

The Machine Learning Solutions Architect Handbook

A comprehensive guide to AI use cases, enterprise ML platform and solutions design on AWS, AI risk management, and generative AI

Second Edition



David Ping



The Machine Learning Solutions Architect Handbook

Copyright © 2023 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

Early Access Publication: The Machine Learning Solutions Architect Handbook

Early Access Production Reference: B20836

Published by Packt Publishing Ltd.

Livery Place

35 Livery Street

Birmingham

B3 2PB, UK

ISBN: 978-1-80512-250-0

www.packt.com

Table of Contents

1. [The Machine Learning Solutions Architect Handbook, Second Edition: A comprehensive guide to AI use cases, enterprise ML platform and solutions design on AWS, AI risk management, and generative AI](#)
2. [1 Machine Learning and Machine Learning Solutions Architecture](#)
 - I. [Join our book community on Discord](#)
 - II. [What are AI and ML?](#)
 - i. [Supervised ML](#)
 - ii. [Unsupervised ML](#)
 - iii. [Reinforcement learning](#)
 - III. [ML versus traditional software](#)
 - IV. [ML life cycle](#)
 - i. [Business understanding and ML problem framing](#)
 - ii. [Data understanding and data preparation](#)
 - iii. [Model training and evaluation](#)
 - iv. [Model deployment](#)
 - v. [Model monitoring](#)
 - vi. [Business metric tracking](#)
 - V. [ML challenges](#)
 - VI. [ML solutions architecture](#)
 - i. [Business understanding and ML transformation](#)
 - ii. [Identification and verification of ML techniques](#)
 - iii. [System architecture design and implementation](#)
 - iv. [ML platform workflow automation](#)
 - v. [Security and compliance](#)
 - VII. [Testing your knowledge](#)
 - VIII. [Summary](#)
3. [2 Business Use Cases for Machine Learning](#)
 - I. [Join our book community on Discord](#)
 - II. [ML use cases in financial services](#)
 - i. [Capital markets front office](#)
 - ii. [Capital markets back office operations](#)
 - iii. [Risk management and fraud](#)

iv. [Insurance](#)

III. [ML use cases in media and entertainment](#)

i. [Content development and production](#)

ii. [Content management and discovery](#)

iii. [Content distribution and customer engagement](#)

IV. [ML use cases in healthcare and life sciences](#)

i. [Medical imaging analysis](#)

ii. [Drug discovery](#)

iii. [Healthcare data management](#)

V. [ML use cases in manufacturing](#)

i. [Engineering and product design](#)

ii. [Manufacturing operations – product quality and yield](#)

iii. [Manufacturing operations – machine maintenance](#)

VI. [ML use cases in retail](#)

i. [Product search and discovery](#)

ii. [Target marketing](#)

iii. [Sentiment analysis](#)

iv. [Product demand forecasting](#)

VII. [ML use cases in automotive](#)

i. [Autonomous vehicle](#)

ii. [Advanced driver assistance systems \(ADAS\)](#)

VIII. [Summary](#)

4. [3 Machine Learning Algorithms](#)

I. [Join our book community on Discord](#)

II. [Technical requirements](#)

III. [How machines learn](#)

IV. [Overview of ML algorithms](#)

i. [Consideration for choosing ML algorithms](#)

ii. [Algorithms for classification and regression problems](#)

iii. [Algorithms for clustering](#)

iv. [Algorithms for time series analysis](#)

v. [Algorithms for recommendation](#)

vi. [Algorithms for computer vision problems](#)

vii. [Algorithms for natural language processing \(NLP\) problems](#)

viii. [Generative AI algorithms](#)

V. [Hands-on exercise](#)

- i. [Problem statement](#)
 - ii. [Dataset description](#)
 - iii. [Setting up a Jupyter Notebook environment](#)
 - iv. [Running the exercise](#)
- VI. [Summary](#)
5. [4 Data Management for Machine Learning](#)
- I. [Join our book community on Discord](#)
 - II. [Technical requirements](#)
 - III. [Data management considerations for ML](#)
 - IV. [Data management architecture for ML](#)
 - V. [Data storage and management](#)
 - i. [AWS Lake Formation](#)
- VI. [Data ingestion](#)
- i. [Kinesis Firehose](#)
 - ii. [AWS Glue](#)
 - iii. [AWS Lambda](#)
- VII. [Data cataloging](#)
- i. [AWS Glue Catalog](#)
 - ii. [Custom data catalog solution](#)
- VIII. [Data processing](#)
- i. [AWS Glue ETL](#)
 - ii. [Amazon Elastic Map Reduce \(EMR\)](#)
 - iii. [AWS Lambda](#)
- IX. [ML data versioning](#)
- i. [S3 partitions](#)
 - ii. [Versioned S3 buckets](#)
 - iii. [Purpose-built data version tools](#)
- X. [ML feature store](#)
- XI. [Data serving for client consumption](#)
- i. [Consumption via API](#)
 - ii. [Consumption via data copy](#)
- XII. [Special databases for ML](#)
- i. [Vector Database](#)
 - ii. [Graph Databases](#)
- XIII. [Data pipeline](#)
- i. [AWS Glue workflows](#)

- ii. [AWS Managed Airflow](#)

XIV. [Authentication and authorization](#)

XV. [Data governance](#)

- i. [Data Lineage](#)

- ii. [Other data governance measures](#)

XVI. [Hands-on exercise – data management for ML](#)

- i. [Creating a data lake using Lake Formation](#)

- ii. [Creating a data ingestion pipeline](#)

- iii. [Creating a Glue catalog](#)

- iv. [Discovering and querying data in the data lake](#)

- v. [Creating an Amazon Glue ETL job to process data for ML](#)

- vi. [Building a data pipeline using Glue workflows](#)

XVII. [Summary](#)

6. [5 Open Source Machine Learning Libraries](#)

I. [Join our book community on Discord](#)

II. [Technical requirements](#)

III. [Core features of open source machine learning libraries](#)

IV. [Understanding the scikit-learn machine learning library](#)

- i. [Installing scikit-learn](#)

- ii. [Core components of scikit-learn](#)

V. [Understanding the Apache Spark ML machine learning library](#)

- i. [Installing Spark ML](#)

- ii. [Core components of the Spark ML library](#)

VI. [Understanding the TensorFlow deep learning library](#)

- i. [Installing Tensorflow](#)

- ii. [Core components of TensorFlow](#)

VII. [Hands-on exercise – training a TensorFlow model](#)

VIII. [Understanding the PyTorch deep learning library](#)

- i. [Installing PyTorch](#)

- ii. [Core components of PyTorch](#)

IX. [Hands-on exercise – building and training a PyTorch model](#)

X. [Summary](#)

7. [6 Kubernetes Container Orchestration Infrastructure Management](#)

I. [Join our book community on Discord](#)

II. [Technical requirements](#)

III. [Introduction to containers](#)

IV. Kubernetes overview and core concepts

- i. Namespaces
- ii. Pods
- iii. Deployment
- iv. Kubernetes Job
- v. Kubernetes custom resources (CRs) and operators
- vi. Services

V. Networking on Kubernetes

- i. Service mesh

VI. Security and access management

- i. Network security
- ii. Authentication and authorization to APIs
- iii. Running ML workloads on Kubernetes

VII. Hands-on – creating a Kubernetes infrastructure on AWS

- i. Problem statement
- ii. Lab instruction

VIII. Summary

8. 7 Open Source Machine Learning Platforms

- I. Join our book community on Discord
- II. Core components of an ML platform
- III. Open source technologies for building ML platforms
 - i. Implementing a data science environment
 - ii. Building a model training environment
 - iii. Registering models with a model registry
 - iv. Serving models using model serving services
 - v. Monitoring models in production
 - vi. Managing ML features
 - vii. Feast
 - viii. Automating ML pipeline workflows

IV. Building end-to-end ML platform

- i. ML platform based strategy
- ii. ML component based strategy

V. Summary

9. 8 Building a Data Science Environment Using AWS ML Services

- I. Join our book community on Discord
- II. Technical requirements

III. [SageMaker overview](#)

IV. [Data science environment architecture using SageMaker](#)

- i. [Onboarding SageMaker users](#)
- ii. [Launching a notebook](#)
- iii. [Preparing data](#)
- iv. [Creating, storing, and sharing features](#)
- v. [Training machine learning models](#)
- vi. [Tuning machine learning models](#)
- vii. [Deploying machine learning models for testing](#)

V. [Automating experimentation and model building](#)

VI. [Best practices for building data science environment](#)

VII. [Hands-on exercise – building a data science environment using AWS services](#)

- i. [Problem statement](#)
- ii. [Dataset](#)
- iii. [Lab instructions](#)

VIII. [Summary](#)

10. [9 Building an Enterprise ML Architecture with AWS ML Services](#)

- I. [Join our book community on Discord](#)
- II. [Technical requirements](#)
- III. [The personas of ML platforms and their requirement](#)
 - i. [ML platform builder](#)
 - ii. [Platform user and operator](#)
 - iii. [Common workflow of an ML initiative](#)
 - iv. [Platform capability requirements](#)

IV. [Key requirements for an enterprise ML platform](#)

V. [Enterprise ML architecture pattern overview](#)

VI. [Model training environment](#)

- i. [Model training engine using SageMaker](#)
- ii. [Automation support](#)
- iii. [Model training life cycle management](#)

VII. [Model hosting environment deep dive](#)

- i. [Inference engine](#)
- ii. [Authentication and security control](#)
- iii. [Monitoring and logging](#)

VIII. [Adopting MLOps for ML workflows](#)

- i. [Components of the MLOps architecture](#)
 - ii. [Monitoring and logging](#)
 - iii. [Best practices in building and operating ML platform](#)
- IX. [Hands-on exercise – building an MLOps pipeline on AWS](#)
- i. [Creating a CloudFormation template for the ML training pipeline](#)
 - ii. [Creating a CloudFormation template for the ML deployment pipeline](#)
- X. [Summary](#)
11. [12 AI Risk Management](#)
- I. [Join our book community on Discord](#)
 - II. [Understanding AI risk scenarios](#)
 - i. [Bias and discrimination](#)
 - ii. [Misinformation and misinterpretation](#)
 - iii. [Lack of interpretability](#)
 - iv. [Unintended consequences](#)
 - v. [Adversarial Attacks](#)
 - vi. [Privacy violation and sensitive data exposure](#)
 - vii. [Third party risk](#)
 - viii. [Model testing risk](#)
 - III. [The regulatory landscape around AI risk management](#)
 - IV. [Understanding AI risk management](#)
 - i. [Placing trust in AI systems](#)
 - V. [Creating risk management framework for trustworthy AI](#)
 - i. [Governance oversight Principles](#)
 - ii. [AI risk management framework](#)
 - VI. [Applying risk management across AI lifecycle](#)
 - i. [Business problem identification and definition](#)
 - ii. [Data acquisition and management](#)
 - iii. [Experimentation and model development](#)
 - iv. [AI system deployment and operations](#)
 - VII. [Designing ML platform for governance and risk considerations](#)
 - i. [Data and model documentation](#)
 - ii. [Lineage and reproducibility](#)
 - iii. [Observability and auditing](#)
 - iv. [Scalability and performance](#)

v. [Data quality](#)

VIII. [Summary](#)

12. [13 Bias, Explainability, Privacy, and Adversarial attacks](#)

I. [Join our book community on Discord](#)

II. [Understanding bias](#)

i. [Bias detection and mitigation](#)

III. [Understanding ML explainability](#)

IV. [Understanding security and privacy preserving ML](#)

V. [Understanding Adversarial Attacks](#)

i. [Evasion attacks](#)

ii. [Data poisoning attacks](#)

iii. [Clean-label Backdoor Attack](#)

iv. [Model extraction attack](#)

VI. [Defense against adversarial attacks](#)

i. [Robustness-based methods](#)

ii. [Detector-based method](#)

iii. [Open-source tools for adversarial attacks and defenses](#)

VII. [Hands-on Lab – Detecting bias, explaining model, and training privacy-preserving models](#)

i. [Overview of the scenario](#)

ii. [Detecting bias in the training dataset](#)

iii. [Explaining feature importance for trained model](#)

iv. [Training privacy preserving models](#)

VIII. [Summary](#)

13. [14 Progressing the ML journey](#)

I. [Join our book community on Discord](#)

II. [Understanding ML journey and AI capabilities](#)

i. [ML adoption stages](#)

ii. [AI/ML maturity and assessment](#)

iii. [AI/ML operating models](#)

III. [Solving ML Journey challenges](#)

i. [Developing AI vision and strategy](#)

ii. [Getting started with first AI/ML initiative](#)

iii. [Solving scaling challenge with AI/ML adoption](#)

IV. [Summary](#)

The Machine Learning Solutions Architect Handbook, Second Edition: A comprehensive guide to AI use cases, enterprise ML platform and solutions design on AWS, AI risk management, and generative AI

Welcome to Packt Early Access. We're giving you an exclusive preview of this book before it goes on sale. It can take many months to write a book, but our authors have cutting-edge information to share with you today. Early Access gives you an insight into the latest developments by making chapter drafts available. The chapters may be a little rough around the edges right now, but our authors will update them over time. You can dip in and out of this book or follow along from start to finish; Early Access is designed to be flexible. We hope you enjoy getting to know more about the process of writing a Packt book.

1. Chapter 1: Machine Learning and Machine Learning Solutions Architecture
2. Chapter 2: Business Use Cases for Machine Learning
3. Chapter 3: Machine Learning Algorithms
4. Chapter 4: Data Management for Machine Learning
5. Chapter 5: Open-Source Machine Learning Libraries
6. Chapter 6: Kubernetes Container Orchestration Infrastructure
7. Chapter 7: Open-Source ML Platforms
8. Chapter 8: Building a Data Science Environment using AWS ML Services
9. Chapter 9: Building Enterprise ML Architecture with AWS ML Services
10. Chapter 10: Advanced ML Engineering
11. Chapter 11: Building ML solutions with AWS AI Services
12. Chapter 12: ML Risk Management
13. Chapter 13: Bias, Explainability, Privacy, and Adversarial Attacks

14. Chapter 14: Progressing Through the ML Journey
15. Chapter 15: ML Milestones and Research Trends

1 Machine Learning and Machine Learning Solutions Architecture

Join our book community on Discord

<https://packt.link/EarlyAccessCommunity>



The field of **artificial intelligence (AI)** and **machine learning (ML)** has had a long history. Over the last 70+ years, ML has evolved from checker game-playing computer programs in the 1950s to advanced AI capable of beating the human world champion in the game of *Go*. And more recently, Generative AI (GAI) technology such as ChatGPT has been taking the industry by the storm, generating huge interest among company executives and consumers alike, promising new ways to transform businesses such as drug discovery, new media content, financial report analysis, and consumer product design. Along the way, the technology infrastructure for ML has also evolved from a single machine/server for small experiments and models to highly complex end-to-end ML platforms capable of training, managing, and deploying tens of thousands of ML models. The hyper-growth in the AI/ML field has resulted in the creation of many new professional roles, such as **MLOps engineering**, **AI/ML product management**, and **ML software engineering**, **AI risk manager**, and **AI strategist** across a range of industries. **Machine learning solutions architecture (ML solutions architecture)** is another relatively new discipline that is playing an increasingly critical role in the full end-to-end ML life cycle as ML projects become increasingly complex in terms of *business impact*, *science sophistication*, and the *technology landscape*. This chapter talks about the basic concepts of ML and where ML solutions

architecture fits in the full data science life cycle. You will learn the three main types of ML, including **supervised**, **unsupervised**, and **reinforcement** learning. We will discuss the different steps it will take to get an ML project from the ideation stage to production and the challenges faced by organizations, such as use case identification, data quality issues, and shortage of ML talent, when implementing an ML initiative. Finally, we will finish the chapter by briefly discussing the core focus areas of ML solutions architecture, including system architecture, workflow automation, and security and compliance.

Upon completing this chapter, you should be able to identify the three main ML types and what type of problems they are designed to solve. You will understand the role of an ML solutions architect and what business and technology areas you need to focus on to support end-to-end ML initiatives. In this chapter, we are going to cover the following main topics:

- What is ML, and how does it work?
- The ML life cycle and its key challenges
- What is ML solutions architecture, and where does it fit in the overall life cycle?

What are AI and ML?

AI can be defined as a machine demonstrating intelligence similar to that of human natural intelligence, such as distinguishing different types of flowers through vision, understanding languages, or driving cars. Having AI capability does not necessarily mean a system has to be powered only by ML. An AI system can also be powered by other techniques, such as rule-based engines. ML is a form of AI that learns how to perform a task using different learning techniques, such as learning from examples using historical data or learning by trial and error. An example of ML would be using historical credit decision data to train an ML model for making credit decisions. The current type of AI/ML capabilities that we work and interact with is called artificial narrow intelligence (ANI). This means that these AI/ML systems are not capable to performing general intelligent tasks and are limited to specific tasks they are designed for. On the other hand,

artificial general intelligence (AGI) refers to systems that can perform any intellectual task that a human can. These systems would be able to reason, learn, and solve problems in a way similar to human intelligence. While AGI is not yet a reality, the latest advancement in generative AI are exhibiting some behaviors we would expect from an AGI system. Deep learning (DL) is a subset of ML that uses a large number of artificial neurons (known as an **artificial neural network**) to learn, which is similar to how a human brain learns. An example of a deep learning-based solution is the **Amazon Echo virtual assistant**. To better understand how ML works, let's first talk about the different approaches taken by machines to learn. They are as follows:

- **Supervised machine learning**
- **Unsupervised machine learning**
- **Reinforcement learning**

Let's have a look at each one of them in more detail.

Supervised ML

Supervised ML is a type of ML where, when training an ML model, an ML algorithm is provided with the input data features (for example, the size and zip code of houses) and the answers, also known as **labels** (for example, the prices of the houses). A dataset with labels is called a **labeled dataset**.

Supervised ML can be thought of as learning through examples. To understand what this means, let's use an example of how we humans learn to distinguish different objects. Say you are first provided with a number of pictures of different flowers and their names. You are then told to study the characteristics of the flowers, such as the shape, size, and color for each provided flower name. After you have gone through a number of different pictures for each flower, you are then given flower pictures without the names and asked to distinguish them. Based on what you have learned previously, you should be able to tell the names of flowers if they possess the characteristics of the previously learned flowers. In general, the more training pictures with variations you have looked at during the learning time, the more accurate you will likely be when you try to name flowers in

the new pictures. Conceptually, this is how supervised ML works. The following figure (*Figure 1.1*) illustrates supervised machine learning using a computer vision algorithm. The labelled dataset of different flowers is provided to the algorithm for model training. During the training process, the algorithm learns to recognize patterns in the input data and map them to the correct labels. After the model is trained, it will then be able to correctly classify new and unseen flower images.

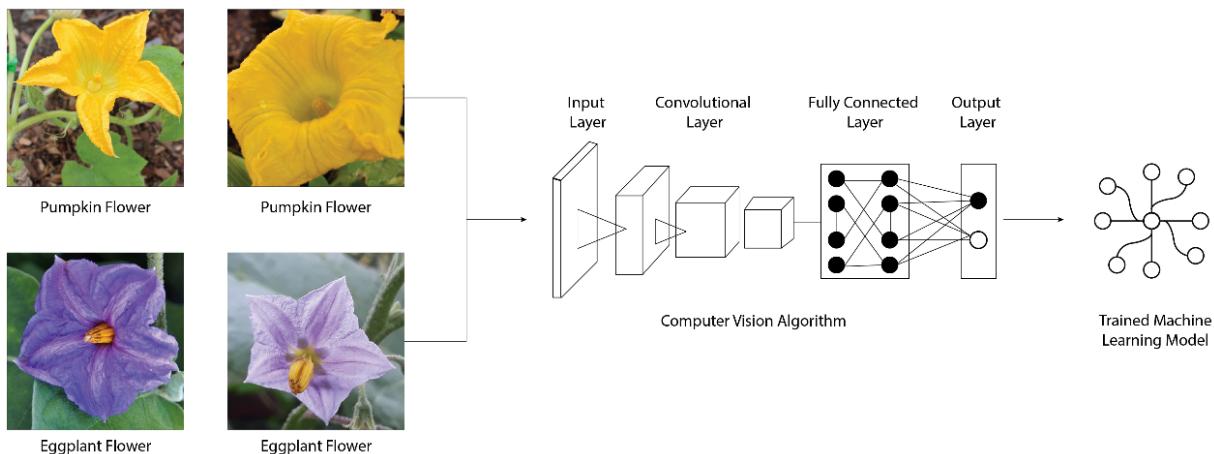


Figure 1.1: Supervised ML

Supervised ML is mainly used for classification tasks that assign a label from a discrete set of categories to an example (for example, telling the names of different objects) and regression tasks that predict a continuous value (for example, estimating the value of something given supporting information). In the real world, the majority of ML solutions are based on supervised ML techniques. The following are some examples of ML solutions that use supervised ML:

- Classifying documents into different document types automatically, as part of a document management workflow. The typical business benefits of ML-based document processing are the reduction of manual effort to save cost, faster processing time, and higher processing quality.
- Assessing the sentiment of news articles to help understand the market perception of a brand or product or facilitate investment decisions.

- Automating the objects or faces detection in images as part of a media image processing workflow. The business benefits this delivers are cost-saving from the reduction of human labor, faster processing, and higher accuracy.
- Predicting the probability that someone will default on a bank loan. The business benefits this delivers are faster decision-making on loan application reviews and approvals, lower processing costs, and a reduced impact on a company's financial statement due to loan defaults.

Unsupervised ML

Unsupervised ML is a type of ML where an ML algorithm is provided with input data features without labels. This means it can be used to discover previously unknown insights and correlations in the data. Let's continue with the flower example, however in this case, you are now only provided with the pictures of the flowers and not their names. In this scenario, you will not be able to figure out the names of the flowers, regardless of how much time you spend looking at the pictures. However, through visual inspection, you should be able to identify the common characteristics (for example, color, size, and shape) of different types of flowers across the pictures, and group flowers with common characteristics in the same group. This is similar to how unsupervised ML works. Specifically, in this particular case, you have performed the **clustering** task in unsupervised ML:

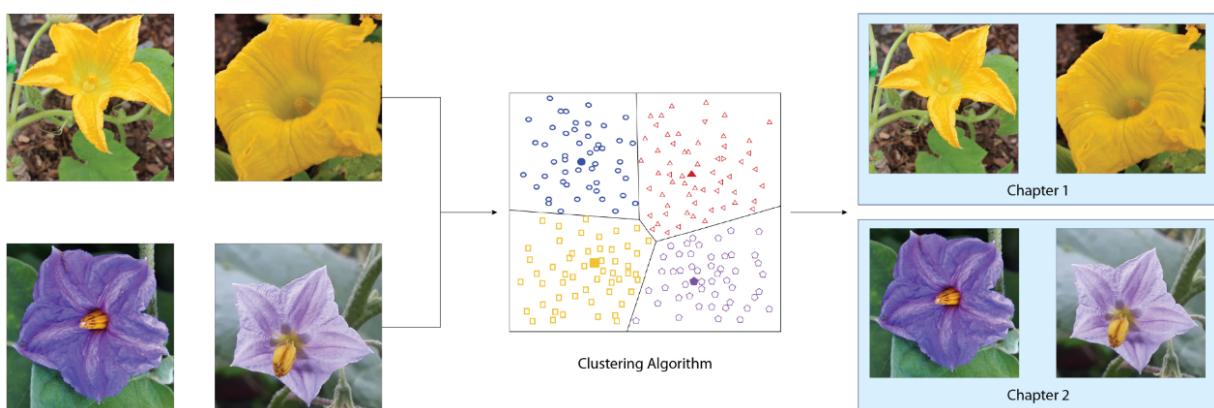


Figure 1.2: Unsupervised ML

In addition to the clustering technique, there are many other techniques in unsupervised ML. Another common and useful unsupervised ML technique is **dimensionality reduction**, where a smaller number of transformed features represent the original set of features while maintaining the critical information from the original features so that they can be largely reproduced in the number of data dimensions and size. To understand this more intuitively, let's take a look at *Figure 1.3*:

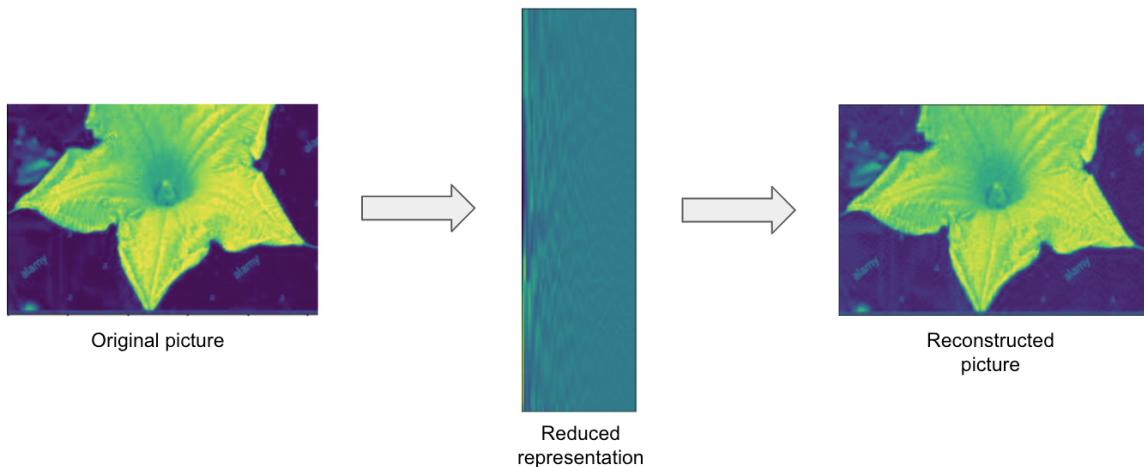


Figure 1.3: Reconstruction of an image from reduced features

In this figure, the original picture on the left is transformed to the reduced representation in the middle. While the reduced representation does not look like the original picture at all, it still maintains the critical information about the original picture, so that when the picture on the right is reconstructed using the reduced representation, the reconstructed image looks almost the same as the original picture. The process that transforms the original picture to the reduced representation is called dimensionality reduction. The main benefits of dimensionality reduction are reduction of the training dataset and helping speed up the model training. Dimensionality reduction also helps visualize high dimensional datasets in lower dimensions (for example, reducing the dataset to three dimensions to

be plotted and visually inspected). Unsupervised ML is mainly used for recognizing underlying patterns within a dataset. Since unsupervised learning is not provided with actual labels to learn from, its predictions have greater uncertainties than predictions using the supervised ML approach. The following are some real-life examples of unsupervised ML solutions:

- **Customer segmentation for target marketing:** By using customer attributes such as demographics and historical engagement data. The data-driven customer segmentation approach is usually more accurate than human judgment, which can be biased and subjective.
- **Computer network intrusion detection:** By detecting outlier patterns that are different from normal network traffic patterns. Detecting anomalies in network traffic manually and rule-based processing is extremely challenging due to the high volume and changing dynamics of traffic patterns.
- **Reducing the dimensions of datasets:** To visualize them in a 2D or 3D environment to help understand the data better and more easily.

Reinforcement learning

Reinforcement learning is a type of ML where an ML model learns by trying out different actions in an environment and adjusts its future behaviors sequentially based on the received responses for its actions. The goal of reinforcement learning is to learn a policy that maximizes the cumulative rewards over time. For example, suppose you are playing a space invader video game for the first time without knowing the game's rules. In that case, you will initially try out different actions randomly using the controls, such as moving left and right or shooting the canon. As different moves are made, you will see responses to your moves, such as getting killed or killing the invader, and you will also see your score increase or decrease. Through these responses, you will know what a good move is versus a bad move in order to stay alive and increase your score. After much trial and error, you will eventually be a very good player of the game. This is basically how reinforcement learning works.

A very popular example of reinforcement learning is the AlphaGo computer program, which uses mainly reinforcement learning to learn how to play the game of Go. Figure 1.4 shows the flow of reinforcement learning where an agent (for example, the player of a space invader game) takes actions (for example, moving the left/right control) in the environment (for example, the current state of the game) and receives rewards or penalties (score increase/decrease). As a result, the agent will adjust its future moves to maximize the rewards in the future states of the environment. This cycle continues for a very large number of rounds, and the agent will improve and become better over time:

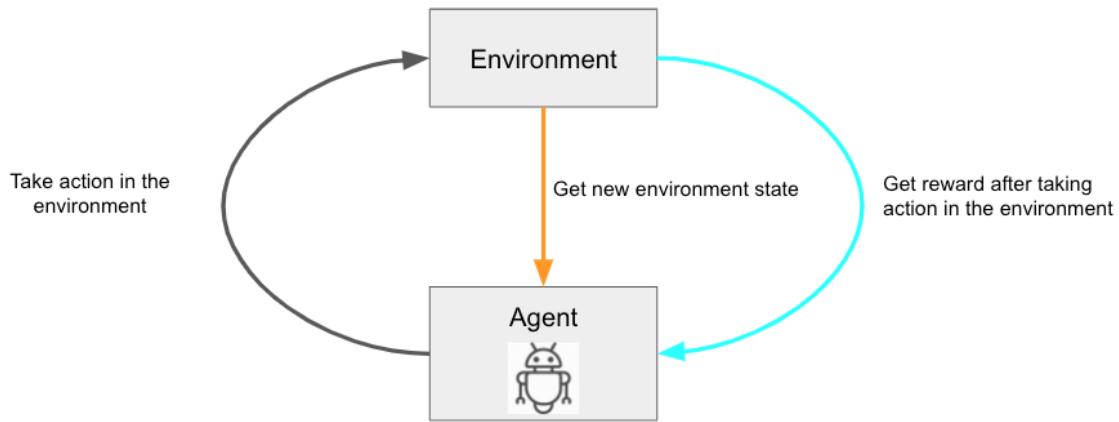


Figure 1.4: Reinforcement learning

There are many practical use cases for reinforcement learning in the real world. The following are some examples for reinforcement learning:

- Robots or self-driving cars learn how to walk or navigate in unknown environments by trying out different moves and responding to the received results.
- A recommendation engine optimizes product recommendations through adjustments based on the feedback of the customers to different product recommendations.
- A truck delivery company optimizes the delivery route of its fleet to determine the delivery sequence required to achieve the best rewards,

such as lowest the cost or shortest time.

ML versus traditional software

Before I started working in the field of AI/ML, I spent many years building computer software platforms for large financial services institutions. Some of the business problems I worked on had complex rules, such as identifying companies for comparable analysis for investment banking deals, or creating a master database for all the different companies' identifiers from the different data providers. We had to implement hardcoded rules in database-stored procedures and application server backends to solve these problems. We often debated if certain rules made sense or not for the business problems we tried to solve. As rules changed, we had to reimplement the rules and make sure the changes did not break anything. To test for new releases or changes, we often replied to human experts to exhaustively test and validate all the business logic implemented before the production release. It was a very time-consuming and error-prone process and required a significant amount of engineering, testing against the documented specification, and rigorous change management for deployment every time new rules were introduced, or existing rules needed to be changed. We often replied to users to report business logic issues in production, and when an issue was reported in production, we sometimes had to open up the source code to troubleshoot or explain the logic of how it worked. I remember I often asked myself if there were better ways to do this. After I started working in the field of AI/ML, I started to solve many similar challenges using ML techniques. With ML, I did not need to come up with complex rules that often require deep data and domain expertise to create or maintain the complex rules for decision making. Instead, I focused on collecting high-quality data and used ML algorithms to learn the rules and patterns from the data directly. This new approach eliminated many of the challenging aspects of creating new rules (for example, a deep domain expertise requirement, or avoiding human bias) and maintaining existing rules. To validate the model before the production release, we could examine model performance metrics such as **accuracy**. While it still required data science expertise to interpret the model metrics against the

nature of the business problems and dataset, it did not require exhaustive manual testing of all the different scenarios. When a model was deployed into production, we would monitor if the model performed as expected by monitoring any significant changes in production data versus the data we have collected for model training. We would collect new labels for production data and test the model performance periodically to ensure its predictive power had not degraded. To explain why a model made a decision the way it did, we did not need to open up source code to re-examine the hardcoded logic. Instead, we would rely on ML techniques to help explain the relative importance of different input features to understand what factors were most influential in the decision-making by the ML models. The following figure (*Figure 1.5*) shows a graphical view of the process differences between developing a piece of software and training an ML model:

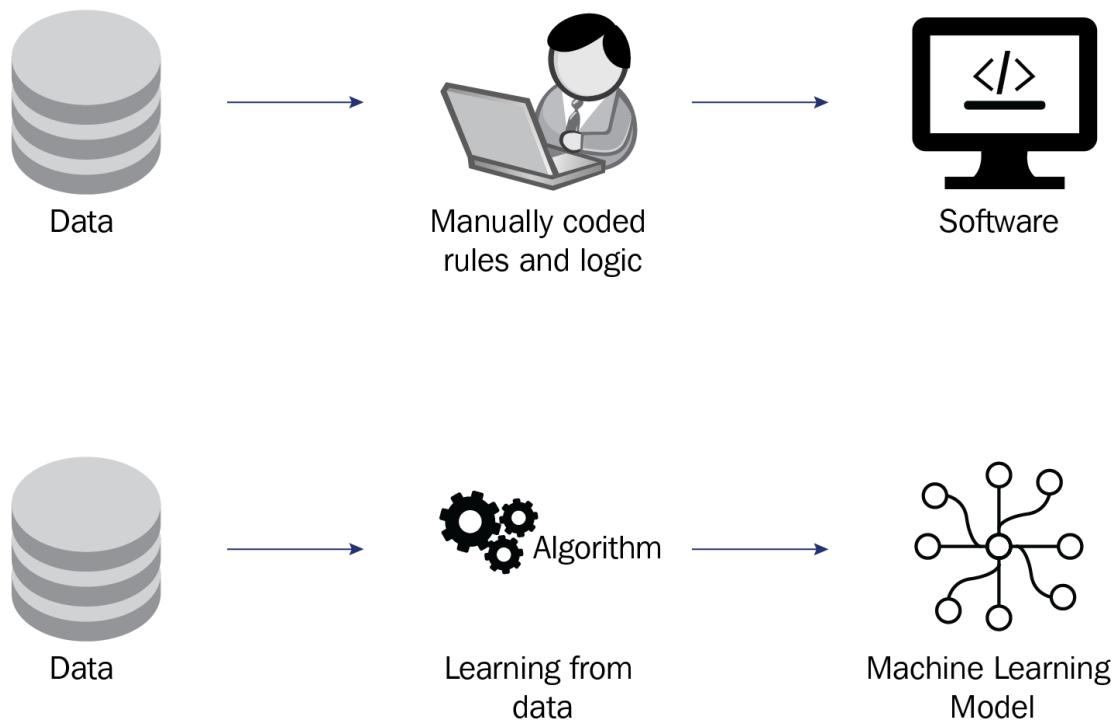


Figure 1.5: ML and computer software

Now that you know the difference between ML and traditional software, it is time to dive deep into understanding the different stages in an ML life

cycle.

ML life cycle

One of the early ML projects that I worked on was a fascinating yet daunting sports predictive analytics problem for a major league brand. I was given a list of predictive analytics outcomes to think about to see if there were ML solutions for the problems. I was a casual viewer of the sport; I didn't know anything about the analytics to be generated, nor the rules of the games in needed detail. I was provided with some sample data, but had no idea what to do with it. The first thing I started to work on was an immersion of the sport itself. I delved into the intricacies of the game, studying the different player positions and events that make up each game and play. Only after armed with the newfound domain knowledge, the data started to make sense. Together with the stakeholder, we evaluated the impact of the different analytics outcomes, assessed the modeling feasibility based on the data we had. With clear understanding of the data, we came up with a couple of top ML analytics with the most business impact to focus on. We also decided how they would be integrated into the existing business workflow, and how they would be measured on their impacts. Subsequently, I delved deeper into the data and ascertain what information was available and what was lacking. I processed and prepared the dataset based on a few of ML algorithms I had considered and conducted experiments to determine the best approach. I lacked a tool to track the different experiment results, so I had to document what I have done manually. After some initial rounds of experimentation, it became evident that existing data was not sufficient to train a high-performance model. Hence, I decided to build a custom deep learning model to incorporate data of different modalities. The data owner was able to provide additional datasets I required, and after more experiments with custom algorithms and significant data preparations and feature engineering, I eventually trained a model that met the business objectives. After completing the model, another hard challenge began – deploying and operationalizing the model in production and integrate it into the existing business workflow and system architecture. We engaged in many architecture and engineering discussions and eventually built out a

deployment architecture for the model. As you can see from my personal experience, the journey from business idea to ML production deployment involved many steps. A typical life cycle of an ML project follows formal structure, which include several essential stages like business understanding, data acquisition and understanding, data preparation, model building, model evaluation, and model deployment. Since a big component of the life cycle is experimentation with different datasets, features, and algorithms, the whole process can be highly iterative. Furthermore, it is essential to note that there is no guarantee of a successful outcome. Factors such as the availability and quality of data, feature engineering techniques (the process of using domain knowledge to extract useful features from raw data), and the capability of the learning algorithms, among others, can all affect the final results.

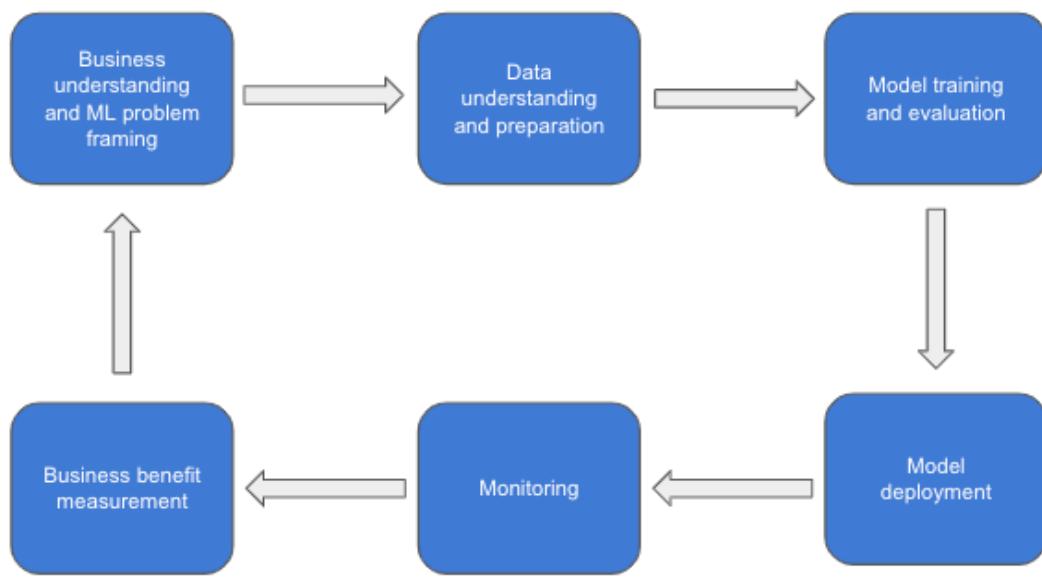


Figure 1.6: ML life cycle

The previous figure (Figure 1.6) illustrates the key steps in ML projects, and in the subsequent sections, we will delve into each of these steps in greater details.

Business understanding and ML problem framing

The first stage in the life cycle is **business understanding**. This stage involves the understanding of the business goals and defining business metrics that can measure the project's success. For example, the following are some examples of business goals:

- Cost reduction for operational processes, such as document processing.
- Mitigation of business or operational risks, such as fraud and compliance.
- Product or service revenue improvements, such as better target marketing, new insight generation for better decision making, and increased customer satisfaction.

