Use guard rails and service control policies (SCPs)** to enforce best practices for each environment type. Limit infrastructure access to administrators.
- **Verify all sensitive data has access through restricted, isolated environments**. Ensure this by examining network isolation, dedicated resources, and service dependencies.
- **Secure ML algorithm implementation** using a restricted Amazon SageMaker Jupyter notebook environment. Secure model training and hosting containers by following the security processes required for your organization.
- **Build a secure enterprise machine learning platform on AWS.**

Documents

- Security in Amazon SageMaker
- Use Amazon SageMaker Notebook Instances
- Amazon SageMaker with Guardrails on AWS

Blogs

- Setting up secure, well-governed machine learning environments on AWS
- Securing Amazon SageMaker Studio Connectivity using a private VPC
- Enable self-service, secured data science using Amazon SageMaker notebooks and AWS Service Catalog
- Accelerating Machine Learning Development with Data Science as a Service from Change Healthcare

Videos

- AWS re:Invent 2020: Implementing MLOps practices with Amazon SageMaker

Examples

- AWS ML Ops Framework
- SageMaker Projects Walkthrough

MLSEC-09: Secure inter-node cluster communications

For frameworks such as Tensorflow, it's common to share information like coefficients as part of the inter-node cluster communications. The algorithms require that exchanged information stay synchronized across nodes. Secure this information through encryption in transit.

Implementation plan

- **Enable inter-node encryption in SageMaker** — Distributed computing environments can be used for training, testing, and analysis. Some data can be transmitted between nodes that traverse wide networks, or even the Internet. Enable inter-node encryption through the appropriate controls for the technology choices made. You can instruct SageMaker to automatically encrypt inter-container communication for your training job. The data that passed between nodes is then passed over an encrypted tunnel without requiring any data encrypting and decrypting.

- **Enable encryption in transit in Amazon EMR** — There are many applications and execution engines in the Hadoop ecosystem, providing a variety of tools to match the needs of your ML and analytics workloads. Amazon EMR has distributed cluster capabilities and is also an option for running training jobs on the data that is either stored locally on the cluster or in Amazon S3. Amazon EMR makes it easy to create and manage fully configured, elastic clusters of Amazon EC2 instances running Hadoop and other applications in the Hadoop ecosystem. Amazon EMR provides security configurations to set up data encryption at rest while stored on Amazon S3 and local Amazon EBS volumes. It also allows the set up of Transport Layer Security (TLS) certificates for the encryption of data in transit.

Documents

- Protect Communications Between ML Compute Instances in a Distributed Training Job
- Amazon EMR Management guide - Security Data Protection & Encryption Options
- Apache Hadoop on Amazon EMR

Blogs

- Building secure machine learning environments with Amazon SageMaker
- Encrypt data in transit using a TLS custom certificate provider with Amazon EMR
- Secure Amazon EMR with Encryption

Examples

- Protect Communications Between ML Compute Instances in a Distributed Training Job
- TF Encrypted

MLSEC-10: Protect against data poisoning threats

Protect against data injection and data manipulation that pollutes the training dataset. Data injections add corrupt training data that will result in incorrect model and outputs. Data manipulations change existing data (for example labels) that can result in inaccurate and weak predictive models. Identify and address corrupt data and inaccurate models using security methods and anomaly detection algorithms.

Implementation plan

- **Use only trusted data sources for training data** — Verify that you have sufficient audit controls to replay activity and determine where a change occurred, by whom, and at what time. Before training, validate the quality of training data to look for strong outliers and potentially incorrect labels.
- **Look for underlying shifts in the patterns and distributions in training data** — Using monitoring of data drift, derive the impact to prediction variance. These skews can be an indicator of underlying data drift, and can provide an early warning of unauthorized access targeting the training data.
- **Identify model updates that negatively impact the results before moving them to production** — Determine if the retrained model results are different from the past model iteration. Use past test data and previous model iterations as a baseline.
- **Have a rollback plan** — Using versioned training data and versioned models, make sure you can revert to a known good working model in a failure scenario.
- **Use low-entropy classification cases** — Look for significant, unexpected changes. Determine the bounds of thresholds, identify classifications that you do not expect to see, and alert if the retrained model exceeds them.

Documents

- Amazon SageMaker Model Monitor - Monitor for Bias Drift in Production Model
- Amazon SageMaker model registry now supports rollback of deployed models
- SageMaker Model Registry Approve

Blogs

- Automated monitoring of your machine learning models with Amazon SageMaker Model Monitor and sending predictions to human review workflows using Amazon A2I
- Amazon SageMaker Model Monitor – Fully Managed Automatic Monitoring For Your Machine Learning Models
- New – Amazon SageMaker Pipelines Brings DevOps Capabilities to your Machine Learning Projects
- 7 ways to improve security of your machine learning workflows

Videos

- AWS re:Invent 2020: Detect machine learning (ML) model drift in production

Examples

- Inawisdom: Machine Learning and Automated Model Retraining with SageMaker
- Amazon SageMaker Workshop - DevOps workflow

Reliability pillar – Best Practices

The reliability pillar encompasses the ability of a workload to perform its intended function correctly and consistently when it's expected to. This section includes best practices to consider during model development that includes model training, tuning, and model performance evaluation.

Best practices

- MLREL-07: Enable CI/CD/CT automation with traceability (p. 54)
- MLREL-08: Ensure feature consistency across training and inference (p. 55)
- MLREL-09: Ensure model validation with relevant data (p. 56)

MLREL-07: Enable CI/CD/CT automation with traceability

Enable source code, data, and artifact version control of ML workloads to enable roll back to a specific version. Incorporate continuous integration (CI), continuous delivery (CD), and continuous training (CT) practices to ML workload operations. This will enable automation with added traceability.

Implementation plan

- **Use Amazon SageMaker Pipelines** — Manual changes to a system can cost additional time and impair reproducibility. Changes to ML workload should be conducted, tracked and rolled back automatically. MLOps is a collection of best practices around integrating and deploying reproducible, auditable changes. MLOps increases your productivity while automating all facets of your ML development cycle (MLDC). Amazon SageMaker Pipelines is the first purpose-built, continuous integration (CI), continuous delivery (CD), and continuous training (CT) service. With SageMaker Pipelines, create, automate, and manage end-to-end ML workflows at scale.

Documents

- AWS MLOps Framework
- SageMaker Pipelines Overview
- Why Should You Use MLOps?
- Continuous Delivery for Machine Learning on AWS
- Practicing CI/CD on AWS

Blogs

- Implementing a continual learning machine learning pipeline with Amazon SageMaker, AWS Glue DataBrew and SAP S/4HANA
- Build a CI/CD pipeline for deploying custom machine learning models using AWS services

Videos

- AWS re:Invent 2020: How to create fully automated ML workflows with Amazon SageMaker Pipelines
- Inawisdom: Machine Learning and Automated Model Retraining with SageMaker

Examples

- Amazon Sagemaker MLops (with classic CI/CD tools) Workshop
- Amazon SageMaker secure MLOps

MLREL-08: Ensure feature consistency across training and inference

Ensure consistent, scalable, and highly available features between training and inference using a feature storage. This results in reducing the training-serving skew by keeping feature consistency between training and inference.

Implementation plan

- **Use Amazon SageMaker Feature Store** — Create, share, and manage features for ML development using SageMaker Feature Store. The Feature Store is a centralized store for features and associated metadata so features can be easily discovered and reused. The online store is used for low latency, real-time inference use cases. The offline store is used for training and batch inference. The Feature Store reduces the repetitive data processing and curation work required to convert raw data into features for training an ML algorithm. Features generated will be used for both training and inference, reducing the training-serving skew. The Feature Store enables feature consistency, feature standardization, and the ability to integrate with Amazon SageMaker Pipelines.

Documents

- Get started with Amazon SageMaker Feature Store

Blogs

- Store, Discover, and Share Machine Learning Features with Amazon SageMaker Feature Store
- Using streaming ingestion with Amazon SageMaker Feature Store to make ML-backed decisions in near-real time

Videos

- AWS re:Invent 2020: Amazon SageMaker Feature Store: Store, discover, & share features for ML apps
- Introducing Amazon SageMaker Feature Store - AWS re:Invent 2020

Examples

- Amazon SageMaker Feature Store: Introduction to Feature Store
- https://github.com/aws/amazon-sagemaker-examples/tree/master/sagemaker-featurestore
- https://github.com/aws/amazon-sagemaker-examples/blob/master/sagemaker-featurestore/feature_store_introduction.ipynb
- https://github.com/aws-samples/amazon-sagemaker-feature-store-streaming-aggregation

MLREL-09: Ensure model validation with relevant data

Put processes in place to include real and representative data for testing and validation. Data that does not include all possible patterns and scenarios will result in failures once model is in production.

Implementation plan

- **Use Amazon SageMaker Experiments** — Your models should be tested and validated using data that is representative of what they will encounter in production. This data can include both real-world data and engineered data. You should account for all scenarios in your training data so that you can avoid errors when your model is deployed to production. Use Amazon SageMaker Experiments to organize, track, compare, and evaluate your machine learning experiments.
- **Use Amazon SageMaker Model Monitor** — Consider implementing a plan to periodically test endpoints for deviations in model quality. Early detection of deviations can help you determine when to take corrective actions. SageMaker Model Monitor continually monitors the quality of Amazon SageMaker ML models in production. With Model Monitor, you can set alerts that notify you when there are deviations in the model quality.

Documents

- Manage Machine Learning with Amazon SageMaker Experiments
- Evaluating ML Models
- Cross-Validation of Machine Learning Models
- Amazon SageMaker Model Monitor

Blogs

- Build an automatic data profiling and reporting solution with Amazon EMR, AWS Glue, and Amazon QuickSight
- Test data quality at scale with Deequ
- Amazon SageMaker Model Monitor – Fully Managed Automatic Monitoring For Your Machine Learning Models

Examples

- Amazon SageMaker Model Monitor

Performance Efficiency pillar – Best Practices

The performance efficiency pillar focuses on the efficient use of computing resources to meet requirements and the maintenance of that efficiency as demand changes and technologies evolve. This section includes best practices to consider during model development that includes model training, tuning and model performance evaluation.

Best practices

- MLPER-06: Optimize training and inference instance types (p. 57)
- MLPER-07: Explore alternatives for performance improvement (p. 58)
- MLPER-08: Establish a model performance evaluation pipeline (p. 59)
- MLPER-09: Establish feature statistics (p. 59)
- MLPER-10: Perform a performance trade-off analysis (p. 60)

MLPER-06: Optimize training and inference instance types

Determine how the model type and data velocity affect the choice of training and inference instance types. Identify the right instance type that supports memory intensive training, or compute intensive with high throughput and low latency real-time inference. The speed of model inferences is directly impacted by model complexity. Selection of high compute instances can accelerate inference speed. GPUs are often the preferred processor type to train many deep learning models. CPUs are often sufficient for the inference workloads.

Implementation plan

- **Experiment with alternative instance types to train and deploy** — Use Amazon SageMaker Experiments to organize, track, compare, and evaluate your machine learning experiments. Learn which instance types are most appropriate for the ML algorithm and the use case. In Amazon SageMaker, start with the default or example instance types, and then scale to optimize performance and cost. Use multiple instances for training for large datasets to take advantage of scale.