To measure the success, you may use specific business metrics such as the number of hours reduced in a business process, an increased number of true positive frauds detected, a conversion rate improvement from target marketing, or the number of churn rate reductions. This is an essential step to get right to ensure there is sufficient justification for an ML project and that the outcome of the project can be successfully measured. After you have defined the business goals and business metrics, you need to evaluate if there is an ML solution for the business problem. While ML has a wide scope of applications, it is not always an optimal solution for every business problem.

Data understanding and data preparation

The saying that "data is the new oil" holds particularly true for ML. Without the required data, you cannot move forward with an ML project. That's why the next step in the ML life cycle is **data acquisition, understanding, and preparation**. Based on the business problems and ML approach, you will need to gather and comprehend the available data to determine if you have the right data and data volume to solve the ML problem. For example, suppose the business problem to address is credit card fraud detection. In that case, you will need datasets such as historical credit card transaction data, customer demographics, account data, device usage data, and networking access data. Detailed data analysis is then necessary to determine if the dataset features and quality are sufficient for the modeling

tasks. You also need to decide if the data needs labeling, such as `fraud` or `not-fraud`. During this step, depending on the data quality, a significant amount of data wrangling might be performed to prepare and clean the data and to generate the dataset for model training and model evaluation, depending on the data quality.

Model training and evaluation

Using the training and validation datasets established, a data scientist must run a number of experiments using different ML algorithms and dataset features for feature selection and model development. This is a highly iterative process and could require a numerous run of data processing and model development runs to find the right algorithm and dataset combination for optimal model performance. In addition to model performance, factors such as data bias and model explainability may need to be considered to comply with internal or regulatory requirements. Prior to deployment into production, the model quality must be validated using the relevant technical metrics, such as the **accuracy score**. This is usually accomplished using a **holdout dataset**, also known as a **test dataset**, to gauge how the model performs on unseen data. It is crucial to understand which metrics are appropriate for model validation, as they vary depending on the ML problems and the dataset used. For example, model accuracy would be a suitable validation metric for a document classification use case if the number of document types is relatively balanced. However, model accuracy would not be a good metric to evaluate the model performance for a fraud detection use case – this is because the number of frauds is small and even if the model predicts `not-fraud` all the time, the model accuracy could still be very high.

Model deployment

After the model is fully trained and validated to meet the expected performance metric, it can be deployed into production and the business workflow. There are two main deployment concepts here. The first involves deployment of the model itself to be used by a client application to generate

predictions. The second concept is to integrate this prediction workflow into a business workflow application. For example, deploying the credit fraud model would either host the model behind an API for real-time prediction or as a package that can be loaded dynamically to support batch predictions. Moreover, this prediction workflow also needs to be integrated into business workflow applications for fraud detection, which might include the fraud detection of real-time transactions, decision automation based on prediction output, and fraud detection analytics for detailed fraud analytics.

Model monitoring

ML life cycle does not end with model deployment. Unlike software, whose behavior is highly deterministic since developers explicitly code its logic, an ML model could behave differently in production from its behavior in model training and validation. This could be caused by changes in the production data characteristics, data distribution, or the potential manipulation of request data. Therefore, model monitoring is an important post-deployment step for detecting model drift or data drift.

Business metric tracking

The actual business impact should be tracked and measured as an ongoing process to ensure the model delivers the expected business benefits. This may involve comparing the business metrics before and after the model deployment, or A/B testing where a business metric is compared between workflows with or without the ML model. If the model does not deliver the expected benefits, it should be re-evaluated for improvement opportunities. This could also mean framing the business problem as a different ML problem. For example, if churn prediction does not help improve customer satisfaction, then consider a personalized product/service offering to solve the problem.

ML challenges

Over the years, I have worked on many real-world problems using ML solutions and encountered different challenges faced by the different industries during ML adoptions. I often get the same question when working on ML projects: *We have a lot of data – can you help us figure out what insights we can generate using ML?* I refer to companies with this question as having the *business use case challenge*. Not being able to identify business use cases for ML is a very big hurdle for many companies. Without a properly identified business problem and its value proposition and benefit, it becomes difficult to initiate an ML project. In my conversations with different companies across their industries, data-related challenges emerge as frequent issue. This includes data quality, data inventory, data accessibility, data governance, and data availability. This problem affects both data-poor and data-rich companies and is often exacerbated by data silos, data security, and industry regulations. The shortage of data science and ML talent is another major challenge I have heard from many companies. Companies, in general, are having a tough time attracting and retaining top ML talents, which is a common problem across all industries. As ML platforms become more complex and the scope of ML projects increases, the need for other ML-related functions starts to surface. Nowadays, in addition to just data scientists, an organization would also need function roles for ML product management, ML infrastructure engineering, and ML operations management. Based on my experiences, I have observed that cultural acceptance of ML-based solutions is another significant challenge for broad adoption. There are individuals who perceive ML as a threat to their job functions, and their lack of knowledge in ML makes them hesitant to adopt these new methods in their business workflow. The practice of ML solutions architecture aims to help solve some of the challenges in ML. In the next section, we will explore ML solutions architecture and its role in the ML life cycle.

ML solutions architecture

When I initially worked with companies as an ML solutions architect, the landscape was quite different from what it is now. The focus was mainly on data science and modeling, and the problems at hand were small in scope.

Back then, most of the problems could be solved using simple ML techniques. The datasets were also small, and the infrastructure required was also not demanding. The scope of the ML initiative at these companies was limited to a few data scientists or teams. As an ML architect at that time, I primarily needed to have solid data science skills and general cloud architecture knowledge to get the job done. In the more recent years, the landscape of ML initiatives has become more intricate and multifaceted, necessitating involvement from a broader range of functions and personas at companies. My engagement has expanded to include discussions with business executives about ML strategies and organizational design to facilitate the broad adoption of AI/ML throughout their enterprises. I have been tasked with designing more complex ML platforms, utilizing a diverse range of technologies for large enterprises across many that meet stringent security and compliance requirements. ML workflow orchestration and operations have become increasingly crucial topics of discussion, and more and more companies are looking to train large ML models with enormous amounts of training data. The number of ML models trained and deployed by some companies has skyrocketed to tens of thousands from a few dozen models in just a few years. Furthermore, sophisticated and security-sensitive customers have sought guidance on topics such as ML privacy, model explainability, and data and model bias. As an ML solutions architect, I've noticed that the skills and knowledge required to be successful in this role have evolved significantly. Trying to navigate the complexities of a business, data, science, and technology landscape can be a daunting task. As an ML solutions architect, I have seen firsthand the challenges that companies face in bringing all these pieces together. In my view, ML solutions architecture is an essential discipline that serves as a bridge connecting the different components of an ML initiative. Drawing on my years of experience working with companies of all sizes and across diverse industries, I believe that an ML solutions architect plays a pivotal role in identifying business needs, developing ML solutions to address these needs, and designing the technology platforms necessary to run these solutions. By collaborating with various business and technology partners, an ML solutions architect can help companies unlock the full potential of their data and realize tangible benefits from their ML initiatives. The following figure

(Figure 1.7) illustrates the core functional areas covered by the ML solutions architecture.

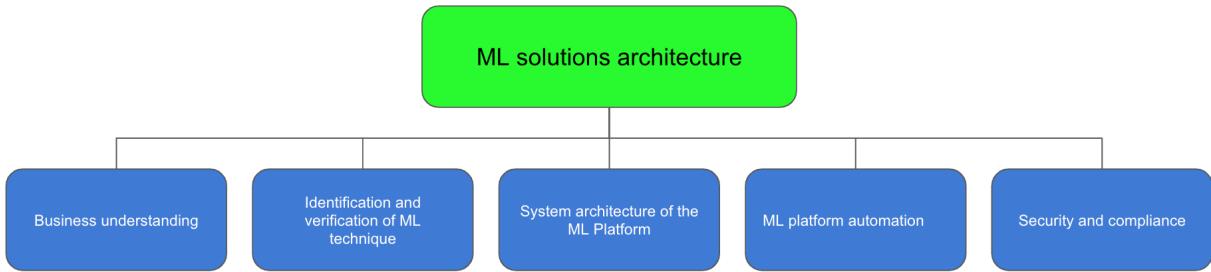


Figure 1.7: ML solutions architecture coverage

In the following sections, we will explore each of these areas in greater details:

- **Business understanding:** Business problem understanding and transformation using AI and ML.
- **Identification and verification of ML techniques:** Identification and verification of ML techniques for solving specific ML problems.
- **System architecture of the ML technology platform:** System architecture design and implementation of the ML technology platforms.
- **ML platform automation:** ML platform automation technical design.
- **Security and compliance:** Security, compliance, and audit considerations for the ML platform and ML models.

Business understanding and ML transformation

The goal of the business workflow analysis is to identify inefficiencies in the workflows and determine if ML can be applied to help eliminate pain points, improve efficiency, or even create new revenue opportunities. Picture this: you are tasked with improving a call center's operations. You know there are inefficiencies that need to be addressed, but you're not sure where to start. That's where business workflow analysis comes in. By analyzing the call center's workflows, you can identify pain points such as long customer wait times, knowledge gaps among agents, and the inability to

extract customer insights from call recordings. Once you have identified these issues, you can determine what data is available and which business metrics need to be improved. This is where ML comes in. You can use ML to create virtual assistants for common customer inquiries, transcribe audio recordings to allow for text analysis, and detect customer intent for product cross-sell and up-sell. But sometimes, you need to modify the business process to incorporate ML solutions. For example, if you want to use call recording analytics to generate insights for cross-selling or up-selling products, but there's no established process to act on those insights, you may need to introduce an automated target marketing process or a proactive outreach process by the sales team.

Identification and verification of ML techniques

Once you have come up with a list of ML options, the next step is to determine if the assumption behind ML approach is valid. This could involve conducting a simple **Proof of Concept (POC)** modeling to validate the available dataset and modeling approach, or technology POC using pre-built AI services, or testing of ML frameworks. For example, you might want to test the feasibility of text transcription from audio files using an existing text transcription service or build a customer propensity model for a new product conversion from a marketing campaign. It is worth noting that ML solutions architecture does not focus on developing new machine algorithms, a job best suited for applied data scientists or research data scientists. Instead, ML solutions architecture focuses on identifying and applying ML algorithms to address a range of ML problems such as predictive analytics, computer vision, or natural language processing. Also, the goal of any modeling task here is not to build production-quality models, but rather to validate the approach for further experimentations by full-time applied data scientists.

System architecture design and implementation

The most important aspect of ML solutions architecture role is the technical architecture design of the ML platform. The platform will need to provide the technical capability to support the different phases of the ML cycle and

personas, such as data scientists and ops engineers. Specifically, an ML platform needs to have the following core functions:

- **Data explorations and experimentation:** Data scientists use ML platforms for data exploration, experimentation, model building, and model evaluation. ML platforms need to provide capabilities such as data science development tools for model authoring and experimentation, data wrangling tools for data exploration and wrangling, source code control for code management, and a package repository for library package management.
- **Data management and large-scale data processing:** Data scientists or data engineers will need the technical capability to store, access, and process large amounts of data for cleansing, transformation, and feature engineering.
- **Model training infrastructure management:** ML platforms will need to provide model training infrastructure for different modeling training using different types of computing resources, storage, and networking configurations. It also needs to support different types of ML libraries or frameworks, such as **scikit-learn**, **TensorFlow**, and **PyTorch**.
- **Model hosting/serving:** ML platforms will need to provide the technical capability to host and serve the model for prediction generations, either for real-time, batch, or both.
- **Model management:** Trained ML models will need to be managed and tracked for easy access and lookup, with relevant metadata.
- **Feature management:** Common and reusable features will need to be managed and served for model training and model serving purposes.

ML platform workflow automation

A key aspect of ML platform design is **workflow automation** and **continuous integration/continuous deployment (CI/CD)**. ML is a multi-step workflow – it needs to be automated, which includes data processing, model training, model validation, and model hosting. Infrastructure provisioning automation and self-service is another aspect of automation design. Key components of workflow automation include the following:

- **Pipeline design and management:** The ability to create different automation pipelines for various tasks, such as model training and model hosting.
- **Pipeline execution and monitoring:** The ability to run different pipelines and monitor the pipeline execution status for the entire pipeline and each of the steps.
- **Model monitoring configuration:** The ability to monitor the model in production for various metrics, such as data drift (where the distribution of data used in production deviates from the distribution of data used for model training), model drift (where the performance of the model degrades in the production compared with training results), and bias detection (the ML model replicating or amplifying bias towards certain individuals).

Security and compliance

Another important aspect of ML solutions architecture is the security and compliance consideration in a sensitive or enterprise setting:

- **Authentication and authorization:** The ML platform needs to provide authentication and authorization mechanisms to manage the access to the platform and different resources and services.
- **Network security:** The ML platform needs to be configured for different network security to prevent unauthorized access.
- **Data encryption:** For security-sensitive organizations, data encryption is another important aspect of the design consideration for the ML platform.
- **Audit and compliance:** Audit and compliance staff need the information to help them understand how decisions are made by the predictive models if required, the lineage of a model from data to model artifacts, and any bias exhibited in the data and model. The ML platform will need to provide model explainability, bias detection, and model traceability across the various datastore and service components, among other capabilities.

Testing your knowledge

Great job! You have reached to the end of the chapter. Now, let's put your newly acquired knowledge to the test and see if you've understood and retained the information presented. Take a look at the list of the following scenarios and determine which of the three ML types can be applied (*supervised*, *unsupervised*, or *reinforcement*):

1. There is a list of online feedback on products. Each comment has been labeled with a sentiment class (for example, `positive`, `negative`, `neutral`). You have been asked to build an ML model to predict the sentiment of new feedback.
2. You have historical house pricing information and details about the house, such as zip code, number of bedrooms, house size, and house condition. You have been asked to build an ML model to predict the price of a house.
3. You have been asked to identify potentially fraudulent transactions on your company's e-commerce site. You have data such as historical transaction data, user information, credit history, devices, and network access data. However, you don't know which transaction is fraudulent or not.

Take a look at the following questions on the ML life cycle and ML solutions architecture to see how you would answer them:

1. There is a business workflow that processes a request with a set of well-defined decision rules, and there is no tolerance to deviate from the decision rules when making decisions. Should you consider ML to automate the business workflow?
2. You have deployed an ML model into production. However, you do not see the expected improvement in the business KPIs. What should you do?
3. There is a manual process that's currently handled by a small number of people. You found an ML solution that can automate this process, however, the cost of building and running the ML solution is higher

than the cost saved from automation. Should you proceed with the ML project?

4. As an ML solutions architect, you have been asked to validate an ML approach for solving a business problem. What steps would you take to validate the approach?

Summary

You now have a solid understanding of various concepts such as AI, ML, and the essential steps of the end-to-end ML life cycle. Additionally, you have gained insight into the core functions of ML solutions architecture and how it plays a crucial role in the success of an ML project. With your newfound knowledge, you can differentiate between different types of ML and identify their application in solving business problems. Moreover, you have learned that it is crucial to have a deep understanding of business and data to achieve success in an ML project, besides modeling and engineering. Lastly, you have gained an understanding of the significance of ML solutions architecture and how it fits into the ML life cycle. In the upcoming chapter, we will dive into various ML use cases across different industries, such as financial services and media and entertainment, to gain further insights into the practical applications of ML.

2 Business Use Cases for Machine Learning

Join our book community on Discord

<https://packt.link/EarlyAccessCommunity>



As a **machine learning (ML)** practitioner, It is essential for me to develop a deep understanding of different businesses to have effective conversations with the business and technology leaders. This should not come as a surprise since the ultimate goal for any **machine learning solution architecture (ML solution architecture)** is to solve practical business problems with science and technology solutions. Therefore, one of the main areas of focus for ML solution architecture is to develop a broad understanding of different business domains, workflows, and relevant data. Without this understanding, it would be challenging to make sense of the data and design and develop practical ML solutions for business problems. In this chapter, we will explore various real-world ML use cases across several industry verticals. We will examine the key business workflows and challenges faced by industries such as financial services and retail, and how ML technologies can help solve these challenges. The aim of this chapter is not to make you to an expert in any particular industry or its ML use case and techniques, but rather to make you aware of real-world ML use cases in the context of business requirements and workflows. After reading this chapter, you will be able to apply similar thinking in your line of business and be able to identify ML solutions. Specifically, we will cover the following topics in this chapter:

- ML use cases in financial services
- ML use cases in media and entertainment
- ML use cases in healthcare and life sciences
- ML use cases in manufacturing
- ML use cases in retail
- ML use cases in automotive

ML use cases in financial services

The **Financial Services Industry (FSI)** has always been at the forefront of technological innovation, and ML adoption is no exception. In recent years, we have seen a range of ML solutions being implemented across different business functions within financial services. For example, in capital markets, ML is being used across front, middle, and back offices to aid investment decisions, trade optimization, risk management, and transaction settlement processing. In insurance, companies are using ML to streamline underwriting, prevent fraud, and automate claim management. While in banking, banks are using it to improve customer experience, combat fraud, and facilitate loan approval decisions. In the following sections, we will explore different core business areas within financial services and how ML can be applied to overcome some of these business challenges.

Capital markets front office

In finance, the front office is the revenue-generating business area that include customer-facing roles such as securities sales, traders, investment bankers, and financial advisors. Front office departments offer products and services such as **Merger and Acquisition (M&A)** and IPO advisory, wealth management, and the trading of financial assets such as equity (e.g., stocks), fixed income (e.g., bonds), commodities (e.g., oil), and currency products. Let's now examine some specific business functions in the front office area.

Sales trading and research

In sales trading, a firm's sales staff monitors investment news such as earnings reports or M&A activities to identify investment opportunities for their institutional clients. The trading staff then execute the trades for their clients, also known as agency trading. Additionally, trading staff can execute trades for their firm, which is known as **prop trading**. As trading staff often deal with large quantities of securities, it is crucial to optimize the trading strategy to acquire the shares at favorable prices without driving up prices. Research teams support sales and trading staff by analyzing equities and fixed income assets and providing recommendations. Algorithmic trading is another type of trading that uses a computer to execute trades automatically based on predefined logic and market conditions. The following diagram illustrates the business flow of a sales trading desk and how different players interact to complete a trading activity:

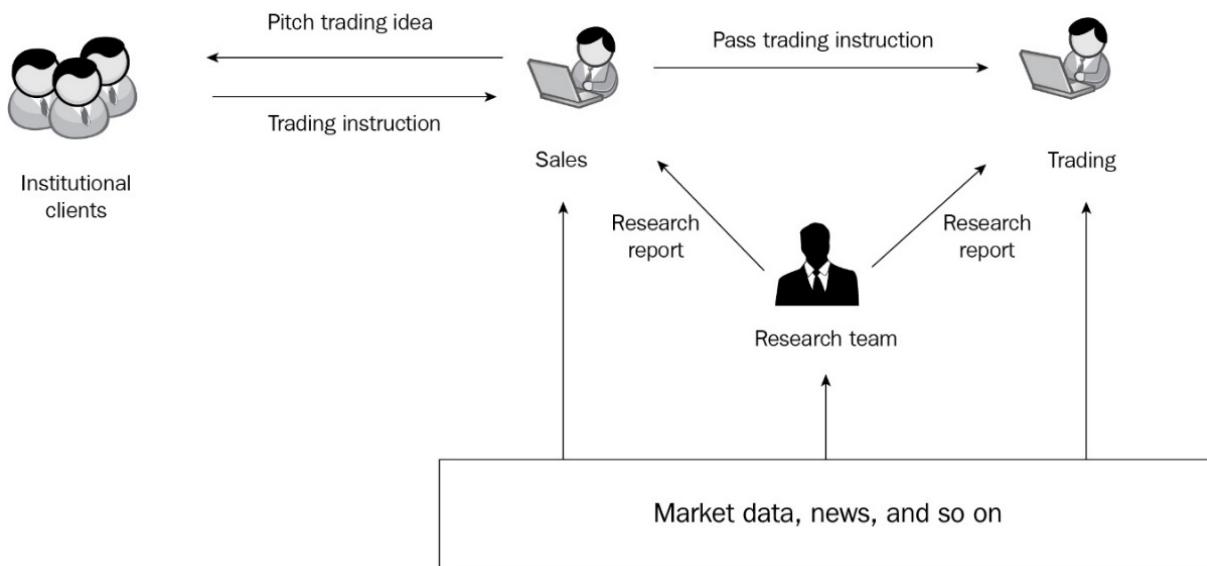


Figure 2.1: Sales, trading, and research

In the domain of sales trading and research, there are several core challenges that professionals in this industry face on a regular basis. These challenges revolve around generating accurate market insights, making informed investment decisions, and achieving optimal trading executions. The followings are examples of these challenges:

- Tight timeline faced by research analysts to deliver research reports.
- Collecting and analyzing large amounts of market information to develop trading strategies and make trading decisions.
- Monitoring the markets continuously to adjust trading strategies.
- Achieving the optimal trading at the preferred price without driving the market up or down.

Sales trading and research offer numerous opportunities for ML. By leveraging **Natural language processing (NLP)** and increasingly Large Language (LLM) models, key entities such as people, events, organizations, and places can be automatically extracted from various data sources such as SEC filing, news announcements, and earnings call transcripts. NLP can also help discover relationships between entities and assess market sentiments toward a company and its stock by analyzing large amounts of news, research reports, and earning calls to inform trading decisions.

Natural language generation (NLG) powered by Large Language Models can assist with narrative writing and report generation, while computer vision has been used to help identify market signals from alternative data sources such as satellite images to understand business patterns such as retail traffic. In trading, ML models can sift through large amounts of data to discover patterns to inform trading strategies, such as pair trading, using data points such as company fundamentals, trading patterns, and technical indicators. In trade execution, ML models can help estimate trading cost and identify optimal trading execution strategies and execution path to minimize costs and optimize profits. Financial services companies generate massive amounts of time series data, such as prices of different financial instruments, which can be used to discover market signals and estimate market trends. As a result, ML has been adopted for use cases such as financial time series classification and forecasting financial instruments and economic indicators.

Investment banking

When corporations, governments, and institutions need access to capital to fund business operations and growth, they engage investment bankers for

capital raising (e.g, selling of stocks or bonds) services. The following diagram illustrates the relationship between investment bankers and investors. In addition to capital raising, the investment bankers also engage in M&A advisory to assist their clients in negotiating and structuring merger and acquisition deals from start to finish. Investment banking staff take on many activities such as financial modeling, business valuation, pitch book generation, and transaction document preparation to complete and execute an investment banking deal. Additionally, they are responsible for general relationship management and business development management activities.

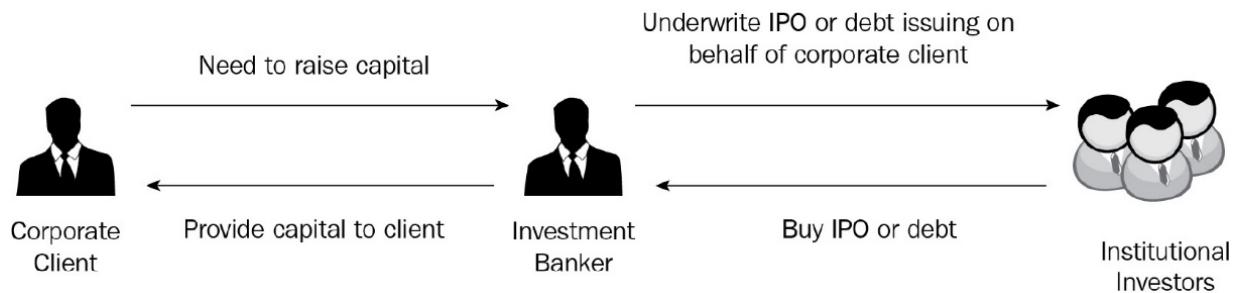


Figure 2.2: Investment banking workflow

The investment banking workflow poses a significant challenge in searching and analyzing large amounts of **structured** (earning, cashflow, estimates) and **unstructured** data (annual reports, filing, news, and internal documents). Typical junior bankers spend many hours searching for documents that might contain useful information and manually extract information from the documents to prepare pitch books or perform financial modeling. To tackle this labor-intensive problem, investment banks have been exploring and adopting ML solutions. One such solution is using NLP to extract structured tabular data automatically from large amounts of PDF documents. Specifically, **named entity recognition (NER)** techniques can help with automatic entity extraction from documents. ML-based reading comprehension and question answering technology can assist bankers in finding relevant information from large volumes of text quickly and accurately using natural human questions instead of simple text string matching. Documents can also be automatically tagged with metadata and classified using the ML technique to improve document management and

information retrievals. Additionally, ML can help solve other challenges in investment banking, such as linking company identifiers from different data sources and resolving different variations of company names.

Wealth management

The **wealth management (WM)** business involves advising their clients with wealth planning and structuring to grow and preserve their clients' wealth. Unlike the investment advisory-focused brokerage firms, WM firms also offer tax planning, wealth preserving, and estate planning to meet their client's more complex financial planning goals. WM firms engage clients to understand their life goals and spending patterns and design customized financial planning solutions for their clients. However, WM firms face various challenges in their operations, such as:

- WM clients are demanding more holistic and personalized financial planning strategies for their WM needs.
- WM clients are becoming increasingly tech-savvy, and many are demanding new channels of engagement in addition to direct client-advisor interactions.
- WM advisors need to cover increasingly more clients while maintaining the same personalized services and planning.
- WM advisors need to keep up with market trends, diverse client needs, and increasingly complex financial products and service portfolio to meet client needs.

WM firms are adopting ML-based solutions to offer more personalized services to their clients. By analyzing clients' transaction history, portfolio details, conversation logs, investment preferences, and life goals, ML models are built to recommend the most suitable investment products and services. These models take into account the clients' likelihood of accepting an offer and business metrics such as expected value to suggest the next best action. This enables wealth management firms to offer tailored financial planning solutions to their clients. The following diagram illustrates the concept of the Next Best Action method:

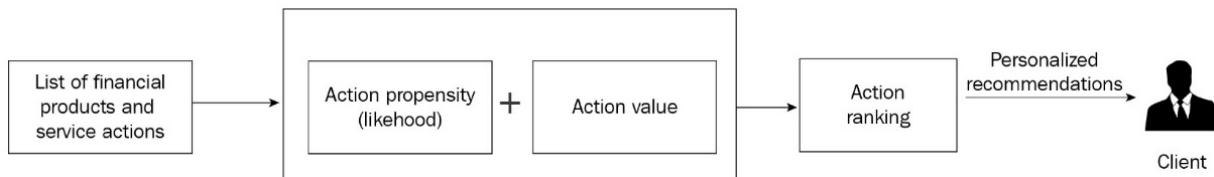


Figure 2.3: Next Best Action recommendation

WM firms are also increasingly leveraging AI and ML to enhance client engagement and experience, automate routine tasks, and equip financial advisors (FAs) with the right knowledge during client engagements. For instance, firms are building virtual assistants that provide personalized answers to client inquiries and automatically fulfill their requests. FAs are being equipped with AI-based solutions that can transcribe audio conversations to text for text analysis, assess clients' sentiment, and alert FAs of potential customer churn. Additionally, intelligent search and question-answering techniques are being adopted to enable FAs to quickly and accurately find relevant information during client engagements.

Capital markets back office operations

The back office is the backbone of financial services companies. While it may not be client-facing, it handles critical support activities such as trade settlement, record keeping, and regulatory compliance. As a result, it's an area that has been quick to adopt machine learning. With the financial benefits and cost-saving it can bring, not to mention its ability to improve regulatory compliance and internal controls, it's no surprise that ML is transforming the back office. Let's explore some of the business processes where ML can make a significant impact.

Net Asset Value review

Financial services companies that offer Mutual Funds and ETFs need to accurately reflect the values of the funds for trading and reporting purposes. They use a **Net Asset Value (NAV)** calculation, which is the value of an entity's assets minus its liability, to represent the value of the fund. NAV is

the price at which an investor can buy and sell the fund. Every day after the market closes, fund administrators must calculate the NAV price with 100% accuracy, which involves five critical steps:

1. Stock reconciliation
2. Reflection of any corporate actions
3. Pricing the instrument
4. Booking, calculating, and reconciling fees and interest accruals, as well as cash reconciliation
5. NAV/price validation

The NAV review process is depicted in the following diagram:



Figure 2.4: Net Asset Value review process

Step 5 is the most vital because if it is done incorrectly, the fund administrator could be liable, which can result in monetary compensations to investors. However, traditional methods of flagging exceptions using fixed thresholds often result in a high number of false positives, wasting analysts' time. Due to the large volumes of data involved in the investigation and review process, including instrument prices, fees, interest, assets, cash positions, and corporate actions data, efficient and accurate methods are essential. The main objective of the NAV validation step is to detect pricing exceptions, which can be treated as an anomaly detection challenge. To identify potential pricing irregularities and flag them for further human investigation, financial services companies have implemented ML-based anomaly detection solutions. This approach has demonstrated a substantial reduction in false positives and saved a significant amount of time for human reviewers.

Post-trade settlement failure prediction

After the front office executes a trade, several post-trade processes must be completed to finalize the trade, such as settlement and clearance. During post-trade settlement, buyers and sellers compare trade details, approve the transaction, update records of ownership, and arrange for securities and cash to be transferred. Although, most trade settlements are handled automatically using **straight-through processing**. Some trade settlements may fail due to various reasons, such as a seller's failure to deliver securities. When this occurs, brokers may need to use their reserves to complete the transaction. To ensure the stockpile is set at the correct level so that valuable capital can be used elsewhere, predicting settlement failure is critical. The following diagram illustrates the workflow for the trade where buyers and sellers buy and sell their securities at exchange through their respective brokerage firms:

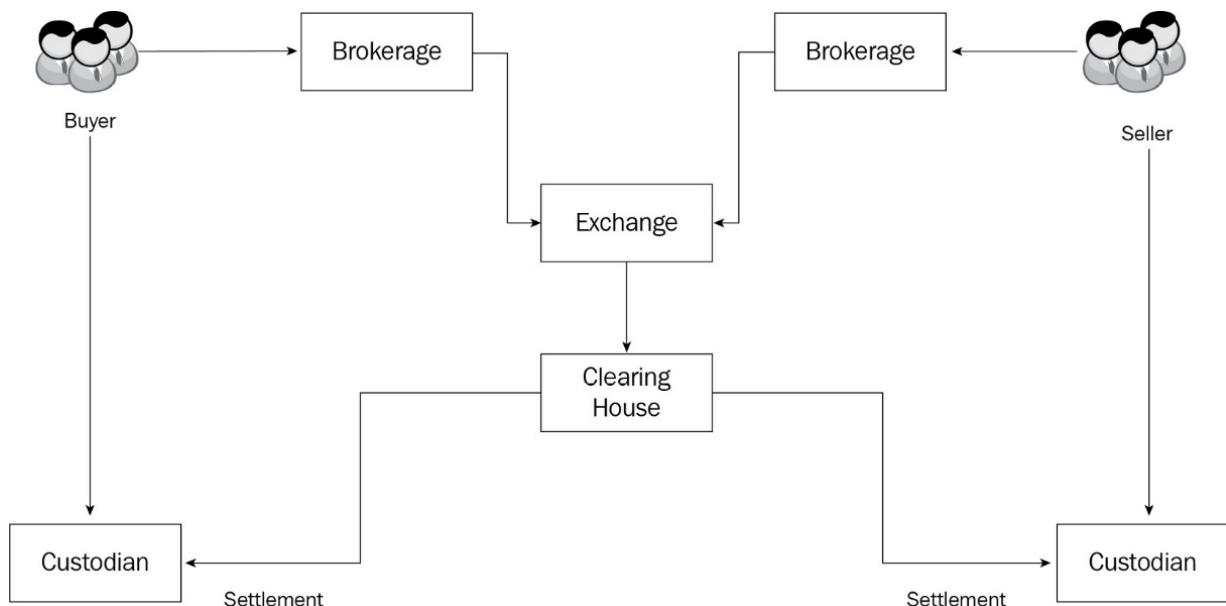


Figure 2.5: Trading workflow

After the trade is executed, a clearing house such as DTCC would handle the clearance and settlement of the trades with the respective custodians for the buyer and sellers. Brokerage houses aim to optimize transaction rates and reduce capital expenditure costs by maintaining the right amount of

stockpile reserve. To achieve this, ML models are utilized to predict trade failures early in the process. With these predictions, brokers can take preventive or corrective actions to prevent or resolve trade failures.

Risk management and fraud

The middle office of financial services firms, including investment banks and commercial banks, encompasses risk management and fraud prevention. Due to their significant financial and regulatory implications, these areas are among the top areas for ML adoption in financial services. ML has many use cases in fraud prevention and risk management, such as detecting anti-money laundering, monitoring trade activities, identifying credit card transaction fraud, and uncovering insurance claim fraud. In the following sections, we will examine some of these use cases in more detail.

Anti-money laundering

Financial institutions are obligated to prevent money laundering by detecting activities that aid illegal money laundering. Anti-money laundering (AML) regulations require financial services companies to devote substantial resources to combat AML activities. Traditionally, rule-based systems have been used to detect AML activities, but they have a limited view and can only detect well-known frauds from the past. Furthermore, it is challenging to include a large number of features to be evaluated in a rule-based system, and it is difficult to keep the rules up to date with new changes. Machine learning-based solutions have been leveraged in multiple areas of AML, such as:

- Network link analysis to reveal the complex social and business relationships among different entities and jurisdictions.
- Clustering analysis to find similar and dissimilar entities to spot trends in criminal activity patterns.
- Deep learning-based predictive analytics to identify criminal activity.
- NLP to gather as much information as possible for the vast number of entities from unstructured data sources.

The following diagram illustrates the data flow for AML analysis, the reporting requirements for regulators, and internal risk management and audit functions:

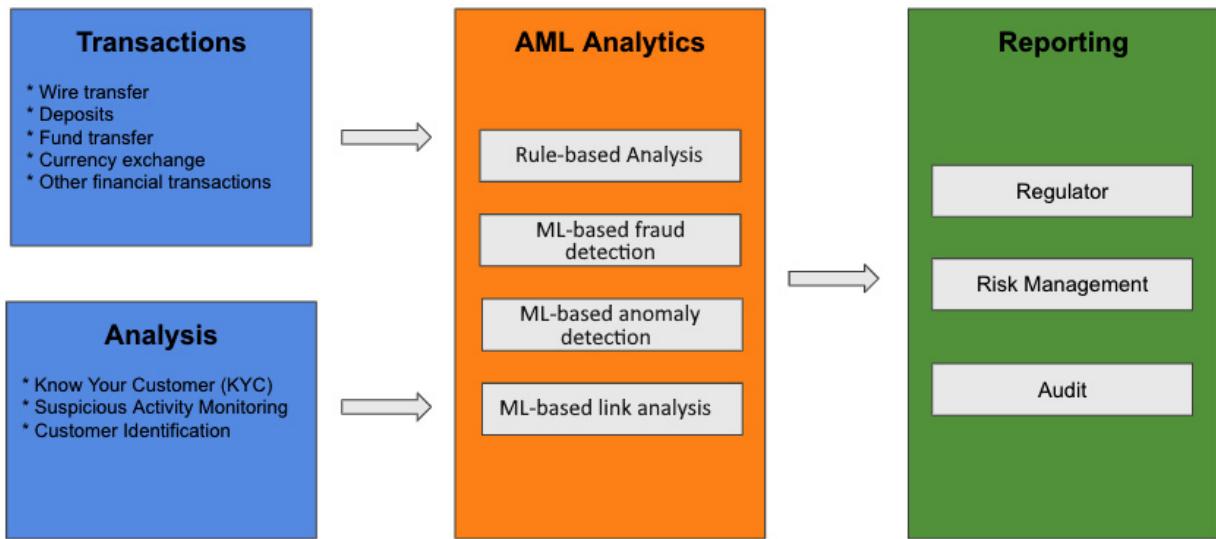


Figure 2.6: Anti-money laundering detection flow

An AML platform takes data from many different sources, including transactions data and internal analysis data such as **Know Your Customer (KYC)** and **Suspicious Activity** data. This data is processed and fed into different rule and ML-based analytics engines to monitor fraudulent activities. The findings can be then sent to internal risk management and auditing, as well as regulators.

Trade surveillance

Traders at financial firms are intermediaries who buy and sell securities and other financial instruments on behalf of their clients. They execute orders and advise clients on entering and exiting financial positions. To prevent market abuse by traders or financial institutions, **trade surveillance** is employed to identify and investigate potential market abuse. Examples of market abuse include market manipulation, such as the dissemination of false and misleading information, manipulating trading volumes through large amounts of wash trading, and insider trading through the disclosure of

non-public information. Financial institutions are required to comply with market abuse regulations such as **Market Abuse Regulation (MAR)**, **Markets in Financial Instruments Directive II (MiFID II)**, and other internal compliance to protect themselves from reputational and financial damages. Enforcing trade surveillance can be challenging due to high noise/signal ratios and many false positives, which increases the cost of case processing and investigations. One typical approach to approach abuse detection is to build complex rule-based systems with different fixed thresholds for decision making. There are multiple ways to frame trade surveillance problems as ML problems, including:

- Framing the abuse detection of activities as a classification problem to replace rule-based systems.
- Framing data extraction information such as entities (for example, restricted stocks) from unstructured data sources (for example, emails and chats) as NLP entity extraction problems.
- Transforming entity relationship analysis (for example, trader-trader collaborations in market abuse) as machine learning-based network analysis problems.
- Treating abusive behaviors as anomalies and using unsupervised ML techniques for anomaly detection.

Many different datasets can be useful for building ML models for trade surveillance such as **P&L** information, positions, order book details, e-communications, linkage information among traders and their trades, market data, trading history, and details such as counterparty details, trade price, order type, and exchanges. The following diagram illustrates the typical data flow and business workflow for trade surveillance management within a financial services company:

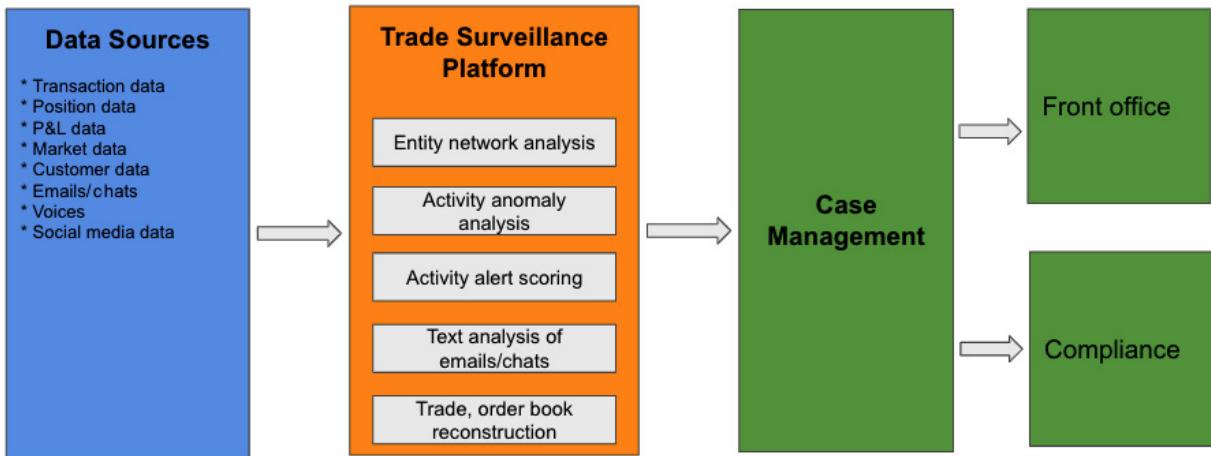


Figure 2.7: Trade surveillance workflow

A trade surveillance system monitors many different data sources, and it feeds its findings to both the front office and compliance department for further investigation and enforcement.