Documents

- Use Amazon SageMaker Notebook Instances
- Train a Model with Amazon SageMaker
- Deploy a Model on SageMaker Hosting Services
- Distributed training libraries
- Deploy Models for Inference

Blogs

- Learn how to select ML instances on the fly in Amazon SageMaker Studio
- Optimizing costs for machine learning with Amazon SageMaker
- Optimizing I/O for GPU performance tuning of deep learning training in Amazon SageMaker
- Right-sizing resources and avoiding unnecessary costs in Amazon SageMaker

Videos

- How to choose the right instance type for ML inference
- The right instance type in Amazon SageMaker

Examples

- Amazon SageMaker Examples – end_to_end

MLPER-07: Explore alternatives for performance improvement

Perform benchmarking to improve the machine learning model performance. Benchmarking in ML involves evaluation and comparison of ML workloads with different algorithms, features, and architecture resources. It enables identifying the combination with optimal performance.

Options you can use when benchmarking include:

- Use more data to broaden the statistical range and hone the precision of the model.
- Apply feature engineering to extract important signals in the data for the model.
- Make alternative algorithm selections for an optimal fit to the specifics of the data.
- Ensemble methods that combine the different advantages of multiple models.
- Tune the hyperparameters for a given algorithm to calibrate the model for the data.

Implementation plan

- **Use Amazon SageMaker Experiments to optimize algorithms and features** — Begin with a simple architecture, obvious features, and a simple algorithm to establish a baseline. Amazon SageMaker provides built-in algorithms for developing a baseline model. Use Amazon SageMaker Experiments to organize, track, compare, and evaluate your machine learning experiments. Test different algorithms with increasing complexity to observe whether performance is improved. Combine models into an ensemble to increase accuracy, but consider the potential loss of efficiency as a trade-off. Refine the features by selection and modify parameters to optimize model performance. Tune the model's hyperparameters to optimize performance using Amazon SageMaker Hyperparameter Optimization to automate the search.

Documents

- Use Amazon SageMaker Built-in Algorithms
- Improving Model Accuracy
- Evaluating ML Models
- Feature Processing with Spark ML and Scikit-learn
- Perform Automatic Model Tuning with SageMaker
- Amazon SageMaker Experiments: Track and Compare Tutorial

Blogs

- Running multiple HPO jobs in parallel on Amazon SageMaker
- Optimizing portfolio value with Amazon SageMaker automatic model tuning
- Utilizing XGBoost training reports to improve your models

Videos

- Tune your ML models to the highest accuracy with automatic model tuning
- Organize, Track, and Evaluate ML Training Runs With Amazon SageMaker Experiments

Examples

- Feature Engineering Immersion Day Workshop
- Improving Forecast Accuracy with Machine Learning
- Ensemble Predictions From Multiple Models

MLPER-08: Establish a model performance evaluation pipeline

Capture key metrics related to model performance using an end-to-end performance pipeline to evaluate the success of a model. Choose specific metrics based on the use case and the business KPIs. Sample key metrics include training or validation errors, and prediction accuracy. Specific model performance metrics include RMSE, accuracy, precision, recall, F1 score, and AUROC. Establish a fully automated performance testing pipeline system to initiate evaluation every time there is an updated model or data.

Implementation plan

- **Create an end-to-end workflow with Amazon SageMaker Pipelines** — Start with a workflow template to establish an initial infrastructure for model training and deployment. SageMaker Pipelines helps you automate different steps of the ML workflow. These steps include data loading, data transformation, training, tuning, and deployment. With SageMaker Pipelines, you can share and reuse workflows to recreate or optimize models, helping you scale ML throughout your organization. Within SageMaker Pipelines, the SageMaker Model Registry tracks the model versions and respective artifacts. These artifacts include the metadata and lineage data collected throughout the model development lifecycle. SageMaker Model Registry can also enable automating model deployment with CI/CD.

Documents

- Define a Pipeline
- Register and Deploy Models with Model Registry

Blogs

- Building, automating, managing, and scaling ML workflows using Amazon SageMaker Pipelines
- Extend Amazon SageMaker Pipelines to include custom steps using callback steps

Videos

- Introducing Amazon SageMaker Pipelines
- How to create fully automated ML workflows with Amazon SageMaker Pipelines

Examples

- SageMaker Pipelines – Immersion day

MLPER-09: Establish feature statistics

Establish key statistics to measure changes in the data that affect model outcomes. The effect of data changes on model inferences depends on the sensitivity of the model to data features. Analyze the feature importance and sensitivity of the model to select the features to monitor. Monitor the statistics of features that most strongly affect inferences. Place acceptability limits on the range of data to alert when important features drift outside the statistical range of the training data. Significant drifts in important features would suggest model re-training.

Implementation plan

- **Analyze and evaluate data** — Use Amazon SageMaker Data Wrangler to analyze the distribution of the selected features. After training the model, map out the regions in feature space where the predictions change abruptly and where the predictions are invariant. Establish a baseline for monitoring the data with Amazon SageMaker Model Monitor. Perform a sensitivity analysis of changes in the feature values near the decision boundaries of the model. Analyze the feature importance to understand how new data will affect the model's predictions. Amazon SageMaker Experiments will help to organize model testing. Use Amazon SageMaker Clarify to check for data biases and imbalances. Monitor the statistics of data used in production inferences. Consider retraining the model if the features are outside the original distribution of the training data.

Documents

- Amazon SageMaker Data Wrangler: Analyze and Visualize
- Amazon SageMaker Model Monitor
- Detect Pretraining Data Bias
- Amazon SageMaker Clarify: Detect bias in ML models and understand model predictions

Blogs

- Exploratory data analysis, feature engineering, and operationalizing your data flow into your ML pipeline with Amazon SageMaker Data Wrangler
- Develop and deploy ML models using Amazon SageMaker Data Wrangler and Amazon SageMaker Autopilot
- Amazon SageMaker Model Monitor – Fully Managed Automatic Monitoring For Your Machine Learning Models
- How Clarify helps machine learning developers detect unintended bias

Videos

- Prepare data for machine learning with ease, speed, and accuracy
- Detect machine learning (ML) model drift in production
- Accelerate data preparation with Amazon SageMaker Data Wrangler

Examples

- Feature Engineering, Immersion Day Workshop

MLPER-10: Perform a performance trade-off analysis

Perform alternative trade-off analysis to obtain optimal performance and accuracy for a given use-case data and business requirement.

- **Accuracy versus complexity trade-off**: The simpler a machine learning model is, the more explainable are its predictions. Deep learning predictions can potentially outperform linear regression or a decision tree algorithm, but at the cost of added complexity in interpretability and explainability.
- **Bias versus fairness trade-off**: Define a process for managing risks of bias and fairness in model performance. Business value most often aligns with models that have considered historical or

sampling biases in the training data. Further consideration should be given to the disparate impact of inaccurate model predictions. For example, underrepresented groups are often more impacted by historical biases, which might perpetuate unfair practices.

- **Bias versus variance trade-off (supervised ML):** The goal is to achieve a trained model with the lowest bias versus variance tradeoff for a given dataset. To help overcome bias and variance errors, you can use:
  - Cross validation
  - Use of more data
  - Regularization
  - Dimension reduction (Principal Component Analysis)
  - Stop training early
- **Precision versus recall trade-off (supervised ML):** This analysis can be important when precision is more important than recall or vice versa. For example, optimization of precision is more important when the goal is to reduce false positives. However, optimization of recall is more important when the goal is to reduce false negatives. It's not possible to have both high precision and high recall—if one is increased, the other decreases. A trade-off analysis helps identify the optimal option for analysis.

Implementation plan

- **Construct alternate workflows to optimize all aspects of business value** — Complex models might deliver high accuracy. They also can be slower to return an inference and more difficult to deploy at the edge. If the business requirements include low-latency, then the model might need to be simplified with lower complexity. Identify how trade-offs affect accuracy and the latency of inferences. Test these trade-offs using Amazon SageMaker Experiments to keep track of each model type. Amazon SageMaker Clarify provides explanations of the data, models, and monitoring used to assess predictions. It can measure biases during each stage of the ML lifecycle. Provided explanations will help understanding how fairness affects the business use case. It can also evaluate if the accuracy for full population is valued more than optimal accuracy for each subgroup.

Documents

- Evaluating ML Models
- AI Fairness and Explainability Whitepaper

Blogs

- Amazon SageMaker Experiments – Organize, Track And Compare Your Machine Learning Trainings
- Amazon SageMaker Clarify Detects Bias and Increases the Transparency of Machine Learning Models

Videos

- Machine learning and society: Bias, fairness, and explainability

Cost Optimization pillar – Best Practices

The cost optimization pillar includes the continual process of refinement and improvement of a system over its entire lifecycle. From the initial design of your very first proof of concept to the ongoing operation of production workloads, adopting the practices in this document can enable you to build and operate cost-aware systems that achieve business outcomes and minimize costs, thus allowing your business to maximize its return on investment. This section includes best practices to consider while developing models that includes training, tuning, and model performance evaluation.

Best practices

- MLCOST-12: Select optimal computing instance size (p. 62)
- MLCOST-13: Select local training for small scale experiments (p. 62)
- MLCOST-14: Select an optimal ML framework (p. 63)
- MLCOST-15: Use automated machine learning (p. 64)
- MLCOST-16: Use distributed training (p. 64)
- MLCOST-17: Stop resources when not in use (p. 65)
- MLCOST-18: Start training with small datasets (p. 66)
- MLCOST-19: Use warm-start and checkpointing hyperparameter tuning (p. 67)
- MLCOST-20: Use hyperparameter optimization technologies (p. 68)

MLCOST-12: Select optimal computing instance size

Right size the training instances according to the ML algorithm used for maximum efficiency and cost reduction. Use debugging capabilities to understand the right resources to use during training. Simple models might not train faster on larger instances because they might not be able to benefit from increased parallelism. These models might even train slower due to the high GPU communication overhead. Start with smaller instances and scale as necessary.

Implementation plan

- **Use Amazon SageMaker Experiments** — Amazon EC2 provides a wide selection of instance types optimized to fit different use cases. Machine learning workloads can use either a CPU or a GPU instance. Select an instance type from the available EC2 instance types depending on the needs of your ML algorithm. Experiment with both CPU and GPU instances to learn which one gives you the best cost configuration. Amazon SageMaker lets you use a single instance or a distributed cluster of GPU instances. Use Amazon SageMaker Experiments to evaluate alternative options, and identify the size resulting in optimal outcome. With the pricing broken down by time and resources, you can optimize the cost of Amazon SageMaker and only pay for what is needed.

Blogs

- Right-sizing resources and avoiding unnecessary costs in Amazon SageMaker
- Save on inference costs by using Amazon SageMaker multi-model endpoints

Videos

- AWS re:Invent 2019: The right instance type in Amazon SageMaker, ft. Texas Instruments
- AWS re:Invent 2020: How to choose the right instance type for ML inference

Examples

- Amazon SageMaker Multi-Model Endpoints using XGBoost

MLCOST-13: Select local training for small scale experiments

Evaluate the requirements to train an ML model in the cloud versus on a local machine. Use local option when experimenting across different algorithms and configurations with small data sizes. For large data, launch a cloud-based training cluster with one or more compute instances. Right size the compute instances in the training cluster based on the workload.

Implementation plan

- **Use Amazon SageMaker** — While experimenting with training a model with small datasets, use Amazon SageMaker notebook local mode. This will train your model on the notebook instance itself, instead of on a separate managed training cluster. You can iterate and test your work without having to wait for a new training or hosting cluster each time. This saves both time and cost associated with creating a managed training cluster.