Credit risk

Banks face the potential risk of borrowers not being able to make required loan payments when issuing loans to businesses and individuals. This results in financial loss for banks, including both principal and interest from activities such as mortgage and credit card loans. To mitigate this default risk, banks utilize credit risk modeling to evaluate the risk of making a loan, focusing on two main aspects:

- The probability that the borrower will default on the loan.
- The impact on the lender's financial situation.

Traditional human-based reviews of loan applications are slow and error-prone, resulting in high loan processing costs and lost opportunities due to incorrect and slow loan approval processing. The following diagram depicts a typical business workflow for credit risk assessment and its various decision points within the process:

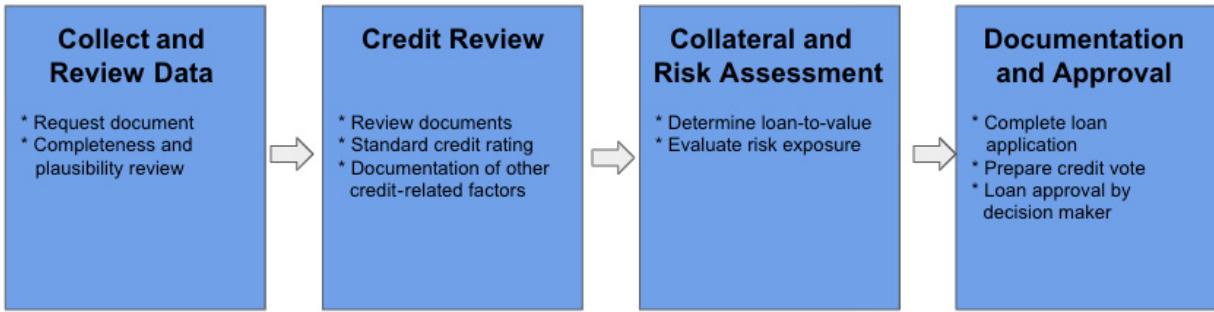


Figure 2.8: Credit risk approval workflow

To reduce credit risk associated with loans, many banks have widely adopted ML techniques to predict loan default and associated risk scores more accurately and quickly. The credit risk management modeling process involves collecting financial information from borrowers, such as income, cash flow, debt, assets and collaterals, the utilization of credits, and other information such as loan type and loan payment behaviors. However, this process can involve analyzing large amounts of unstructured data from financial statements. To address this challenge, machine learning-based solutions such as **Optical Character Recognition (OCR)** and NLP information extraction and understanding have been widely adopted for automated intelligence document processing.

Insurance

The insurance industry comprises various sub-sectors, each offering distinct insurance products, such as life insurance, property and casualty insurance, and accident and health insurance. Apart from insurance companies, insurance technology providers also play a critical role in the industry. Most insurance companies have two primary business processes, namely, insurance underwriting and insurance claim management.

Insurance underwriting

Insurance companies evaluate the risks of offering insurance coverage to individuals and assets through a process known as **insurance underwriting**. Using actuarial data and insurance software, insurance

companies determine the appropriate insurance premium for the risks they are willing to undertake. The underwriting process varies depending on the insurance products offered. For instance, the steps involved in underwriting property insurance are typically as follows:

1. The customer files an insurance application through an agent or insurance company directly.
2. The underwriter at the insurance company assesses the application by considering different factors such as the applicant's loss and insurance history, actuarial factors to determine whether the insurance company should take on the risk, and what the price and premium should be for the risk. Then, they make an additional adjustment to the policy, such as coverage amount and deductibles.
3. If the application is accepted, then an insurance policy is issued.

During the underwriting process, a large amount of data needs to be collected and reviewed by an underwriter, who estimates the risk of a claim based on the data and personal experience to determine a justifiable premium. However, human underwriters are limited in their ability to review only a subset of data and can introduce personal bias into the decision-making process. In contrast, ML models can analyze vast amounts of data and make more accurate, data-driven decisions regarding risk factors such as claim probability and outcome, while also making faster decisions than human underwriters. Additionally, ML models can utilize large amounts of historical data and risk factors to generate recommended premiums for policies, reducing the amount of time needed for assessment.

Insurance claim management

Insurance claim management involves the process of evaluating claims made by policyholders and providing compensation for the losses incurred, as specified in the policy agreement. The specific steps in the claim process can vary depending on the type of insurance. For example, in the case of property insurance, the following steps are usually followed:

1. The insured person submits a claim, along with supporting evidence like photographs of the damage and a police report (in case of automobiles).
2. An adjuster is assigned by the insurance company to assess the extent of the damage.
3. The adjuster evaluates the damage, carries out fraud assessments, and sends the claim for payment approval.

Some of the main challenges that are faced in the insurance claim management process are as follows:

- Time-consuming manual effort is needed for the damaged/lost item inventory process and data entry.
- The need for speedy claim damage assessment and adjustment.
- Insurance fraud.

Insurance companies collect a lot of data during the insurance claim process, such as property details, items damage data and photos, the insurance policy, the claims history, and historical fraud data.

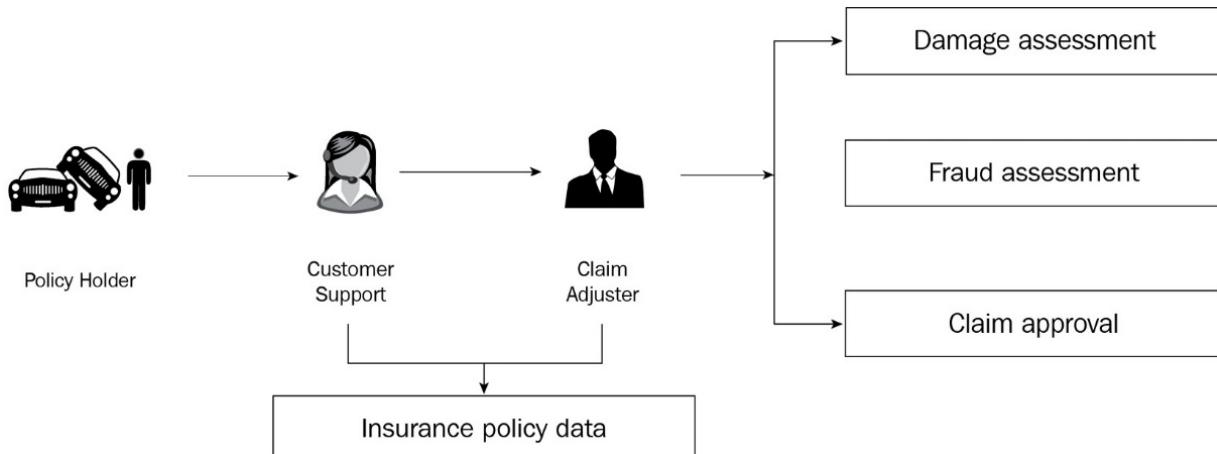


Figure 2.9: Insurance claim management workflow

ML can help automate manual processes, such as data extraction from documents and identification of insured objects from pictures, which can reduce the amount of manual effort required for data collection. For damage assessment, ML can be used to estimate the cost of repair and replacement,

which can speed up the claim processing. Additionally, ML can be used to detect exceptions in insurance claims and predict potential fraud, which can help to identify cases for further investigation in the fight against insurance fraud.

ML use cases in media and entertainment

The **media and entertainment (M&E)** industry encompasses the production and distribution of various forms of content, such as films, television, streaming content, music, games, and publishing. The industry has undergone significant changes due to the growing prevalence of streaming and **over-the-top (OTT)** content delivery over traditional broadcasting. M&E customers, having access to ever-increasing selection of media content, are shifting their consumption habits and demanding more personalized and enhanced experiences across different devices, anytime, anywhere. The industry is also characterized by intense competition, and to remain competitive, M&E companies need to identify new monetization channels, improve user experience, and improve operational efficiency. The workflow for media production and distribution is illustrated in the diagram below:

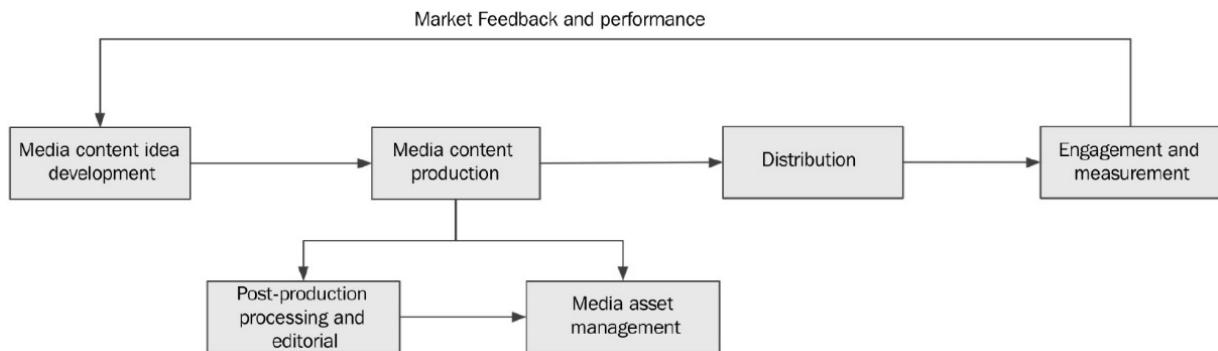


Figure 2.10: Media production and distribution workflow

In recent years, I have seen M&E companies increasingly adopting ML in the different stages of the media life cycle, such as content generation and content distribution, to improve efficiency and spur business growth. For instance, ML has been used to enhance content management and search,

develop new content development, optimize monetization, and enforce compliance and quality control.

Content development and production

During the initial planning phase of the film production life cycle, content producers need to make decisions on the next content based on factors such as estimated performance, revenue, and profitability. To aid this process, filmmakers have adopted ML-based predictive analytics models to help predict the popularity and profitability of new ideas by analyzing factors such as casts, scripts, the past performance of different films, and target audience. This allows producers to quickly eliminate ideas with limited market potential and focus their effort on developing more promising and profitable ideas. To support personalized content viewing needs, content producers often segment long video content into smaller micro-segments around certain events, scenes, or actors, so that they can be distributed individually or repackaged into something more personalized to individual preferences. This ML-based approach can be used to create video clips by detecting elements such as scenes, actors, and events for the different target audiences with varying tastes and preferences.

Content management and discovery

M&E companies with vast digital content assets need to curate their content to create new content for new monetization opportunities. To support this, these companies need rich metadata for the digital assets to enable different content to be searched and discovered. Consumers also need to search for content easily and accurately for different usages, such as for personal entertainment or research. Without metadata tagging, discovering relevant content is quite challenging. As part of digital asset management workflow, many companies hire humans to review and tag this content with meaningful metadata for discovery. However, manual tagging is very costly and time-consuming, leading to insufficient metadata for effective content management and discovery. Computer vision models can automatically tag image and video content for items such as objects, genres, people, places, or themes. ML models can also interpret the meaning of textual content such

as topics, sentiment, entities, and sometimes video. Audio content can be transcribed using ML techniques into text for additional text analysis. Machine learning-based text summarization can help you summarize long text as part of the content metadata generation. The following diagram illustrates where ML-based analysis solutions can be incorporated into the media asset management flow:

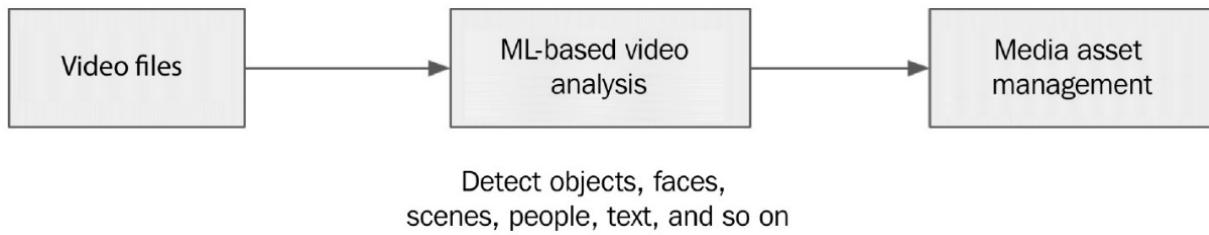


Figure 2.11: ML-based media analysis workflow

Machine learning-based solutions are being increasingly adopted by M&E companies to streamline media asset management workflows. Overall, these solutions can lead to significant time and cost saving for M&E companies while improving user experience for consumers.

Content distribution and customer engagement

Nowadays, media content such as films and music are increasingly being distributed through digital **video on demand (VOD)** and live streaming on different devices, bypassing traditional media such as DVDs and broadcasting, providing consumers with a variety of media content choices. As a result, media companies are facing challenges in customer acquisition and retention. To keep users engaged and on their platforms, M&E companies are focusing on highly personalized product features and content. One effective way to achieve highly personalized engagement is through content recommendation engine, which use viewing and engagement behavior data to train ML models that target individuals based on their preferences and viewing patterns. This allows for a diverse range of media content to be recommended to users, including videos, music, and games.

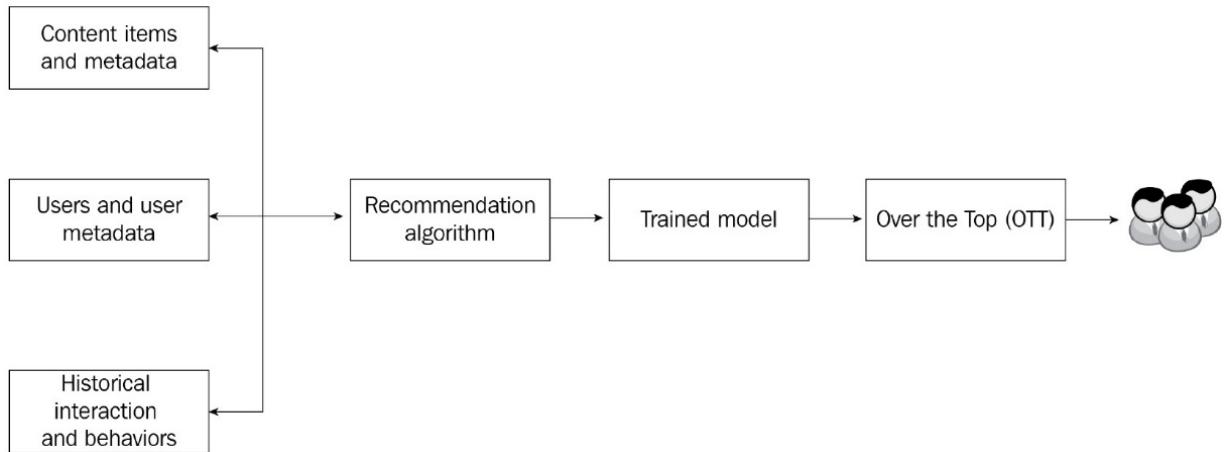


Figure 2.12: Recommendation ML model training

Recommendation technologies have been around for many years and have improved greatly over the years. Nowadays, recommendation engines can learn patterns using multiple data inputs, including historical interactions, sequential patterns, and the metadata associated with the users and content. Modern recommendation engines can also learn from the user's real-time behaviors/decisions and make dynamic recommendations based on real-time user behaviors and decisions.

ML use cases in healthcare and life sciences

The healthcare and life science industry is one of the largest and most important industries in the world, serving millions of people globally. The industry encompasses a wide range of sectors, each with its own unique set of challenges and opportunities. One of the most significant sectors within healthcare and life science is the drugs sector, which includes biotechnology firms, pharmaceutical companies, and manufacturers of genetics drugs. These companies are responsible for developing and producing medications to treat various illnesses and diseases, ranging from minor ailments to life-threatening conditions. They invest heavily in research and development to discover new drugs and therapies, often requiring significant financial resources and years of clinical trials before a product can be brought to market. Another important sector within healthcare and life science is the medical equipment industry, which manufactures a wide range of products

ranging from standard equipment such as syringes and bandages to hi-tech equipment such as MRI machines and surgical robots. These companies are at the forefront of innovation, constantly developing new technologies to improve patient outcomes and advance medical practices. Managed healthcare is another critical sector within the healthcare and life science industry. These companies provide health insurance policies, covering medical costs for their policyholders. This sector faces many challenges, such as rising healthcare costs and changing regulations, and it requires careful management and planning to provide affordable and effective coverage to policyholders. Health facilities, such as hospitals, clinics, and labs, are another significant sector within healthcare and life science. These facilities provide medical care and services to patients, ranging from routine check-ups to complex surgeries. They require significant resources to operate, such as skilled medical staff, state-of-the-art equipment, and advanced technologies. Government agencies, such as the Centers for Disease Control and Prevention (CDC) and the Food and Drug Administration (FDA), play a critical role in regulating and overseeing the healthcare and life science industry. They are responsible for ensuring the safety and efficacy of drugs and medical devices, monitoring public health issues, and developing policies to promote public health and safety. Top of Form In recent years, the healthcare and life science industry has seen a significant increase in the adoption of AI and ML. These technologies have been used to address complex challenges in the industry, such as improving patient outcomes, reducing costs, and accelerating drug discovery and development. With the availability of large amounts of health data, including electronic health records, genomics data, and medical imaging, ML algorithms can extract meaningful insights and patterns to inform clinical decision-making, disease diagnosis, and treatment planning. In this way, ML is transforming the healthcare and life science industry, enabling practitioners and researchers to make more informed decisions and improve patient outcomes.

Medical imaging analysis

Medical imaging is the process and technique of creating a visual representation of the human body for medical analysis. Medical

professionals, such as radiologists and pathologists, use medical imaging to assist with medical condition assessments and prescribe medical treatments. However, the increasing demand for medical imaging analysis has led to a shortage of qualified professionals to review these images. This challenge has been partially addressed through the adoption of ML in medical imaging analysis. One ML-based solution involves treating medical imaging analysis as a computer vision object detection and classification problem. For example, in the case of cancer cell detection, cancerous tissues can be identified and labeled in the existing medical images as training data for computer vision algorithms. Once trained, these models can be used to automate the screening of a large number of X-ray images, highlighting those that are important for the pathologists to review. This approach has the potential to improve the efficiency and accuracy of medical imaging analysis, reducing the workload of medical professionals and improving patient outcomes. The following diagram illustrates the process of training a computer vision model using labeled image data in medical imaging analysis:

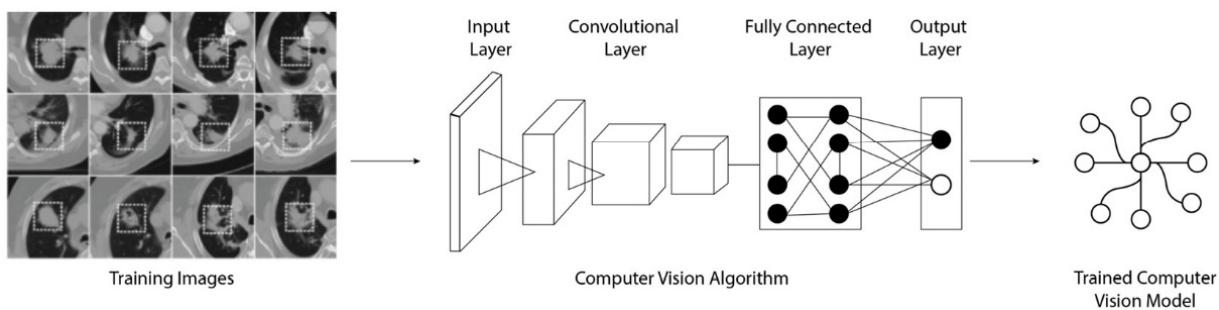


Figure 2.13: Using computer vision for cancer detection

Medical image analysis can be enhanced by combining image data with other clinical data, such as a patient's medical history, laboratory test results, and genetic data. This combination of data can improve the accuracy of medical diagnosis and enable early detection of diseases. For example, in the case of breast cancer, medical imaging can be combined with other clinical data, such as age, family history, and hormonal status, to develop a more accurate prediction model for breast cancer risk. The combined data can be fed into machine learning algorithms to train a joint

model that takes into account all the available information to make more accurate predictions. The machine learning model can learn complex patterns and relationships between various features in the data, including the images, and predict the likelihood of the presence of a particular condition. The use of non-image data in conjunction with medical images has the potential to provide a more comprehensive understanding of a patient's health status and allow for earlier and more accurate diagnosis of diseases. In addition, it can help medical professionals develop more effective treatment plans for patients based on their specific health conditions.

Drug discovery

The process of drug discovery and development is a crucial aspect of the healthcare and life science industry. The first stage is discovery and development, which involves identifying a lead compound that can target a particular protein or gene to act as a drug candidate. This process typically involves basic research in fields such as molecular biology, biochemistry, and pharmacology. Once a lead compound has been identified, it undergoes preclinical research to determine its efficacy and safety. This stage involves extensive laboratory testing and animal studies to understand the pharmacokinetics and pharmacodynamics of the drug. The ultimate goal is to identify the most promising drug candidates to move forward into clinical development. Clinical development is the next stage, which involves clinical trials and volunteer studies to fine-tune the drug and optimize its dosage, safety, and efficacy. This stage is divided into three phases, with each phase becoming progressively larger and more expensive. The goal is to demonstrate that the drug is both safe and effective for its intended use. After the clinical development stage, the drug undergoes FDA review, where it is evaluated holistically to either approve or reject it. This involves a rigorous evaluation of the drug's safety, efficacy, and manufacturing processes. Finally, post-market monitoring is carried out to ensure the safety of the drug once it has been approved and is available to the public. This involves ongoing monitoring of adverse reactions, side effects, and other safety concerns. In the field of drug discovery and development, machine learning (ML) has emerged as a powerful tool in recent years. ML

techniques can be used for various purposes, such as predicting the efficacy and toxicity of a drug candidate, identifying new drug targets. Additionally, ML can aid in one of the key challenges in drug discovery: understanding protein folding. Protein folding is the process by which a protein molecule assumes its functional three-dimensional shape. ML algorithms can analyze the complex interactions between protein molecules and predict their folding patterns. This can provide insights into the mechanisms of diseases and facilitate the discovery of new drugs targeting specific proteins. By leveraging large and complex datasets, ML can accelerate the drug development process, reduce costs, and improve the safety and efficacy of new drugs.

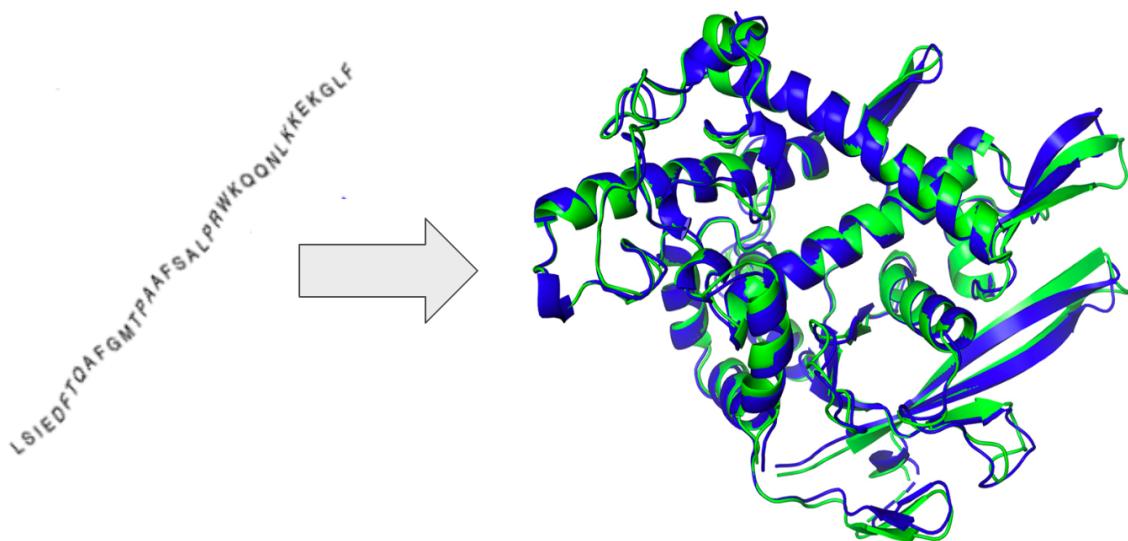


Figure 2.14: Predicting protein 3D structure

ML has been used to optimize clinical trials, such as identifying potential cohorts for clinical trials, an important step in the drug discovery process. By analyzing large amounts of patient data, ML models can help identify groups of patients that are most likely to benefit from a particular treatment. For example, in cancer research, ML has been used to analyze genetic and clinical data from patients to identify specific patient subgroups that may respond better to a particular drug. ML models can also help optimize clinical trial design by predicting the likelihood of success for a particular

trial. For instance, ML algorithms can be used to analyze historical clinical trial data to identify factors that are associated with successful trials, such as patient characteristics, dosage, and treatment duration. This information can then be used to design more effective trials in the future.

Healthcare data management

Every day, the healthcare industry generates and collects a vast amount of patient healthcare data, which comes in various formats, such as handwritten notes, insurance claim data, recorded medical conversations, and medical images, such as X-rays. This data is crucial for developing a comprehensive view of patients or supporting medical coding for medical billing processes. However, extracting valuable insights from these sources often requires significant manual processing, which is both expensive and error-prone, often carried out by people with health domain expertise. Consequently, a substantial amount of patient healthcare data remains unutilized in its original form. Machine learning-based approaches have been adopted to automate this process and improve the accuracy and efficiency of data processing. For instance, natural language processing (NLP) models can extract information from unstructured medical notes, while computer vision algorithms can analyze medical images to detect and diagnose diseases. This enables healthcare organizations to obtain valuable insights from patient healthcare data that was previously untapped. The process of extracting information from unstructured data sources using machine learning is depicted in the following diagram, showcasing the flow of data and the integration of ML into different healthcare tasks, such as medical coding and clinical decision support.

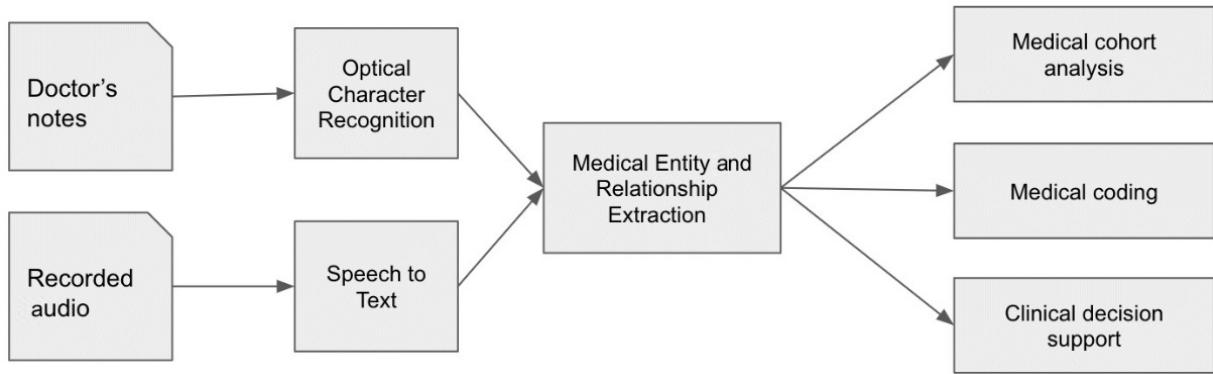


Figure 2.15: Medical data management

Overall, the adoption of ML-based solutions in healthcare is enabling healthcare organizations to unlock valuable insights from unstructured data sources, which can help improve patient outcomes, optimize resource utilization, and reduce costs.

ML use cases in manufacturing

The manufacturing industry is a vast sector that is responsible for creating a wide range of physical products, such as consumer goods, electronics, automobiles, furniture, building materials, and more. Each sub-sector of manufacturing requires a specific set of tools, resources, and expertise to successfully produce the desired products. The manufacturing process generally involves several stages, including product design, prototyping, production, and post-manufacturing service and support. During the design phase, manufacturers work on conceptualizing and planning the product. This includes defining the product's features, materials, and production requirements. In the prototyping stage, a small number of products are created to test their functionality and performance. Once the product design has been finalized, manufacturing and assembling takes place. This is the stage where raw materials are transformed into finished products. Quality control is a critical aspect of the manufacturing process, as manufacturers need to ensure that each product meets the required standards and specifications. Finally, post-manufacturing service and support involves activities such as repair and maintenance, customer support, and product

upgrades. The goal is to provide ongoing value to customers and to ensure the product continues to perform optimally. The following diagram illustrates the typical business functions and flow in the manufacturing sector:

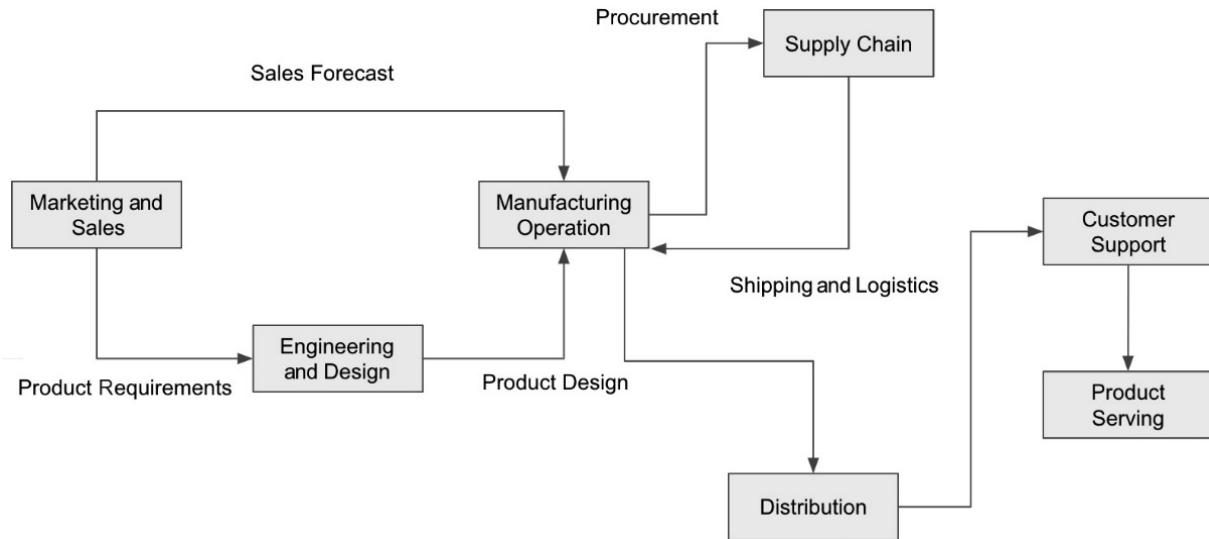


Figure 2.16: Manufacturing business process flow

AI and ML have become crucial tools in the manufacturing industry, driving significant improvements in various stages of the manufacturing process. For instance, machine learning algorithms are used to forecast sales, which allows companies to make informed decisions about production volume and material procurement. This, in turn, leads to more efficient inventory management, reduction of waste, and increased profitability. Moreover, predictive machine maintenance is another area where AI and ML are making significant contributions. With the use of machine learning algorithms, manufacturers can analyze data from sensors and other sources to predict equipment failure before it occurs. This helps avoid unplanned downtime, reduces maintenance costs, and improves overall equipment effectiveness. Quality control is another crucial area where AI and ML have made significant improvements. By analyzing data from sensors and cameras, machine learning algorithms can identify defective products or parts in real-time, allowing for timely intervention to address issues in the manufacturing process. In addition, AI and ML have

been instrumental in automating various tasks in the manufacturing process. This includes the use of robots for assembling products, performing quality checks, and handling material movement. This not only improves manufacturing quality and yield but also helps ensure worker safety by reducing the risk of accidents in hazardous work environments. Finally, AI and ML can also be used to optimize supply chain management, improving overall operational efficiency and reducing costs. Machine learning algorithms can analyze data from multiple sources to identify inefficiencies and bottlenecks in the supply chain, enabling manufacturers to make data-driven decisions that improve production planning, inventory management, and distribution.

Engineering and product design

Product design is a crucial aspect of the manufacturing process, where designers aim to create products that are both functional and appealing to consumers. During the design phase, designers need to strike a balance between their creative vision, the practical needs of the market, and the constraints of production. To achieve this, they may create multiple versions of a new product concept that cater to different needs and constraints. For instance, in the fashion industry, designers may analyze customer preferences in terms of color, texture, and style to develop new apparel designs and graphics that meet those demands. The manufacturing industry has been leveraging generative design ML technology to assist with new product concept design. For example, Generative AI, a type of machine learning, can be used in product design to generate a vast number of possible design variations that meet specific constraints and requirements. By inputting design constraints, such as cost, materials, and production capabilities, generative AI can produce thousands of design options that meet those criteria. This approach can significantly speed up the product design process and also enable designers to explore a broader range of design possibilities. In addition to generative design, ML techniques have proven to be invaluable in analyzing market requirements and estimating the potential of new products. By leveraging various data sources such as customer feedback, market trends, and competitor analysis, ML algorithms can predict the demand for new products accurately. Furthermore, ML

models can analyze large amounts of data quickly, enabling businesses to stay ahead of the competition by identifying new market opportunities and trends. ML algorithms can also identify customer preferences, such as color, texture, style, and functionality, to guide the product development process. The ability of ML to analyze complex data sets can also provide insights into the underlying factors that influence consumer behavior and product preferences. This information can help businesses refine their product design and marketing strategies, leading to increased sales and revenue.

Manufacturing operations – product quality and yield

In the manufacturing industry, quality control is crucial to ensure that the products meet the required standards and specifications. However, relying solely on human inspection can be time-consuming and expensive. That's why the adoption of computer vision-based technology has been a game-changer in the quality control process. Computer vision models can be trained using machine learning algorithms to identify defects and flaws in manufactured products. For instance, in the automotive industry, computer vision algorithms can detect even the slightest surface scratches, dents, or deformations that might affect the vehicle's performance. Moreover, computer vision-based technology can be applied to different stages of the manufacturing process, such as monitoring assembly lines, detecting defects in finished goods, and identifying problems with raw materials. The use of AI-powered systems in quality control not only enhances efficiency and reduces costs, but also ensures consistency and accuracy in the inspection process.

Manufacturing operations – machine maintenance

Regular maintenance is crucial for industrial manufacturing equipment and machinery to ensure smooth operations and prevent unexpected failures. However, traditional maintenance practices that follow a regular maintenance schedule can be costly and may not always detect potential problems. Fortunately, machine learning-based predictive maintenance analytics have emerged as a solution to help manufacturers forecast potential problems in advance and reduce the risk of unforeseen equipment

failures. By analyzing a variety of data, including telemetry data collected by Internet of Things (IoT) sensors, machine learning algorithms can predict whether a piece of equipment is likely to fail within a certain time window. The maintenance crew can then take proactive measures to prevent equipment failure and avoid costly repairs or replacements. This approach not only minimizes the risk of unplanned outages but also reduces overall maintenance costs and downtime.

ML use cases in retail

The retail industry is a sector that sells consumer products directly to customers, either through physical retail stores or online platforms. Retailers acquire their merchandise from wholesale distributors or manufacturers directly. Over the years, the retail industry has undergone significant changes. The growth of e-commerce has outpaced that of traditional retail business, compelling brick-and-mortar stores to adapt and innovate in-store shopping experiences to remain competitive. Retailers are exploring new approaches to enhance the shopping experience across both online and physical channels. Recent developments such as social commerce, augmented reality, virtual assistant shopping, smart stores, and 1:1 personalization have become key differentiators in the retail industry. The retail industry is currently undergoing a transformation fueled by AI and ML technologies. Retailers are utilizing these technologies to optimize inventory, predict consumer demand, and deliver personalized and immersive shopping experiences. AI and ML algorithms can provide personalized product recommendations and enable virtual reality shopping, making it possible for shoppers to try on clothes virtually. Additionally, AI and ML technologies are being employed to enable cashier-less store shopping and to prevent fraudulent activities and shoplifting. Overall, the retail industry's adoption of AI and ML technologies is expected to enhance the shopping experience and enable retailers to meet the evolving needs and expectations of their customers.

Product search and discovery

Online shopping has simplified the purchasing process for consumers, but searching for a product online can sometimes be difficult when you only have a picture and no information on the item's name or features. This is where deep learning-powered visual search technology comes in handy. This technology allows consumers to quickly identify similar-looking products by simply uploading a picture of the item they are looking for. Visual search technology works by creating a digital representation of the item's pictures, also known as encoding or embedding, and storing it in a high-performance item index. When a shopper needs to find a similar-looking item using a picture, the new picture is encoded into a digital representation and searched against the item index using an efficient distance-based comparison. The system then returns the items that are closest to the target item. With visual search technology, consumers can easily find what they are looking for, even if they do not know the correct search terms. This technology has become increasingly popular among e-commerce retailers, and the architecture for building an ML-based image search capability is continually evolving to improve the accuracy and efficiency of visual search. The following diagram illustrates an architecture for building an ML-based image search capability:

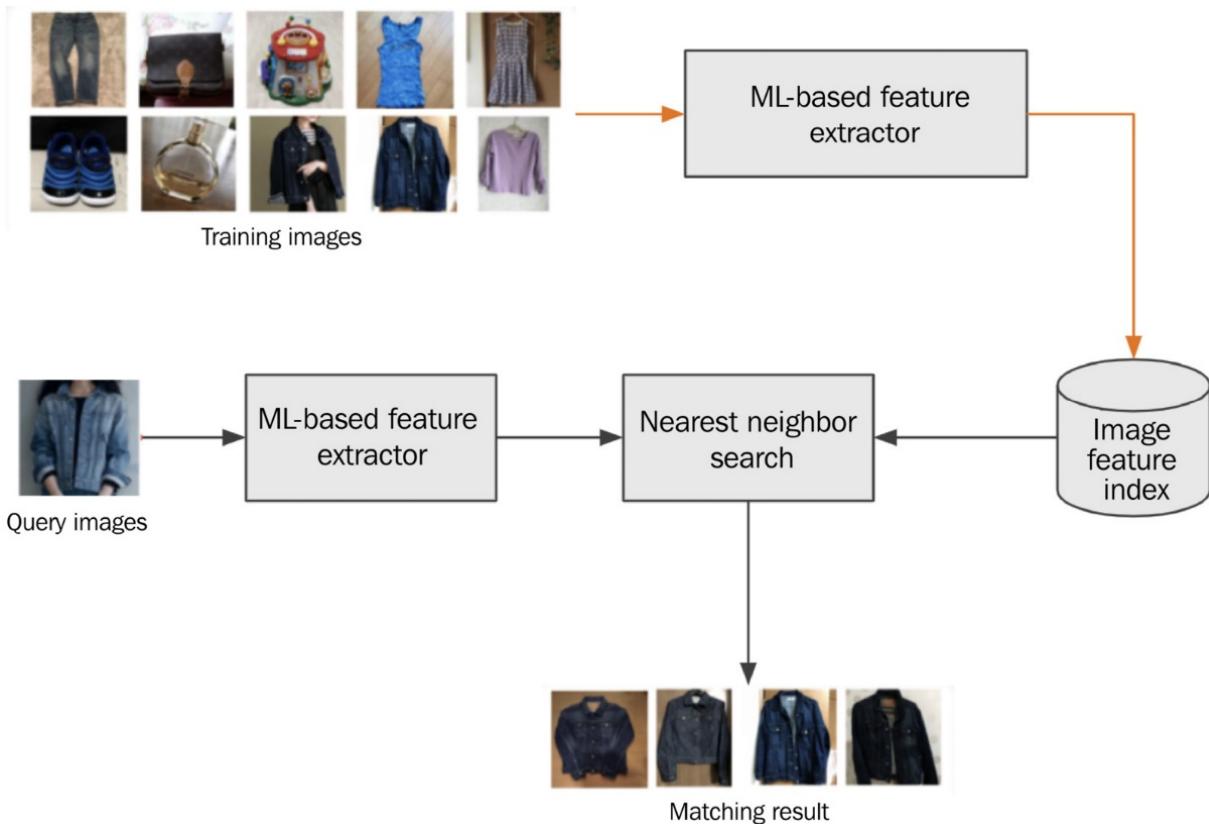


Figure 2.18: Image search architecture

Visual search-based recommendations have been adopted by many large e-commerce sites such as Amazon.com to enhance the shopping experience.