  During training, Amazon SageMaker retrieves the container you specify from Amazon ECR and prepares it to run on a training instance. For smaller data sets, Amazon SageMaker supports file mode for training, which downloads the training data from the S3 bucket to the EBS volume attached to the training instance. This allows the algorithm to read its training data from the local file system without integrating directly with Amazon S3. By using containers and copying objects from Amazon S3, Amazon SageMaker enables network isolation of your algorithms and models during training and hosting.

  Experimentation can also occur outside of a notebook, for example on a local machine. From your local machine, you can use the SageMaker SDK to train and deploy models on AWS.

Blogs

- Use the Amazon SageMaker local mode to train on your notebook instance

Videos

- Train with Amazon SageMaker on your local machine

Examples

- Multiple examples showing how to run SageMaker in local mode

MLCOST-14: Select an optimal ML framework

Organize, track, compare and evaluate machine learning (ML) experiments and model versions. Identify the most cost-effective and optimal combination of instance types and ML frameworks. Examples of ML frameworks include: Tensorflow, PyTorch, and Scikit-learn.

Implementation plan

- **Use Amazon SageMaker Experiments** — Amazon SageMaker Experiments lets you organize, track, compare, and evaluate your machine learning experiments. Using this service, you can try out various ML frameworks and see which one gives you the most cost-effective performance. AWS Deep Learning AMIs and AWS Deep Learning Containers enable you to use several open-source ML frameworks for training on your infrastructure. AWS Deep Learning AMIs have popular deep learning frameworks and interfaces preinstalled including TensorFlow, PyTorch, Apache MXNet, Chainer, Gluon, Horovod, and Keras. The AMI or container can be launched on powerful infrastructure that has been optimized for ML performance. SageMaker also allows you to bring your own container where you can use any framework you choose.

Documents

- Manage Machine Learning with Amazon SageMaker Experiments
- Use Machine Learning Frameworks, Python, and R with Amazon SageMaker

Blogs

- Right-sizing resources and avoiding unnecessary costs in Amazon SageMaker
- Amazon SageMaker Experiments – Organize, Track And Compare Your Machine Learning Trainings

MLCOST-15: Use automated machine learning

Use automated data analyzer systems when building a model. These systems experiment with and select the best algorithm from the list of high-performing algorithms. They automatically test different solutions and parameter settings to achieve optimal models. The automated system speeds up the process, while eliminating manual experimentation and comparisons.

Implementation plan

- **Use Amazon SageMaker Autopilot** — Amazon SageMaker Autopilot automates key tasks of an automatic machine learning (AutoML) process. Autopilot explores your data, selects the algorithms relevant to your problem type, and prepares the data to facilitate model training and tuning. Autopilot applies a cross-validation resampling procedure automatically to all candidate algorithms. It tests the ability of these algorithms to predict using data they have not been trained on. Autopilot simplifies your machine learning experience by automating the key tasks that constitute an AutoML process. It ranks all of the optimized models tested by their performance. Autopilot finds the best performing model that you can deploy at a fraction of the time normally required.

Documents

- Automate model development with Amazon SageMaker Autopilot

Blogs

- Amazon SageMaker Autopilot – Automatically Create High-Quality Machine Learning Models With Full Control And Visibility
- Customizing and reusing models generated by Amazon SageMaker Autopilot

Videos

- Automatically Build, Train, and Tune ML Models With Amazon SageMaker Autopilot
- Use Autopilot to automate and explore the machine learning process

Examples

- Customer Churn Prediction with Amazon SageMaker Autopilot
- SageMaker Autopilot

MLCOST-16: Use distributed training

Enable distributed training for a faster training time, when an algorithm allows it. Use multiple instances in a training cluster. Use managed services to help ensure all training instances are automatically shut down when training is completed.

Implementation plan

- **Use Amazon SageMaker** — The distributed data parallel library in Amazon SageMaker (the library) can be used to training large deep learning models that were previously difficult to train due to GPU memory limitations. The library automatically and efficiently splits a model across multiple GPUs and instances. It coordinates model training, allowing you to increase prediction accuracy by creating larger models with more parameters.
  - The library extends the SageMaker training capabilities on deep learning models with near-linear scaling efficiency. This enables achieving fast time-to-train with minimal code changes.
  - The library optimizes your training job for AWS network and Amazon EC2 instance topology.
  - The library takes advantage of gradient updates to communicate between nodes with a custom allreduce algorithm.
  - Increasing deep learning model size (layers and parameters) can result in better accuracy, however there is a limit to the maximum model size fitting a single GPU. When training deep learning models, GPU memory limitations can be a bottleneck in the following ways:
    - You can limit the size of the model you train. Given that larger models tend to achieve higher accuracy, this directly translates to trained model accuracy.
    - You can limit the batch size you train with, leading to lower GPU utilization and slower training.
  - To overcome the limitations associated with training a model on a single GPU, you can use model parallelism to distribute and train your model on multiple computing devices.

Documents

- SageMaker's Distributed Data Parallel Library
- SageMaker's Distributed Model Parallel
- Distributed Training

Blogs

- New – Data Parallelism Library in Amazon SageMaker Simplifies Training on Large Datasets
- How Latent Space used the Amazon SageMaker model parallelism library to push the frontiers of large-scale transformers
- The science behind Amazon SageMaker’s distributed-training engines

Videos

- AWS re:Invent 2020: Train billion-parameter models with model parallelism on Amazon SageMaker
- AWS re:Invent 2020: Fast training and near-linear scaling with DataParallel in Amazon SageMaker

Examples

- Distributed Training
- Distributed training using Amazon SageMaker Distributed Data Parallel library and debugging using Amazon SageMaker Debugger
- SageMaker developer guide on distributed training

MLCOST-17: Stop resources when not in use

Stop resources that are not in use to reduce cost. For example, hosted Jupyter environments used to explore small samples of data, can be stopped when not actively in use. Where practical, commit the work, stop them, and restart when needed. The same approach can be used to stop the computing and the data storage services.

Implementation plan

- **Use CloudWatch Billing Alarms** — You can monitor your estimated AWS charges by using Amazon CloudWatch. When you enable the monitoring of estimated charges for your AWS account, the estimated charges are calculated and sent several times daily to CloudWatch as metric data. Use this feature to receive notifications when your resource charge exceeds a threshold amount.
- **Use SageMaker Lifecycle Configuration** — A lifecycle configuration provides shell scripts that run when you create the notebook instance or whenever you start one. When you create a notebook instance, you can create a new lifecycle configuration and its scripts or apply ones that you already have. Use a lifecycle configuration script to access AWS services from your notebook. These scripts enable checking notebook instance activities and shut them down if idle.

- **Use Amazon SageMaker Studio auto shutdown** — Amazon SageMaker Studio provides a unified, web-based visual interface for performing all ML development steps, making data science teams more productive. Idle SageMaker Studio notebooks can be detected and stopped using an auto-shutdown JupyterLab extension that can be installed manually or automatically. You can shut down individual resources, including notebooks, terminals, kernels, applications, and instances.

Documents

- Shut Down Resources
- Lifecyle configuration scripts in SageMaker
- CloudWatch Billing Alarms

Blogs

- Save costs by automatically shutting down idle resources within Amazon SageMaker Studio

Videos

- Fully-Managed Notebook Instances with Amazon SageMaker

Examples

- sagemaker-studio-auto-shutdown-extension

MLCOST-18: Start training with small datasets

Start experimentation with smaller datasets on a small compute instance (or local system). This will enable iterating quickly at low cost. After the experimentation period, scale up to train with the full dataset available on a separate compute cluster. Choose the appropriate storage layer for training data based on the performance requirements.

Implementation plan

- **Use SageMaker notebooks** — Notebooks are a popular way to explore and experiment with data in small quantities. Iterating with a small sample of the dataset locally and then scaling to train on the full dataset in a distributed manner is common in machine learning. In AWS, Amazon SageMaker notebook instances provide a hosted Jupyter environment that can be used to explore small samples of data. Stop the notebook instances when you are not actively using them. Where practical, commit your work, stop them, and restart them when you need them again. Storage is persisted and you can use lifecycle configuration to automate package installation or repository synchronization.

Documents

- Customize a Notebook Instance Using a Lifecycle Configuration Script
- Use Amazon SageMaker Notebook Instances
- Start Notebook Instance
- Stop Notebook Instance

MLCOST-19: Use warm-start and checkpointing hyperparameter tuning

Where feasible, use warm start hyperparameter tuning. Warm start can consist of using a parent job for a model trained previously or using transfer learning. Warm start of hyperparameter tuning jobs eliminates the need to start a tuning job from scratch. Create a new hyperparameter tuning job that is based on selected parent jobs or pre-trained models. Use checkpointing capabilities to restart a training job from the last saved checkpoint. Reuse previous trainings as prior knowledge, or use checkpointing to accelerate the tuning process and reduce the cost.

Implementation plan

- **Use warm-start hyperparameter tuning** — Use warm start to start a hyperparameter tuning job using one or more previous tuning jobs as a starting point. The results of previous tuning jobs are used to inform which combinations of hyperparameters to search over in the new tuning job. Hyperparameter tuning uses Bayesian or random search to choose combinations of hyperparameter values from ranges that you specify.
- **Use checkpointing hyperparameter tuning** — Use checkpoints in Amazon SageMaker to save the state of ML models during training. Checkpoints are snapshots of the model and can be configured by the callback functions of ML frameworks. You can use the saved checkpoints to restart a training job from the last saved checkpoint.

Documents

- Run a Warm Start Hyperparameter Tuning Job
- Use checkpoints in Amazon SageMaker

Blogs

- Amazon SageMaker Automatic Model Tuning becomes more efficient with warm start of hyperparameter tuning jobs
- Running multiple HPO jobs in parallel on Amazon SageMaker

Videos

- Tune Your ML Models to the Highest Accuracy with Amazon SageMaker Automatic Model Tuning

Examples

- Automatic Model Tuning : Warm Starting Tuning Jobs

MLCOST-20: Use hyperparameter optimization technologies

Use automatic hyperparameter tuning to run many training jobs and find the best version of your model. Use the algorithm and ranges of hyperparameters that you specify. Use appropriate hyperparameter ranges, as well as metrics that are realistic and meet the business requirements.

Implementation plan

- **Use SageMaker automatic model tuning** — SageMaker automatic model tuning, also known as hyperparameter tuning, finds the optimal model by running many training jobs on your dataset. It uses the algorithm and ranges of hyperparameters that you specify. It then chooses the hyperparameter values that result in a model that performs the best, as measured by a metric that you choose. To create a new hyperparameter optimization (HPO) tuning job for one or more algorithms, you need to define the settings for the tuning job. Create training job definitions for each algorithm being tuned, and configure the resources for the tuning job.

Documents

- Perform automatic model tuning with SageMaker
- Create a HPO tuning job

Blogs

- Running multiple HPO jobs in parallel on SageMaker

Videos

- Automatic model tuning with SageMaker

Examples

- Large HPO polling examples

ML lifecycle phase - Deployment

After you have trained, tuned, and evaluated your model, then you can deploy the model into production. You can make predictions and inferences against the deployed model. Use a manual governance process. This ensures the model has fully been tested and evaluated before it's released to production.

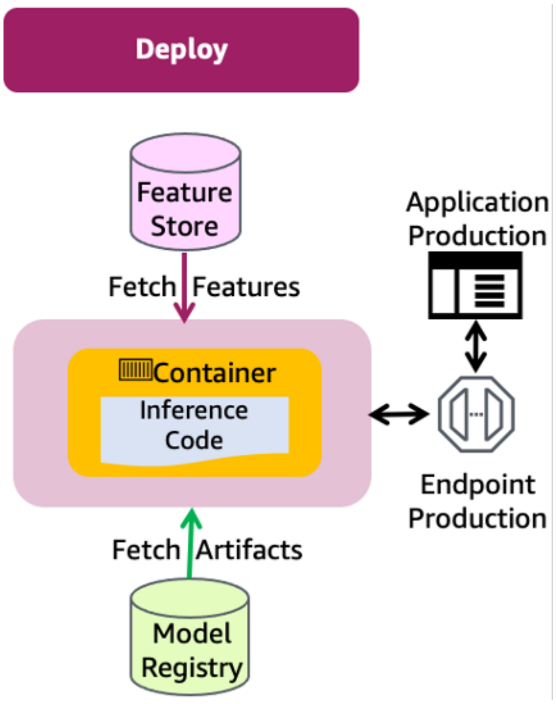

Figure 15 – Deployment architecture diagram

Figure 15 illustrates the deploy production phase of the ML lifecycle. Features are retrieved from the feature store, and artifacts from the model registry, and the inference code container from the container repository.

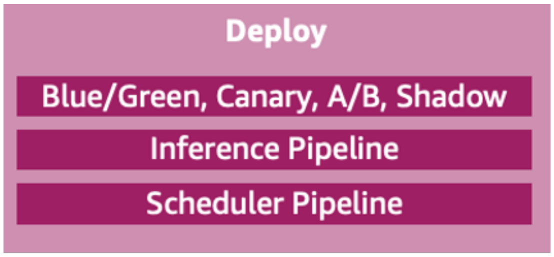

Figure 16: Deployment main components

Figure 16 lists key components of production deployment including:

- **Blue/green, canary, A/B, shadow deployment/testing** — Deployment and testing strategies that reduce downtime and risks when releasing a new or updated version.
  - The **blue/green** deployment technique provides two identical production environments. You can use this technique when you need to deploy a new version of the model to production. Once testing is done on the green environment, live application traffic is directed to the green environment. Then the blue environment is maintained as backup.
  - With a **canary** deployment, first deploy the new release to a small group of users. Other users continue to use the previous version until you're satisfied with the new release. Then, you can gradually roll the new release out to all users.
  - **A/B** testing strategy enables deploying changes to a model. Direct a defined portion of traffic to the new model. Direct the remaining traffic to the old model. A/B testing is similar to canary testing, but has larger user groups and a longer time scale, typically days or even weeks.

- With **shadow** deployment strategy, the new version is available alongside the old version. The input data is run through both versions. The older version is used for servicing the production and the new one is used for testing and analysis.
- **Inference pipeline** — Figure 17 shows the inference pipeline that automates capturing of the prepared data, performing predictions and post-processing for real-time or batch inferences.

- **Scheduler pipeline** — Deployed model is representative of the latest data patterns. Re-training at intervals can minimize the risk of data and concept drifts. A scheduler can initiate a re-training at business defined intervals. Data prepare, CI/CD/CT, and feature pipelines will also be active during this process. This is shown in Figure 17.

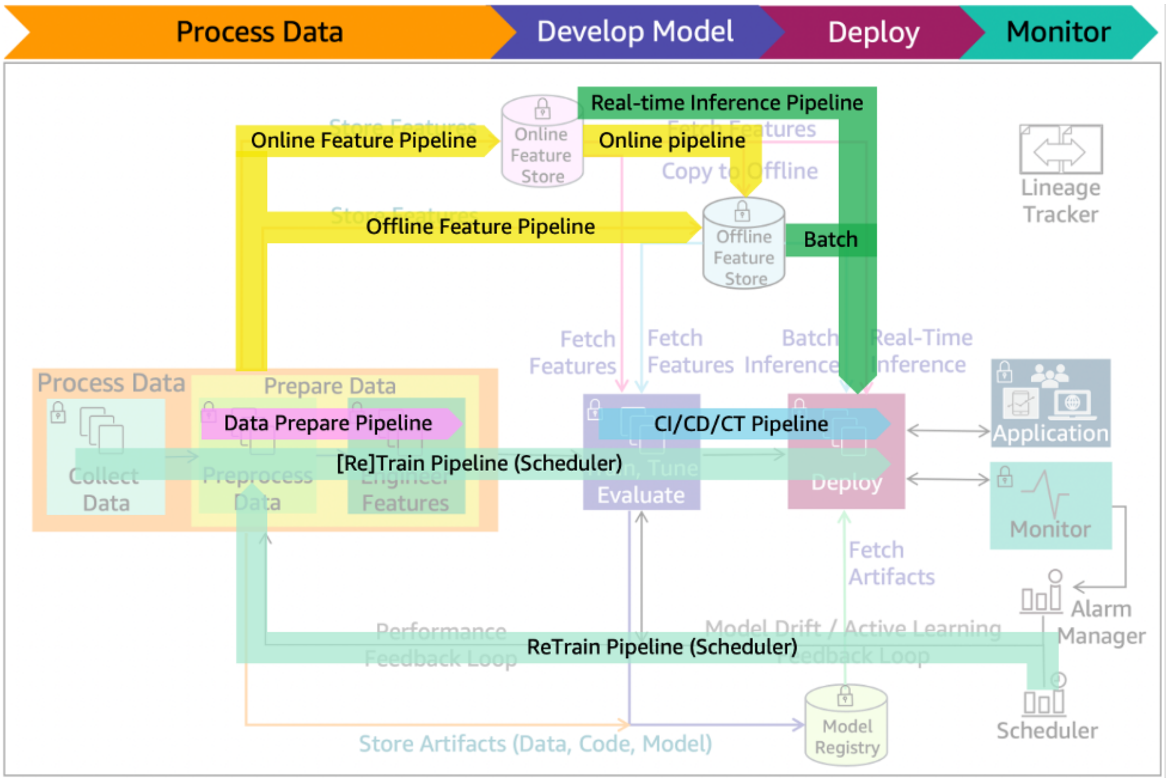

Figure 17: ML lifecycle with scheduler re-train, and batch/real-time Inference pipelines

Topics

- Operational Excellence pillar – Best Practices (p. 70)
- Security pillar – Best Practices (p. 72)
- Reliability pillar – Best Practices (p. 73)
- Performance Efficiency pillar – Best Practices (p. 75)
- Cost Optimization pillar – Best Practices (p. 76)

Operational Excellence pillar – Best Practices

The operational excellence pillar includes the ability to run and monitor systems to deliver business value and to continually improve supporting processes and procedures. This section includes best practices to consider while deploying a model in production.

Best practices

- MLOE-11: Establish deployment environment metrics (p. 71)

MLOE-11: Establish deployment environment metrics

Measure machine learning operations metrics to determine the performance of a deployed environment. These metrics include memory and CPU/GPU usage, disk utilization, ML endpoint invocations, and latency.

Implementation plan

- **Record performance-related metrics** — Use a monitoring and observability service to record performance-related metrics. These metrics can include database transactions, slow queries, I/O latency, HTTP request throughput, service latency, and other key data.
- **Analyze metrics when events or incidents occur** — Use monitoring dashboards and reports to understand and diagnose the impact of an event or incident. These views provide insight into what portions of the workload are not performing as expected.
- **Establish key performance indicators (KPIs) to measure workload performance** — Identify the KPIs that indicate whether the workload is performing as intended. An API-based workload might use overall response latency as an indication of overall performance. For example, an ecommerce site might choose to use the number of purchases as its KPI.
- **Use monitoring to generate alarm-based notifications** — Monitor metrics for the defined key performance indicators (KPIs) and generate alarms automatically when the measurements are outside expected boundaries.
- **Review metrics at regular intervals** — As routine maintenance, or in response to events or incidents, review what metrics are collected. Use these reviews to identify the metrics that were key in addressing issues. Identify any additional tracked metrics that would help to identify, address, or prevent issues.
- **Monitor and alarm proactively** — Use key performance indicators (KPIs), combined with monitoring and alerting systems, to proactively address performance-related issues. Use alarms to initiate automated actions to remediate issues where possible. Escalate the alarm to those able to respond if an automated response is not possible. Use a system to predict expected KPI values, and alert when thresholds are exceeded. Use a tool that can automatically halt or roll back deployments if KPIs are outside of the expected values.
- **Use Amazon CloudWatch** — Use Amazon CloudWatch metrics for SageMaker to determine the memory, CPU usage, and disk utilization. Set up CloudWatch Dashboards to visualize the environment metrics and establish CloudWatch alarms to initiate a notification via Amazon SNS (Email, SMS, WebHook) to notify on events occurring in the execution environment.
- **Use Amazon EventBridge** — Consider defining an automated workflow using Amazon EventBridge to respond automatically to events. These events can include training job status changes, endpoint status changes, and increasing the compute environment capacity after it crosses a defined threshold (such as CPU or disk utilization).
- **Use AWS Application Cost Profiler** — Use AWS Application Cost Profiler to report the cost per tenant (model/user).

Documents

- DevOps and AWS

Videos

- DevOps at Amazon

Security pillar – Best Practices

The security pillar encompasses the ability to protect data, systems, and assets to take advantage of cloud technologies to improve your security. This section includes best practices to consider for model deployment to production.

Best practices

- MLSEC-11: Protect against adversarial and malicious activities (p. 72)

MLSEC-11: Protect against adversarial and malicious activities

Add protection inside and outside of the deployed code to detect malicious inputs that might result in incorrect predictions. Automatically detect unauthorized changes by examining the inputs in detail. Repair and validate the inputs before they are added back to the pool.

Implementation plan

- **Evaluate the robustness of the algorithm** — Evaluate your use case and determine bad predictions or classifications. Evaluate the robustness of the algorithm against increasingly perturbed inputs to understand susceptibility to manipulated inputs.
- **Build for robustness from the start** — Select diverse features to improve the algorithm's ability to handle outliers. Consider pairing models in an ensemble for increased diversity in decisions. Consider using ensembles to build in robustness around decision points.
- **Identify repeats** — Detect similar repeated inputs to the model to indicate possible threats to the decision boundaries. This can take the form of model brute forcing, where threats iterate only a limited set of variables to determine what influences decision points and derive feature importance.
- **Lineage tracking** — If retraining on untrusted or invalidated inputs, make sure any model skew is traced back to the data and pruned before retraining a replacement model.
- **Use secure inference API endpoints** — Host the model so that a consumer of the model can perform inference against it securely. Enable consumers using the API to define the relationship, restrict access to the base model, and provide monitoring of model interactions.

Documents

- Deep ensembles
- Empirical demonstration of deterministic overconfidence
- Making Machine Learning Robust Against Adversarial Inputs

Blogs

- 7 ways to improve security of your machine learning workflows

Videos

- Security and Privacy of Machine Learning

Examples

- Evasion Attacks against Banking Fraud Detection Systems
- SecML - Evasion Attacks against Machine Learning
- Adversarial Robustness Libraries

- SecML: A library for Secure and Explainable Machine Learning

Reliability pillar – Best Practices

The reliability pillar encompasses the ability of a workload to perform its intended function correctly and consistently when it's expected to. This section includes best practices to consider for model deployment to production.