Target marketing

Retailers employ various marketing campaigns and advertising tactics, including direct marketing emails or digital advertisements, to attract potential customers with incentives or discounts based on their demographics. The success of such campaigns relies heavily on effectively targeting the right customers to achieve a high conversion rate while minimizing advertising costs and reducing customer disturbances. Machine learning models have been developed to optimize the effectiveness of marketing campaigns. These models use customer data and various demographic factors to identify potential customers who are most likely to convert, as well as to determine the most appropriate messaging and

incentives for each customer segment. By leveraging machine learning techniques, retailers can improve the accuracy and efficiency of their marketing campaigns, resulting in a higher return on investment. **Segmentation** is one traditional way to understand the different customer segments to help improve marketing campaigns' effectiveness. There are different ways to do segmentations with machine learning, such as unsupervised clustering of customers based on data such as basic demographic data. This allows you to group customers into several segments and create unique marketing campaigns for each segment. A more effective target marketing approach is to use highly personalized **user-centric marketing campaigns**. They work by creating accurate individual profiles using large amounts of individual behavior data such as historical transaction data, responses data to historical campaigns, and alternative textual data such as social media data. Highly personalized campaigns with customized marketing messages can be generated using these personal profiles for a higher conversion rate. The ML approach to user-centric target marketing predicts the conversion rate, such as the **click-through rate (CTR)**, for different users and sends ads to users with a high conversion rate. This can be a classification or regression problem by learning the relationship between the user features and the probability of conversion. Contextual advertising is a targeted marketing technique that displays ads that are relevant to the content on a web page. It involves placing display or video ads on websites that match the advertisement's content, which can improve the ad's effectiveness. For instance, a cooking product advertisement may be placed on a cooking recipe website to reach a highly engaged audience. Machine learning can assist with identifying the context of an ad to ensure it is placed appropriately. For instance, computer vision models can analyze video ads to detect objects, people, and themes to extract contextual information and match them to the website's content. By utilizing contextual advertising, marketers can increase the chances of their ads resonating with their target audience and achieving a higher click-through rate.

Sentiment analysis

Understanding consumer perception of their brand is crucial for retail businesses as it can have a significant impact on their success. With the rise of online platforms, consumers have become more vocal about their experiences and opinions, making it easier for retailers to monitor their brand reputation. Retailers are adopting various techniques, including soliciting feedback from their shoppers and monitoring social media channels, to assess their customer's emotions and sentiments towards their brand and products. By effectively analyzing sentiment, retailers can identify areas for improvement, such as operational or product enhancements, as well as mitigate potentially malicious attacks on their brand reputation. Sentiment analysis is a text classification problem that involves using labeled text data, such as product reviews, to determine whether the sentiment is positive, negative, or neutral. Machine learning algorithms, including deep learning-based algorithms, can be used to train models that can detect sentiment in a piece of text. These models can then be used to automatically classify new text data, such as social media posts or customer feedback, to help retailers understand the overall sentiment towards their brand and products. By leveraging sentiment analysis, retailers can gain valuable insights into customer preferences, identify areas for improvement, and make data-driven decisions to improve their overall customer experience.

Product demand forecasting

Retail businesses rely on inventory planning and demand forecasting to manage inventory costs while maximizing revenue and avoiding out-of-stock situations. Traditional methods for demand forecasting, such as buyer surveys, expert opinions, and projections based on past demands, have limitations in accuracy and reliability. To address these limitations, retailers are turning to statistical and machine learning (ML) techniques such as regression analysis and deep learning. These approaches can use historical demand and sales data, as well as other related data such as prices, holidays, special events, and product attributes, to create more accurate and data-driven demand forecasts. Deep learning-based algorithms can be particularly effective in producing accurate demand forecasts by incorporating multiple data sources into the model. This approach involves training an ML model

to recognize patterns and relationships in the data to generate highly accurate forecasts. The result is more reliable inventory planning that helps retailers optimize their inventory while maximizing revenue. The following diagram illustrates the concept of building a deep learning model using multiple data sources to generate forecasting models:

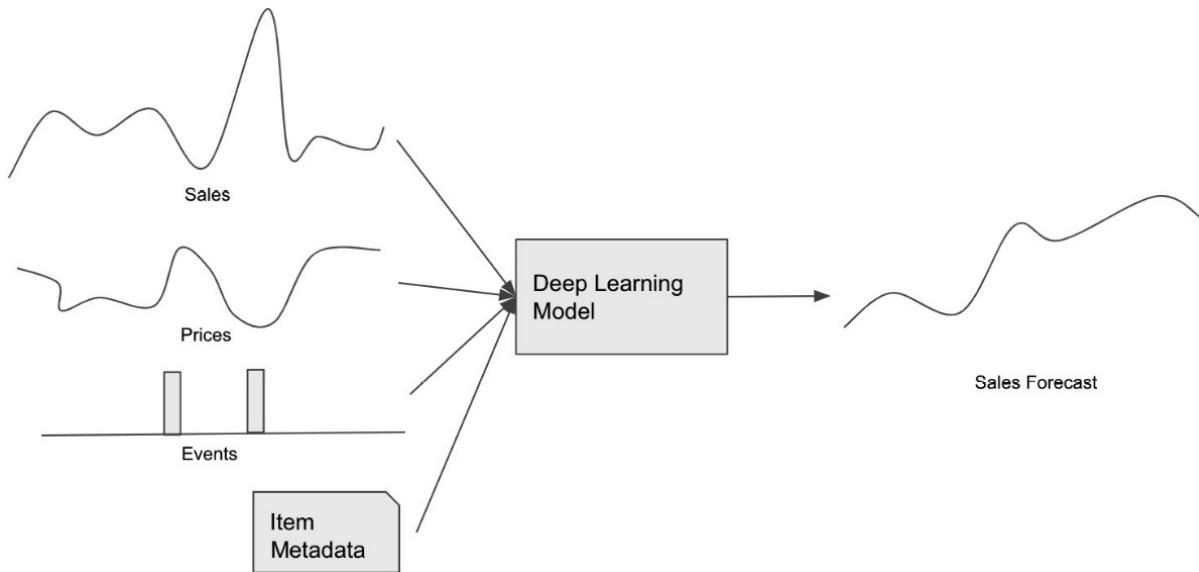


Figure 2.19: Deep learning-based forecasting model

ML-based forecasting models can generate both point forecasts (a number) of probabilistic forecasts (a forecast with a confidence score). Many retail businesses use ML to generate baseline forecasts, and then professional forecasters review them and make adjustments based on their expertise and other factors.

ML use cases in automotive

The automotive industry has undergone significant transformation in recent years, with technology playing a key role in shaping its evolution. AI and ML have emerged as powerful tools for automakers and suppliers to improve efficiency, safety, and customer experience. From production lines to connected cars, AI and ML are being used to automate processes, optimize operations, and enable new services and features.

Autonomous vehicle

One of the most significant applications of AI and ML in the automotive industry is in autonomous driving. Automakers and tech companies are leveraging these technologies to build self-driving vehicles that can safely navigate roads and highways without human intervention. AI and ML algorithms are used to process data from sensors, cameras, and other inputs to make real-time decisions and actions, such as braking or changing lanes. The system architecture of an autonomous vehicle (AV) consists of 3 main stages: 1/ perception and localization, 2/ decision and planning, and 3/ control, as illustrated in the diagram below:

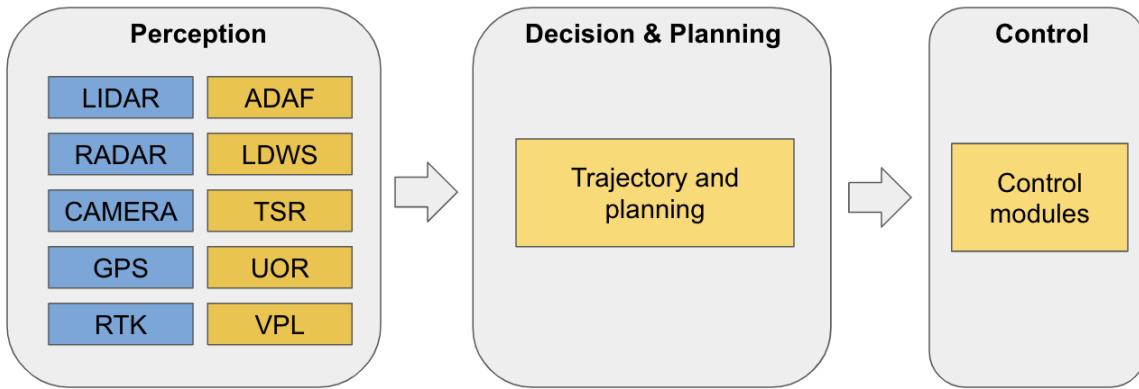


Figure 2.20: Autonomous vehicle system architecture

Perception and localization

Perception is a crucial stage in autonomous driving, where the AV gathers information about its surroundings through a variety of sensors and determines its own position in relation to the environment. The AV employs sensors such as RADAR, LIDAR, cameras, and real-time kinematic (RTK) systems to capture data from the surrounding environment. This sensory data is then fed into recognition modules for further processing. One of the key components in the perception stage is the adaptive detection and recognition framework (ADAF), which utilizes ML algorithms to detect and classify objects such as pedestrians, vehicles, and obstacles in the AV's

vicinity. Additionally, the AV incorporates modules such as Lane Departure Warning System (LDWS), Traffic Sign Recognition (TSR), Unknown Obstacles Recognition (UOR), and Vehicle Positioning and Localization (VPL) to enhance its perception capabilities. The perception stage forms a fundamental building block in the overall autonomous driving system. The accuracy and reliability of the perception module significantly impact the AV's ability to perceive and interpret its environment. Advances in ML algorithms, sensor technology, and sensor fusion techniques continue to improve the perception capabilities of autonomous vehicles, enabling them to operate safely and effectively in diverse and complex driving scenarios.

Decision and planning

The Decision and Planning stage is a crucial aspect of autonomous driving, which controls the motion and behavior of the autonomous vehicle based on the data collected during the perception stage. Artificial Intelligence (AI) and Machine Learning (ML) technologies play a vital role in this stage, which can be considered the brain of the AV. By analyzing data from sensors such as RADAR, LIDAR, and cameras, the Decision and Planning stage uses algorithms to determine the optimal path for the AV to follow. AI/ML can help enhance the path planning process by taking into account various factors, such as real-time map information, traffic patterns, and user inputs, to make informed decisions. Through prediction and forecasting techniques, AVs can anticipate the actions of other road users and plan accordingly. AI/ML algorithms can also aid in obstacle avoidance by continuously monitoring the environment, detecting potential hazards, and making real-time adjustments to the vehicle's trajectory to avoid collisions. The Decision and Planning stage serves as the intelligence behind the AV's actions, allowing it to make informed choices based on real-time and historical data. With advancements in AI and ML, decision-making algorithms have become increasingly sophisticated, enabling AVs to navigate complex scenarios and respond effectively to dynamic traffic conditions.

Control

The Control module in autonomous driving plays a vital role in translating the decisions made by the Decision and Planning stage into physical actions that control the autonomous vehicle (AV). Artificial Intelligence (AI) and Machine Learning (ML) techniques are applied in this module to enhance the control mechanisms and optimize the AV's performance. One area where AI/ML can be applied in the Control module is in adaptive control systems. By utilizing sensor data and real-time feedback, AI algorithms can dynamically adjust the control inputs to ensure the AV operates smoothly and safely. ML models can learn from past driving experiences and optimize control actions based on various driving conditions, such as different road surfaces, weather conditions, and traffic patterns. Moreover, reinforcement learning techniques can be employed in the Control module to enable the AV to learn optimal control policies through trial and error. By interacting with the environment and receiving feedback on the outcomes of its actions, the AV can iteratively improve its control strategies, leading to more efficient and effective driving behavior.

Advanced driver assistance systems (ADAS)

In addition to autonomous driving, AI and ML are also being used to enhance the driving experience, with features such as advanced driver assistance systems (ADAS). ADAS leverages computer vision, sensor fusion, and AI techniques to detect and interpret the surrounding environment in real-time. By analyzing data from cameras, radars, and other sensors, ADAS can identify potential hazards on the road, including pedestrians, cyclists, and other vehicles. This allows the system to issue warnings to the driver or even take autonomous corrective actions to mitigate risks. For example, lane departure warning systems alert drivers when they unintentionally deviate from their lane, while automatic emergency braking systems can autonomously apply the brakes to prevent or reduce the severity of a collision. ADAS technologies not only enhance safety but also contribute to reducing accidents and saving lives.

Summary

Throughout this chapter, we have explored various industries and the ways in which they are utilizing machine learning to solve business challenges and drive growth. From finance and healthcare to retail and automotive, we have seen how ML algorithms can improve processes, generate insights, and enhance the customer experience. As we move into the next chapter, we will delve deeper into the mechanics of machine learning, exploring the fundamental concepts behind how machines learn and some of the most widely used algorithms in the field. This will provide you with a solid foundation for understanding how ML is applied in practice across a range of industries.

3 Machine Learning Algorithms

Join our book community on Discord

<https://packt.link/EarlyAccessCommunity>



While ML algorithm design may not be the primary role of ML solutions architects, it is still essential for them to possess a comprehensive understanding of common real-world ML algorithms and their applications in solving business problems. This knowledge empowers ML solutions architects to identify suitable data science solutions and design the necessary technology infrastructure for deploying these algorithms effectively. By familiarizing themselves with a range of ML algorithms, ML solutions architects can grasp the strengths, limitations, and specific use cases of each algorithm. This enables them to evaluate business requirements accurately and select the most appropriate algorithmic approach to address a given problem. Whether it's classification, regression, clustering, or recommendation systems, understanding the underlying algorithms equips architects with the knowledge required to make informed decisions. In this chapter, we will explore the fundamentals of machine learning (ML) and delve into common ML and deep learning algorithms. We'll cover tasks such as classification, regression, object detection, recommendation, forecasting, and natural language generation. By understanding the core principles and applications of these algorithms, you'll gain the knowledge to identify suitable ML solutions for real-world problems. This chapter aims to equip you with the expertise to make informed decisions and design effective ML solutions across various domains.

Technical requirements

You need a personal computer (**Mac** or **Windows**) to complete the hands-on exercise portion of this chapter. You need to download the dataset from <https://www.kaggle.com/mathchi/churn-for-bank-customers>. Additional instructions will be provided in the *Hands-on exercise* section.

How machines learn

In Chapter 1, "Machine Learning and Machine Learning Solutions Architecture," we discussed the self-improvement capability of ML algorithms through data processing and parameter updates, leading to the generation of models akin to compiled binaries in computer source code. But how does an algorithm actually learn? In essence, ML algorithms learn by optimizing an objective function, also known as a loss function, which involves minimizing or maximizing it. An objective function can be seen as a business metric, such as the disparity between projected and actual product sales. The aim of optimization is to reduce this disparity. To achieve this, an ML algorithm iterates and processes extensive historical sales data (training data), adjusting its internal model parameters until the gaps between projected and actual values are minimized. This process of finding the optimal model parameters is referred to as optimization, with mathematical routines specifically designed for this purpose known as optimizers. To illustrate the concept of optimization, let's consider a simple example of training an ML model to predict product sales based on its price. In this case, we can use a linear function as the ML algorithm, represented as follows:

$$sales = W * price + B$$

In this example, our objective is to minimize the disparity between the predicted and actual sales values. To achieve this, we employ the **mean square error** (MSE) as the loss function for optimization. The specific task is to determine the optimal values for the model parameters W and B , commonly referred to as weight and bias. The weight assigns a relative significance to each input variable, while the bias represents the average output value. Our aim is to identify the W and B values that yield the lowest MSE in order to enhance the accuracy of the sales predictions:

$$Error = \frac{1}{n} \sum_{i=1}^n (predicted_i - actual_i)^2$$

There are multiple techniques available to solve ML optimization problems. Among them, gradient descent and its variations are widely used for optimizing neural networks and various other ML algorithms. Gradient descent is an iterative approach that involves calculating the rate of error change (gradient) associated with each input variable. Based on this gradient, the model parameters (W and B in this example) are updated step by step to gradually reduce the error. The learning rate, a hyperparameter of the ML algorithm, controls the magnitude of parameter updates at each iteration. This allows for fine-tuning the optimization process. The figure below illustrates the optimization of the W value using gradient descent:

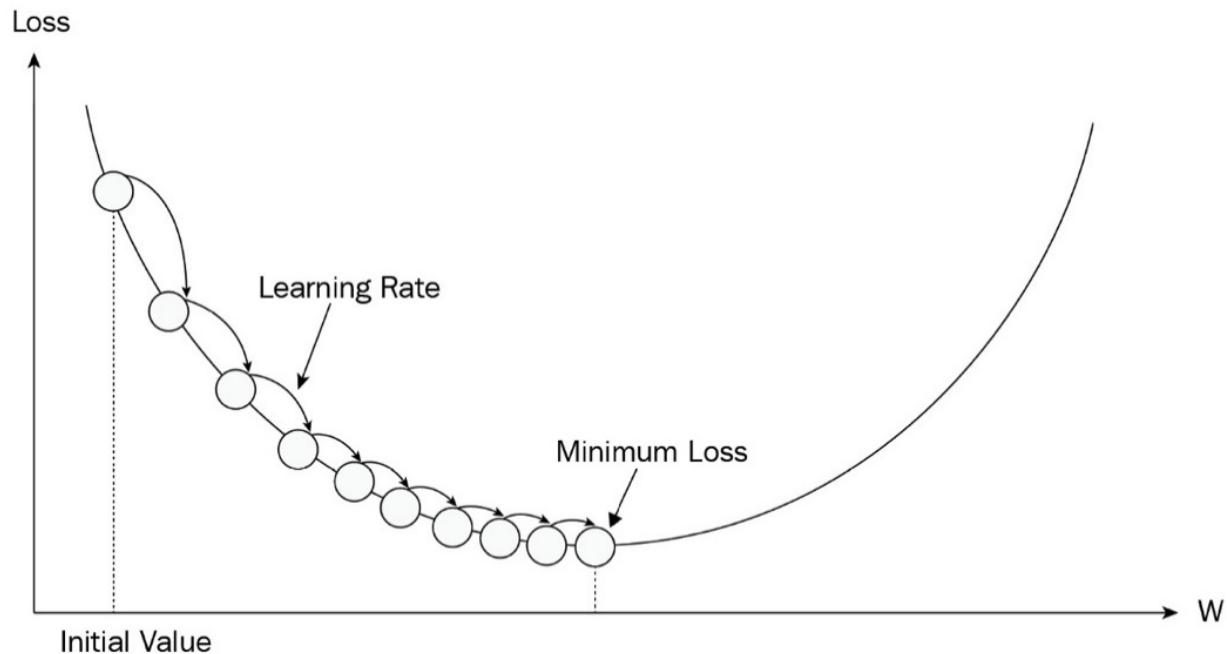


Figure 3.1: Gradient descent

The gradient descent optimization process involves several key steps:

1. Initialize the value of W randomly.
2. Calculate the error (loss) using the assigned value of W .
3. Compute the gradient (rate of change) of the error with respect to the loss function. The gradient can be positive, zero, or negative.
4. If the gradient is positive or negative, update the value of W in a direction that reduces the error in the next iteration. In this example, we move W to the right to increase its value.
5. Repeat steps 2 to 4 until the gradient becomes zero, indicating that the optimal value of W has been reached and convergence has been achieved.

In addition to gradient descent, alternative optimization techniques like the normal equation can be used to find optimal parameters for ML algorithms such as linear regression. Unlike the iterative approach of gradient descent, the normal equation offers a one-step analytical solution for calculating the coefficients of linear regression models. Other ML algorithms may also have algorithm-specific optimization methods for model training, which will be discussed in the next section. Now, we have briefly discussed about the fundamental concept of how machines learn. Next, let's delve into each individual ML algorithms.

Overview of ML algorithms

The field of machine learning has seen the development of numerous algorithms, with ongoing research and innovation from both academia and industry. In this section, we will explore several well-known traditional and deep learning algorithms, examining their applications across various types of machine learning problems. Before we delve into these algorithms, it's important to discuss the factors to consider when selecting an appropriate algorithm for a given task.

Consideration for choosing ML algorithms

When choosing a machine learning algorithm, there are several key considerations to keep in mind:

- **Problem type:** Different algorithms are better suited for specific types of problems. For example, classification algorithms are suitable for tasks where the goal is to categorize data into distinct classes, while regression algorithms are used for predicting continuous numerical values. Understanding the problem type is crucial in selecting the most appropriate algorithm.
- **Dataset size:** The size of your dataset can impact the choice of algorithm. Some algorithms perform well with small datasets, while others require large amounts of data to generalize effectively. If you have limited data, simpler algorithms with fewer parameters may be preferable to avoid overfitting.
- **Feature space:** Consider the number and nature of features in your dataset. Some algorithms can handle high-dimensional feature spaces, while others are more suitable for datasets with fewer features. Feature engineering and dimensionality reduction techniques can also be applied to enhance algorithm performance.
- **Computational efficiency:** The computational requirements of an algorithm should be taken into account, especially if you have large datasets or limited computational resources. Some algorithms are computationally expensive and may not be feasible for certain environments. Time complexity and space complexity are quantitative measures used to assess the efficiency of machine learning algorithms. Big O notation represents the upper bound estimation for time and space requirements. For example, linear search has a time complexity of $O(N)$, while binary search has $O(\log N)$. Understanding these complexities helps evaluate algorithm efficiency and scalability, aiding in algorithm selection for specific tasks.
- **Interpretability:** Depending on your application, the interpretability of the algorithm's results may be important. Some algorithms, such as decision trees or linear models, offer easily interpretable

outcomes, while others, like deep neural networks, provide more complex and abstract representations.

- **Algorithm complexity and assumptions:** Different algorithms make different assumptions about the underlying data distribution. Consider whether these assumptions are valid for your dataset. Additionally, the complexity of the algorithm can impact its ease of implementation, training time, and ability to handle noisy or incomplete data.

By considering these factors, you can make an informed decision when selecting a machine learning algorithm that best suits your specific problem and available resources.

Algorithms for classification and regression problems

The vast majority of ML problems today primarily involve classification and regression. In the upcoming section, we'll explore common algorithms used for classification and regression tasks.

Linear regression algorithm

Linear regression algorithms are developed to solve regression problems by predicting continuous values based on independent inputs. They find wide applications in various practical scenarios, such as estimating product sales based on price or determining crop yield based on rainfall and fertilizer. Linear regression utilizes a linear function of a set of coefficients and input variables to predict a scalar output. The formula for the linear regression is expressed as follows:

$$f(x) = W_1 * X_1 + W_2 * X_2 + \dots + W_n * X_n + \epsilon$$

In the linear regression equation, the X s represent the input variables, W s denote the coefficients, and ϵ represents the error term. Linear regression aims to estimate the output value by calculating the weighted sum of the inputs, assuming a linear relationship between the output and inputs. The intuition behind linear regression is to find a line or hyperplane that can estimate the value for a set of input values. Linear regression can work efficiently with small datasets, offering interpretability through the coefficients' assessment of input and output variables. However, it may not perform well with complex, nonlinear datasets. Additionally, linear regression assumes independence among input features and struggles when there is co-linearity (the value of one feature influences the value of another feature), as it becomes challenging to assess the significance of correlated features.

Logistic regression algorithm

Logistic regression is commonly employed for binary classification tasks. It can predict the probability of an event occurring, such as whether a person will click on an advertisement or qualify for a loan. Logistic regression is a valuable tool in real-world scenarios where the outcome is binary and requires estimating the likelihood of a particular class. By utilizing a logistic function, this algorithm maps the input variables to a probability score, enabling effective classification decision-making. Logistic regression is a statistical model used to estimate the probability of an event or outcome, such as transaction fraud or passing an exam. It is a linear model similar to linear regression, but with a different output transformation. The goal of logistic regression is to find a decision boundary, represented by a line or hyperplane, that effectively separates the two classes of data points. By applying a logistic function to the linear combination of input variables, logistic regression ensures that the predicted output falls within the range of 0 and 1, representing the probability of belonging to a particular class. The following formula is the function for the logistic regression, where X is a linear combination of input variables ($b + wx$). Here, the w is the regression coefficient: Top of Form Bottom of Form

$$f(x) = \frac{1}{1 + e^{-x}}$$

Like linear regression, logistic regression offers fast training speed and interpretability as its advantages. However, due to its linear nature, logistic regression is not suitable for solving problems with complex non-linear relationships.

Decision tree algorithm

Decision trees find extensive application in various real-world ML scenarios, including heart disease prediction, target marketing, and loan default prediction. They are versatile and can be used for both classification and regression problems. A decision tree is motivated by the idea that data can be divided hierarchically based on rules, leading to similar data points following the same decision path. It achieves this by splitting the input data using different features at different branches of the tree. For example, if age is a feature used for splitting at a branch, a conditional check like age > 50 would be used to divide the data. The decision of which feature to split on and where to split is made using algorithms such as the Gini purity index and information gain. The Gini index measures the probability of misclassification, while information gain quantifies the reduction in entropy resulting from the split. In this book, we won't delve into specific algorithm details. However, the general concept of decision tree involves experimenting with various split options and conditions, calculating metric values (e.g., information gain) for each split option, and selecting the option that yields the highest value. During prediction, input data traverses the tree based on the learned branching logic, and the final prediction is determined by the terminal node (leaf node). Refer to Figure 3.2 for an example structure of a decision tree.

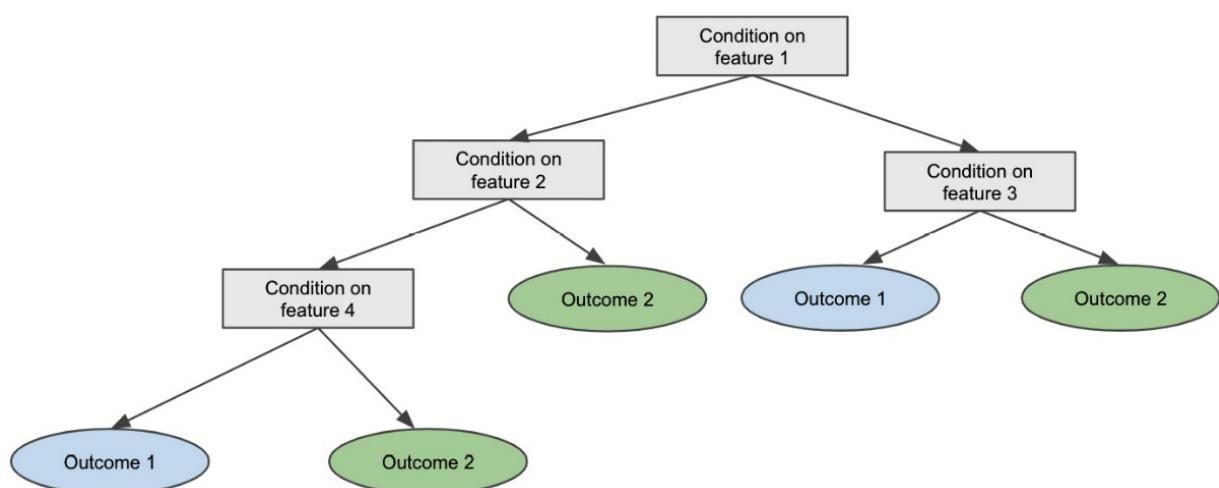


Figure 3.2: Decision tree

The main advantage of decision trees over linear regression and logistic regression is their ability to capture non-linear relationships and interactions between features. Decision trees can handle complex data patterns and are not limited to linear relationships between input variables and the output. They can represent decision boundaries that are more flexible and can handle both numerical and categorical features. A decision tree is advantageous in that it can handle data with minimal preprocessing, accommodate both categorical and numerical features, and handle missing values and varying feature scales. It is also highly interpretable, allowing for easy visualization and analysis of decision paths. Furthermore, decision trees are computationally efficient. However, they can be sensitive to outliers and prone to **overfitting**, particularly when dealing with a large number of features and noisy data. Overfitting occurs when the model memorizes the training data but performs poorly on unseen data. A notable limitation of decision trees and tree-based algorithms is their inability to extrapolate beyond the range of training inputs. For instance, if a housing price model is trained on square footage data ranging from 500 to 3,000 sq ft, a decision tree would be unable to make predictions beyond 3,000 sq ft. In contrast, a linear model would be capable of capturing the trend and making predictions beyond the observed range.

Random forest algorithm

Random forest algorithms are widely employed in various real-world applications across e-commerce, healthcare, and finance sectors. They are particularly valuable for classification and regression tasks. Real-world examples of these tasks include insurance underwriting decisions, disease prediction, loan payment default prediction, and targeted marketing efforts. The versatility of random forest algorithms allows them to be applied in a wide range of industries to address diverse business challenges. As discussed in the preceding decision tree section, a decision tree uses a single tree to make its decisions, and the root node of the tree (the first feature to split the tree) has the most influence on the final decision. The motivation behind this random forest is that combining the decisions of multiple trees can lead to improved overall performance. The way that a random forest works is to create multiple smaller **subtrees**, also called **weaker learner trees**, where each subtree uses a random subset of all the features to come to a decision, and the final decision is made by either majority voting (for classification) or averaging (for regression). This process of combining the decision from multiple models is also referred to as **ensemble learning**. Random forest algorithms also allow you to introduce different degrees of randomness, such as **bootstrap sampling**, which involves using the same sample multiple times in a single tree. This helps make the model more generalized and less prone to overfitting. The following figure illustrates how the random forest algorithm processes input data instances using multiple subtrees and combines their outputs.

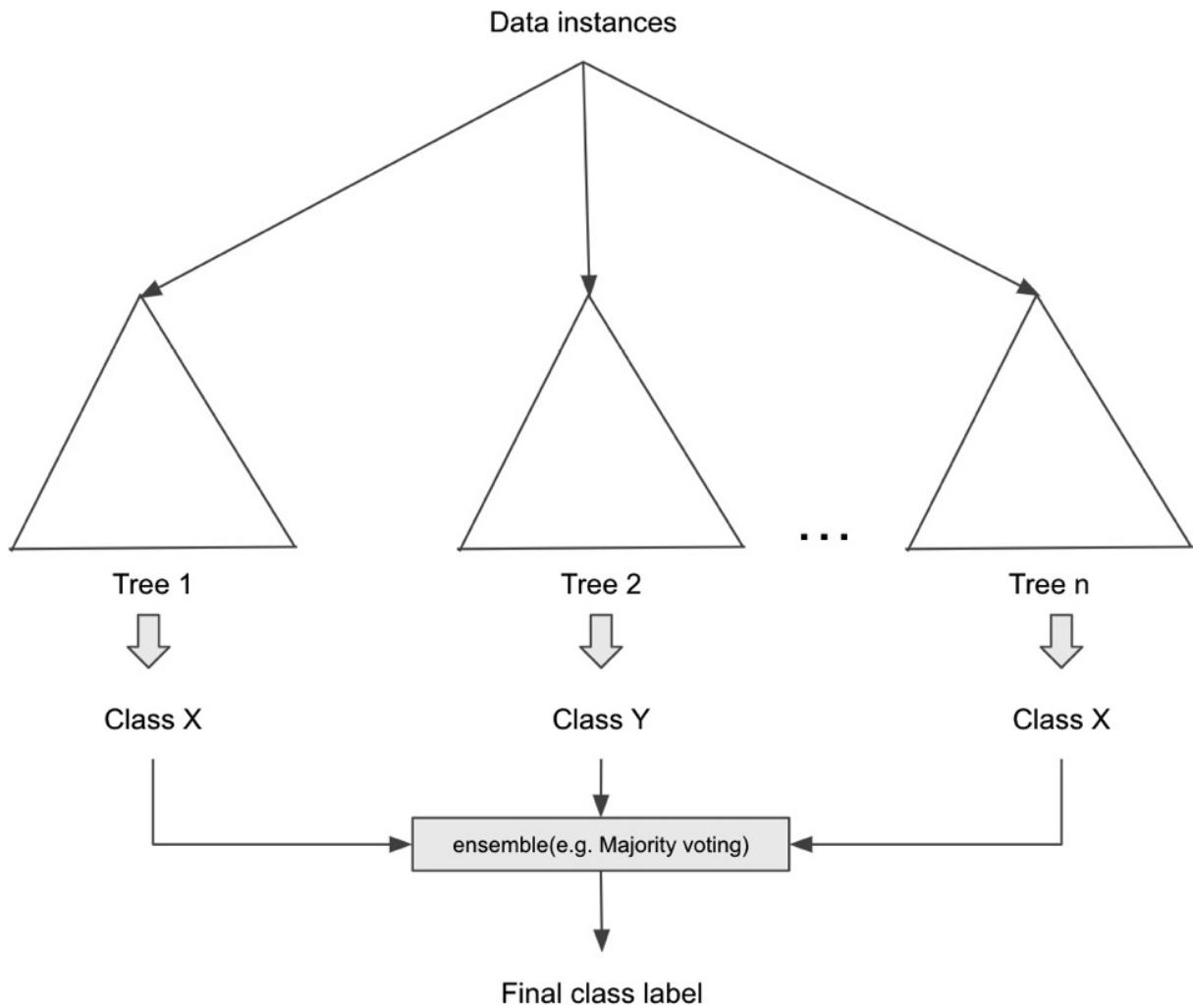


Figure 3.3: Random forest

Random forests have several advantages over decision trees. They provide improved accuracy by combining the predictions of multiple trees through majority voting or averaging. They also reduce overfitting by introducing randomness in the model and using diverse subsets of features. Random forests handle large feature sets better by focusing on different aspects of the data. They are robust to outliers and provide feature importance estimation. Additionally, random forests support parallel processing for training large datasets across multiple machines. The limitations of random forests include reduced interpretability compared to decision trees, longer training and prediction times, increased memory usage, and the need for hyperparameter tuning.

Gradient boosting machine and XGBoost algorithms

Gradient boosting and XGBoost are also popular multi-tree-based ML algorithms used in various domains like credit scoring, fraud detection, and insurance claim prediction. Unlike random forests that combine results from weaker learner trees at the end, gradient boosting sequentially aggregates results from different trees. Random forests utilize parallel independent weaker learners, while gradient boosting employs a sequential approach where each weaker learner tree corrects the errors of the previous tree. Gradient boosting offers more hyperparameters to fine-tune and can achieve superior performance with proper tuning.

It also allows for custom loss functions, providing flexibility in modeling real-world scenarios. Refer to the following figure for an illustration of how gradient boosting trees operate:

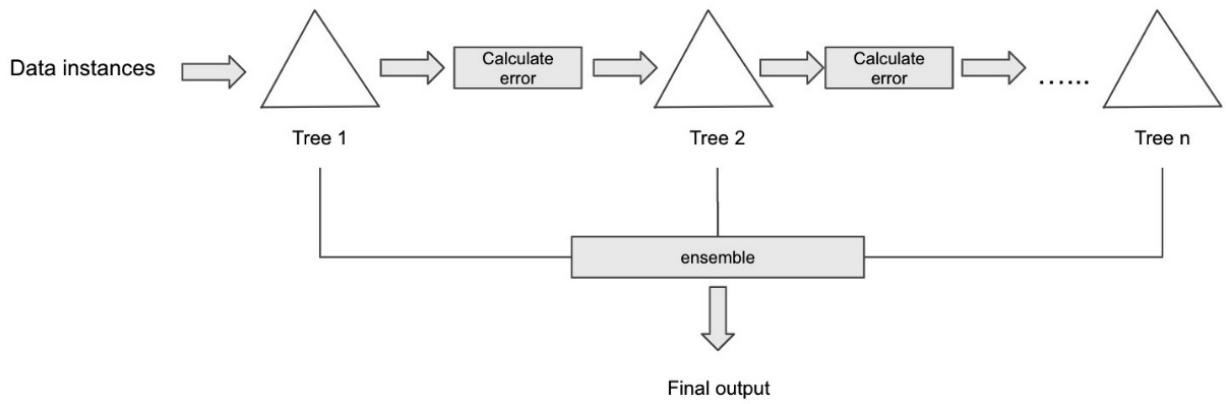


Figure 3.4: Gradient boosting

Gradient boosting offers several key advantages. Firstly, it excels in handling imbalanced datasets, making it highly suitable for tasks such as fraud detection and risk management. Secondly, it has the potential to achieve higher performance than other algorithms when properly tuned. Additionally, gradient boosting supports custom loss functions, providing flexibility in modeling real-world applications. Lastly, it can effectively capture complex relationships in the data and produce accurate predictions. Gradient boosting, despite its advantages, also has some limitations to consider. Firstly, due to its sequential nature, it lacks parallelization capabilities, making it slower in training compared to algorithms that can be parallelized. Secondly, gradient boosting is sensitive to noisy data, including outliers, which can lead to overfitting and reduced generalization performance. Lastly, the complexity of gradient boosting models can make them less interpretable compared to simpler algorithms like decision trees, making it challenging to understand the underlying relationships in the data. XGBoost, a widely-used implementation of gradient boosting, has gained popularity for its success in Kaggle competitions. While it shares the same underlying concept as gradient boosting, XGBoost offers several improvements. It enables training a single tree across multiple cores and CPUs, leading to faster training times. XGBoost incorporates powerful regularization techniques to mitigate overfitting and reduce model complexity. It also excels in handling sparse datasets. In addition to XGBoost, other popular variations of gradient boosting trees include LightGBM and CatBoost.

K-nearest neighbor algorithm

K-nearest neighbor (K-NN) is a versatile algorithm used for both classification and regression tasks. It is also employed in search systems and recommendation systems. The underlying assumption of K-NN is that similar items tend to have close proximity to each other in the feature space. To determine this proximity, distances between different data points are measured, often using metrics like Euclidean distance. In the case of classification, K-NN starts by loading the training data along with their respective class labels. When a new data point needs to be classified, its distances to the existing data points are calculated, typically using Euclidean distance. The K nearest neighbors to the new data point are identified, and their class labels are retrieved. The class label for the new data point is then determined through majority voting, where the most frequent class among the K nearest neighbors is assigned to the new data point. For regression tasks, K-NN follows a similar approach. The distances between the new data point and the existing data points are computed, and the K nearest neighbors are selected. The predicted scalar value for the new data point is obtained by averaging the values of the K closest data points. One advantage of K-NN is its simplicity and lack of the need for training or tuning with hyperparameters, apart from selecting the value of K. The algorithm directly uses the loaded data points. The results of K-NN are also easily explainable, as each

prediction can be understood by examining the properties of the nearest neighbors. However, K-NN has some limitations. As the number of data points increases, the complexity of the model grows, and predictions can become slower, especially with large datasets. K-NN is not suitable for high-dimensional datasets, as the concept of proximity becomes less meaningful in higher-dimensional spaces. The algorithm is also sensitive to noisy data and missing data, requiring outlier removal and data imputation techniques to handle such cases effectively.

Multi-layer perceptron (MLP) network

As mentioned earlier, an artificial neural network (ANN) emulates the learning process of the human brain. The brain comprises numerous interconnected neurons that process information. Each neuron in a network processes inputs (electrical impulses) from another neuron, processes and transforms the inputs, and sends the output to neurons in the network. This mimics the behavior of human neurons. Below is an illustration depicting a human neuron:

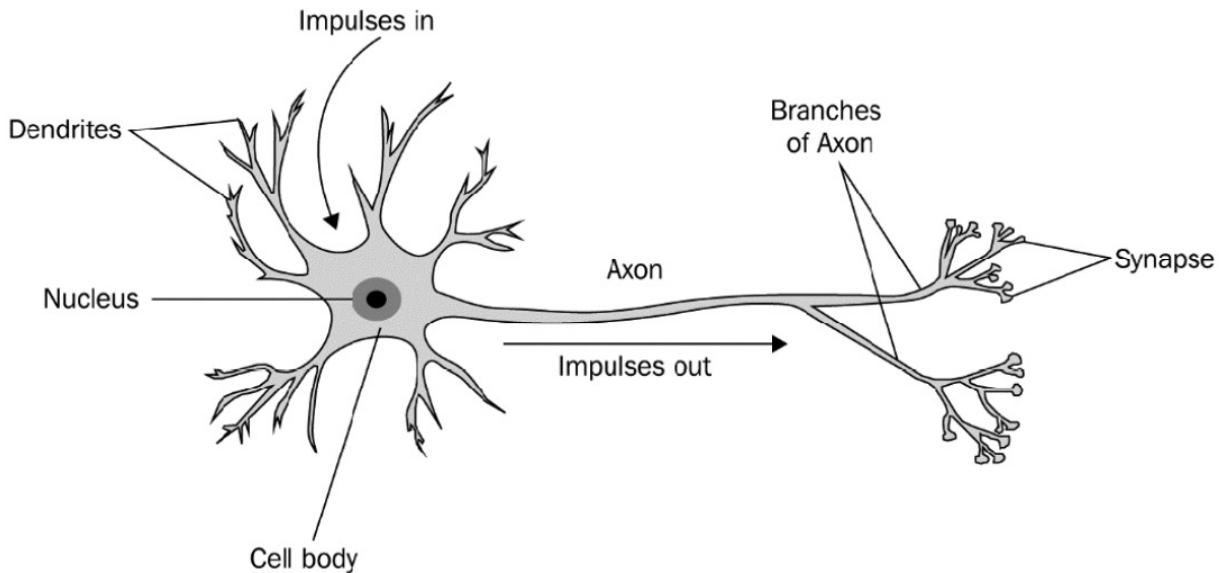


Figure 3.5: Human brain neuron

An artificial neuron operates in a similar manner. The following diagram illustrates an artificial neuron, which consists of a linear function combined with an activation function. The activation function modifies the output of the linear function, such as compressing it within a specific range, such as 0 to 1 (sigmoid activation), -1 to 1 (tanh activation), or maintaining values above 0 (ReLU). The activation function is employed to capture non-linear relationships between inputs and outputs. Alternatively, each neuron can be viewed as a linear classifier, akin to logistic regression.

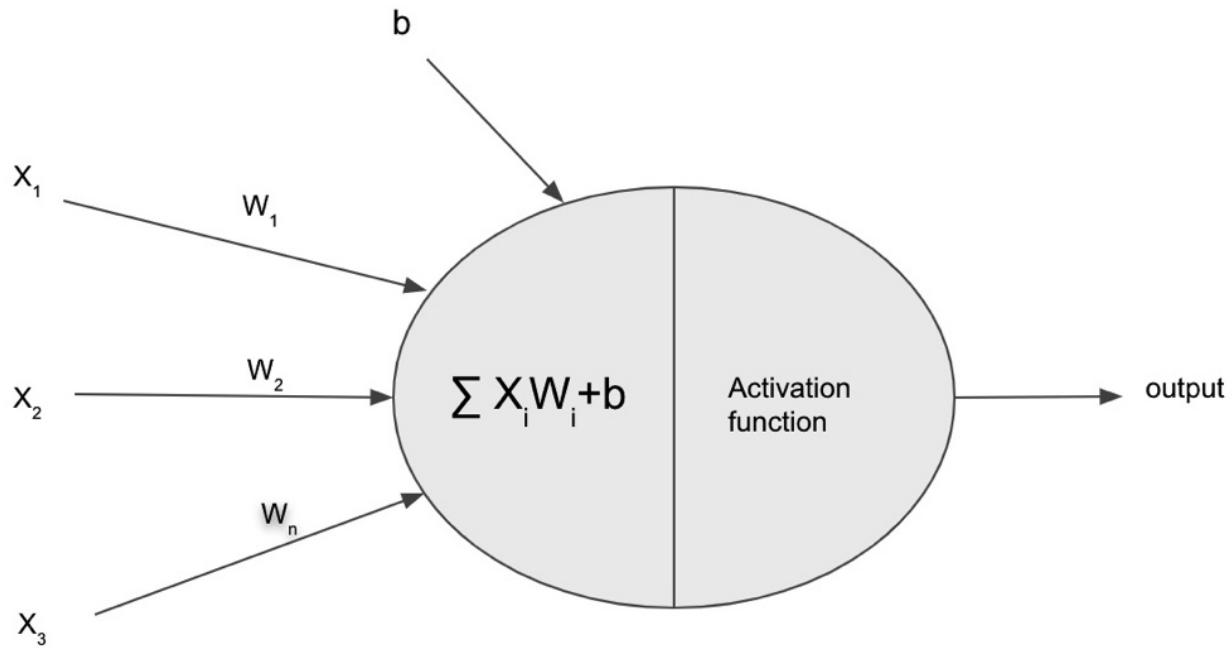


Figure 3.6: Artificial neuron

When you stack a large number of neurons into different layers (*input layer*, *hidden layers*, and *output layer*) and connect all of the neurons together between two adjacent layers, we have an ANN called **multi-layer perceptron (MLP)**. Here, the term *perceptron* means *artificial neuron*, and it was originally invented by Frank Rosenblatt in 1957. The idea behind MLP is that each hidden layer will learn some higher-level representation (features) of the previous layer, and those higher-level features capture the more important information in the previous layer. When the output from the final hidden layer is used for prediction, the network has extracted the most important information from the raw inputs for training a classifier or regressor. The following figure shows the architecture of an MLP network:

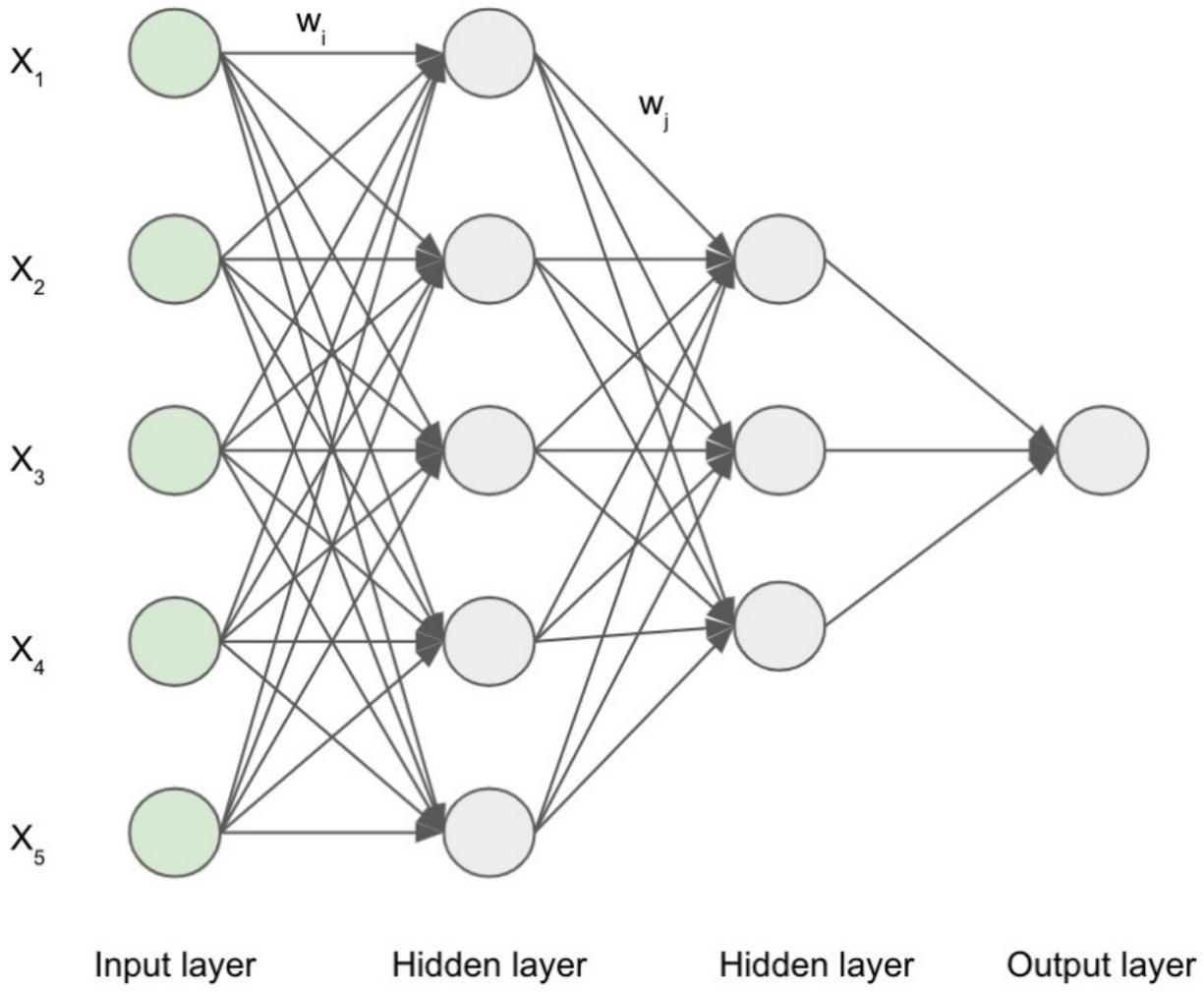


Figure 3.7: Multi-layer perceptron

During model training, the weights (W) of each neuron in every layer are adjusted using gradient descent to optimize the training objective. This adjustment process is known as backpropagation. It involves propagating the total error back through the network, attributing a portion of the error to each neuron based on its contribution. This allows for fine-tuning of the weights in each neuron, ensuring that every neuron in every layer influences the final output to improve overall performance. MLP is a versatile neural network suitable for both classification and regression tasks, similar to random forest and XGBoost. While commonly applied to tabular data, it can also handle diverse data formats like images and text. MLP excels in capturing intricate nonlinear patterns within the dataset and exhibits efficient computational processing, thanks to its parallelization capabilities. However, MLP typically demands a larger training dataset to achieve optimal performance compared to traditional machine learning algorithms.

Algorithms for clustering

Clustering is a data mining method that involves grouping items together based on their shared attributes. One practical application of clustering is to create customer segments by analyzing demographics, transaction history, or behavior data. Various clustering algorithms exist, and we will focus on the K-means clustering algorithm in this section.

K-means algorithm

The K-means algorithm is widely employed in real-world applications, including customer segmentation analysis, document classification based on document attributes, and insurance fraud detection. It is a versatile algorithm that can effectively group data points in various domains for different purposes. K-means aims to group similar data points together in clusters, and it is an unsupervised algorithm, meaning it doesn't rely on labeled data. The algorithm begins by randomly assigning K centroids, which represent the centers of the clusters. It then iteratively adjusts the assignment of data points to the nearest centroid and updates the centroids to the mean of the data points in each cluster. This process continues until convergence, resulting in well-defined clusters based on similarity. K-means clustering offers several advantages, including its simplicity and ease of understanding, making it accessible to beginners. It is computationally efficient and can handle large datasets effectively. The resulting clusters are interpretable, providing valuable insights into the underlying patterns in the data. K-means is versatile and applicable to various types of data, including numerical, categorical, and mixed attribute datasets. However, there are some drawbacks to consider. Selecting the optimal number of clusters (K) can be subjective and challenging. The algorithm is sensitive to the initial placement of centroids, which can lead to different cluster formations. K-means assumes spherical clusters with equal variance, which may not hold true in all cases. It is also sensitive to outliers and struggles with non-linear data relationships.

Algorithms for time series analysis

A time series consists of a sequence of data points recorded at successive time intervals. It is commonly used to analyze and predict trends in various domains, such as finance and sales. Time series analysis allows us to understand past patterns and make future predictions based on the relationship between current and past values. Forecasting in time series relies on the assumption that future values are influenced by previous observations at different time points. Time series data exhibits several important characteristics, including trend, seasonality, and stationarity. Trend refers to the long-term direction of the data, whether it shows an overall increase or decrease over time. It helps to identify the underlying pattern and understand the general behavior of the time series. Seasonality, on the other hand, captures repeating patterns within a fixed interval, often occurring in cycles or seasons. It helps to identify regular fluctuations that repeat over specific time periods, such as daily, weekly, or yearly patterns. Stationarity refers to the property of a time series where statistical properties, such as mean and variance, remain constant over time. Stationarity is crucial because many forecasting techniques assume that the underlying data is stationary. Non-stationary time series can lead to inaccurate or unreliable forecasts. Therefore, it is important to assess and address the stationarity of a time series before applying forecasting techniques.

ARIMA algorithm

The **autoregressive integrated moving average** (ARIMA) algorithm finds practical applications in various real-world scenarios, including budget forecasting, sales forecasting, patient visit forecasting, and customer support call volume forecasting. ARIMA is a powerful tool for analyzing and predicting time series data, allowing organizations to make informed decisions and optimize their operations in these areas. By leveraging historical patterns and trends in the data, ARIMA enables accurate forecasts and assists businesses in effectively managing their resources and planning for the future. ARIMA operates on the premise that the value of a variable in a given period is influenced by its own previous values (autoregressive), the deviations from the mean follow a pattern based on previous deviations (moving average), and trend and seasonality can be eliminated by differencing (calculating the differences between consecutive data points). This differencing process aims to transform the time series into a stationary state, where statistical properties like mean and variance remain constant over time. These three components of ARIMA can be mathematically represented using the following formulas:

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \dots + \phi_p y_{t-p} + \varepsilon_t,$$

Where the **autoregressive (AR)** component is expressed as a regression of previous values (also known as lags)

$$y_{t-1} \cdot \dots \cdot y_{t-p}$$

The constant C represents a drift:

$$y_t = c + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \dots + \theta_q \varepsilon_{t-q},$$

The **moving average (MA)** component is expressed as a weighted average of forecasting errors for the previous time periods, where it represents a constant:

$$y_t' = y_t - y_{t-1}$$

The **integrated component** (time series differencing) of a time series can be expressed as the difference between the values in one period from the previous period. ARIMA is a suitable choice for forecasting single time series (univariate) data as it doesn't rely on additional variables. It outperforms simpler forecasting techniques like simple moving average, exponential smoothing, or linear regression. Additionally, ARIMA provides interpretability, allowing for a clear understanding of the underlying patterns. However, due to its backward-looking nature, ARIMA may struggle to accurately forecast unexpected events. Furthermore, being a linear-based model, ARIMA may not effectively capture complex non-linear relationships in time series data.

DeepAR algorithm

Deep learning-based forecasting algorithms offer solutions to the limitations of traditional models like ARIMA. They excel at capturing complex non-linear relationships and can effectively utilize multivariate datasets. These models enable the training of a global model, allowing for a single model to handle multiple similar target time series. This eliminates the need for creating separate models for each individual time series, providing a more efficient and scalable approach. **Deep Autoregressive (DeepAR)** is a state-of-the-art forecasting algorithm based on neural networks, designed to handle large datasets with multiple similar target time series. It has the capability to incorporate related time series, such as product prices or holiday schedules, to enhance the accuracy of its forecasting models. This feature proves particularly valuable when

dealing with spiky events triggered by external variables, allowing for more precise and reliable predictions. DeepAR utilizes a **recurrent neural network** (RNN) as its underlying model to capture patterns in the target time series. It goes beyond single-variable forecasting by incorporating multiple target time series and additional external supporting time series. Instead of considering individual values, the RNN takes input vectors representing the values of various variables at each time period. By jointly learning the patterns of these combined vectors over time, DeepAR can effectively capture the intrinsic non-linear relationships and shared patterns among the different time series. This approach enables DeepAR to train a single global model that can be used for forecasting across multiple similar target time series. DeepAR excels in handling complex multivariate datasets; however, it performs best when trained with large amounts of data. It is particularly useful in real-world scenarios involving large-scale retail forecasting for numerous items, where external factors like marketing campaigns and holiday schedules need to be taken into account. By leveraging its capability to model multiple variables simultaneously, DeepAR can provide accurate predictions and insights in such practical use cases.

Algorithms for recommendation

The recommender system has gained widespread adoption in various industries, including retail, media and entertainment, finance, and healthcare. It is an essential ML technology that predicts a user's preference for items, primarily relying on user or item attribute similarities or user-item interactions. Over the years, the field of recommendation algorithms has evolved significantly. In the following section, we will explore some commonly used algorithms in the realm of recommender systems.

Collaborative filtering algorithm

Collaborative filtering is a popular recommendation algorithm that leverages the notion that individuals with similar interests or preferences in one set of items are likely to have similar interests in other items as well. By analyzing the collective experiences and behaviors of different users, collaborative filtering can effectively recommend items to individual users based on the preferences of similar users. This approach taps into the experiences of the crowd to provide personalized and relevant recommendations. The following figure illustrates an item-user interaction matrix in the context of movie ratings. As you can see, it is a **sparse matrix**. This means that there are many empty entries in the matrix, which is expected as it is unlikely for any individual to have watched every movie:

	User 1	User 2	User 3	User 4	User n
Movie 1	5			4		
Movie 2	4		2			4
...						4
Movie n			3			

User-item interaction matrix

Figure 3.8: User-item interaction matrix for collaborative filtering

Matrix factorization is a technique commonly used in collaborative filtering for recommendation systems. It involves learning vector representations, or embeddings, for both users and items in the user-item interaction matrix. The goal is to approximate the original matrix by taking the product of the learned user and item embedding matrices. This allows us to predict the missing entries in the matrix, which represent the likely ratings that a user would give to unseen items. To make predictions, we simply compute the dot product between the user embedding and the item embedding. Embedding is a fundamental concept in machine learning that plays a crucial role in various domains. It involves creating numerical representations for entities, such as words or objects, in a manner that captures their semantic similarity. These representations are organized in a multi-dimensional space, where similar entities are positioned closer to each other. By using embeddings, we can uncover the underlying latent semantics of the objects, enabling more effective analysis and modeling. In the upcoming sections, we will delve deeper into embedding techniques and their applications in NLP algorithms.

Multi-arm bandit/contextual bandit algorithm

Collaborative filtering-based recommender systems heavily rely on prior interaction data between identified users and items to make accurate recommendations. However, these systems face challenges when there is a lack of prior interactions or when the user is anonymous, resulting in a cold start problem. To address this issue, one approach is to utilize a **multi-arm bandit** (MAB) based recommendation system. This approach draws inspiration from the concept of trial and error, similar to a gambler simultaneously playing multiple slot machines and observing which machine yields the best overall return. By employing reinforcement learning techniques, MAB-based recommendation systems dynamically explore and exploit different recommendations to optimize the user experience, even in the absence of substantial prior interaction data. An MAB algorithm operates under the paradigm of online learning, where it does not have pre-existing training data to train a model prior to deployment. Instead, it incrementally learns and adapts as data becomes available. In the initial stages of MAB learning, the model recommends all available options (such as products on an e-commerce site) with equal probabilities to users. As users begin to interact with a subset of the items and provide feedback (rewards), the MAB model adjusts its strategy. It starts to offer items that have yielded higher rewards (e.g., more user interactions) more frequently, exploiting the knowledge of their positive performance. However, the model also continues to allocate a smaller percentage of recommendations to new items, aiming to explore their potential for receiving interactions. This balance between exploration (offering new items) and exploitation (offering items with known rewards) is a fundamental tradeoff in MAB algorithms.

Algorithms for computer vision problems

Computer vision refers to the ability of computers to interpret and understand visual representations, such as images, in order to perform various tasks like object identification, text detection, face recognition, and activity detection. These tasks rely on pattern recognition, where images are labeled with object names and bounding boxes, and computer vision models are trained to recognize these patterns and make predictions on new images. Computer vision technology finds numerous applications in practical domains such as content management, security, augmented reality, self-driving cars, medical diagnosis, sports analytics, and quality inspection in manufacturing. In the following section, we will delve deeper into a few neural network architectures specifically designed for computer vision tasks.

Convolution neural network

A **convolutional neural network** (CNN) is a deep learning architecture specifically designed for processing and analyzing image data. It takes inspiration from the functioning of the animal visual cortex. In the visual cortex, individual neurons respond to visual stimuli within specific subregions of the visual field. These

subregions, covered by different neurons, partially overlap to cover the entire visual field. Similarly, in a CNN, different filters are applied to interact with subregions of an image, capturing and responding to the information within that region. This allows the CNN to extract meaningful features and patterns from the image data. A CNN architecture consists of multiple layers that repeat in a pattern. Each layer has different sublayers with specific functions. The convolutional layer plays a crucial role in feature extraction from input images. It utilizes convolutional filters, which are matrices defined by height and width, to extract relevant features. These convolutional layers process the input images by convolving them with the filters, producing feature maps that are passed to the next layer in the network. The pooling layer, found after one or multiple convolutional layers, reduces the dimensionality of the extracted features. It combines multiple outputs into a single output, resulting in a more compact representation. Two commonly used pooling techniques are max pooling, which selects the maximum value from the outputs, and average pooling, which calculates the average value. Following the convolutional and pooling layers, a fully connected layer is employed to combine and flatten the outputs from the previous layer. This layer aggregates the extracted features and feeds them into an output layer, typically used for tasks like image classification. The architecture of a CNN is illustrated in the following figure, showcasing the flow of information through the various layers:

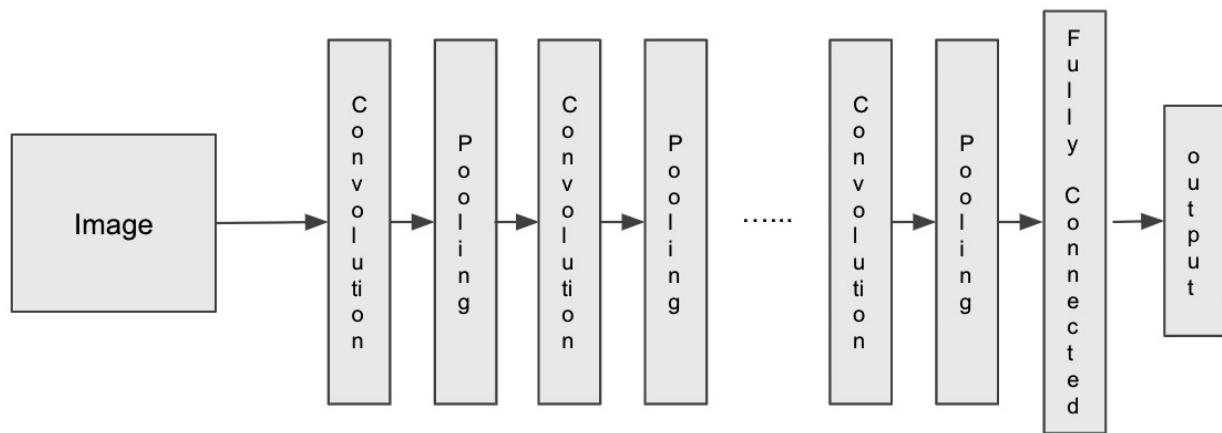


Figure 3.9: CNN architecture

CNN-based models offer efficient training due to their high degree of parallelism. This is particularly advantageous for tasks involving large-scale image data, where parallel processing can significantly accelerate training time. While CNNs are primarily used for computer vision tasks, their success has led to their application in other domains as well, including natural language processing. By adapting the principles of convolution and hierarchical feature extraction, CNNs have shown promise in tasks such as text classification and sentiment analysis. This demonstrates the versatility and effectiveness of CNN-based models beyond their traditional application in computer vision.

ResNet

As computer vision tasks grow in complexity, the addition of more layers in CNNs enhances their capability for image classification by enabling the learning of increasingly intricate features. However, as the number of layers increases in a CNN architecture, performance may deteriorate. This is commonly referred to as the **vanishing gradient** problem, where signals originating from the initial inputs, including crucial information, gradually diminish as they traverse through multiple layers of the CNN. **Residual networks (ResNet)** address the vanishing gradient problem by implementing a layer skipping technique. Rather than processing signals sequentially through each layer, ResNet introduces skip connections that allow signals to bypass certain layers. This can be visualized as a highway with fewer exits, enabling the signals from earlier

layers to be preserved and carried forward without loss. The ResNet architecture is depicted in the following figure.

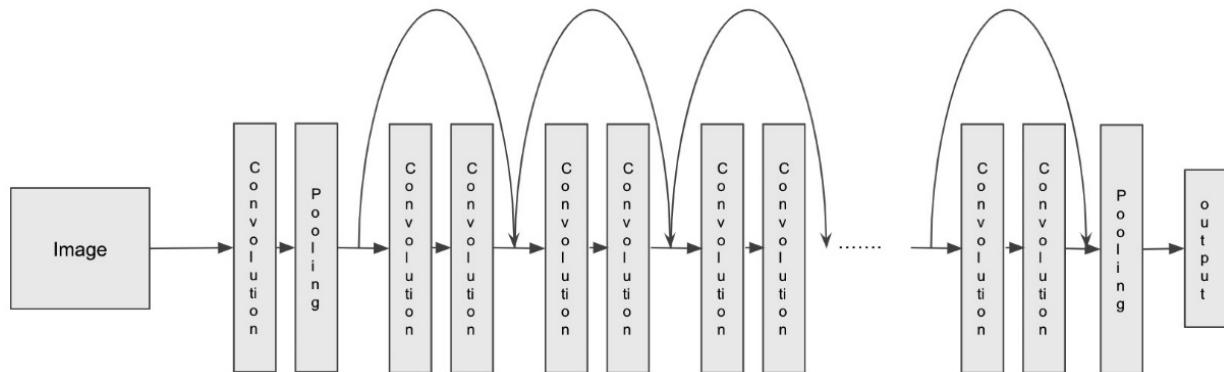


Figure 3.10: ResNet architecture

ResNet can be used for different computer vision tasks such as *image classification*, *object detection* (detecting all objects in a picture), and producing models with much higher accuracy than a vanilla CNN network. However, a potential disadvantage of ResNet is increased computational complexity due to the introduction of skip connections. The additional connections require more memory and computational resources, making training and inference more computationally expensive compared to shallower architectures.

Algorithms for natural language processing (NLP) problems

NLP focuses on the relationship between computers and human language. It involves the processing and analysis of extensive amounts of natural language data with the objective of enabling computers to comprehend the meaning behind human language and extract valuable information from it. NLP encompasses a wide range of tasks within the field of data science. Some of these tasks include document classification, topic modeling, converting speech to text, generating speech from text, extracting entities from text, language translation, understanding and answering questions, reading comprehension, and language generation. ML algorithms cannot process raw text data directly. To train NLP models effectively, it is necessary to convert the words within an input text into numerical representations within the context of other words, sentences, or documents. Before the advancement of embedding, there were two widely used methods for representing the relevance of words in a text: bag-of-words (BOW) and term frequency-inverse document frequency (TF-IDF). BOW is simply the count of a word appearing in a text (document). For example, if the input documents are **I need to go to the bank to make a deposit and I am taking a walk along the river bank**, and you count the number of appearances for each unique word in each input document, you will get 1 for the word *I*, and 3 for the word *to* in the first document, as an example. If we have a vocabulary for all the unique words in the two documents, the vector representation for the first document can be **[1 1 3 1 1 1 1 0 0 0 0]**, where each position represents a unique word in the vocabulary (for example, the first position represents the word *I*, and the third position represents the word *to*). Now, this vector can be fed into an ML algorithm to train a model such as text classification. The main idea behind BOW is that a word that appears more frequently has stronger weights in a text. TF-IDF has two components. The first component, *TF*, is the ratio of the number of times a vocabulary word appears in a document over the total number of words in the document. Using the preceding first document, the word *I* would have a TF value of $1/11$ for the first sentence, and the word *walk* would have a TF value of $0/11$, since *walk* does not appear in the first sentence. While TF measures the importance of a word in the context of one text, the IDF component measures the importance of a word across all the documents. Mathematically, it is the log of the ratio of the number of documents over the number of documents where a

word appears. The final value of TF-IDF for a word would be the *TF* term multiplied by the *IDF* term. In general, TF-IDF works better than BOW. Although BOW and TF-IDF are useful for NLP tasks, they lack the ability to capture the semantic meaning of words and often result in large and sparse input vectors. This is where the concept of embedding plays a crucial role. Embedding is a technique used to generate low-dimensional representations (mathematical vectors) for words or sentences, which capture the semantic meaning of the text. The underlying idea is that words or sentences with similar semantic meanings tend to occur in similar contexts. In a multi-dimensional space, the mathematical representations of semantically similar entities are closer to each other than those with different meanings. For instance, if we consider sports-related words like soccer, tennis, and bike, their embeddings would be close to each other in the high-dimensional embedding space, measured by metrics like cosine similarity. The embedding vector represents the intrinsic meaning of the word, with each dimension representing a specific attribute associated with the word. Visualizing embeddings in a multidimensional space shows the proximity of related entities. The following diagram provides a visual depiction of the closeness in this multidimensional space:

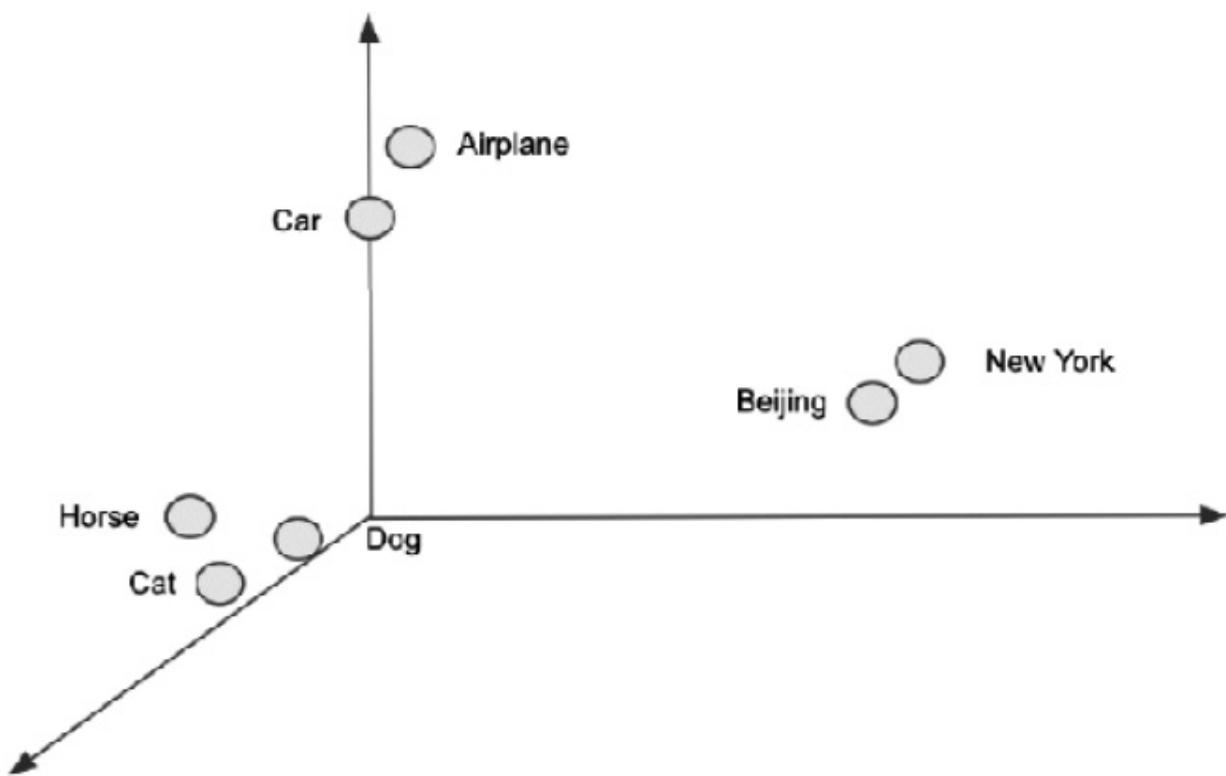


Figure 3.11: Embedding representation

Nowadays, embeddings have become a crucial component for achieving good results in most NLP tasks. Compared to other techniques like simple word counts, embeddings offer more meaningful representations of the underlying text. This has led to their widespread adoption in various machine learning algorithms designed for NLP. In the following section, we will delve into several of these algorithms, exploring their specific applications and benefits in the context of NLP tasks.

Word2Vec

Thomas Mikolov created **Word2Vec** in 2013. It supports two different techniques for learning embedding: **continuous bag-of-words (CBOW)** and **continuous-skip-gram**. CBOW tries to predict a word for a given window of surrounding words, and continuous-skip-gram tries to predict surrounding words for a given

word. The training dataset for Word2Vec could be any running text available, such as **Wikipedia**. The process of generating a training dataset for CBOW is to run a sliding window across running text (for example, a window of five words) and choose one of the words as the target and the rest as inputs (the order of words is not considered). In the case of continuous-skip-gram, the target and inputs are reversed. With the training dataset, the problem can be turned into a multi-class classification problem, where the model will learn to predict the classes (for example, words in the vocabulary) for the target word and assign each predicted word with a probability distribution. Word2Vec embeddings can be trained using a straightforward one-hidden-layer MLP network. In this approach, the input to the MLP network is a matrix that represents the neighboring words, while the output is a probability distribution for the target words. During training, the weights of the hidden layer are optimized, and once the training process is complete, these weights serve as the actual embeddings for the words. The resulting embeddings capture the semantic relationships and contextual meanings of the words, enabling them to be effectively utilized in various natural language processing tasks. As large-scale word embedding training can be expensive and time-consuming, Word2Vec embeddings are usually trained as a pre-training task so that they can be readily used for downstream tasks such as text classification or entity extraction. This approach of using embeddings as features for downstream tasks is called a **feature-based application**. There are pre-trained embeddings (for example, Tomas Mikolov's Word2Vec and Stanford's GloVe) in the public domain that can be used directly. The embeddings are a 1:1 mapping between each word and its vector representation.

BERT

Word2Vec produces a fixed embedding representation for each word in the vocabulary, disregarding the contextual variations in meaning. However, words can have different meanings depending on the specific context in which they are used. For instance, the word "bank" can refer to a financial institution or the land alongside a body of water. To address this issue, contextualized word embeddings have been developed. These embeddings take into account the surrounding words or the overall context in which a word appears, allowing for a more nuanced and context-aware representation. By considering context, these embeddings capture the diverse meanings a word can have, enabling more accurate and context-specific analyses in downstream tasks. **BERT**, which stands for **Bidirectional Encoder Representations from Transformers**, is a language model that takes context into consideration by the following:

- Predicting randomly masked words in sentences (the context) and taking the order of words into consideration. This is also known as **language modeling**.
- Predicting the next sentence from a given sentence.

Released in 2018, this context-aware embedding approach provides better representation for words and can significantly improve language tasks such as *reading comprehension*, *sentiment analysis*, and *named entity resolution*. Additionally, BERT generates embeddings at subword levels (a segment between a word and a character, for example, the word *embeddings* is broken up into *em*, *bed*, *ding*, and *s*). This allows it to handle the **out-of-vocabulary (OOV)** issue, another limitation of Word2Vec, which only generates embeddings on known words and will treat OOV words simply as unknown. To obtain word embeddings with BERT, the process differs from the straightforward word-to-vector mapping used in Word2Vec. Instead, sentences are inputted into a pre-trained BERT model, and embeddings are extracted dynamically. This approach generates embeddings that are contextualized within the context of the given sentences. Besides word-level embeddings, BERT is also capable of producing embeddings for entire sentences. **Pre-training** is the term used to describe the process of learning embeddings using input tokens, and the diagram below illustrates the components involved in a BERT model for this purpose.

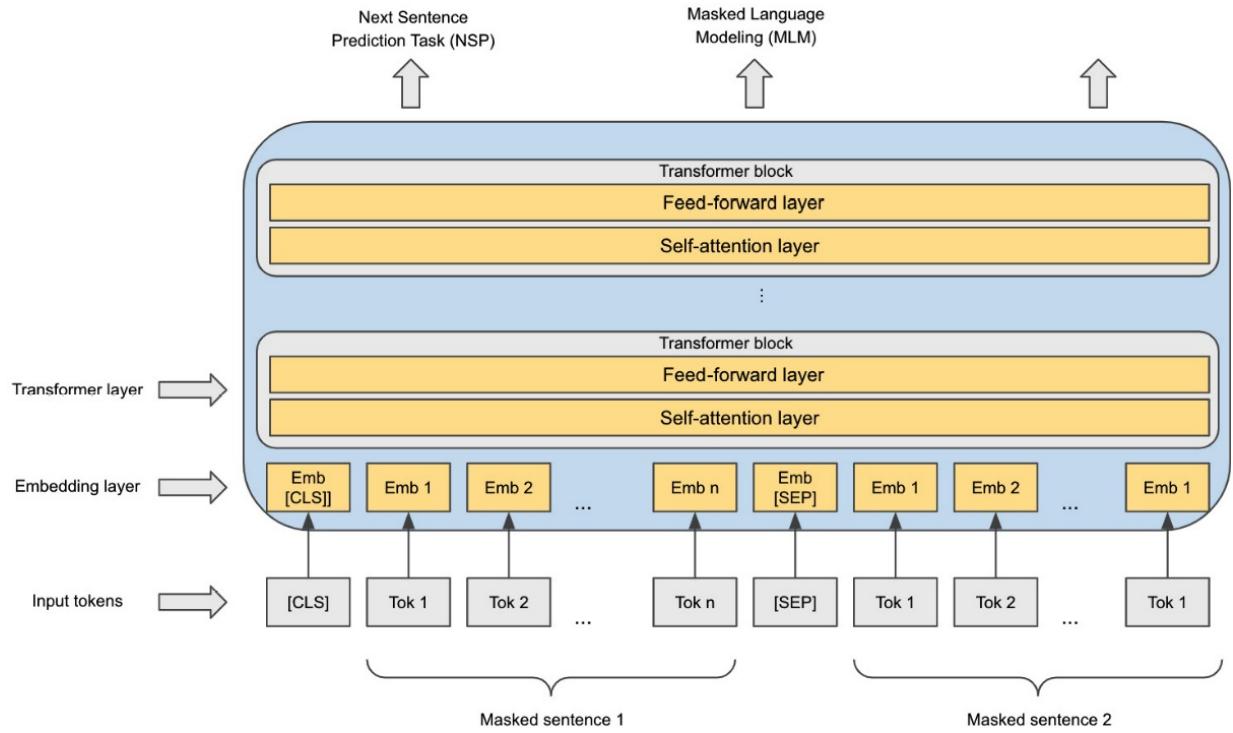


Figure 3.13: BERT model pre-training

Architecturally, BERT mainly uses a building block called a **transformer**. A transformer has a stack of encoders and a stack of decoders inside it, and it transforms one sequence of inputs into another sequence. Each encoder has two components:

1. A self-attention layer mainly calculates the strength of the connection between one token (represented as a vector) and all other tokens in the input sentence, and this connection helps with the encoding of each token. One way to think about self-attention is which words in a sentence are more connected than other words in a sentence. For example, if the input sentence is *The dog crossed a busy street*, then we would say the words *dog* and *crossed* have stronger connections with the word *The* than the word *a* and *busy*, which would have strong connections with the word *street*. The output of the self-attention layer is a sequence of vectors; each vector represents the original input token as well as the importance it has with other words in the inputs.
2. A feed-forward network layer (single hidden layer MLP) extracts higher-level representation from the output of the self-attention layer.