Best practices

- MLREL-10: Automate endpoint changes through a pipeline (p. 73)
- MLREL-11: Use an appropriate deployment and testing strategy (p. 73)

MLREL-10: Automate endpoint changes through a pipeline

Automate changes to a model endpoint through a pipeline that integrates with a change management tracking system. Automation reduces overhead and manual intervention. Versioned pipeline inputs and artifacts allow you to track the changes and automatically rollback after a failed change.

Implementation plan

- **Use Amazon SageMaker Pipelines** — Manual change management is error prone, and incurs a high manpower cost. Use automated pipelines to deploy changes to your model endpoints. Deploying changes through a pipeline is a safe engineering method that enables consistency. Amazon SageMaker Pipelines is the purpose-built, easy-to-use continuous integration and continuous delivery (CI/CD) service for machine learning. With SageMaker Pipelines, you can create, automate, and manage end-to-end ML workflows at scale.

Documents

- SageMaker Pipelines Overview
- What is a SageMaker Project?

Blogs

- Build a CI/CD pipeline for deploying custom machine learning models using AWS services
- Building, automating, managing, and scaling ML workflows using Amazon SageMaker Pipelines

Videos

- AWS re:Invent 2020: How to create fully automated ML workflows with Amazon SageMaker Pipelines
- AWS on Air 2020: AWS What's Next ft. Amazon SageMaker Pipelines

Examples

- Comparing model metrics with SageMaker Pipelines and SageMaker Model Registry

MLREL-11: Use an appropriate deployment and testing strategy

Run a tradeoff analysis across available and relevant deployment/testing strategies and select the one that meets your business requirements. Examples of deployment/testing strategies include blue/green, canary, shadow, and A/B testing. Implement metrics that evaluate model performance to identify when a rollback or rollforward is required. When architecting for rollback or rollforward, evaluate the following for each model:

- Where is the model artifact stored?
- Are model artifacts versioned?
- What changes are included in each version?
- For a deployed endpoint, what version of the model is deployed?

Implementation plan

- **Blue/green deployments** make use of a primary and secondary set of production environment. The blue environment is the existing infrastructure. New changes are deployed to a brand-new infrastructure stack, called the green stack. Once the deployment is complete, traffic is redirected from the blue stack to the green stack. For more details, review the Blue/Green Deployments on AWS whitepaper.
- In a **canary deployment**, new changes are rolled out in a limited fashion to the existing production environment. The purpose of a canary deployment is to reduce the risk of deploying a new version that impacts the workload. The method incrementally deploys the new version, making it visible to new users in a slow fashion. As you gain confidence in the deployment, you can deploy it to replace the current version in its entirety. For more details, review performing a canary-based deployment.
- **A/B testing** is when you deploy a newer model to an endpoint alongside an existing model. You split the traffic between these two versions for an evaluation period. You then measure the performance of the new model against the existing model and, if it performs as expected, you then fully deploy the new model gradually by increasing traffic to it. For more details, review test models in production in the SageMaker documentation.
- A **shadow deployment** is where you release a new software version alongside an existing software version, then shift the traffic to the new version without impacting your production traffic. You can then evaluate the new version with production data without impacting your users.

Documents

- Amazon SageMaker – Testing models in production – Model A/B test example
- Blue/Green Deployment on AWS
- Perform a canary-based deployment using the blue/green strategy and AWS Lambda

Blogs

- A/B Testing ML models in production using Amazon SageMaker
- Dynamic A/B testing for machine learning models with Amazon SageMaker MLOps projects
- Deploy shadow ML models in Amazon SageMaker
- Safely deploying and monitoring Amazon SageMaker endpoints with AWS CodePipeline and AWS CodeDeploy

Videos

- AWS re:Invent 2020: Canaries in the code mines: Monitoring deployment pipelines

Examples

- Amazon SageMaker A/B Testing Pipeline

- Amazon SageMaker Safe Deployment Pipeline
- ML Model Shadow Deployment Strategy on AWS

Performance Efficiency pillar – Best Practices

The performance efficiency pillar focuses on the efficient use of computing resources to meet requirements and the maintenance of that efficiency as demand changes and technologies evolve. This section includes best practices to consider during model deployment to production.

Best practices

- MLPER-11: Evaluate machine learning deployment option (cloud versus edge) (p. 75)

MLPER-11: Evaluate machine learning deployment option (cloud versus edge)

Evaluate if machine learning applications require near-instantaneous inference results. Offering the lowest latency possible might require the removal of costly roundtrips to the nearest API endpoints. A reduction in latency can be achieved by running the inference directly on the device itself (on the *edge*). A common use-case for such a requirement is predictive maintenance in factories.

Implementation plan

- **Evaluate cloud or device deployment options** — Training and optimizing machine learning models require massive computing resources, so it is a natural fit for the cloud. Inference takes a lot less computing power and is often done in real time when new data is available. When getting inference results with very low latency, ensure that your IoT applications can respond quickly to local events. Evaluate and choose the option to meet your business requirements.
  - Amazon SageMaker Neo enables ML models to be trained once and then run anywhere in the cloud and at the edge. Amazon SageMaker Neo consists of a compiler and a runtime. The compilation API reads models exported from various frameworks, converts them into framework-agnostic representations, and generates optimized binary code. The runtime for each target platform then loads and runs the compiled model.
  - AWS IoT Greengrass enables ML inferences on edge devices using models trained in the cloud. These models can be built using Amazon SageMaker, AWS Deep Learning AMI, or AWS Deep Learning Containers. These models can be stored in Amazon S3 before deployed on edge devices.

Documents

- AWS IoT Greengrass ML Inference
- Amazon SageMaker Edge Manager
- Getting Started with Neo on Edge Devices

Blogs

- Machine Learning at the Edge: Using and Retraining Image Classification Models with AWS IoT Greengrass
- Monitor and Manage Anomaly Detection Models on a fleet of Wind Turbines with Amazon SageMaker Edge Manager
- SageMaker Edge Manager Simplifies Operating Machine Learning Models on Edge Devices
- Amazon SageMaker Neo Helps Detect Objects and Classify Images on Edge Devices
- Machine Learning at the Edge with AWS Outposts and Amazon SageMaker

Videos

- Machine Learning at the Edge
- Getting Started Using Machine Learning at the Edge
- Train ML Models Once, Run Anywhere in the Cloud & at the Edge with Amazon SageMaker Neo

Examples

- AWS IoT Greengrass flow examples

Cost Optimization pillar – Best Practices

The cost optimization pillar includes the continual process of refinement and improvement of a system over its entire lifecycle. From the initial design of your very first proof of concept to the ongoing operation of production workloads, adopting the practices in this document can enable you to build and operate cost-aware systems that achieve business outcomes and minimize costs, thus allowing your business to maximize its return on investment. This section includes best practices to consider for model deployment to production.

Best practices

- MLCOST-21: Use an inference pipeline (p. 76)

MLCOST-21: Use an inference pipeline

Use an inference pipeline to process requests for inferences on data. An inference pipeline combines preprocessing, predictions, and post-processing on real-time and batch inference requests. Inference pipelines can be composed of any ML framework, built-in algorithm, or custom containers.

Implementation plan

- **Use Amazon SageMaker** — Amazon SageMaker inference pipeline is composed of a linear sequence of two to 15 containers to process inference requests. Use an inference pipeline to define and deploy any combination of pretrained SageMaker built-in algorithms and your own custom algorithms packaged in Docker containers. Use an inference pipeline to run pre-processing, predictions, and post-processing on real-time and batch inferences. Inference pipelines can be composed of any ML framework, built-in algorithm, or custom containers that you can use on Amazon SageMaker. Build data processing pipelines with a suite of feature transformers in the SparkML and Scikit-learn framework containers. Deploy these as part of the inference pipelines to reuse data processing code and simplify management of your ML processes. These inference pipelines are fully managed and can combine preprocessing, predictions, and post-processing as part of a data science process.

Documents

- Deploy an Inference Pipeline
- Run Real-time Predictions with an Inference Pipeline

Blogs

- Preprocess input data before making predictions using Amazon SageMaker inference pipelines and Scikit-learn
- Using Amazon SageMaker inference pipelines with multi-model endpoints

Videos

- AWS Summit Tel Aviv 2019 | Building Machine Learning Inference Pipelines at Scale

Examples

- Inference Pipeline with Scikit-learn and Linear Learner
- Inference Pipeline with Custom Containers and xgBoost

ML lifecycle phase – Monitoring

The model monitoring system must capture data, compare that data to the training set, define rules to detect issues, and send alerts. This process repeats on a defined schedule, when initiated by an event, or when initiated by human intervention. The issues detected in the monitoring phase include: data quality, model quality, bias drift, and feature attribution drift.

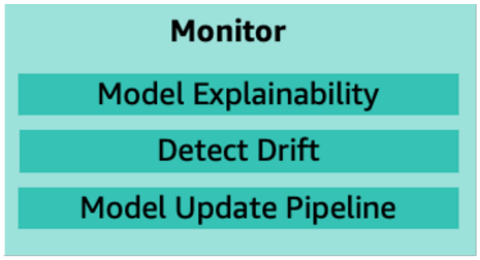

Figure 18: Post deployment monitor main components

Figure 18 lists key components of monitoring including:

- **Model explainability** — Monitoring system uses *explainability* to evaluate the soundness of the model and if the predictions can be trusted.
- **Detect drift** — Monitoring system detects data and concept drifts. Data drift is the significant changes to data distribution compared to data used for training. Concept drift is when the properties of target variables change. Any kind of drift will result in model performance degradation. This will initiate an alert and send it to alarm manager system.
- **Model update pipeline** — If alarm manager identifies any violations, it launches the model update pipeline for a re-train. This can be seen in Figure 19. Data prepare, CI/CD/CT, and feature pipelines will also be active during this process.

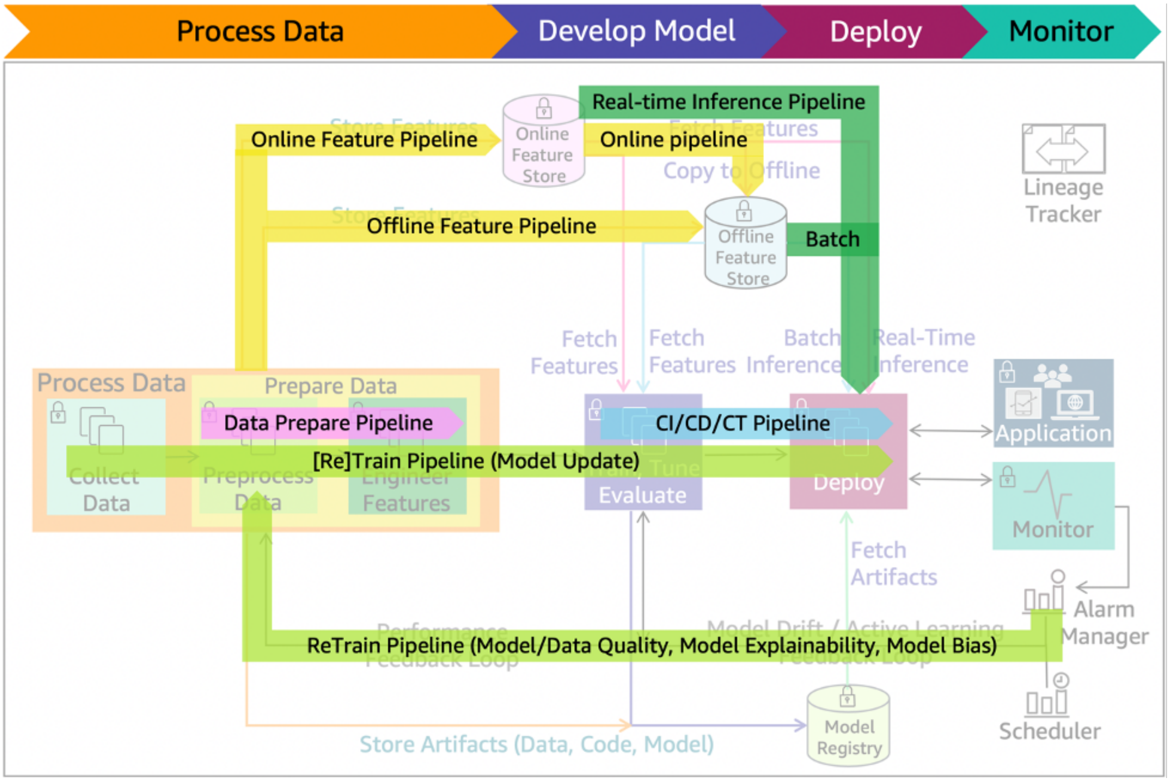

Figure 19: ML lifecycle with model update re-train and batch/real-time Inference pipelines

Topics

- Operational Excellence pillar – Best Practices (p. 78)
- Security pillar – Best Practices (p. 80)
- Reliability pillar – Best Practices (p. 82)
- Performance Efficiency pillar – Best Practices (p. 84)
- Cost Optimization pillar – Best Practices (p. 89)

Operational Excellence pillar – Best Practices

The operational excellence pillar includes the ability to run and monitor systems to deliver business value and to continually improve supporting processes and procedures. This section includes best practices to consider for monitoring models in production.

Best practices

- MLOE-12: Enable monitoring health of model endpoint (p. 78)
- MLOE-13: Synchronize architecture and configuration, and check for skew across environments (p. 79)

MLOE-12: Enable monitoring health of model endpoint

Establish model endpoint monitoring. Identify and react to any potential issues or opportunities for improvement. Monitor metrics that measure the operational health of the underlying compute resources hosting the endpoint and the health of endpoint responses. Ensure traceability of hosting metrics back to versioned inputs for analysis.

Implementation plan

- **Use Amazon SageMaker Model Monitor** — Continually monitor the quality of Amazon SageMaker ML models in production and compare with the results from training using SageMaker Model Monitor.
- **Use Amazon CloudWatch** — Amazon SageMaker Model Monitor automatically sends metrics to Amazon CloudWatch so that you can gather and analyze usage statistics for your ML models.
- **Use Amazon SageMaker Clarify** — Identify various types of bias that can emerge during model training or when the model is in production. This helps improve your ML models by detecting potential bias and helping explain the predictions that the models make. SageMaker Clarify helps explain how these models make predictions using a feature attribution approach. It also monitors inferences that the models make in production for bias or feature attribution drift. SageMaker Clarify provides tools to help you generate model governance reports that you can use to inform risk and compliance teams, and external regulators.

Documents

- Amazon SageMaker Model Monitor
- Monitoring Amazon ML with Amazon CloudWatch Metrics
- How Amazon CloudWatch works
- Clarify, Fairness and Explainability

Blogs

- Architect and build the full machine learning lifecycle with AWS: An end-to-end Amazon SageMaker demo

Videos

- Introducing Amazon SageMaker Clarify, part 1 - Bias detection - AWS re:Invent 2020
- Introducing Amazon SageMaker Clarify, part 2 - Model explainability - AWS re:Invent 2020
- AWS re:Invent 2020: Understand ML model predictions & biases with Amazon SageMaker Clarify

Examples

- Explainability and bias detection with Amazon SageMaker Clarify
- Monitoring bias drift and feature attribution drift Amazon SageMaker Clarify

MLOE-13: Synchronize architecture and configuration, and check for skew across environments

Ensure all systems and configurations are identical across development and deployment phases. Otherwise, the same algorithm can result in different inference results depending on differences in system architectures. Ensure that the model gets the same range of accuracy in development, staging, and production environments. Perform this check as part of the normal promotion process.

Implementation plan

- **Use AWS CloudFormation** — AWS CloudFormation gives you an easy way to model a collection of related AWS and third-party resources. CloudFormation provisions these resources quickly and consistently, and manages them throughout their lifecycles, by treating infrastructure as code. You can use a CloudFormation template to create, update, and delete an entire stack as a single unit, as often as you need to, instead of managing resources individually. You also can manage and provision stacks across multiple AWS accounts and AWS Regions. This will synchronize your architecture and configuration across environments.

- **Use Amazon SageMaker Model Monitor** — Continually monitor the quality of Amazon SageMaker machine learning models in production and compare with the results from training using SageMaker Model Monitor. Set alerts that notify you when there are deviations in the model quality. Early and proactive detection of these deviations enables you to take corrective actions. These actions can include retraining models, auditing upstream systems, and fixing quality issues without having to monitor models manually or build additional tools. Model Monitor provides monitoring capabilities that do not require coding. You also have the flexibility to monitor models by adding code to provide custom analysis.

Documents

- Amazon SageMaker Model Monitor
- Evaluating ML Models
- Cross-Validation of Machine Learning Models
- Implement infrastructure as code

Blogs

- Amazon SageMaker Model Monitor – Fully Managed Automatic Monitoring for your Machine Learning Models
- AWS CloudFormation – Create Your AWS Stack From a Recipe

Examples

- Introduction to Amazon SageMaker Model Monitor
- Model Monitor Visualization

Security pillar – Best Practices

The security pillar encompasses the ability to protect data, systems, and assets to take advantage of cloud technologies to improve your security. This section includes best practices to consider when monitoring models in production.

Best practices

- MLSEC-12: Restrict access to intended legitimate consumers (p. 80)
- MLSEC-13: Monitor human interactions with data for anomalous activity (p. 81)

MLSEC-12: Restrict access to intended legitimate consumers

Use least-privileged permissions to invoke the deployed model endpoint. For consumers who are external to the workload environment, provide access via a secure API.

Implementation plan

- **Use secure inference API endpoints** — Host the model so that a consumer of the model can perform inference against it securely. Enable consumers using the API to define the relationship, restrict access to the base model, and provide monitoring of model interactions.

- **Secure inference endpoints** — Only authorized parties should make inferences against the ML model. Treat inference endpoints as you would any other HTTPS API. Ensure that you follow guidance from the AWS Well-Architected Framework to provide network controls, such as restricting access to specific IP ranges, and bot control. The HTTPS requests for these API calls should be signed, so that the requester identity can be verified, and the requested data is protected in transit.

Documents

- Amazon Machine Learning Key Concepts - Real-time Predictions
- Give SageMaker Hosted Endpoints Access to Resources in Your Amazon VPC
- Register and Deploy Models with Model Registry

Blogs

- Integrating machine learning models into your Java-based microservices
- How Financial Institutions can use AWS to Address Regulatory Reporting
- Secure deployment of Amazon SageMaker resources

Videos

- AWS re:Invent 2019: End-to-End machine learning using Spark and Amazon SageMaker

Examples

- Amazon SageMaker secure MLOps
- Accelerating Machine Learning Development with Data Science as a Service from Change Healthcare

MLSEC-13: Monitor human interactions with data for anomalous activity

Ensure that data access logging is enabled. Audit for anomalous data access events, such as access events from abnormal locations, or activity exceeding the baseline for that entity. Use services and tools that support anomalous activity alerting, and combine their use with data classification to assess risk. Evaluate using services to aid in monitoring data access events.

Implementation plan

- **Enable data access logging** — Verify that you have data access logging for all human CRUD (create, read, update, and delete) operations, including the details of who accessed what elements, what action they took, and at what time.
- **Classify your data** — Use Amazon Macie for protecting and classifying training and inference data in Amazon S3. Amazon Macie is a fully managed security service. It uses ML to automatically discover, classify, and protect sensitive data in AWS. The service recognizes sensitive data, such as personally identifiable information (PII) or intellectual property. Amazon Macie provides visibility into how this data is being accessed or moved. Amazon Macie continually monitors data access activity for anomalies. It generates detailed alerts when it detects a risk of unauthorized access or inadvertent data leaks.
- **Monitor and protect** — Use Amazon GuardDuty to monitor for malicious and unauthorized activities. This will enable protecting AWS accounts, workloads, and data stored in Amazon S3.

Documents

- Amazon SageMaker Incident Response - Logging & Monitoring
- Amazon GuardDuty S3 Finding Types - which aid in anomaly detection for S3 resource access events.

Blogs

- Building a Self-Service, Secure, & Continually Compliant Environment on AWS
- How to Use New Advanced Security Features for Amazon Cognito user pools

Videos

- Protect Your Data in S3 with Amazon Macie and Amazon GuardDuty - AWS Online Tech Talks
- AWS re:Invent 2020: Protecting sensitive data with Amazon Macie and Amazon GuardDuty

Examples

- Controlling and auditing data exploration activities with Amazon SageMaker Studio and AWS Lake Formation

Reliability pillar – Best Practices

The reliability pillar encompasses the ability of a workload to perform its intended function correctly and consistently when it's expected to. This section includes best practices to consider while monitoring deployed models in production.

Best practices

- MLREL-12: Allow automatic scaling of the model endpoint (p. 82)
- MLREL-13: Ensure a recoverable endpoint with a managed version control strategy (p. 83)

MLREL-12: Allow automatic scaling of the model endpoint

Implement capabilities that allow the automatic scaling of model endpoints. This helps ensure the reliable processing of predictions to meet changing workload demands. Include monitoring on endpoints to identify a threshold that initiates the addition or removal of resources to support current demand. After a request to scale is received, put in place a solution to scale backend resources supporting that endpoint.

Implementation plan

- **Configure automatic scaling for Amazon SageMaker Endpoints** — Amazon SageMaker supports automatic scaling (autoscaling) for your hosted models. SageMaker Endpoints can be configured with autoscaling. This ensures that as traffic increases in your application your endpoint can maintain the same level of service availability. Automatic scaling is a key feature of the cloud. It allows you to automatically provision new resources horizontally to handle increased user demand or system load. Automatic scaling is also a key component of creating event-driven architectures and is a necessary capability of any distributed system.
- **Use Amazon Elastic Inference** — With Amazon Elastic Inference, you can choose the CPU instance in AWS that is best suited to the overall compute and memory needs of your application. Separately configure the right amount of GPU-powered inference acceleration, allowing you to efficiently utilize resources and reduce costs.

- **Use Amazon Elastic Inference with EC2 Auto Scaling** — When you create an Auto Scaling group, you can specify the information required to configure the Amazon EC2 instances. This includes Elastic Inference accelerators. To do this, specify a launch template with your instance configuration and the Elastic Inference accelerator.

Documents

- Automatically Scale Amazon SageMaker Model
- What is Amazon Elastic Inference?

Blogs

- Configuring autoscaling inference endpoints in Amazon SageMaker

Videos

- Build, Train and Deploy ML Models at Scale with Amazon SageMaker
- Deploy Your ML Models to Production at Scale with Amazon SageMaker

Examples

- Automatically Scale Amazon SageMaker Models

MLREL-13: Ensure a recoverable endpoint with a managed version control strategy

Ensure an endpoint responsible for hosting model predictions, and all components responsible for generating that endpoint are fully recoverable. Some of these components include model artifacts, container images, and endpoint configurations. Ensure all required components are version controlled, and traceable in a lineage tracker system.