Inside the decoder, there is also a self-attention layer and feed-forward layer, plus an extra encoder-decoder layer that helps the decoder to focus on the right places in the inputs. In the case of BERT, only the encoder part of the transformer is used. BERT can be used for a number of NLP tasks, including *question answering*, *text classification*, *named entity extraction*, and *text summarization*. It achieved state-of-the-art performance in many of the tasks when it was released. BERT pre-training has also been adopted for different domains, such as scientific text and biomedical text, to understand domain-specific languages. The following figure showcases how a pre-trained BERT model is used to train a model for a question-answering task using the fine-tuning technique:

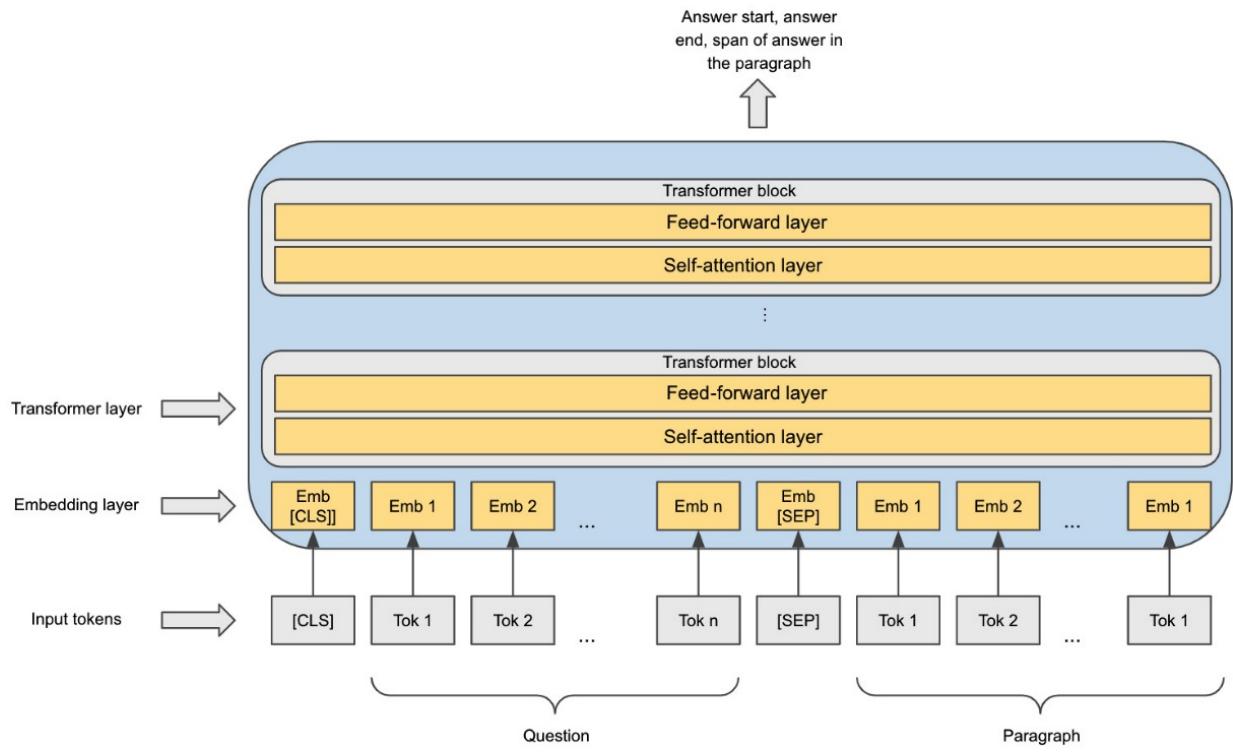


Figure 3.14: BERT fine-tuning

While BERT's pre-trained embeddings can be extracted for downstream tasks such as text classification and question answering, a more straightforward way to use its pre-trained embeddings is through a technique called **fine-tuning**. With fine-tuning, an additional output layer is added to the BERT network to perform a specific task, such as question answering or entity extraction. During fine-tuning, the pre-trained model is loaded, and you plug in the task-specific input (for example, question/passage pairs in question answering) and output (start/end and span for the answers in the passage) to fine-tune a task-specific model. With fine-tuning, the pre-trained model weights are updated.

Generative pre-trained transformer (GPT)

Unlike BERT, which requires fine-tuning using a large domain-specific dataset for the different downstream NLP tasks, the **Generative Pre-trained Transformer (GPT)**, developed by **OpenAI**, can learn how to perform a task with just seeing a few examples (or no example). This learning process is called **few-shot learning** or **zero-shot learning**. In a few-shot scenario, the GPT model is provided with a few examples, a task description, and a prompt, and the model will use these inputs and start to generate output tokens one by one. For instance, when using GPT-3 for translation, the task description could be "translate English to Chinese," and the training data would consist of a few examples of Chinese sentences translated from English sentences. To translate a new English sentence using the trained model, you provide the English sentence as a prompt, and the model generates the corresponding translated text in Chinese. It's important to note that few-shot or zero-shot learning does not involve updating the model's parameter weights, unlike the fine-tuning technique. GPT, like BERT, utilizes the Transformer architecture as its primary component and employs a training approach called next word prediction. This entails predicting the word that should follow a given sequence of input words. However, GPT diverges from BERT in that it exclusively employs the Transformer decoder block, whereas BERT employs the Transformer encoder block. Like BERT, GPT incorporates masked words to learn embeddings. However, unlike BERT, which randomly masks words and predicts the missing ones, GPT restricts the self-attention calculation to exclude words to the right of the

target word. This approach is referred to as masked self-attention. GPT and its chatbot interface, ChatGPT, have showcased exceptional capabilities in numerous traditional NLP tasks like language modeling, language translation, and question answering. Additionally, they have proven their efficacy in pioneering domains such as generating programming code or machine learning code, composing website content, and answering questions. Consequently, GPT has paved the way for a novel AI paradigm known as Generative AI.

Generative AI algorithms

Although technologies like ChatGPT have popularized Generative AI, the concept of generative models is not new. Generative Adversarial Networks (GANs), a prominent example of generative AI technology, have been around for years and have been successfully applied in various real-world domains, with image synthesis being a notable application. Generative AI has become one of the most transformative AI technologies, and I have devoted the entire chapter 15 to delve deeper into real-world generative AI use cases, practical technology solutions, and ethical considerations. In this chapter, we will familiarize ourselves with several generative AI algorithms, expanding on the information previously covered regarding GPT.

Generative adversarial network (GAN)

The Generative Adversarial Network is a type of generative model designed to generate realistic data instances, such as images. It employs a two-part network consisting of a Generator and a Discriminator. The Generator network is responsible for generating instances, while the Discriminator network learns to distinguish between real and fake instances generated by the Generator. This adversarial setup encourages the Generator to continually improve its ability to produce increasingly realistic data instances.

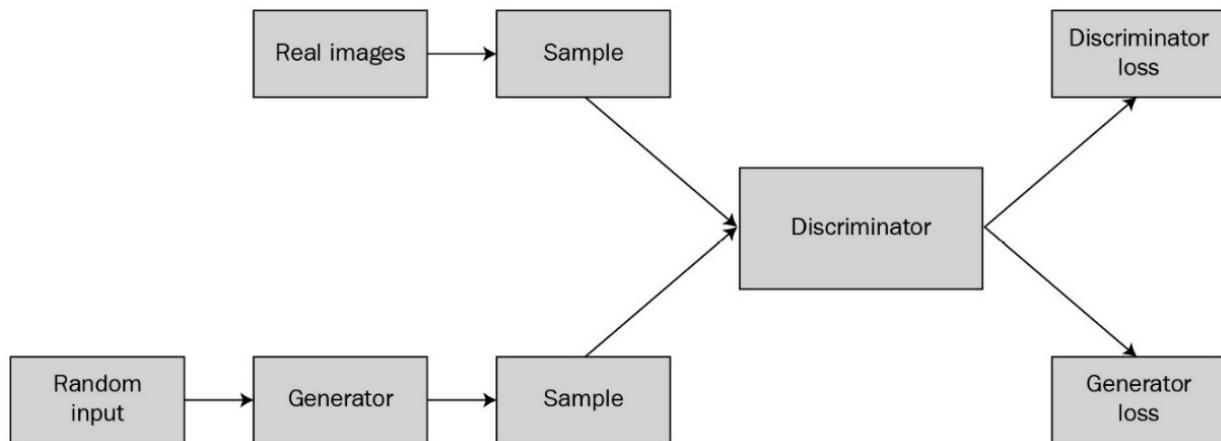


Figure 3.16: Generative adversarial network

During the training process, the Discriminator network in a Generative Adversarial Network is exposed to two different data sources: one from a real dataset, which serves as positive examples, and one from the Generator network, which generates synthetic or fake samples. The Discriminator is trained to classify and distinguish between the real and fake samples, optimizing its loss to accurately predict the source of each sample. Conversely, the Generator network is trained to produce synthetic data that appears indistinguishable from real data, aiming to deceive the Discriminator. The Generator is penalized when the Discriminator correctly identifies its generated data as fake. Both networks learn and update their parameters using backpropagation, enabling them to improve iteratively. During the generation phase, the

Generator is provided with random inputs to produce new synthetic samples. Throughout training, the Generator and Discriminator networks are alternately trained in a connected manner, allowing them to learn and optimize their performance as a unified system. GAN has had a lot of success in generating realistic images that can fool humans. For example, it can be applied to many applications, such as translating sketches to realistic-looking images, converting text inputs and generating images corresponding to the text, and generating realistic human faces.

Large Language Model (LLM)

LLMs are a class of generative AI model that is capable of generating text, translating languages, producing creative content, and providing informative answers to questions. LLMs are trained on extensive datasets comprising text and code, with billions of model parameters, allowing them to understand and learn the statistical patterns and relationships between words and phrases. For example, GPT-3, an LLM, has 175 billion parameters after training. This training enables LLMs to effectively process and generate human-like text across a wide range of applications. While GPT serves as a notable example of an LLM, the open-source community and other companies have developed other LLMs in recent years, mainly using the similar Transformer based architecture. LLMs are also called foundation models. Unlike traditional ML models, which are trained for specific tasks, foundation models are pre-trained with massive dataset and can handle multiple tasks. In addition, foundation models can be fine-tuned and adapted for additional tasks. The remarkable capabilities and adaptability of foundation models have found many exciting AI use cases that were not easily solvable before. Now, let's briefly review a few other popular foundation models:

- **Google Pathways Language Model (PaLM)** is a 540-Billion parameters, decoder only Transformer model. It offers similar capabilities as GPT, including text generation, translation, code generation, question answering, summarization, and support for creating chatbot. PaLM is trained using a new architecture called Pathways. Pathways is a modular architecture, meaning that it is composed of modules, each is responsible for a specific task.
- **Meta Large Language Model Meta AI (LLaMA)** is an LLM available in multiple sizes from 7 billion parameters to 65 billion parameters. While it is a smaller model comparing to GPT and PaLM, it offers several advantages such as requiring fewer computational resources. LLaMA offers similar capabilities as other LLMs such as generating creative text, answering questions, and solving mathematical problems. Meta has issued a noncommercial license for LLaMA, emphasizing its usage in research contexts. LLaMA has also been found to perform extremely, when it is fine-tuned with additional training data.
- **Big Science BLOOM** is a 176-billion LLM that can generate text in 46 different languages and 13 programming languages. The development of BLOOM involved the collaborative efforts of more than 1,000 researchers from over 70 countries and 250 institutions. As part of the Responsible AI License, individuals and institutions who agree to its terms can use and build upon the model on their local machines or cloud platforms. The model is easily accessible within the Hugging Face ecosystem.
- **Bloomberg BloombergGPT** – While general purpose LLMs like GPT and LLaMA can perform well in a range of tasks across different domain domains, some domains such as financial services and life science require domain specific LLMs to solve tough domain focused problems. BloombergGPT is an example of domain focused LLMs that are specifically trained for industries. BloombergGPT represents a significant advancement in applying this technology to finance. The model will enhance existing financial NLP tasks such as sentiment analysis, named entity recognition, news classification, and question answering, among others. Drawing from their extensive collection and curation resources, Bloomberg utilized its four-decade archive of financial language documents, resulting in a comprehensive 363 billion token dataset comprised of English financial documents. This dataset was augmented with a 345 billion token public dataset, yielding a training corpus with over 700 billion tokens.

While these LLM models have exhibited remarkable capabilities, they also come with significant limitations, including generating misinformation (hallucinations) and toxic content, and displaying potential biases. LLMs are also consume significantly more resources to train and run. It is worth noting that while LLMs can help solve some new problems, many of the common problems (e.g., name entity extraction, document classification, sentiment analysis) have been solved with existing NLP techniques, which remain to be highly viable options for those tasks.

Diffusion Model

Recently, AI's remarkable ability to generate high resolution photo-realistic images and never-seen-before generative art or manipulate images with precision has garnered significant attention. Behind all these amazing capabilities is a new type of deep learning models called diffusion model. Building upon the foundations of deep learning and GANs, the diffusion model introduces a novel approach to generating high-quality, realistic data instances. Unlike GAN, which try to generate realistic fake image by fooling a discriminator network, diffusion model works by first adding noises to the input data (e.g., images) incrementally over many steps until the input data is unrecognizable, a process called diffusion steps. The model is then trained to reverse the diffusion steps from noise to the original data. In more technical terms, the training process of a diffusion model involves optimizing a set of learnable parameters through backpropagation. The model learns to generate realistic samples by maximizing the likelihood of the training data given a sequence of diffusion steps. This iterative process allows the model to capture complex dependencies, intricate patterns, structures and generate highly realistic and diverse data instances. The following figure illustrates this process:

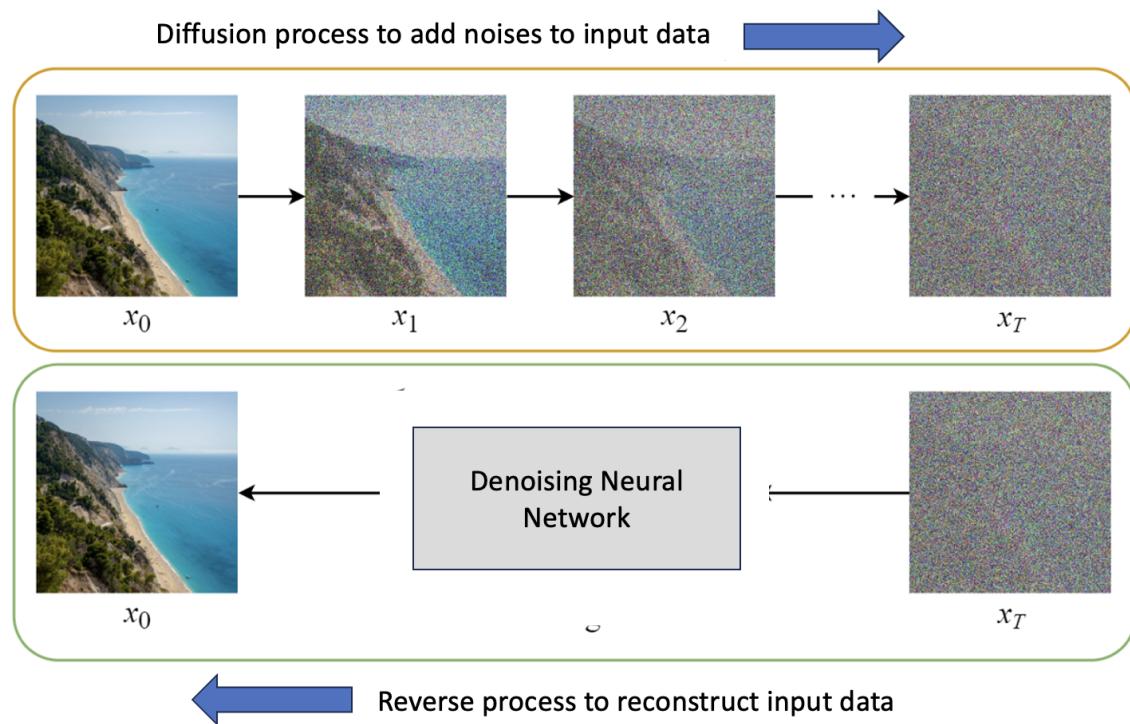


Figure 3.17: How diffusion model works

Furthermore, the diffusion model offers flexibility and controllability in the generation process. By adjusting the diffusion steps, one can control the trade-off between sample quality and diversity. This allows users to fine-tune the model to suit their specific needs, whether it's emphasizing fidelity to the training data or

encouraging more creative and novel outputs. Compared to GANs, diffusion models can generate more realistic images and are more stable than GANs. The diffusion model has shown great promise in various domains, including computer vision, natural language processing, and audio synthesis. Its ability to generate high-quality data with fine-grained details has opened up exciting possibilities for applications such as image generation, video prediction, text generation, and more. The open-source community and private companies have developed many models based on this diffusion approach. Two of more popular models worth mentioning are Stable Diffusion and DALL-E 2.

- **DALL-E 2 By OpenAI** - DALL-E 2 is a text-to-image model developed by OpenAI. DALL-E 2 was trained using a dataset of images and text descriptions. First released in Jan 2022, DALL-E 2 has shown remarkable capabilities in generating or manipulating images from text descriptions. It has also been applied for inpainting (modifying regions in images), outpainting (extending an image), and image-to-image translation. The images generated by DALL-E 2 are often indistinguishable from real images, and they can be used for a variety of purposes, such as creating art, generating marketing materials. From a model training perspective, DALL-E 2 training comprises of 2 key steps:
 - **Linking textual semantics and visual representations:** This step learns how a piece of text, such as "a man wearing a hat" is semantically linked to an actual image of "a man wearing a hat". To do this, DALL-E 2 uses a model called Contrastive Language-Image Pre-training (CLIP). CLIP is trained with hundreds of millions of images and their associated descriptions. After it is trained, it can output text-conditioned visual encoding given a piece of text description. You can learn more about CLIP at <https://openai.com/research/clip>.
 - **Generating images from visual embeddings:** This step learns to reverse the image from the visual embeddings generated by the CLIP. For this step, DALL-E 2 uses a model called GLIDE, which is based the Diffusion Model. You can learn more about GLIDE at <https://arxiv.org/abs/2112.10741>.

After the model is trained, DALL-E 2 can generate new image that closely related to an input text description.

- **Stable Diffusion by Stability AI** - Stable diffusion an algorithm developed by Compvis (the Computer Vision research group at Ludwig Maximilian University of Munich) and sponsored primarily by Stability AI. The model is also a text-to-image model trained using a dataset of real images and text descriptions, which allows the model to generate realistic images using text descriptions. First released in August 2022, Stable diffusion has been shown to be effective at generating high-quality images from text descriptions. Architecturally, it employs a CLIP encoder to condition the model on text descriptions, and it uses UNET as the denoising neural network to generate image from the visual encoding. It is an open-source model with released code and model weights to the public. You can get more detail on Stable diffusion at <https://github.com/CompVis/stable-diffusion>.

As powerful as the diffusion models are, there are also concerns around using diffusion models such as copywrite infringement, and creation of harmful images.

Hands-on exercise

In this hands-on exercise, we will build a **Jupyter Notebook** environment on your local machine and build and train an ML model in your local environment. The goal of the exercise is to get some familiarity with the installation process of setting up a local data science environment, and learn how to analyze the data, prepare the data, and train an ML model using one of the algorithms we covered in the preceding sections. First, let's take a look at the problem statement.

Problem statement

Before we start, let's first review the business problem that we need to solve. A retail bank has been experiencing a high customer churn rate for its retail banking business. To proactively implement preventive measures to reduce potential churn, the bank needs to know who the potential churners are, so the bank can target those customers with incentives directly to prevent them from leaving. From a business perspective, it is far more expensive to acquire a new customer than offering incentives to keep an existing customer. As an ML solutions architect, you have been tasked to run some quick experiments to validate the ML approach for this problem. There is no ML tooling available, so you have decided to set up a Jupyter environment on your local machine for this task.

Dataset description

You will use a dataset from the Kaggle site for bank customers' churn for modeling. You can access the dataset at <https://www.kaggle.com/mathchi/churn-for-bank-customers>. The dataset contains 14 columns for features such as credit score, gender, and balance, and a target variable column, **Exited**, to indicate if a customer churned or not. We will review those features in more detail in later sections.

Setting up a Jupyter Notebook environment

Now, let's set up a local data science environment for data analysis and experimentation. We will be using the popular Jupyter Notebook on your local computer. Setting up a Jupyter Notebook environment on a local machine consists of the following key components:

- **Python:** Python is a general-purpose programming language, and it is one of the most popular programming languages for data science work. Installation instructions can be found at <https://www.python.org/downloads>.
- **PIP:** PIP is a Python package installer used for installing different Python library packages, such as ML algorithms, data manipulation libraries, or visualization. Installation instructions can be found at <https://pip.pypa.io/en/stable/installation/>.
- **Jupyter Notebook:** Jupyter Notebook is a web application designed for authoring documents (called notebooks) that contain code, description, and/or visualizations. It is one of the most popular tools for data scientists to do experimentation and modeling. Installation instructions can be found at <https://jupyter.org/install>.

Running the exercise

1. With your environment configured, let's get started with the actual data science work. First, download the data files:
 - A. Let's create a folder called **MLSALab** on your local machine to store all the files. You can create the folder anywhere on your local machine as long as you can get to it. I have a Mac, so I created one directly inside the default user **Documents** folder.
 - B. Create another subfolder called **Lab1-bankchurn** under the **MLSALab** folder.
 - C. Visit the <https://www.kaggle.com/mathchi/churn-for-bank-customers> site and download the data file (an archive file) and save it in the **MSSALab/Lab1-bankchurn** folder. Create a Kaggle account if you do not already have one. Extract the archive file inside the folder, and you will see a file called **churn.csv**. You can now delete the archive file.
2. Launch Jupyter Notebook:
 - A. Inside the Terminal window (or the Command Prompt window for Windows systems), navigate to the **MLSALab** folder and run the following command to start the Jupyter Notebook server on your machine:

```
jupyter notebook
```

A browser window will open up and display the Jupyter Notebook environment (see the following screenshot). Detailed instructions on how Jupyter Notebook works is out of scope for this lab. If you are not familiar with how Jupyter Notebook works, you can easily find information on the internet:

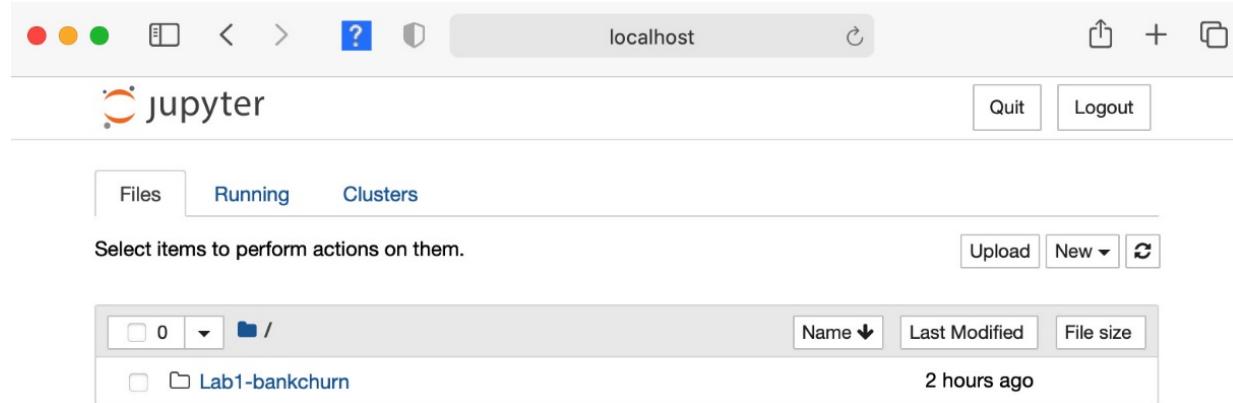


Figure 3.17: Jupyter Notebook

1. Click on the **Lab1-bankchurn** folder and you will see the **churn.csv** file.
1. Experimentation and model building: Now, let's create a new data science notebook inside the Jupyter Notebook environment. To do this, you click on the **New** dropdown and select **Python 3** (see following screenshot):

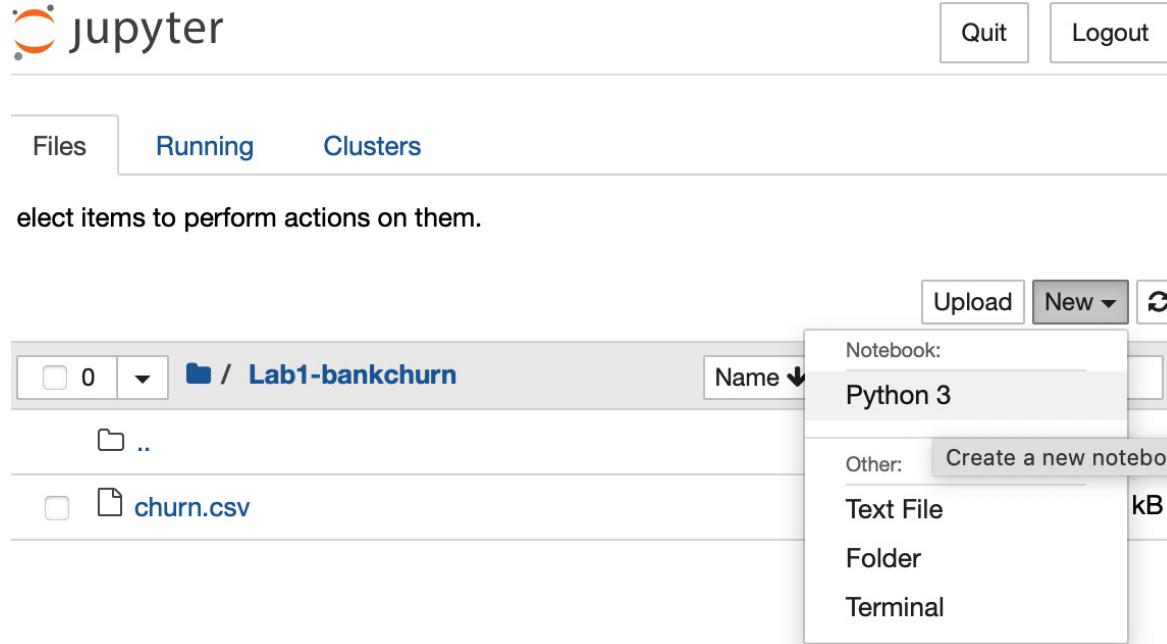


Figure 3.18: Creating a new Jupyter notebook

2. You will see a screen similar to the following screenshot. This is an empty notebook that we will use to explore data and build models. The section next to **In []:** is called a **cell**, and we will enter our code into the cell. To run the code in the cell, you click on the **Run** button on the toolbar. To add a new cell, you click on the **+** button on the toolbar:

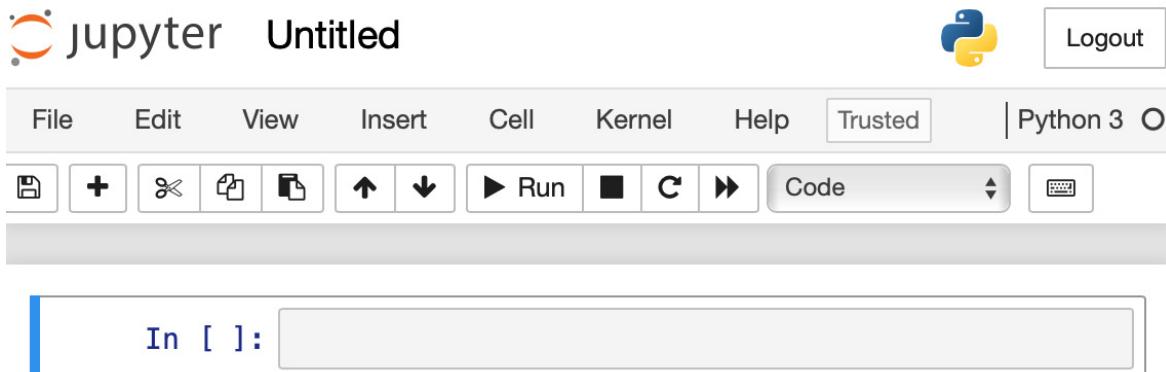


Figure 3.19: Empty Jupyter notebook

3. Add a new cell by clicking on the + button in the toolbar, enter the following code block inside the first empty cell, and run the cell by clicking on the **Run** button in the toolbar. This code block downloads a number of Python packages for data manipulation (**pandas**), visualization (**matplotlib**), and model training and evaluation (**scikit-learn**). We will cover scikit-learn in greater detail in *Chapter 5, Open Source Machine Learning Libraries*. We will use these packages in the following sections:

```
! pip3 install pandas
! pip3 install matplotlib
! pip3 install scikit-learn
```

1. Now, we can load and explore the data. Add the following code block in a new cell to load the Python library packages and load the data from the `churn.csv` file. You will see a table with 14 columns, where the **Exited** column is the target column:

```
import pandas as pd
churn_data = pd.read_csv("churn.csv")
churn_data.head()
```

1. You can explore the dataset using a number of tools to understand information with the commands that follow, such as *dataset statistics*, the *pairwise correlation between different features*, and *data distributions*. The `describe()` function returns basic statistics about the data such as mean, standard deviation, min, and max, for each numerical column. The `hist()` function plots the histogram for the selected columns, and `corr()` calculates the correlation matrix between the different features in the data. Try them out one at a time in a new cell to understand the data:

```
# The following command calculates the various statistics for the features.
churn_data.describe()
# The following command displays the histograms for the different features.
# You can replace the column names to plot the histograms for other features
churn_data.hist(['CreditScore', 'Age', 'Balance'])
# The following command calculate the correlations among features
churn_data.corr()
```

1. The dataset needs transformations in order to be used for model training. The following code block will convert the `Geography` and `Gender` values from categorical strings to ordinal numbers so they can be taken by the ML algorithm later. Please note that model accuracy is not the main purpose of this exercise, and we are performing ordinal transformation for demonstration purposes. Copy and run the following code block in a new cell:

```

from sklearn.preprocessing import OrdinalEncoder
encoder_1 = OrdinalEncoder()
encoder_2 = OrdinalEncoder()
churn_data['Geography_code'] = encoder_1.fit_transform(churn_data[['Geography']])
churn_data['Gender_code'] = encoder_2.fit_transform(churn_data[['Gender']])

```

1. There are some columns not needed for model training. We can drop them using the following code block:

```
churn_data.drop(columns = ['Geography', 'Gender', 'RowNumber', 'Surname'], inplace=True)
```

1. Now, the dataset has only the features we care about. Next, we need to split the data for training and validation. We also prepare each dataset by splitting the target variable, `Exited`, from the rest of the input features. Enter and run the following code block in a new cell:

```

# we import the train_test_split class for data split
from sklearn.model_selection import train_test_split
# Split the dataset into training (80%) and testing (20%).
churn_train, churn_test = train_test_split(churn_data, test_size=0.2)
# Split the features from the target variable "Exited" as it is required for model training
# and validation later.
churn_train_X = churn_train.loc[:, churn_train.columns != 'Exited']
churn_train_y = churn_train['Exited']
churn_test_X = churn_test.loc[:, churn_test.columns != 'Exited']
churn_test_y = churn_test['Exited']

```

1. We are ready to train the model. Enter and run the following code block in a new cell. Here, we will use the random forest algorithm to train the model, and the `fit()` function kicks off the model training:

```

# We will use the Random Forest algorithm to train the model
from sklearn.ensemble import RandomForestClassifier
bank_churn_clf = RandomForestClassifier(max_depth=2, random_state=0)
bank_churn_clf.fit(churn_train_X, churn_train_y)

```

1. Finally, we will test the accuracy of the model using the `test` dataset. Here, we get the predictions returned by the model using the `predict()` function, and then use the `accuracy_score()` function to calculate the model accuracy using the predicted values (`churn_prediction_y`) and the true values (`churn_test_y`) for the test dataset:

```

# We use the accuracy_score class of the sklearn library to calculate the accuracy.
from sklearn.metrics import accuracy_score
# We use the trained model to generate predictions using the test dataset
churn_prediction_y = bank_churn_clf.predict(churn_test_X)
# We measure the accuracy using the accuracy_score class.
accuracy_score(churn_test_y, churn_prediction_y)

```

Congratulations! You have successfully installed a Jupyter data science environment on your local machine and trained a model using the random forest algorithm. You have validated that an ML approach could potentially solve this business problem.

Summary

In this chapter, we have explored various machine learning (ML) algorithms that can be applied to solve different types of ML problems. By now, you should have a good understanding of which algorithms are suitable for specific tasks. Additionally, you have set up a basic data science environment on your local machine, utilized the scikit-learn ML libraries to analyze and preprocess data, and successfully trained an

ML model. In the upcoming chapter, our focus will shift to the intersection of data management and the ML life cycle. We will delve into the significance of effective data management and discuss how to build a comprehensive data management platform on AWS (Amazon Web Services) to support downstream ML tasks. This platform will provide the necessary infrastructure and tools to streamline data processing, storage, and retrieval, ultimately enhancing the overall ML workflow.

4 Data Management for Machine Learning

Join our book community on Discord

<https://packt.link/EarlyAccessCommunity>



As an ML solutions architecture practitioner, I often receive requests for guidance on designing data management platforms for ML workloads. Although data management platform architecture is typically treated as a separate technical discipline, it plays a crucial role in ML workloads. To create a comprehensive ML platform, ML solutions architects must understand the essential data architecture considerations for machine learning and be familiar with the technical design of a data management platform that caters to the needs of data scientists and automated ML pipelines. In this chapter, we will explore the intersection of data management and ML, discussing key considerations for designing a data management platform specifically tailored for ML. We will delve into the core architecture components of such a platform and examine relevant AWS technologies and services that can be used to build it.

Technical requirements

In this chapter, you will need access to an AWS account and AWS services such as **Amazon S3**, **Amazon Lake Formation**, **AWS Glue**, and **AWS Lambda**. If you do not have an AWS account, follow the official AWS website's instructions to create an account.

Data management considerations for ML

Data management is a broad and complex topic. Many organizations have dedicated data management teams and organizations to manage and govern the various aspects of a data platform. Historically, data management primarily revolved around fulfilling the requirements of transactional systems and analytics systems. However, as ML solutions gain prominence, there are now additional business and technology factors to consider when it comes to data management platforms. The advent of ML introduces new requirements and challenges that necessitate an evolution in data management practices to effectively support these advanced solutions. To understand where data management intersects with the ML workflow, let's bring back the ML life cycle, as illustrated in the following figure:

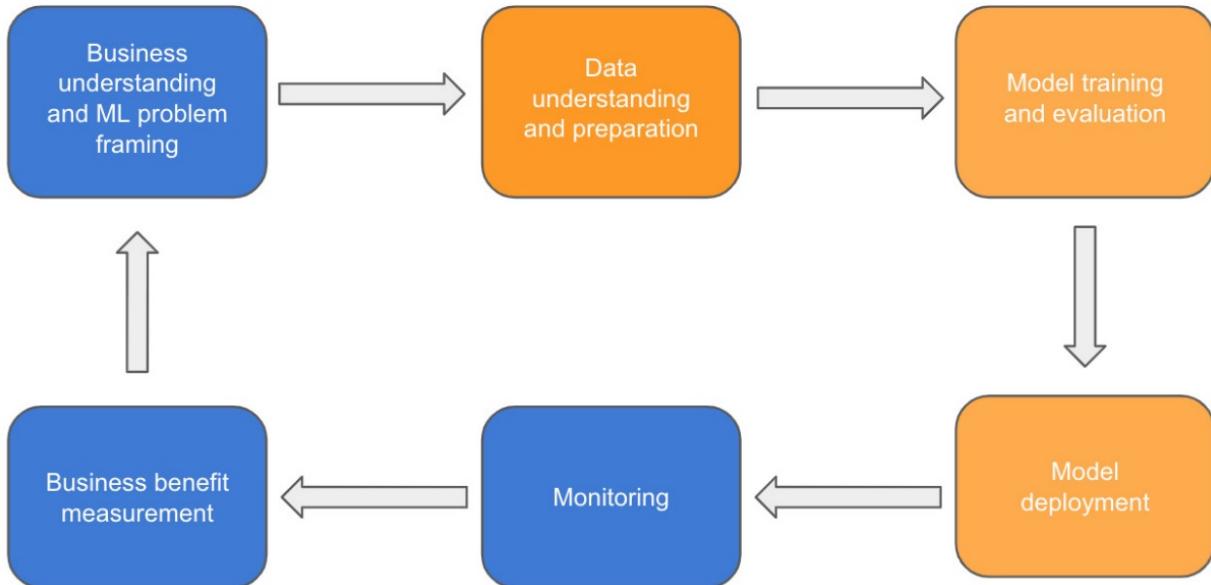


Figure 4.1: Intersection of data management and the ML life cycle

At a high level, data management intersects with the ML life cycle in three stages: *data understanding and preparation*, *model training and evaluation*, and *model deployment*. During the *data understanding and preparation* stage, data scientists undertake several essential tasks. They begin by identifying relevant data sources that contain datasets suitable for their modeling tasks. Exploratory data analysis is then performed to gain insights into the dataset, including data statistics, correlations between features, and data sample distributions. Additionally, data preparation for model training and validation is crucial, involving a series of steps that typically include the following:

- Data validation: The data is checked for errors and anomalies to ensure its quality. This includes verifying data range, distribution, data types, and identifying missing or null values.
- Data cleaning: Any identified data errors are fixed or corrected to ensure the accuracy and consistency of the dataset. This may involve removing duplicates, handling missing values, or resolving inconsistencies.
- Data enrichment: Additional value is derived from the data through techniques like joining different datasets or transforming the data. This helps generate new signals and insights that can enhance the modeling process.
- Data labeling: For supervised ML model training, training and testing dataset need to labeled by human annotators or ML model accurately. This critical step is necessary to guarantee the development and validation of high-quality models.

The data management capabilities needed during this stage encompass the following aspects:

- Dataset discovery: The capability to search and locate curated datasets using relevant metadata like dataset name, description, field name, and data owner.
- Data access: The ability to access both raw and processed datasets for performing exploratory data analysis. This ensures data scientists can explore and analyze the data effectively.
- Querying and retrieval: The capability to run queries against selected datasets to obtain details such as statistical information, data quality metrics, and data samples. Additionally, the ability to retrieve data from the data management platform to a data science environment for further processing and feature engineering.
- Scalable data processing: The ability to execute data processing operations on large datasets efficiently. This ensures that data scientists can handle and process substantial amounts of data during model development and experimentation.

During the stage of model training and validation, data scientists are responsible for generating a training and validation dataset to conduct formal model training. To facilitate this process, the following data management

capabilities are essential:

1. Data processing and automated workflows: A data management platform should provide robust data processing capabilities along with automated workflows. This enables the conversion of raw or curated datasets into training and validation datasets in various formats suitable for model training.
2. Data repository and versioning: An efficient data management platform should offer a dedicated data repository to store and manage the training and validation datasets. Additionally, it should support versioning, allowing data scientists to keep track of different iterations and modifications made to the datasets, along with the versions of the code and trained ML models.
3. Data labeling: For supervised ML model training, training and testing dataset need to be labeled by human annotators or ML model accurately. This critical step is necessary to guarantee the development and validation of high-quality models. This is a highly labor-intensive task, requiring purpose-built software tools to do it at scale.
4. ML features/embeddings generation and storage: Some ML features/embeddings (e.g., averages, sums, text embeddings) need to be pre-computed for one or more downstream model training tasks. These features/embeddings often need to be managed using purpose-built tools for efficient access and re-use.
5. Dataset provisioning for model training: The platform should provide mechanisms to serve the training and validation datasets to the model training infrastructure. This ensures that the datasets are accessible by the training environment, allowing data scientists to train models effectively.