Implementation plan

- **Implement MLOps best practices with Amazon SageMaker Pipelines and Projects** — Amazon SageMaker Pipelines is a service for building machine learning pipelines. It automates developing, training, and deploying models in a versioned, predictable manner. Amazon SageMaker Projects enable teams of data scientists and developers to collaborate on machine learning business problems. A SageMaker project is an AWS Service Catalog provisioned product that enables you to easily create an end-to-end ML solution. SageMaker Projects entities include pipeline executions, registered models, endpoints, datasets, and code repositories.
- **Use infrastructure as code (IaC) tools** — Use AWS CloudFormation to define and build your infrastructure, including your model endpoints. Store your CloudFormation code in git repositories, such as AWS CodeCommit, so that you can version control your infrastructure code.
- **Use Amazon Elastic Container Registry (Amazon ECR)** — Store your containers in Amazon ECR, an artifact repository for Docker containers. Amazon ECR automatically creates a version hash for your containers as you update them allowing you to roll back to previous versions.

Documents

- AWS CloudFormation
- Infrastructure as Code

- SageMaker Pipelines Overview
- What is AWS CodeCommit?
- What is a SageMaker Project?
- What is AWS Service Catalog?
- What is Amazon Elastic Container Registry

Blogs

- How to manage Amazon SageMaker code with AWS CodeCommit
- Building, automating, managing, and scaling ML workflows using Amazon SageMaker Pipelines
- Multi-account model deployment with Amazon SageMaker Pipelines
- Automate feature engineering pipelines with Amazon SageMaker

Videos

- Infrastructure as Code on AWS - AWS Online Tech Talks
- AWS re:Invent 2020: How to create fully automated ML workflows with Amazon SageMaker Pipelines
- Introducing Amazon SageMaker Pipelines - AWS re:Invent 2020
- AWS re:Invent 2020: Implementing MLOps practices with Amazon SageMaker

Examples

- CI/CD Pipeline for AWS CloudFormation templates on AWS
- Amazon SageMaker MLOps

Performance Efficiency pillar – Best Practices

The performance efficiency pillar focuses on the efficient use of computing resources to meet requirements and the maintenance of that efficiency as demand changes and technologies evolve. This section includes best practices to consider while monitoring models in production.

Best practices

- MLPER-12: Evaluate model explainability (p. 84)
- MLPER-13: Evaluate data drift (p. 85)
- MLPER-14: Monitor, detect, and handle model performance degradation (p. 86)
- MLPER-15: Establish an automated re-training framework (p. 87)
- MLPER-16: Review for updated features (p. 88)
- MLPER-17: Include human-in-the-loop monitoring (p. 88)

MLPER-12: Evaluate model explainability

Evaluate model performance as constrained by the explainability requirements of the business. Compliance requirements, business objectives, or both might require that the inferences from a model be directly explainable. Evaluate the explainability needs, and the trade-off between explainability and model complexity. Then select the model type or evaluation metrics. This approach provides transparency into the reasons that a particular inference was attained given the input data.

Implementation plan

- **Use Amazon SageMaker Clarify to explain model results** — Amazon SageMaker Clarify helps improve your ML models by detecting potential bias and helping explain the predictions that models make. It helps you identify various types of bias in data that can emerge during model training or in production. SageMaker Clarify helps explain how these models make predictions using a feature attribution approach. It also monitors inferences that the models make in production for bias or feature attribution drift. The fairness and explainability functions provided by SageMaker Clarify helps you build less biased and more understandable machine learning models. It also provides tools to help you generate model governance reports that you can use to inform risk and compliance teams, and external regulators.

Documents

- Model Explainability
- Feature Attributions that Use Shapley Values
- What is Fairness and Model Explainability for Machine Learning Predictions?

Blogs

- ML model explainability with Amazon SageMaker Clarify and the SKLearn pre-built container
- Explaining Amazon SageMaker Autopilot models with SHAP
- Human-in-the-loop review of model explanations with Amazon SageMaker Clarify and Amazon A2I

Videos

- Interpretability and explainability in machine learning
- Explaining Credit Decisions with Amazon SageMaker

Examples

- Fairness and Explainability with SageMaker Clarify
- Explainability with Amazon SageMaker Debugger

MLPER-13: Evaluate data drift

Understand the effects of data drift on model performance. In cases where the data has drifted, the model could generate inaccurate predictions. Consider a strategy that monitors and adapts to data drift through re-training.

Implementation plan

- **Use Amazon SageMaker Model Monitor, and SageMaker Clarify** — Amazon SageMaker Model Monitor helps you maintain high-quality ML models by detecting model and concept drift in real time, and sending you alerts so you can take immediate action. Model and concept drift are detected by monitoring the quality of the model. Independent variables (also known as features) are the inputs to an ML model, and dependent variables are the outputs of the model. Additionally, SageMaker Model Monitor is integrated with Amazon SageMaker Clarify to help you identify potential bias in your ML models.

Documents

- Amazon SageMaker Model Monitor
- Amazon SageMaker Clarify - Model Explainability

Blogs

- Amazon SageMaker Clarify Detects Bias and Increases the Transparency of Machine Learning Models

Videos

- Detect machine learning (ML) model drift in production

Examples

- Amazon SageMaker Clarify

MLPER-14: Monitor, detect, and handle model performance degradation

Model Performance could degrade over time for reasons including: data quality, model quality, model bias, and model explainability. Continuously monitor the quality of the ML model in real time. Identify the right time and frequency to retrain and update the model. Configure alerts to notify and initiate actions if any drift in model performance is observed.

Implementation plan

- **Monitor model performance** — Amazon SageMaker Model Monitor continually monitors the quality of Amazon SageMaker machine learning models in production. Establish a baseline during training before model is in production. Collect data while in production and compare changes in model inferences. Observations of drifts in the data statistics will indicate that the model may need to be retrained. The timing of drifts will establish a schedule for retraining. Use SageMaker Clarify to identify model bias. Configure alerting systems with Amazon CloudWatch to send notifications for unexpected bias or changes in data quality.

- **Perform automatic scaling** — Amazon SageMaker automatically monitors core system metrics. It includes capabilities to establish automatic scaling for your hosted model to dynamically adjust underlying compute supporting an endpoint based on demand. This capability ensures that your endpoint can dynamically support demand while reducing operational overhead.
- **Monitor endpoint metrics** — Amazon SageMaker also outputs endpoint metrics for monitoring the usage and health of the endpoint. Amazon SageMaker Model Monitor provides the capability to monitor your ML models in production and provides alerts when data quality issues appear. Create a mechanism to aggregate and analyze model prediction endpoint metrics using services, such as Amazon OpenSearch Service (OpenSearch Service). OpenSearch Service supports Kibana for dashboards and visualization. The traceability of hosting metrics back to versioned inputs allows for analysis of changes that could be impacting current operational performance.

Documents

- Amazon SageMaker Model Monitor
- How Amazon CloudWatch works
- Fairness and Explainability with SageMaker Clarify

Blogs

- Monitoring in-production ML models at large scale using Amazon SageMaker Model Monitor
- ML model explainability with Amazon SageMaker Clarify and the SKLearn pre-built container

Videos

- Understand ML model predictions & biases with Amazon SageMaker Clarify
- Deep Dive on Amazon SageMaker Debugger & Amazon SageMaker Model Monitor
- Detect machine learning (ML) model drift in production

Examples

- Amazon SageMaker Model Monitor Examples - Github

MLPER-15: Establish an automated re-training framework

Monitor the data and the model predictions. Run analyses of model performance against defined metrics to identify errors due to data and concept drift. Automate model re-training to mitigate these errors on fixed scheduled intervals, or when model variance reaches a defined threshold. Automated model re-training can also be started as enough new data becomes available.

Implementation plan

- **Identify retraining opportunities** — Monitor data statistics and ML inferences at production using Amazon SageMaker Model Monitor. If the data drifts beyond a defined threshold, then start retraining. A retraining can also be initiated at defined scheduled intervals to meet business requirements. A retraining can also be started when additional training data is available. AWS supports mechanisms for automatically starting retraining based on new data PUT to an Amazon S3 bucket. Ensure model versioning is supported when incorporating additional data into your models. This enables re-creating an inadvertently deleted model artifact using the combined versions of components used to create the versioned artifact.

- **Use Amazon SageMaker Pipelines** — A retraining pipeline can be developed using Amazon SageMaker Pipelines that enables orchestration using step creation and management.

- **Use AWS Step Functions** — You can also use AWS Step Functions Data Science SDK for Amazon SageMaker to automate training of a machine learning model. Define all the steps in the workflow and set up alerts to start the flow. To detect the presence of new training data in an S3 bucket, AWS CloudTrail combined with Amazon CloudWatch Events allows you to start an AWS Step Function workflow to initiate retraining tasks in your training pipeline.
- **Use third-party tools** — Use third-party deployment orchestration tools, such as Jenkins, that integrate with AWS service APIs to automate model retraining when new data is available.

Documents

- Amazon SageMaker Model Building Pipelines
- Retraining Models on New Data
- Amazon SageMaker Model Monitor
- Train a Machine Learning Model (using AWS Step Functions)
- AWS Step Functions Data Science SDK for Python

Blogs

- Monitoring in-production ML models at large scale using Amazon SageMaker Model Monitor
- Automating model retraining and deployment using the AWS Step Functions Data Science SDK for Amazon SageMaker
- Automating complex deep learning model training using Amazon SageMaker Debugger and AWS Step Functions
- Build a CI/CD pipeline for deploying custom machine learning models using AWS services

Videos

- How to create fully automated ML workflows with Amazon SageMaker Pipelines (29:23)
- Machine Learning and Automated Model Retraining with SageMaker

Examples

- Autopilot, Debugger and Model Monitor – Immersion day
- Amazon SageMaker MLOps

MLPER-16: Review for updated features

Establish a framework to run data exploration and feature engineering at pre-determined time intervals based on data volatility and availability. New features that have not been considered in the model training can affect the accuracy of model inferences.

Implementation plan

- **Explore changing data with Amazon SageMaker Data Wrangler** — Evaluate the rate of change of the business environment to set a schedule for validating and possibly changing model input data and features. Analyze the data using Amazon SageMaker Data Wrangler and explore new features. Establish a team who will periodically evaluate and possibly change features and retrain the model.

Documents

- Prepare ML Data with Amazon SageMaker Data Wrangler
- Amazon SageMaker Model Monitor

Blogs

- Exploratory data analysis, feature engineering, and operationalizing your data flow into your ML pipeline with Amazon SageMaker Data Wrangler

Videos

- Introducing Amazon SageMaker Data Wrangler – AWS re:Invent 2020

MLPER-17: Include human-in-the-loop monitoring

Use human-in-the-loop monitoring to monitor model performance efficiently. When automating decision processes, the human labeling of model results is a reliable quality test for model inferences.

Compare human labels with model inferences to estimate model performance degradation. Perform mitigation as model re-training.

Implementation plan

- **Use Amazon Augmented AI to get human review** — Learn how to design a quality assurance system for model inferences. Establish a team of subject matter experts to audit model inference in production. Use Amazon Augmented AI (Amazon A2I) to get human review of low-confidence predictions or random prediction samples. Amazon A2I uses resources in IAM, SageMaker, and Amazon S3 to create and run your human review workflows.

Documents

- Using Amazon Augmented AI for Human Review

Blogs

- Human-in-the-loop review of model explanations with Amazon SageMaker Clarify and Amazon A2I
- Verifying and adjusting your data labels to create higher quality training datasets with Amazon SageMaker Ground Truth

Videos

- Easily Implement Human in the Loop into Your Machine Learning Predictions with Amazon A2I

Cost Optimization pillar – Best Practices

The cost optimization pillar includes the continual process of refinement and improvement of a system over its entire lifecycle. From the initial design of your very first proof of concept to the ongoing operation of production workloads, adopting the practices in this document can enable you to build and operate cost-aware systems that achieve business outcomes and minimize costs, thus allowing your business to maximize its return on investment. This section includes best practices to consider for monitoring models in production.

Best practices

- MLCOST-22: Monitor usage and cost by ML activity (p. 89)

MLCOST-22: Monitor usage and cost by ML activity

Use cloud resource tagging to manage, identify, organize, search for, and filter resources. Tags help categorize resources by purpose, owner, environment, or other criteria. Associate costs with resources using ML activity categories, such as re-training and hosting, by using tagging to manage and optimize cost in deployment phases. Tagging can be useful for generating billing reports with breakdown of cost by associated resources.

Implementation plan

- **Use AWS tagging** — A tag is a label that you or AWS assigns to an AWS resource. Each tag consists of a key and a value. For each resource, each tag key must be unique, and each tag key can have only one value. You can use tags to organize your resources, and cost allocation tags to track your AWS costs on a detailed level. AWS uses the cost allocation tags to organize your resource costs on your cost allocation report. This will make it easier for you to categorize and track your AWS costs. AWS provides two types of cost allocation tags, an AWS-generated tag and user-defined tags.

Documents

- Tagging AWS resources
- Use Tags to Track and Allocate Amazon SageMaker Studio Notebooks Costs
- Tagging best practices

Blogs

- Optimizing costs for machine learning with Amazon SageMaker

Videos

- How can I tag my AWS resources to divide up my bill by cost center or project?

Conclusion

The Well-Architected ML design principles in this paper provide the guidance for the best practices collection. The technology and cloud agnostic best practices across the Well-Architected pillars provide architectural guidance for each phase of the ML lifecycle. Implementation plans provide guidance on implementing these best practices on AWS. Architecture diagrams demonstrate the lifecycle phases with the supporting technologies, that enable many of the best practices introduced in this paper. The ML lens extends the Well-Architected Framework, and builds specific machine learning best practices upon it. As you work towards building and deploying production ML workloads in AWS, we recommend reviewing the AWS Well-Architected Framework pillar best practices.

Use the Lens to help ensure that your ML workloads are architected with operational excellence, security, reliability, performance efficiency, and cost optimization in mind. Plan early and make informed decisions when designing new workloads. Use the best practices to guide you through building and deploying new workloads faster. Using the lens guidance, evaluate existing workloads regularly to identify, mitigate, and address potential issues early.

References

- AWS Architecture Center
- Architecture Best Practices for Machine Learning
- AWS Well-Architected Framework
- AWS Operational Excellence pillar
- AWS Security pillar
- AWS Reliability pillar
- AWS Performance Efficiency pillar
- AWS Cost Optimization pillar
- Data Lifecycle and Analytics Reference Guide
- Amazon AI Fairness and Explainability Whitepaper
- Overview of Deployment Options on AWS
- Crisp-DM 1.0

Document history and contributors

To be notified about updates to this whitepaper, subscribe to the RSS feed.

update-history-change	update-history-description	update-history-date
Minor update (p. 52)	MLSEC-09 updated.	December 20, 2021
Minor update (p. 51)	Fixed missing content in MLSEC-07.	November 9, 2021
Major update (p. 93)	Whitepaper updated to reflect current best practices and AWS services.	October 12, 2021
Minor changes (p. 93)	Updated links in the whitepaper.	January 21, 2021
Initial publication (p. 93)	Machine Learning Lens first published.	April 16, 2020

Note
To subscribe to RSS updates, you must have an RSS plug-in enabled for the browser that you are using.

Contributors

The following individuals contributed to this document:

- Haleh Najafzadeh — Principal AI/ML Solutions Architect, Well-Architected ML Lens Lead, Amazon Web Services
- Patrick Roberts — Senior Data Scientist, Amazon Web Services
- Raju Penmatcha — Senior AI/ML Specialist, Amazon Web Services
- Jess Clark — Principal Security Engineer, Amazon
- Emily Soward — Data Scientist, Amazon Web Services
- Rich Boyd — Geo Solutions Architect Well-Architected, Amazon Web Services
- Amit Lulla — Senior Solutions Architect ISV, Amazon Web Services
- Ali Arsanjani — Principal AI/ML Specialist Solutions Architect, Amazon Web Services
- Giuseppe Angelo Porcelli — Principal AI/ML Specialist Solutions Architect, Amazon Web Services
- Sireesha Muppala — Principal Solutions Architecture, Amazon Web Services
- Shelbee Eigenbrode — Principal AI/ML Specialist Solutions Architect, Amazon Web Services
- Phi Nguyen — Senior Solutions Architect – AI/ML Framework, Amazon Web Services
- Brian Carlson — Senior Operations Solutions Architect Well-Architected, Amazon Web Services
- Ben Potter — Principal Security Lead Well-Architected, Amazon Web Services
- Aden Leirer — Content Program Manager Well-Architected, Amazon Web Services
- Jeremy Sobek — Technical Curriculum Developer, Amazon Web Services

Best practices arranged by pillar

These are all of the best practices outlined in this paper organized by pillar of the AWS Well-Architected Framework.

Operational excellence pillar

- MLOE-01: Develop right skills with accountability and empowerment (p. 7)
- MLOE-02: Establish ML roles and responsibilities (p. 13)
- MLOE-03: Prepare an ML profile template (p. 14)
- MLOE-04: Establish model improvement strategies (p. 15)
- MLOE-05: Establish a lineage tracker system (p. 16)
- MLOE-06: Establish feedback loops across ML lifecycle phases (p. 17)
- MLOE-07: Profile data to improve quality (p. 34)
- MLOE-08: Document data processing (p. 35)
- MLOE-09: Automate operations through IaC and CaC (p. 50)
- MLOE-10: Establish scalable patterns to access approved public libraries (p. 50)
- MLOE-11: Establish deployment environment metrics (p. 71)
- MLOE-12: Enable monitoring health of model endpoint (p. 78)
- MLOE-13: Synchronize architecture and configuration, and check for skew across environments (p. 79)

Security pillar

- MLSEC-01: Validate ML software privacy and license terms (p. 8)
- MLSEC-02: Ensure least privilege access (p. 18)
- MLSEC-03: Secure data and modeling environment (p. 36)
- MLSEC-04: Protect sensitive data privacy (p. 38)
- MLSEC-05: Enforce data lineage (p. 39)
- MLSEC-06: Keep only relevant data (p. 39)
- MLSEC-07: Detect transfer learning risk (p. 51)
- MLSEC-08: Secure governed ML environment (p. 51)
- MLSEC-09: Secure inter-node cluster communications (p. 52)
- MLSEC-10: Protect against data poisoning threats (p. 53)
- MLSEC-11: Protect against adversarial and malicious activities (p. 72)
- MLSEC-12: Restrict access to intended legitimate consumers (p. 80)
- MLSEC-13: Monitor human interactions with data for anomalous activity (p. 81)

Reliability pillar

- MLREL-01: Discuss and agree on the level of model explainability (p. 9)
- MLREL-02: Use APIs to abstract change from model consuming applications (p. 19)

- MLREL-03: Adopt a machine learning microservice strategy (p. 20)
- MLREL-04: Use a data catalog (p. 40)
- MLREL-05: Use a data pipeline (p. 41)
- MLREL-06: Automate managing data changes (p. 42)
- MLREL-07: Enable CI/CD/CT automation with traceability (p. 54)
- MLREL-08: Ensure feature consistency across training and inference (p. 55)
- MLREL-09: Ensure model validation with relevant data (p. 56)
- MLREL-10: Automate endpoint changes through a pipeline (p. 73)
- MLREL-11: Use an appropriate deployment and testing strategy (p. 73)
- MLREL-12: Allow automatic scaling of the model endpoint (p. 82)
- MLREL-13: Ensure a recoverable endpoint with a managed version control strategy (p. 83)

Performance efficiency pillar

- MLPER-01: Determine key performance indicators, including acceptable errors (p. 10)
- MLPER-02: Understand and manage the available services and resources (p. 21)
- MLPER-03: Review fairness and explainability (p. 22)
- MLPER-04: Define relevant evaluation metrics (p. 23)
- MLPER-05: Use a data lake house architecture (p. 43)
- MLPER-06: Optimize training and inference instance types (p. 57)
- MLPER-07: Explore alternatives for performance improvement (p. 58)
- MLPER-08: Establish a model performance evaluation pipeline (p. 59)
- MLPER-09: Establish feature statistics (p. 59)
- MLPER-10: Perform a performance trade-off analysis (p. 60)
- MLPER-11: Evaluate machine learning deployment option (cloud versus edge) (p. 75)
- MLPER-12: Evaluate model explainability (p. 84)
- MLPER-13: Evaluate data drift (p. 85)
- MLPER-14: Monitor, detect, and handle model performance degradation (p. 86)
- MLPER-15: Establish an automated re-training framework (p. 87)
- MLPER-16: Review for updated features (p. 88)
- MLPER-17: Include human-in-the-loop monitoring (p. 88)

Cost optimization pillar

- MLCOST-01: Define overall return on investment (ROI) and opportunity cost (p. 11)
- MLCOST-02: Use managed services to reduce total cost of ownership (TCO) (p. 12)
- MLCOST-03: Identify if machine learning is the right solution (p. 24)
- MLCOST-04: Enable data and compute proximity (p. 25)
- MLCOST-05: Select optimal algorithms (p. 25)
- MLCOST-06: Tradeoff analysis on custom versus pre-trained models (p. 26)
- MLCOST-07: Enable debugging and logging (p. 27)
- MLCOST-08: Use managed data labeling (p. 44)
- MLCOST-09: Use data wrangler tools for interactive analysis (p. 44)
- MLCOST-10: Enable feature reusability (p. 45)

- MLCOST-11: Establish data bias detection and mitigation (p. 46)
- MLCOST-12: Select optimal computing instance size (p. 62)
- MLCOST-13: Select local training for small scale experiments (p. 62)
- MLCOST-14: Select an optimal ML framework (p. 63)
- MLCOST-15: Use automated machine learning (p. 64)
- MLCOST-16: Use distributed training (p. 64)
- MLCOST-17: Stop resources when not in use (p. 65)
- MLCOST-18: Start training with small datasets (p. 66)
- MLCOST-19: Use warm-start and checkpointing hyperparameter tuning (p. 67)
- MLCOST-20: Use hyperparameter optimization technologies (p. 68)
- MLCOST-21: Use an inference pipeline (p. 76)
- MLCOST-22: Monitor usage and cost by ML activity (p. 89)

Notices

Customers are responsible for making their own independent assessment of the information in this document. This document: (a) is for informational purposes only, (b) represents current AWS product offerings and practices, which are subject to change without notice, and (c) does not create any commitments or assurances from AWS and its affiliates, suppliers or licensors. AWS products or services are provided “as is” without warranties, representations, or conditions of any kind, whether express or implied. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

AWS glossary

For the latest AWS terminology, see the AWS glossary in the *AWS General Reference*.