During the stage of model deployment, the focus shifts towards utilizing the trained models to serve predictions. To support this stage effectively, the following data management capabilities are crucial:

- Serving data for feature processing: The data management platform should be capable of serving the necessary data required for feature processing as part of the input data when invoking the deployed models. This ensures that the models receive the relevant data inputs required for generating predictions.
- Serving pre-computed features/embeddings: In some cases, pre-computed features/embeddings are utilized as inputs when invoking the deployed models. The data management platform should have the capability to serve these pre-computed features seamlessly, allowing the models to incorporate them into the prediction process.

In contrast to traditional data access patterns for transactional or business intelligence solutions, where developers can utilize non-production data in lower environments for development purposes, data scientists typically require access to production data for model development. Having explored the considerations for ML data management, we will now delve deeper into the data management architecture specifically designed for machine learning.

Data management architecture for ML

Depending on the scale of your ML initiatives, it is important to consider different data management architecture patterns to effectively support them. For small-scale ML projects characterized by limited data scope, a small team size, and minimal cross-functional dependencies, a purpose-built data pipeline tailored to meet the specific project requirements can be a suitable approach. For instance, if your project involves working with structured data sourced from an existing data warehouse and a publicly available dataset, you can consider developing a straightforward data pipeline. This pipeline would extract the necessary data from the data warehouse and public domain and store it in a dedicated storage location owned by the project team. This data extraction process can be scheduled as needed to facilitate further analysis and processing. The diagram below illustrates a simplified data management flow designed to support a small-scale ML project:

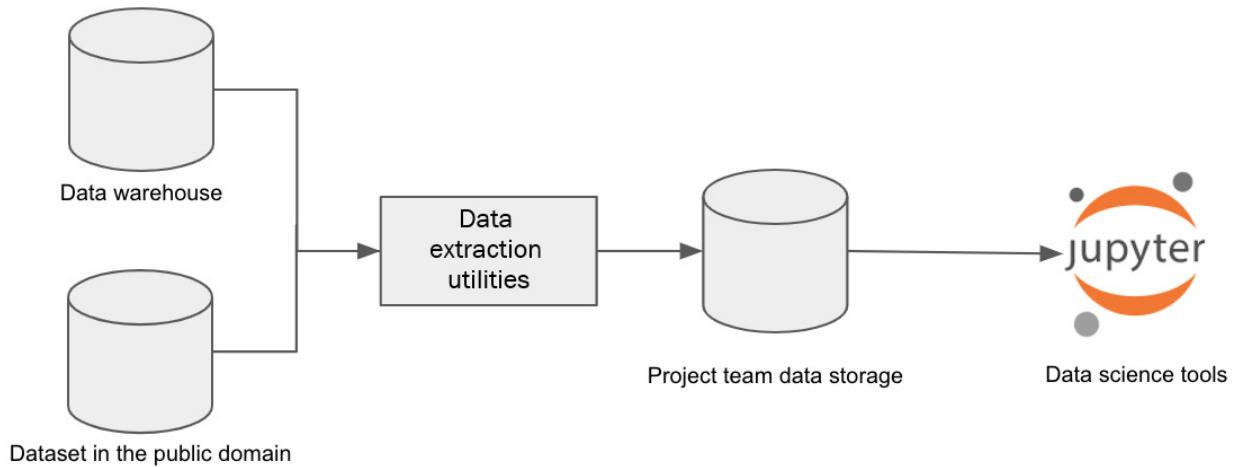


Figure 4.2: Data architecture for an ML project with limited scope

For large-scale ML initiatives at the enterprise level, the data management architecture closely resembles that of enterprise analytics. Both require robust support for data ingestion from diverse sources and centralized management of data for various processing and access requirements. While analytics data management primarily deals with structured data and often relies on an enterprise data warehouse as its core backend, ML data management needs to handle structured, semi-structured, and unstructured data for different ML tasks. Consequently, a data lake architecture is commonly adopted. ML data management is typically an integral part of the broader enterprise data management strategy, encompassing both analytics and ML initiatives. The diagram below illustrates a logical enterprise data management architecture comprising key components such as data ingestion, data storage, data processing, data catalog, data security, and data access:

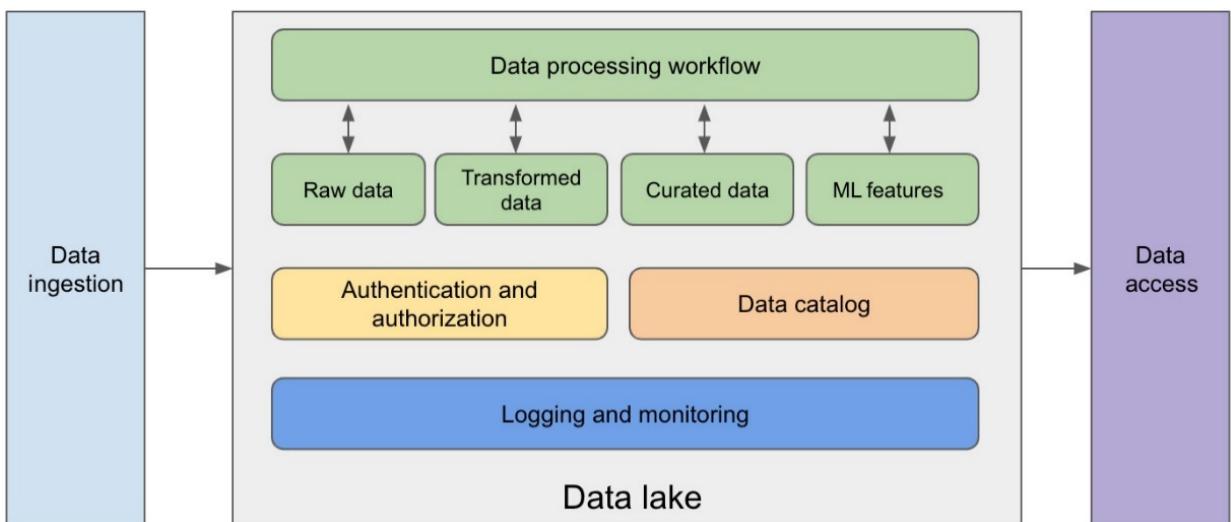


Figure 4.3: Enterprise data management

In the following sections, we will delve into a detailed analysis of each key component, providing an in-depth understanding of their functionalities and implications within a data management architecture built using AWS native services in the cloud. By exploring the specific characteristics and capabilities of these components, we will gain valuable insights into the overall structure and mechanics of an AWS-based data management architecture.

Data storage and management

ML workloads often require data from diverse sources and in various formats, and the sheer volume of data can be substantial, particularly when dealing with unstructured data. To address these requirements, cloud object data storage solutions like Amazon S3 are commonly employed as the underlying storage medium. Conceptually, cloud object storage can be likened to a file storage system that accommodates files of different formats. Moreover, the storage system allows for the organization of files using prefixes, which serve as virtual folders for enhanced object management. It is important to note that these prefixes do not correspond to physical folder structures. The term "object storage" stems from the fact that each file is treated as an independent object, bundled with metadata and assigned a unique identifier. Object storage boasts features such as virtually unlimited storage capacity, robust object analytics based on metadata, API-based access, and cost-effectiveness. To efficiently handle the vast quantities of data stored in cloud object storage, it is advisable to implement a data lake architecture that leverages this storage medium. A data lake, tailored to encompass the entire enterprise or specific line of business, acts as a centralized hub for data management and access. Designed to accommodate limitless data volumes, the data lake facilitates the organization of data across various lifecycle stages, including raw, transformed, curated, and ML feature data. Its primary purpose is to consolidate disparate data silos into a singular repository that enables centralized management and access for both analytics and ML requirements. Notably, a data lake can house diverse data formats, such as structured data from databases, unstructured data like documents, semi-structured data in JSON and XML formats, as well as binary formats encompassing images, videos, and audio files. This capability proves particularly invaluable for ML workloads, as ML often involves working with data in multiple formats. The data lake should be organized by different zones. For example, a *landing zone* should be established as the target for the initial data ingestion from different sources. After data pre-processing and data quality management processing, the data can be moved to the raw data zone. Data in the *raw data zone* can be further transformed and processed to meet different business and downstream consumption needs. To further ensure the reliability of the dataset for usage, the data can be curated and stored in the *curate data zone*. For ML tasks, ML features often need to be pre-computed and stored in an ML feature zone for reuse purposes.

AWS Lake Formation

AWS Lake Formation is a comprehensive data management service offered by AWS, which streamlines the process of building and maintaining a data lake on the AWS platform. The following figure illustrates the core components of AWS Lake Formation:

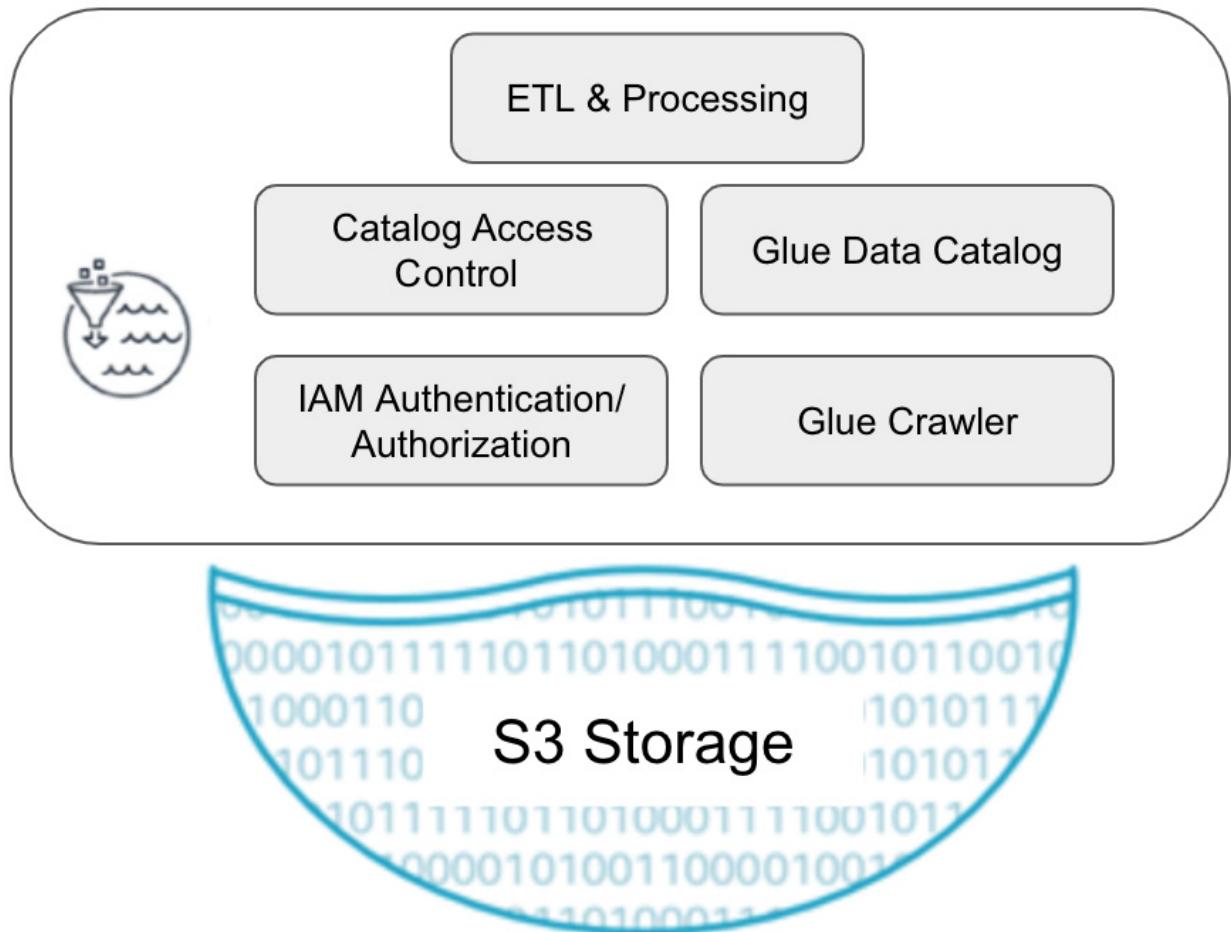


Figure 4.4: AWS Lake Formation

Overall, AWS Lake Formation offers four fundamental capabilities to enhance data lake management:

- Data Source Crawler: This functionality automatically examines data files within the data lake to infer their underlying structure, enabling efficient organization and categorization of the data.
- Data Catalog Creation and Maintenance: AWS Lake Formation facilitates the creation and ongoing management of a data catalog, providing a centralized repository for metadata, enabling easy data discovery and exploration within the data lake.
- Data Transformation Processing: With built-in data transformation capabilities, the service allows for the processing and transformation of data stored in the data lake, enabling data scientists and analysts to work with refined and optimized datasets.
- Data Security and Access Control: AWS Lake Formation ensures robust data security by providing comprehensive access control mechanisms and enabling fine-grained permissions management, ensuring that data is accessed only by authorized individuals and teams.

Lake Formation integrates with AWS Glue, a serverless **Extract, Transform, Load (ETL)** and data catalog service, to provide data catalog management and data ETL processing functionality. We will cover ETL and data catalog components separately in later sections. Lake Formation provides centralized data access management capability for managing data access permissions for the database, tables, or different registered S3 locations. For databases and tables, the permission can be granularly assigned to individual tables and columns and database functions, such as creating tables and inserting records.

Data ingestion

The data ingestion component plays a crucial role in acquiring data from diverse sources, including structured, semi-structured, and unstructured formats, such as databases, knowledge graph, social media, file storage, and IoT devices. Its primary responsibility is to store this data persistently in various storage solutions like object data storage (e.g., Amazon S3), data warehouses, or other data stores. Effective data ingestion patterns should incorporate both real-time streaming and batch ingestion mechanisms to cater to different types of data sources and ensure timely and efficient data acquisition. Various data ingestion technologies and tools cater to different ingestion patterns. For streaming data ingestion, popular choices include Apache Kafka, Apache Spark Streaming, and Amazon Kinesis/Kinesis Firehose. These tools enable real-time data ingestion and processing. On the other hand, for batch-oriented data ingestion, tools like Secure File Transfer Protocol (SFTP) and AWS Glue are commonly used. AWS Glue, in particular, offers support for a wide range of data sources and targets, including Amazon RDS, MongoDB, Kafka, Amazon DocumentDB, S3, and any databases that support JDBC connections. This flexibility allows for seamless ingestion of data from various sources into the desired data storage or processing systems. When making decisions on which tools to use for data ingestion, it is important to assess the tools and technologies based on practical needs. The following are some of the considerations when deciding on data ingestion tools:

- **Data format, size, and scalability:** Take into account the various data formats, data size, and scalability needs. ML projects could be using data from different sources and different formats (e.g., **CSV**, **Parquet**, JSON/XML, documents or image/audio/video files). Determine whether the infrastructure can handle large data volumes efficiently when necessary and scale down to reduce costs during periods of low volume.
- **Ingestion patterns:** Consider the different data ingestion patterns that need to be supported. The tool or combination of several tools should support both batch ingestion patterns (transferring bulk data at specific time intervals) and real-time streaming (processing data such as sensor data or website clickstreams in real time).
- **Data preprocessing capability:** Evaluate whether the ingested data need to be preprocessed before it is stored in the target data repository. Look for tools that offer built-in processing capability or seamless integration with external processing tools.
- **Security:** Ensure that the selected tools provide robust security mechanisms for authentication and authorization to protect sensitive data.
- **Reliability:** Verify that the tools offer failure recovery mechanisms to prevent critical data loss during the ingestion process. If recovery capability is lacking, ensure there is an option to rerun ingestion jobs from the source.
- **Support for different data sources and targets:** The chosen ingestion tools should be compatible with a wide range of data sources, including databases, files, and streaming sources. Additionally, they should provide an API for easy data ingestion.
- **Manageability:** Another important factor to consider is the level of manageability. Does the tool require self-management, or is it a fully managed solution? Take into account the trade-offs between cost and operational complexity before making a decision.

AWS provides several services for data ingestion into a data lake on their platform. These services include Kinesis Data Streams, Kinesis Firehose, AWS Managed Streaming for Kafka, and AWS Glue Streaming, which cater to streaming data requirements. For batch ingestion, options such as AWS Glue, SFTP, and AWS Data Migration Service (DMS) are available. In the upcoming section, we will delve into the usage of Kinesis Firehose and AWS Glue for managing data ingestion processes for data lakes. We will also discuss AWS Lambda, a serverless compute service, for simple and light weight data ingestion alternative.

Kinesis Firehose

Kinesis Firehose is a service that streamlines the process of loading streaming data into a data lake. It is a fully managed solution, meaning you don't have to worry about managing the underlying infrastructure. Instead, you can interact with the service's API to handle the ingestion, processing, and delivery of your data. Kinesis Firehose provides comprehensive support for various scalable data ingestion requirements, including:

- Seamless integration with diverse data sources such as websites, IoT devices, and video cameras. This is achieved through the use of an ingestion agent or ingestion API.
- Versatility in delivering data to multiple destinations, including Amazon S3, Amazon Redshift (an AWS data warehouse service), Amazon ElasticSearch (a managed search engine), and Splunk (a log aggregation and analysis product).
- Seamless integration with AWS Lambda and Kinesis Data Analytics, offering advanced data processing capabilities. With AWS Lambda, you can leverage serverless computing to execute custom functions written in languages like Python, Java, Node.js, Go, C#, and Ruby. For more comprehensive information on the functionality of Lambda, please refer to the official AWS documentation.

The following figure illustrates the data flow with Kinesis Firehose:

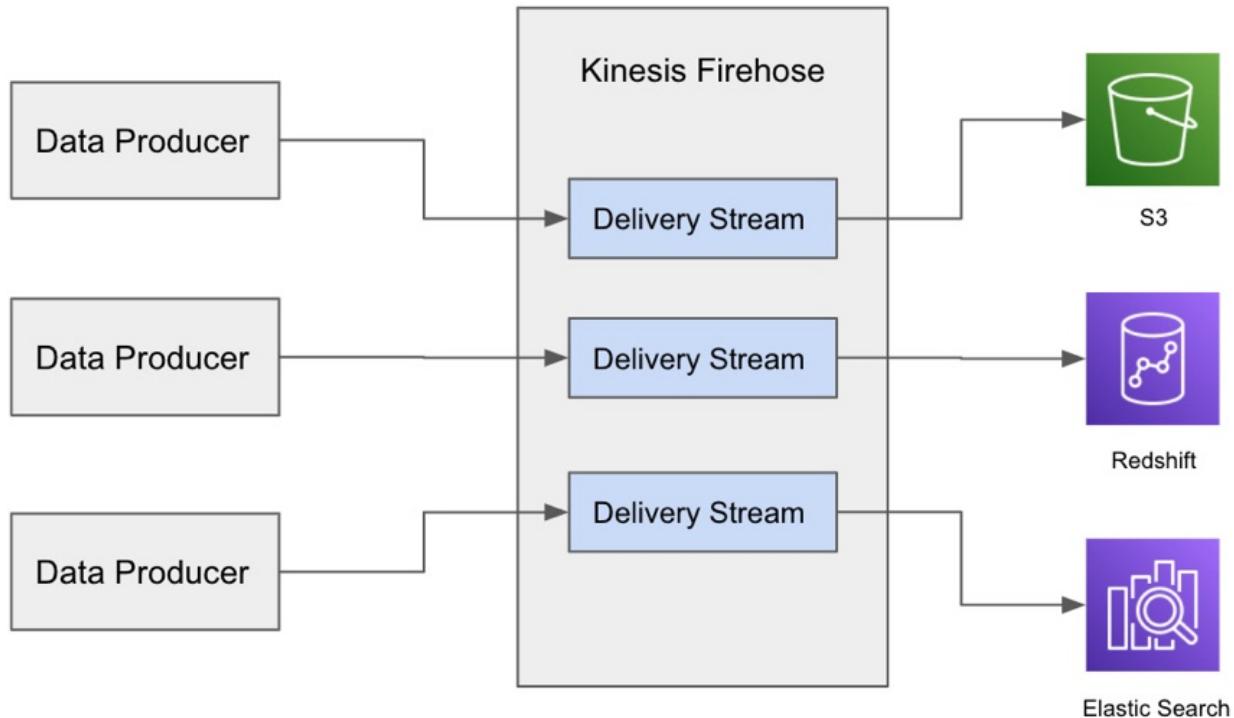


Figure 4.5: Kinesis Firehose data flow

Kinesis operates by establishing delivery streams, which are the foundational components in the Firehose architecture responsible for receiving streaming data from data producers. These delivery streams can be configured with various delivery destinations, such as S3 and Redshift. To accommodate the data volume generated by the producers, you can adjust the throughput of the data stream by specifying the number of shards. Each shard has the capacity to ingest 1 MB/sec of data and can support data read at a rate of 2 MB/sec. Additionally, Kinesis Firehose offers APIs for increasing the number of shards and merging them when needed.

AWS Glue

AWS Glue is a comprehensive serverless ETL service that helps manage data integration and ingestion process for data lakes. It seamlessly connects with various data sources, including transactional databases, data warehouses, and NoSQL databases, facilitating the movement of data to different destinations, such as Amazon S3. This movement can be scheduled or triggered by events. Additionally, AWS Glue offers the capability to process and transform data before delivering it to the target. It provides a range of processing options, such as the Python shell for executing Python scripts and Apache Spark for Spark-based data processing tasks. With AWS Glue, you can efficiently integrate and ingest data into your data lake, benefiting from its fully managed and serverless nature.

AWS Lambda

AWS Lambda is AWS's serverless computing platform. It seamlessly integrates with various AWS services, including Amazon S3. By leveraging Lambda, you can trigger the execution of functions in response to events, such as the creation of a new file in S3. These Lambda functions can be developed to move data from different sources, such as copying data from a source S3 bucket to a target landing bucket in a data lake. It's important to note that AWS Lambda is not specifically designed for large-scale data movement or processing tasks. However, for simpler data ingestion and processing jobs, it proves to be a highly efficient tool.

Data cataloging

Data catalog plays a crucial role in data governance and enables data analysts and scientists to discover and access data stored in a central data storage. It becomes particularly important during the data understanding and exploration phase of the ML life cycle when scientists need to search and comprehend available data for their ML projects. When evaluating a data catalog technology, consider the following key factors:

- **Metadata catalog:** The technology should support a central data catalog for effective management of data lake metadata. This involves handling metadata such as database names, table schemas, and table tags. The Hive metastore catalog is a popular standard for managing metadata catalogs.
- **Automated data cataloging:** The capability to automatically discover and catalog datasets, as well as infer data schemas from various data sources like Amazon S3, relational databases, NoSQL databases, and logs. Typically, this functionality is implemented through a crawler that scans data sources, identifies metadata elements (e.g., column names, data types), and adds them to the catalog.
- **Tagging flexibility:** The ability to assign custom attributes or tags to metadata entities like databases, tables, and fields. This flexibility supports enhanced data search and discovery capabilities within the catalog.
- **Integration with other tools:** Seamless integration of the data catalog with a wide range of data processing tools, enabling easy access to the underlying data. Additionally, native integration with data lake management platforms is advantageous.
- **Search functionality:** A robust search capability across diverse metadata attributes within the catalog. This includes searching by database, table, and field names, custom tags or descriptions, and data types.

When it comes to building data catalogs, there are various technical options available. In this section, we first explore how AWS Glue can be utilized for data cataloging purposes. We will also discuss a Do-It-Yourself (DIY) option for data catalog using standard AWS services such as Lambda and OpenSearch.

AWS Glue Catalog

AWS Glue offers a comprehensive solution for data cataloging, integrating seamlessly with AWS Lake Formation and other AWS services. The Glue catalog can be a drop-in replacement for the Hive metastore catalog, so any Hive metastore-compatible applications can work with the AWS Glue catalog. With AWS Glue, you can automatically discover, catalog, and organize your data assets, making them easily searchable and accessible to data analysts and scientists. Here are some key features and benefits of using AWS Glue for data cataloging:

- **Automated Data Discovery:** AWS Glue provides automated data discovery capabilities. By using data crawlers, Glue can scan and analyze data from diverse structured and semi-structured sources such as Amazon S3, relational databases, NoSQL databases, and more. It identifies metadata information, including table schemas, column names, and data types, which are stored in the AWS Glue Data Catalog.
- **Centralized Metadata Repository:** The AWS Glue Data Catalog serves as a centralized metadata repository for your data assets. It provides a unified view of your data, making it easier to search, query, and understand the available datasets.
- **Metadata Management:** AWS Glue allows you to manage and maintain metadata associated with your data assets. You can define custom tags, add descriptions, and organize your data using databases, tables, and partitions within the Data Catalog.

The metadata hierarchy of the AWS Glue catalog is organized using databases and tables. Databases serve as containers for tables, which hold the actual data. Similar to traditional databases, a single database can house multiple tables, which can be sourced from various data stores. However, each table is exclusively associated with a single database. To query these databases and tables, one can utilize Hive metastore-compatible tools and execute SQL queries. When collaborating with AWS Lake Formation, access permissions to the catalog's databases and tables can be controlled through the Lake Formation entitlement layer.

Custom data catalog solution

Another option for building a data catalog is to create your own with a set of AWS services. Consider this option when you have specific requirements that not met by the purpose-built products. The architecture for this do-it-yourself (DIY) approach involves leveraging services like DynamoDB and Lambda, as depicted in the accompanying diagram:

Figure 4.6: Custom data catalog solution

At a high level, AWS Lambda triggers are used to populate DynamoDB tables with object names and metadata when those objects are put into S3; Amazon OpenSearch Service is used to search for specific assets, related metadata, and data classifications.

Data processing

The data processing functionality of a data lake encompasses the frameworks and compute resources necessary for various data processing tasks, such as data correction, transformation, merging, splitting, and ML feature engineering. Common data processing frameworks include Python shell scripts and Apache Spark. The essential requirements for data processing technology are as follows:

- Integration and compatibility with the underlying storage technology: The ability to seamlessly work with the native storage system simplifies data access and movement between the storage and processing layers.
- Integration with the data catalog: The capability to interact with the data catalog's metastore for querying databases and tables within the catalog.
- Scalability: The capacity to scale compute resources up or down to accommodate changing data volumes and processing velocity requirements.
- Language and framework support: Support for popular data processing libraries and frameworks, such as Python and Spark.
- Batch and real-time processing capabilities: The capability to handle both real-time data streams and bulk data processing in batch mode.

Now, let's examine a selection of AWS services that offer data processing capabilities within a data lake architecture.

AWS Glue ETL

In addition to supporting data movement and data catalogs, the ETL features of AWS Glue can be used for ETL and general-purpose data processing. AWS Glue ETL provides a number of built-in functions for data transformation, such as the drop of the **NULL** field and data filtering. It also provides general processing frameworks for Python and Spark to run Python scripts and Spark jobs. Glue ETL works natively with the Glue catalog for accessing the databases and tables in the catalog. Glue ETL can also access the Amazon S3 storage directly.

Amazon Elastic Map Reduce (EMR)

Amazon EMR is a fully managed big data processing platform on AWS. It is designed for large-scale data processing using the Spark framework and other Apache tools, such as **Apache Hive**, **Apache Hudi**, and **Presto**.

It integrates with the Glue Data Catalog and Lake Formation natively for accessing databases and tables in Lake Formation.

AWS Lambda

AWS Lambda can be used for lightweight data processing tasks or as part of a larger data processing pipeline within the data lake architecture. Lambda can be triggered by real-time events, so it is a good option for real-time data processing.

ML data versioning

In order to establish lineage for model training across training data and ML models, it is crucial to implement version control for the training, validation, and testing datasets. Data versioning control presents challenges as it necessitates the use of appropriate tools and adherence to best practices by individuals. During the model building process, it is common for data scientists to obtain a copy of a dataset, perform cleansing and transformations specific to their needs, and save the modified data as a new version. This poses significant challenges in terms of data management, including duplication and establishing links between the data and its various upstream and downstream tasks. Data versioning for the entire data lake is out of scope for this book. Instead, we will focus on discussing a few architectural options specifically related to versioning control for training datasets.

S3 partitions

In this approach, each newly created or updated dataset is stored in a separate S3 partition with a unique prefix, typically derived from the name of the S3 folder. While this method can lead to data duplication, it offers a clear and simple approach to differentiate between different datasets intended for model training. To maintain data integrity, it is advisable to generate datasets through a controlled processing pipeline that enforces naming standards. The processing pipeline should also track data provenance and record the processing scripts used for data manipulation and feature engineering. Furthermore, the datasets should be configured as read-only for downstream applications, ensuring their immutability. The following example showcases an S3 partition structure, illustrating multiple versions of a training dataset:

```
s3://project1/<date>/<unique version id 1>/train_1.txt
s3://project1/<date>/<unique version id 1>/train_2.txt
s3://project1/<date>/<unique version id 2>/train_1.txt
s3://project1/<date>/<unique version id 2>/train_2.txt
```

In this instance, the two versions of the dataset are segregated using distinct S3 prefixes. To effectively track these training files, it is recommended to employ a database for storing metadata pertaining to these training files. When utilizing these files, it is crucial to establish links between the training datasets, ML training jobs, ML training scripts, and the resulting ML models to establish comprehensive lineage.

Versioned S3 buckets

Amazon S3 offers versioning support for S3 buckets, which can be leveraged to manage different versions of training datasets when enabled. With this approach, each newly created or updated dataset is assigned a unique version ID at the S3 object level. Additionally, it is recommended to utilize a database to store all relevant metadata associated with each version of the training dataset. This enables the establishment of lineage, tracking the journey from data processing to ML model training. The metadata should capture essential information to facilitate comprehensive tracking and analysis.

Purpose-built data version tools

Instead of developing custom solutions for data version control, there are purpose-built tools available for efficient data version management. Here are a few notable options:

- **Git LFS (Large File Storage):** Git LFS extends Git's capabilities to handle large files, including datasets. It stores these files outside the Git repository while retaining versioning information. Git LFS seamlessly integrates with Git and is commonly used for versioning large files in data-centric projects.
- **DataVersionControl (DVC):** DVC is an open-source tool designed specifically for data versioning and management. It integrates with Git and provides features for tracking and managing large datasets. DVC enables lightweight links to actual data files stored in remote storage, such as Amazon S3 or a shared file system. This approach maintains a history of changes and allows easy switching between different dataset versions, eliminating the need for data duplication.
- **Pachyderm:** Pachyderm is an open-source data versioning and data lineage tool. It offers version control for data pipelines, enabling tracking of changes to data, code, and configuration files. Pachyderm supports distributed data processing frameworks like Apache Spark and provides features like reproducibility, data lineage, and data lineage-based branching.

These purpose-built tools streamline the process of data versioning, ensuring efficient tracking and management of datasets.

ML feature store

In large enterprises, it is beneficial to centrally manage common reusable ML features like curated customer profile data and standardized product sales data. This practice helps reduce the ML project life cycle, particularly during the data understanding and data preparation stages. Depending on the specific requirements, there are two main options for managing these reusable ML features. Firstly, you can build custom feature stores that fulfill the fundamental requirements of inserting and looking up organized features for ML model training. These custom feature stores can be tailored to meet the specific needs of the organization. Alternatively, you can opt for commercial-grade feature store products, such as Amazon SageMaker Feature Store. SageMaker Feature Store is a machine learning service offered by AWS, which we will delve into in later chapters. It provides advanced capabilities such as online and offline functionality for training and inference, metadata tagging, feature versioning, and advanced search. These features enable efficient management and utilization of ML features in production-grade scenarios.

Data serving for client consumption

The central data management platform should offer various methods, such as APIs or Hive metastore-based approaches, to facilitate online access to the data. Additionally, it is important to consider data transfer tools that support the movement of data from the central data management platform to other data-consuming environments, catering to different data consumption patterns. It is advantageous to explore tools that either have built-in data serving capabilities or can be seamlessly integrated with external data serving tools. When supplying data to data science environments, there are multiple data serving patterns to consider. In the following discussion, we will explore two prominent data access patterns and their characteristics.

Consumption via API

In this data serving pattern, consumption environments and applications have the capability to directly access data from the data lake. This can be achieved using Hive metastore-compliant tools or through direct access to S3, the underlying storage of the data lake. Amazon provides various services that facilitate this pattern, such as Amazon Athena, a powerful big data query tool, Amazon EMR, a robust big data processing tool, and Amazon Redshift Spectrum, a feature of Amazon Redshift. By leveraging these services, data lake data indexed in Glue catalogs can be queried without the need to make a separate copy of the data. This pattern is particularly suitable when only a subset of the data is required for downstream data processing tasks. It offers the advantage of avoiding data duplication while enabling efficient selection and processing of specific data subsets as part of the overall data workflow.

Consumption via data copy

In this data serving pattern, a specific portion of the data stored in the data lake is replicated or copied to the storage of the consumption environment. This replication allows for tailored processing and consumption based on specific needs. For instance, the latest or most relevant data can be loaded into a data analytics environment such as Amazon Redshift. Similarly, it can be delivered to S3 buckets owned by a data science environment, enabling efficient access and utilization for data science tasks. By replicating the required data subsets, this pattern provides flexibility and optimized performance for different processing and consumption requirements in various environments.

Special databases for ML

In light of emerging machine learning (ML) paradigms like graph neural networks and generative AI, specialized databases have been developed to cater to ML-specific tasks such as link prediction, cluster classification, and retrieval-augmented generation. In the following section, we will delve into two types of databases—vector databases and graph databases—and examine how they are utilized in ML tasks. We will explore their unique characteristics and applications in the context of machine learning.

Vector Database

Vector databases, also known as vector similarity search engines or vector stores, are specialized databases designed to efficiently store, index, and query high-dimensional vectors. These databases are particularly well-suited for ML applications that rely on vector-based computations. In ML, vectors are commonly used to represent data points, embeddings, or feature representations. These vectors capture essential information about the underlying data, enabling similarity search, clustering, classification, and other ML tasks. Vector databases provide powerful tools for handling these vector-based operations at scale. One of the key features of vector databases is their ability to perform fast similarity searches, allowing efficient retrieval of vectors that are most similar to a given query vector. This capability is essential in various ML use cases, such as recommender systems, content-based search, and anomaly detection. There are several vector database providers in the market, each offering their own unique features and capabilities. Some of the prominent ones include:

- **FAISS:** Developed by Facebook AI Research (FAIR), FAISS is an open-source library for efficient similarity search and clustering of dense vectors. It provides highly optimized algorithms and data structures for fast and scalable vector search.
- **Milvus:** Milvus is an open-source vector database designed for managing and serving large-scale vector datasets. It offers efficient similarity search, supports multiple similarity metrics, and provides scalability through distributed computing.
- **Pinecone:** Pinecone is a cloud-native vector database service that specializes in high-performance similarity search and recommendation systems. It offers real-time indexing and retrieval of vectors with low latency and high throughput.
- **Elasticsearch:** Although primarily known as a full-text search and analytics engine, Elasticsearch also provides vector similarity search capabilities through the use of plugins for efficient vector indexing and querying.
- **Weaviate:** Weaviate is an open-source vector database. It allows you to store data objects and vector embeddings from your favorite ML-models, and scale seamlessly into billions of data objects.

These are just a few examples of vector database providers, and the landscape is continuously evolving with new solutions and advancements in the field. When choosing a vector database provider, it's important to consider factors such as performance, scalability, ease of integration, and the specific requirements of your ML use case.

Graph Databases

Graph databases are specialized databases designed to store, manage, and query graph-structured data. In a graph database, data is represented as nodes (entities) and edges (relationships) connecting these nodes, forming a graph-like structure. Graph databases excel at capturing and processing complex relationships and dependencies between entities, making them highly relevant for ML tasks. Graph databases offer a powerful way to model and analyze

data in domains where relationships play a crucial role, such as social networks, recommendation systems, fraud detection, knowledge graphs, and network analysis. They enable efficient traversal of the graph, allowing for queries that explore connections and patterns within the data. In the context of ML, graph databases have multiple applications. One key use case is graph-based feature engineering, where graphs are used to represent relationships between entities, and the graph structure is leveraged to derive features that can enhance the performance of ML models. For example, in a recommendation system, a graph database can represent user-item interactions, and graph-based features can be derived to capture user similarities, item similarities, or collaborative filtering patterns. Graph databases also enable graph-based algorithms, such as graph convolutional networks (GCNs), for tasks like node classification, link prediction, and graph clustering. These algorithms leverage the graph structure to propagate information across nodes and capture complex patterns in the data. Furthermore, graph databases can be used to store and query graph embeddings, which are low-dimensional vector representations of nodes or edges. These embeddings capture the structural and semantic information of the graph and can be input to ML models for downstream tasks, such as node classification or recommendation. Some of the notable graph databases include Neo4J, a popular and widely-used graph database that allows for efficient storage, retrieval, and querying of graph-structured data, and Amazon Neptune, a fully managed graph database service provided by AWS.

Data pipeline

Data pipelines streamline the flow of data by automating tasks such as data ingestion, validation, transformation, and feature engineering. These pipelines ensure data quality and facilitate the creation of training and validation datasets for machine learning models. Numerous workflow tools are available for constructing data pipelines, and many data management tools offer built-in capabilities for building and managing these pipelines.

AWS Glue workflows

AWS Glue workflows provide a native workflow management feature within AWS Glue, enabling the orchestration of various Glue jobs like data ingestion, processing, and feature engineering. Comprised of trigger and node components, a Glue workflow incorporates schedule triggers, event triggers, and on-demand triggers. Nodes within the workflow can be either crawler jobs or ETL jobs. Triggers initiate workflow runs, while event triggers are emitted after the completion of crawler or ETL jobs. By structuring a series of triggers and jobs, workflows facilitate the seamless execution of data pipelines within AWS Glue.

AWS Step Functions

AWS Step Functions is a powerful workflow orchestration tool that seamlessly integrates with various AWS data processing services like AWS Glue and Amazon EMR. It enables the creation of robust workflows to execute diverse steps within a data pipeline, such as data ingestion, data processing, and feature engineering, ensuring smooth coordination and execution of these tasks.

AWS Managed Airflow

AWS Managed Apache Airflow is a fully managed service that simplifies the deployment, configuration, and management of Apache Airflow, an open-source platform for orchestrating and scheduling data workflows. This service offers scalability, reliability, and easy integration with other AWS services, making it an efficient solution for managing complex data workflows in the cloud.

Authentication and authorization

Authentication and authorization are crucial for ensuring secure access to a data lake. Federated authentication, such as AWS Identity and Access Management (IAM), verifies user identities for administration and data consumption purposes. AWS Lake Formation combines the built-in Lake Formation access control with AWS IAM to govern access to data catalog resources and underlying data storage. The built-in Lake Formation permission model utilizes commands like grant and revoke to control access to resources such as databases and

tables, as well as actions like table creation. When a user requests access to a resource, both IAM policies and Lake Formation permissions are evaluated to verify and enforce access before granting it. This multi-layered approach enhances data lake security and governance. There are several personas involved in the administration of the data lake and consumption of the data lake resources, including:

- **Lake Formation administrator:** A Lake Formation administrator has the permission to manage all aspects of a Lake Formation data lake in an AWS account. Examples include granting/revoking permissions to access data lake resources by other users, register data stores in S3, and creating/deleting databases. When you create Lake Formation, you will need to register an administrator. An administrator can be an AWS IAM user or IAM role. You can add more than one administrator to a Lake Formation data lake.
- **Lake Formation database creator:** A Lake Formation database creator has been granted permission to create databases in Lake Formation. A database creator can be an IAM user or IAM role.
- **Lake Formation database user:** A Lake Formation database user can be granted permission to perform different actions against a database. Example permissions include create table, drop table, describe table, and alter table. A database user can be an IAM user or IAM role.
- **Lake Formation data user:** A Lake Formation data user can be granted permission to perform different actions against database tables and columns. Example permissions include insert, select, describe, delete, alter, and drop. A data user can be an IAM user or an IAM role.

Accessing and querying the database and tables in Lake Formation is facilitated through compatible AWS services like Amazon Athena and Amazon EMR. When performing queries using these services, Lake Formation verifies the principals (IAM users, groups, and roles) associated with them to ensure they have the necessary access permissions for the database, tables, and corresponding S3 data location. If the access is granted, Lake Formation issues a temporary credential to the service, enabling it to execute the query securely and efficiently. This process ensures that only authorized services can interact with Lake Formation and perform queries on the data.

Data governance

Data governance encompasses essential practices to ensure the reliability, security, and accountability of data assets. Trustworthy data is achieved through the identification and documentation of data flows, as well as the measurement and reporting of data quality. Data protection and security involve classifying data and applying appropriate access permissions to safeguard its confidentiality and integrity. To maintain visibility into data activities, monitoring and auditing mechanisms should be implemented, allowing organizations to track and analyze actions performed on data, ensuring transparency and accountability in data management.

Data Lineage

To establish and document data lineage during the ingestion and processing of data across different zones, it is important to capture specific data points. When utilizing data ingestion and processing tools like AWS Glue, AWS EMR, or AWS Lambda in a data pipeline, the following information can be captured to establish comprehensive data lineage:

- **Data Source Details:** Include the name of the data source, its location, and ownership information to identify the origin of the data.
- **Data Processing Job History:** Capture the history and details of the data processing jobs involved in the pipeline. This includes information such as the job name, unique identifier (ID), associated processing script, and owner of the job.
- **Generated Artifacts:** Document the artifacts generated as a result of the data processing jobs. For example, record the S3 URI or other storage location for the target data produced by the pipeline.
- **Data Metrics:** Track relevant metrics at different stages of the data processing. This can include the number of records, data size, data schema, and feature statistics to provide insights into the processed data.

To store and manage data lineage information and processing metrics, it is recommended to establish a central data operational data store. AWS DynamoDB, a fully managed NoSQL database, is an excellent technology choice for

this purpose. With its capabilities optimized for low latency and high transaction access, DynamoDB provides efficient storage and retrieval of data lineage records and processing metrics. By capturing and documenting these data points, organizations can establish a comprehensive data lineage that provides a clear understanding of the data's journey from its source through various processing stages. This documentation enables traceability, auditability, and better management of the data as it moves through the pipeline.

Other data governance measures

In addition to managing data lineage, there are several other important measures for effective data governance, including:

- **Data quality:** Automated data quality checks should be implemented at different stages, and quality metrics should be reported. For example, after the source data is ingested into the landing zone, an AWS Glue quality check job can run to check the data quality using tools such as the open source **Deequ** library. Data quality metrics (such as counts, schema validation, missing data, wrong data type, or statistical deviations from the baseline) and reports can be generated for reviews. Optionally, manual or automated operational data cleansing processes should be established to correct data quality issues.
- **Data cataloging:** Create a central data catalog and run Glue crawlers on datasets in the data lake to automatically create an inventory of data and populate the central data catalog. Enrich the catalogs with additional metadata to track other information to support discovery and data audits, such as the business owner, data classification, and data refresh date. For ML workloads, data science teams also generate new datasets (for example, new ML features) from the existing datasets in the data lake for model training purposes. These datasets should also be registered and tracked in a data catalog, and different versions of the data should be retained and archived for audit purposes.
- **Data access provisioning:** A formal process should be established for requesting and granting access to datasets and Lake Formation databases and tables. An external ticketing system can be used to manage the workflow for requesting access and granting access.
- **Monitoring and auditing:** Data access should be monitored, and access history should be maintained. Amazon S3 server access logging can be enabled to track access to all S3 objects directly. AWS Lake Formation also records all accesses to Lake Formation datasets in **AWS CloudTrail** (AWS CloudTrail provides event history in an AWS account to enable governance, compliance, and operational auditing). With Lake Formation auditing, you can get details such as event source, event name, SQL queries, and data output location.

By implementing these key data governance measures, organizations can establish a strong foundation for data management, security, and compliance, enabling them to maximize the value of their data assets while mitigating risks.

Hands-on exercise – data management for ML

In this hands-on exercise, you will go through the process of constructing a simple data management platform for a fictional retail bank. This platform will serve as the foundation for an ML workflow, and we will leverage different AWS technologies to build it. If you don't have an AWS account, you can easily create one by following the instructions at <https://aws.amazon.com/console/>. The data management platform we create will have the following key components:

- A data lake environment for data management
- A data ingestion component for ingesting files to the data lake
- A data discovery and query component
- A data processing component

The following diagram shows the data management architecture we will build in this exercise:

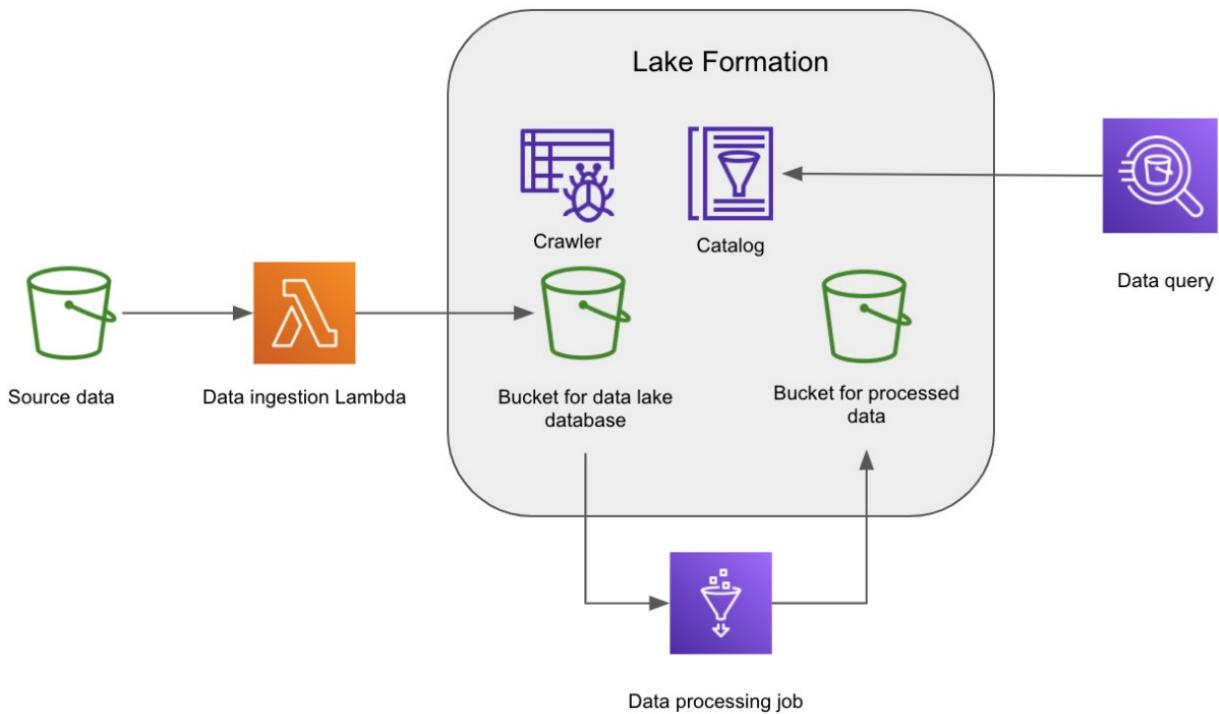


Figure 4.7: Data management architecture for the hands-on exercise

Let's get started with building out this architecture on AWS.

Creating a data lake using Lake Formation

We will build the data lake architecture using AWS Lake Formation. After you log on to the **AWS Management Console**, create an S3 bucket called `MLSA-DataLake-<your initials>`. We will use this bucket as the storage for the data lake. If you get a message that the bucket name is already in use, try adding some random characters to the name to make it unique. If you are not familiar with how to create S3 buckets, follow the instructions at the following link: <https://docs.aws.amazon.com/AmazonS3/latest/user-guide/create-bucket.xhtml> After the bucket is created, follow these steps to get started with creating a data lake:

- 1. Register Lake Formation administrators:** We need to add Lake Formation administrators to the data lake. The administrators will have full permission to manage all aspects of the data lake. To do this, navigate to the Lake Formation management console, click on the **Administrative roles and tasks** link, and you should be prompted to add an administrator. Select "Add myself" and click on **Get started** button.
- 2. Register S3 storage:** Next, we need to register the S3 bucket (`MLSA-DataLake-<your initials>`) you created earlier in Lake Formation so it will be managed and accessible through Lake Formation. To do this, you click on the **Dashboard** link, expand **Data lake setup** and then click on **Register Location**. Browse and select the bucket you created and click on **Register Location**. This S3 bucket will be used by Lake Formation to store data for the databases and manage its access permission.
- 3. Create database:** Now, we are ready to set up a database called `bank_customer_db` for managing retail customers. Before we register the database, let's first create a folder called the `bank_customer_db` folder under the `MLSA-DataLake-<your initials>` bucket. This folder will be used to store data files associated with the database. To do this, you click on the **Create database** button on the Lake Formation dashboard and follow the instructions on the screen to create the database.

You have now successfully created a data lake powered by Lake Formation and created a database for data management. Next, we will create a data ingestion pipeline to move files into the data lake.

Creating a data ingestion pipeline

Now that the database is prepared, we can proceed to ingest data into the newly created database. As mentioned earlier, there are various data sources available, including databases like Amazon RDS, streaming platforms like social media feeds, and logs such as CloudTrail. Additionally, AWS offers a range of services for building data ingestion pipelines, such as AWS Glue, Amazon Kinesis, and AWS Lambda. In this phase of the exercise, we will focus on creating an AWS Lambda function job that will facilitate the ingestion of data from other S3 buckets into our target database:

1. Create a source S3 bucket and download data files:

Let's create another S3 bucket, called `customer-data-source`, to represent the data source where we will ingest the data from. Now, download the sample data files from the following

link: <https://github.com/PacktPublishing/The-Machine-Learning-Solutions-Architect-Handbook/tree/main/Chapter04/Archive.zip>

Then, save it to your local machine. Extract the archived files and upload them to the `customer-data-source` bucket. There should be two files (`customer_data.csv` and `churn_list.csv`).

1. Create Lambda function:

Now, we will create the Lambda function that ingests data from the `customer-data-source` bucket to the `MLSA-DataLake-<your initials>` bucket:

1. To get started, navigate to the AWS Lambda management console, click on the **Functions** link in the left pane, and click on the **Create Function** button in the right pane. Choose **Author from scratch**, then enter **datalake-s3-ingest** for the function name, and select latest **Python version** (e.g., 3.10) as the runtime.
2. On the next screen, click on **Add trigger**, select **S3** as the trigger, and select the **customer-data-source** bucket as the source. For **Event Type**, choose the **Put** event and click on the **Add** button to complete the step. This trigger will allow the Lambda function to be invoked when there is an S3 bucket event, such as saving a file into the bucket.
3. Next, let's create the function by replacing the default function template with the following code block. Replace the **desBucket** variable with the name of the actual bucket:

```
import json
import boto3
def lambda_handler(event, context):
    s3 = boto3.resource('s3')
    for record in event['Records']:
        srcBucket = record['s3']['bucket']['name']
        srcKey = record['s3']['object']['key']
        desBucket = "MLSA-DataLake-<your initials>"
        desFolder = srcKey[0:srcKey.find('.')]
        desKey = "bank_customer_db/" + desFolder + "/" + srcKey
        source= { 'Bucket' : srcBucket,'Key':srcKey}
        dest ={ 'Bucket' : desBucket,'Key':desKey}
        s3.meta.client.copy(source, desBucket, desKey)
    return {
        'statusCode': 200,
        'body': json.dumps('files ingested')
    }
```

1. The new function will also need S3 permission to copy files (*objects*) from one bucket to another. For simplicity, just add the **AmazonS3FullAccess** policy to the **execution IAM role** associated with the function. You can find the IAM role by clicking on the **Permission** tab for the Lambda function.

1. Trigger data ingestion:

You can now trigger the data ingestion process by uploading the `customer_detail.csv` and `churn_list.csv` files to the `customer-data-source` bucket and verify the process completion by checking the `MLSA-DataLake-<your initials>/bank_customer_db` folder for the two files. You have now successfully created an AWS Lambda-based data ingestion pipeline to automatically move data from a source S3 bucket to a target S3 bucket. Next, let's create an AWS Glue catalog using the Glue crawler.

Creating a Glue catalog

To allow discovery and querying of the data in the `bank_customer_db` database, we need to create a data catalog. Here, we will use an AWS Glue crawler to crawl the files in the `bank_customer_db` S3 folder and generate the catalog:

1. Grant permission for Glue:

- A. First, let's grant permission for AWS Glue to access the `bank_customer_db` database. We will create a new IAM role for the Glue service to assume on your behalf. To do this, create a new IAM service role called `AWSGlueServiceRole_data_lake`, and attach the `AWSGlueServiceRole` and `AmazonS3FullAccess` IAM managed policies to it. Make sure you select `Glue` as the service when you create the role. If you are not familiar with how to create a role and attach a policy, follow the instructions at the following link: <https://docs.aws.amazon.com/IAM/latest/UserGuide>.
- B. After the role is created, click on **Data lake permission** in the left pane of the Lake Formation management console and then click the **Grant** button in the right pane.
- C. On the next screen, select **AWSGlueServiceRole_data_lake for IAM users and role**, and **bank_customer_db** under **Named data catalog resources**, choose Super for both **Database permissions** and **Grantable permissions**, and finally click on **Grant**. `AWSGlueServiceRole_data_lake` will be used later for configuring the Glue crawler job.

2. Configure Glue crawler job:

- A. Launch the Glue crawler by clicking on the **Crawler** link in the Lake Formation management console. A new browser tab for Glue will open up. Click on the **Create Crawler** button to get started. Enter **bank_customer_db_crawler** as the name of the crawler. Click on the **Add a data source** button, select **S3** and enter `s3://MLSA-DataLake-<your initials>/bank_customer_db/churn_list/` for the **include path** field.
- B. Click on **Add another data source button** again, this time enter `s3://MLSA-DataLake-<your initials>/bank_customer_db/customer_data/`.
- C. On the next **Configure security settings** screen, select **AWSGlueServiceRole_data_lake for the existing IAM role**, which you used earlier.
- D. On the next **Set output and scheduling** screen, select **bank_customer_db** as the target database, and choose **Run on demand** as the frequency for the crawler schedule.
- E. On the next **Review and create screen**, select **Finish** on the final screen to complete the setup.
- F. On the **Crawler** screen, select the **bank_customer_db_crawler** job you just created, click on **Run crawler**, and wait for the status to say **Ready**.
- G. Navigate back to the Lake Formation management console and click on the **Tables** link. You will now see two new tables created (**churn_list** and **customer_data**).
- H. You have now successfully configured an AWS Glue crawler that automatically discovers table schemas from data files and creates data catalogs for the new data.

You have created the Glue Data Catalog for the newly ingested data. We are now ready to discover and query the data in the data lake.

Discovering and querying data in the data lake

To facilitate the data discovery and data understanding phase of the ML workflow, it is essential to incorporate data discovery and data query capabilities within the data lake. By default, Lake Formation already provides a list

of tags, such as data type classification (for example, CSV), for tables in the database to search. Let's add a few more tags for each table to make it more discoverable:

1. Grant permission to edit the database tables by grant your current user id with `Super` permission for both the `customer_data` and `churn_list` tables.
2. Let's add some metadata to the table fields. Select the `customer_data` table, click on **Edit Schema**, select the `creditscore` field, click on **Edit and Add** to add a column property, and enter the following:
 - 3. **description:** credit score is the FICO score for each customer
4. Follow the same previous steps and add the following column property for the `exited` field in the `churn_list` table:
 - 5. **description:** churn flag
6. We are now ready to do some searches using metadata inside the Lake Formation management console. Try typing the following words separately in the text box for **Find table by properties** to search for tables and see what's returned:
 - **FICO**
 - **csv**
 - **churn flag**
 - **creditscore**
 - **customerid**

Now you have found the table you are looking for, let's query the table and see that actual data. Select the table you want to query and click on the **View data** button in the **Actions** drop-down menu. This should bring you to the **Amazon Athena** screen. You should see a query tab already created, and the query is already executed. The results are displayed at the bottom of the screen. You can run any other SQL query to explore the data further, such as joining the `customer_data` and `churn_list` tables by the `customerid` field:

```
SELECT * FROM "bank_customer_db"."customer_data", "bank_customer_db"."churn_list" where "bank_ci
```

You have now learned how to discover the data in Lake Formation and run queries against the data in a Lake Formation database and tables. Next, let's run a data processing job using the Amazon Glue ETL service to make the data ready for ML tasks.

Creating an Amazon Glue ETL job to process data for ML

The `customer_data` and `churn_list` tables contain features that are useful for ML. However, they need to be joined and processed so they can be used for training ML models. One option is for the data scientists to download these datasets and process them in a Jupyter notebook for model training. Another option is to process the data using a separate processing engine so that the data scientists can work with the processed data directly. Here, we will set up an AWS Glue job to process the data in the `customer_data` and `churn_list` tables and transform them into new ML features that are ready for model training directly:

1. First, create a new S3 bucket called `MLSA-DataLake-Serving-<your initials>`. We will use this bucket to store the output training datasets from the Glue job.
2. Using Lake Formation console, grant **AWSGlueService_Role** with **Super access to tables** `customer_data` and `churn_list`. We will use this role to run the Glue job.
3. To start creating the Glue job, click on the **Jobs** link on the Lake Formation console. This will bring you to the AWS Glue Studio screen. Select **Spark script editor** and click on the **Create** button.
4. On the script editor screen, change the name from `Untitled job` to `customer_churn_process` as the job name.
5. On the **Job details** tab, select **AWSGlueService_Role** as the IAM role. Add a new parameter called `"target_bucket"` and enter the value of the your target bucket for the output files.
6. On the **Script tab** screen, copy the following code blocks to the code section. Make sure to replace `default_bucket` with your own bucket in the code. The following code block first joins the `churn_list` and `customer_data` tables using the `customerid` column as the key, then transforms the gender and geo

columns with an index, creates a new DataFrame with only the relevant columns, and finally saves the output file to an S3 location using the date and generated version ID as partitions. The code uses default values for the target bucket, prefix variables, and generates a date partition and version partition for the S3 location. The job can also accept input arguments for these parameters.

The following code block sets up default configurations, such as `SparkContext` and a default bucket:

```
import sys
from awsglue.utils import getResolvedOptions
from awsglue.transforms import Join
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
import pandas as pd
from datetime import datetime
import uuid
from pyspark.ml.feature import StringIndexer
glueContext = GlueContext(SparkContext.getOrCreate())
logger = glueContext.get_logger()
current_date = datetime.now()
default_date_partition = f"{current_date.year}-{current_date.month}-{current_date.day}"
default_version_id = str(uuid.uuid4())
default_bucket = "<your default bucket name>"
default_prefix = "ml-customer-churn"
target_bucket = ""
prefix = ""
day_partition = ""
version_id = ""
try:
    args = getResolvedOptions(sys.argv, ['JOB_NAME', 'target_bucket', 'prefix', 'day_partition', 'version_id'])
    target_bucket = args['target_bucket']
    prefix = args['prefix']
    day_partition = args['day_partition']
    version_id = args['version_id']
except:
    logger.error("error occurred with getting arguments")
if target_bucket == "":
    target_bucket = default_bucket
if prefix == "":
    prefix = default_prefix
if day_partition == "":
    day_partition = default_date_partition
if version_id == "":
    version_id = default_version_id
```

The following code joins the `customer_data` and `churn_list` tables into a single table using the `customerid` column as the key:

```
# catalog: database and table names
db_name = "bank_customer_db"
tbl_customer = "customer_data"
tbl_churn_list = "churn_list"
# Create dynamic frames from the source tables
customer = glueContext.create_dynamic_frame.from_catalog(database=db_name, table_name=tbl_customer)
churn = glueContext.create_dynamic_frame.from_catalog(database=db_name, table_name=tbl_churn_list)
# Join the frames to create customer churn data frame
customer_churn = Join.apply(customer, churn, 'customerid', 'customerid')
customer_churn.printSchema()
```

The following code block transforms several data columns from string labels to label indices and writes the final file to an output location in S3:

```
# ---- Write out the combined file ----
current_date = datetime.now()
str_current_date = f"{current_date.year}-{current_date.month}-{current_date.day}"
```

```

random_version_id = str(uuid.uuid4())
output_dir = f"s3://{target_bucket}/{prefix}/{day_partition}/{version_id}"
s_customer_churn = customer_churn.toDF()
gender_indexer = StringIndexer(inputCol="gender", outputCol="genderindex")
s_customer_churn = gender_indexer.fit(s_customer_churn).transform(s_customer_churn)
geo_indexer = StringIndexer(inputCol="geography", outputCol="geographyindex")
s_customer_churn = geo_indexer.fit(s_customer_churn).transform(s_customer_churn)
s_customer_churn = s_customer_churn.select('geographyindex', 'estimatedsalary', 'hasrcard', 'numo1')
s_customer_churn = s_customer_churn.coalesce(1)
s_customer_churn.write.option("header", "true").format("csv").mode('Overwrite').save(output_dir)
logger.info("output_dir:" + output_dir)

```

1. Click on **Save** and then the **Run job** button to run the job
2. After the job completes, check the `s3://MLSA-DataLake-Serving-<your initials>/ml-customer-churn/<date>/<guid>/` location in S3 and see whether a new CSV file was generated. Open the file and see whether you see the new processed dataset in the file.

You have now successfully built an AWS Glue job for data processing and feature engineering for ML. Try creating a crawler to crawl the newly processed data in the `MLSA-DataLake-Serving-<your initials>` bucket to make it available in the Glue catalog and run some queries against it. You should see a new table created with multiple partitions (for example, `ml-customer-churn`, `date`, and `GUID`) for the different training datasets. You can query the data by using the `GUID` partition as a query condition.

Building a data pipeline using Glue workflows

Next, we will construct a pipeline that executes a data ingestion job, followed by the creation of a database catalog for the data. Finally, a data processing job will be initiated to generate the training dataset. This pipeline will automate the flow of data from the source to the desired format, ensuring seamless and efficient data processing for ML model training:

1. To start, click on the **Workflows (orchestration)** link in the left pane of the Glue management console.
2. Click on **Add workflow** and enter a name for your workflow on the next screen. Then, click on the **Create workflow** button.
3. Select the workflow you just created and click on **Add trigger**. Select the **Add New** tab, and then enter a name for the trigger and select the **on-demand** trigger type.
4. On the workflow UI designer, you will see a new **Add Node** icon show up. Click on the **Add Node** icon, select the **Crawler** tab, and select **bank_customer_db_crawler**, then, click on **Add**.
5. On the workflow UI designer, click on the **Crawler** icon, and you will see a new **Add Trigger** icon show up. Click on the **Add Trigger** icon, select the **Add new** tab, and select **Start after ANY event** as the trigger logic, and then click on **Add**.
6. On the workflow UI designer, click on the **Add Node** icon, select the **jobs** tab, and select the **customer_churn_process** job.
7. On the workflow UI designer, the final workflow should look like the following diagram:

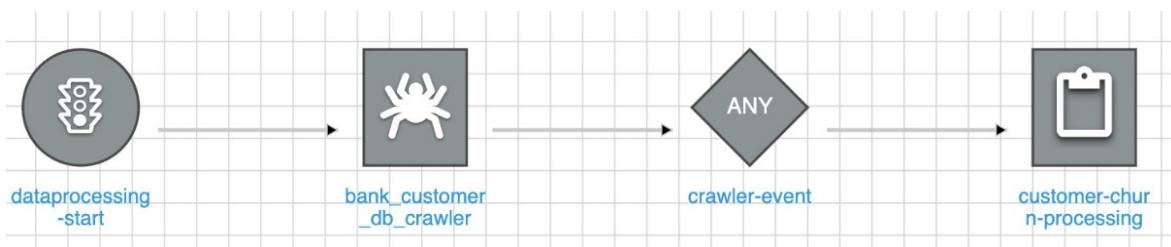


Figure 4.8: Glue data flow design

8. Now, you are ready to run the workflow. Select the workflow and select **Run** from the **Actions** dropdown. You can monitor the running status by selecting the **Run ID** and clicking on **View run details**. You should see something similar to the following screenshot:

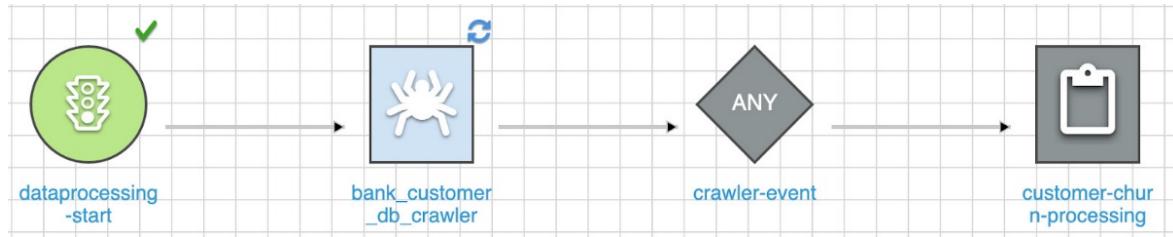


Figure 4.9: Glue workflow execution

9. Try deleting the `customer_data` and `churn_list` tables and re-run the workflow. See whether the new tables are created again. Check the `s3://MLSA-DataLake-Serving-<your initials>/ml-customer-churn/<date>/` S3 location to verify a new folder is created with a new dataset.

Congratulations! You have completed the hands-on lab and learned how to build a simple data lake and its supporting components to allow data cataloging, data querying, and data processing.

Summary

In this chapter, we delved into the considerations for managing data in the context of ML and explored the architecture of an enterprise data management platform for ML. We examined the intersection of data management with the ML life cycle and learned how to design a data lake architecture on AWS. To apply these concepts, we went through the process of building a data lake using AWS Lake Formation. Through hands-on experience, we practiced data ingestion, processing, and cataloging for data discovery, querying, and ML tasks. Additionally, we gained proficiency in using AWS data management tools such as AWS Glue, AWS Lambda, and Amazon Athena. In the next chapter, our focus will shift to the architecture and technologies involved in constructing data science environments using open source tools.

5 Open Source Machine Learning Libraries

Join our book community on Discord

<https://packt.link/EarlyAccessCommunity>



There is a wide range of machine learning (ML) and data science technologies available, encompassing both open source and commercial products. Different organizations have adopted different approaches when it comes to building their ML platforms. Some have opted for in-house teams that leverage open source technology stacks, allowing for greater flexibility and customization. Others have chosen commercial products to focus on addressing specific business and data challenges. Additionally, some organizations have adopted a hybrid architecture, combining open source and commercial tools to harness the benefits of both. As a practitioner in ML solution architecture, it is crucial to be knowledgeable about the available open source ML technologies and their applications in building robust ML solutions. In the upcoming chapters, our focus will be on exploring different open source technologies for experimentation, model building, and the development of machine learning platforms. In this chapter specifically, we will delve into popular machine learning libraries including scikit-learn, Spark, TensorFlow, and PyTorch. We will examine the core capabilities of these libraries and demonstrate how they can be effectively utilized throughout the various stages of a machine learning project lifecycle, encompassing tasks such as data processing, model development, and model evaluation. Moreover, you will have the opportunity to engage in hands-on exercises, gaining practical experience with these machine learning libraries and their application in training models. Specifically, we will be covering the following main topics:

- Core features of open source machine learning libraries
- Understanding the scikit-learn machine learning library
- Understanding the Apache Spark ML machine learning library
- Understanding the TensorFlow machine learning library and hands-on lab
- Understanding the PyTorch machine learning and hands-on lab

Technical requirements

In this chapter, you will need access to your local machine where you installed the **Jupyter** environment from *Chapter 3, Machine Learning Algorithms*. You can find the code samples used in this chapter at <https://github.com/PacktPublishing/The-Machine-Learning-Solutions-Architect-Handbook/tree/main/Chapter05>.

Core features of open source machine learning libraries

Machine learning libraries are software libraries designed to facilitate the implementation of machine learning algorithms and techniques. While they share similarities with other software libraries, what sets them apart is their specialized support for various machine learning functionalities. These libraries typically offer a range of features through different sub-packages, including:

- **Data manipulation and processing:** This includes support for different data tasks such as loading data of different formats, data manipulation, data analysis, data visualization, data transformation, and feature extraction.

- **Model building and training:** This includes support for built-in machine learning algorithms as well as capabilities for building custom algorithms for wide range of ML tasks. Most ML libraries also have built-in support for the commonly used loss functions (such as mean squared error or cross-entropy) and a list of optimizers (such as gradient descent or **adam**) to choose from. Some libraries also provide advanced support for distributed model training across multiple CPU/GPU devices or compute nodes.
- **Model evaluation and validation:** This includes packages for evaluating the performance of trained models, such as model accuracy, precision, recalls, or error rates.
- **Model saving and loading:** This includes support for saving the models to various formats for persistence, and support for loading saved models into memory for predictions.
- **Model serving:** This includes model serving features to expose trained machine learning models behind an API, usually a RESTful API web service.
- **Interpretation:** This includes functionality for interpreting model predictions and feature importance.

Machine learning libraries typically offer support for multiple programming languages, including popular options such as Python, Java, and Scala, catering to diverse user requirements. Python, in particular, has emerged as a prominent language in the field of machine learning, and many libraries provide extensive support for its interface. While the user-facing interface is often implemented in Python, the backend and underlying algorithms of these libraries are primarily written in compiled languages like C++ and Cython. This combination allows for efficient and optimized performance during model training and inference. In the following sections, we will delve into some widely used machine learning libraries to gain a deeper understanding of their features and capabilities.

Understanding the scikit-learn machine learning library

scikit-learn (<https://scikit-learn.org/>) is an open source machine learning library for Python. Initially released in 2007, it is one of the most popular machine learning libraries for solving many machine learning tasks, such as classification, regression, clustering, and dimensionality reduction. scikit-learn is widely used by companies in different industries and academics for solving real-world business cases such as churn prediction, customer segmentation, recommendations, and fraud detection. scikit-learn is built mainly on top of three foundational libraries: **NumPy**, **SciPy**, and **matplotlib**. NumPy is a Python-based library for managing large, multidimensional arrays and matrices, with additional mathematical functions to operate on the arrays and matrices. SciPy provides scientific computing functionality, such as optimization, linear algebra, and Fourier Transform. Matplotlib is used for plotting data for data visualization. scikit-learn is a sufficient and effective tool for a range of common data processing and model-building tasks.

Installing scikit-learn

You can easily install the `scikit-learn` package on different operating systems such as Mac, Windows, and Linux. The `scikit-learn` library package is hosted on the **Python Package Index** site (<https://pypi.org/>) and the **Anaconda** package repository (<https://anaconda.org/anaconda/repo>). To install it in your environment, you can use either the **PIP** package manager or the **Conda** package manager. A package manager allows you to install and manage the installation of library packages in your operating system. To install the `scikit-learn` library using the PIP or Conda package manager, you can simply run `pip install -U scikit-learn` to install it from the **PyPI** index or run `conda install scikit-learn` if you want to use a Conda environment. You can learn more about PIP at <https://pip.pypa.io/> and Conda at <http://docs.conda.io>.

Core components of scikit-learn

The **scikit-learn library** provides a wide range of Python classes and functionality for the various stages of ML life cycle. It consists of several main components, as depicted in the following diagram. By utilizing these components, you can construct machine learning pipelines and perform tasks such as classification, regression, clustering, and more.

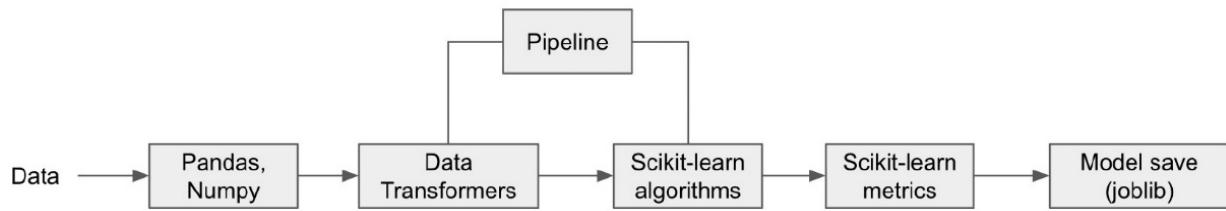


Figure 5.1: scikit-learn components

Now, let's delve deeper into how these components support the different stages of the machine learning life cycle:

- **Preparing data:** For data manipulation and processing, the `pandas` library is commonly used. It provides core data loading and saving functions, as well as utilities for data manipulations such as data selection, data arrangement, and data statistical summaries. `pandas` is built on top of `NumPy`. The `pandas` library also comes with some visualization features such as pie charts, scatter plots, and boxplots.
- **scikit-learn** provides a list of transformers for data processing and transformation, such as imputing missing values, encoding categorical values, normalization, and feature extraction for text and images. You can find the full list of transformers at https://scikit-learn.org/stable/data_transforms.xhtml. Furthermore, you have the flexibility to create custom transformers.
- **Model training:** `scikit-learn` provides a long list of machine learning algorithms (also known as estimators) for classification and regression (for example, logistic regression, K nearest neighbors, and random forest), as well as clustering (for example, k-means). You can find the full list of algorithms at <https://scikit-learn.org/stable/index.xhtml>. The following sample code shows the syntax for using the `RandomForestClassifier` algorithm to train a model using a labeled training dataset:

```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier (max_depth, max_features, n_estimators)
model.fit(train_X, train_y)
```

- **Model evaluation:** `scikit-learn` has utilities for hyperparameter tuning and cross-validation, as well as metrics classes for model evaluations. You can find the full list of model selection and evaluation utilities at https://scikit-learn.org/stable/model_selection.xhtml. The following sample code shows the `accuracy_score` class for evaluating the accuracy of classification models:

```
from sklearn.metrics import accuracy_score
acc = accuracy_score (true_label, predicted_label)
```

- **Model saving:** `scikit-learn` can save model artifacts using Python object serialization (`pickle` or `joblib`). The serialized `pickle` file can be loaded into memory for predictions. The following sample code shows the syntax for saving a model using the `joblib` class:

```
import joblib
joblib.dump(model, "saved_model_name.joblib")
```

- **Pipeline:** `scikit-learn` also provides a pipeline utility for stringing together different transformers and estimators as a single processing pipeline, and it can be reused as a single unit. This is especially useful when you need to preprocess data for modeling training and model prediction, as both require the data to be processed in the same way:

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
pipe = Pipeline([('scaler', StandardScaler()), ('RF', RandomForestClassifier())])
pipe.fit(X_train, y_train)
```

As demonstrated, getting started with `scikit-learn` for experimenting with and constructing machine learning models is straightforward. `scikit-learn` is particularly suitable for typical regression, classification, and clustering

tasks performed on a single machine. However, if you're working with extensive datasets or require distributed training across multiple machines, scikit-learn may not be the optimal choice unless the algorithm supports incremental training, such as `SGDRegressor`. In the following sections, we will explore alternative machine learning libraries that excel in large-scale model training scenarios.

Understanding the Apache Spark ML machine learning library

Apache Spark is an advanced framework for distributed data processing, designed to handle large-scale data processing tasks. With its distributed computing capabilities, Spark enables applications to efficiently load and process data across a cluster of machines by leveraging in-memory computing, thereby significantly reducing processing times. Architecturally, a Spark cluster consists of a master node and worker nodes for running different Spark applications. Each application that runs in a Spark cluster has a driver program and its own set of processes, which are coordinated by the `SparkSession` object in the driver program. The `SparkSession` object in the driver program connects to a cluster manager (for example, Mesos, YARN, Kubernetes, or Spark's standalone cluster manager), which is responsible for allocating resources in the cluster for the Spark application. Specifically, the cluster manager acquires resources on worker nodes called `executors` to run computations and store data for the Spark application. Executors are configured with resources such as the number of CPU cores and memory to meet task processing needs. Once the executors have been allocated, the cluster manager sends the application code (Java JAR or Python files) to the executors. Finally, `SparkContext` sends the tasks to the executors to run. The following diagram shows how a driver program interacts with a cluster manager and executor to run a task:

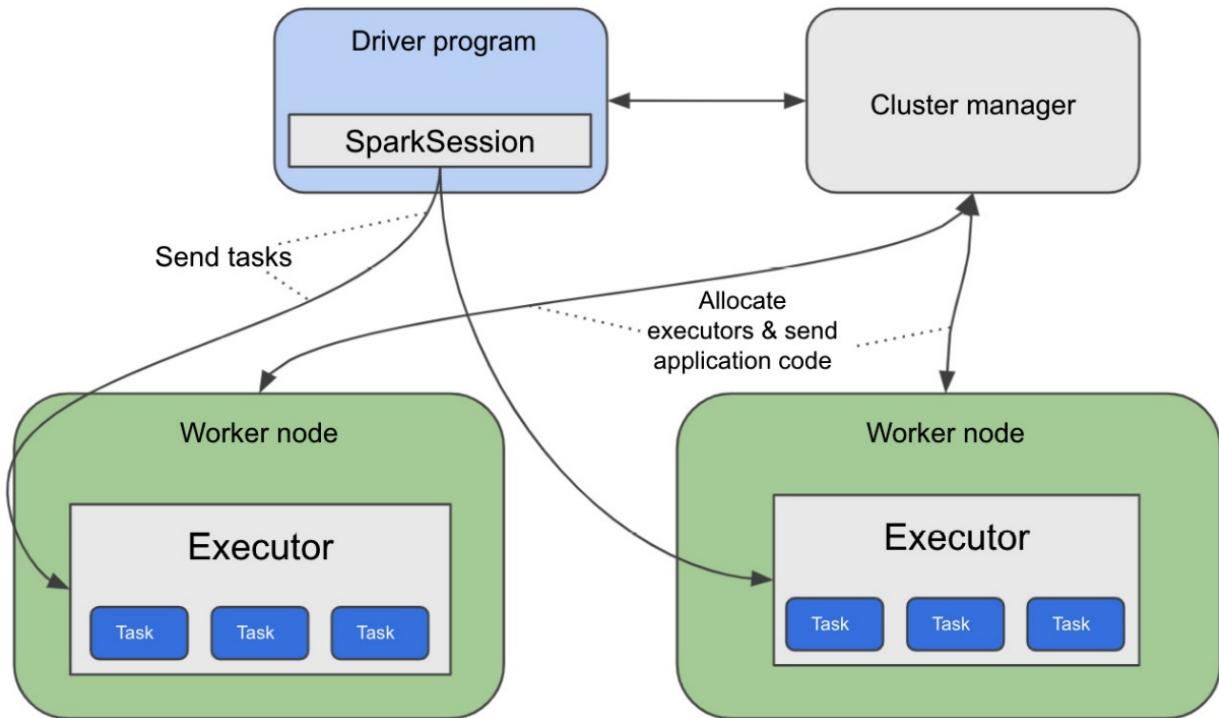


Figure 5.2: Running a Spark application on a Spark cluster

Each Spark application gets its own set of executors, which stay up for the duration of the application. The executors for different applications are isolated from each other, and they can only share data through external data storage. The machine learning package for Spark is called `MLlib`, which runs on top of the distributed Spark architecture. It is capable of processing and training models with a large dataset that does not fit into the memory of a single machine. It provides APIs in different programming languages, including Python, Java, Scala, and R. From a structure perspective, it is very similar to that of the `scikit-learn` library. Spark is highly popular and adopted by companies of all sizes across different industries. Large companies such as **Netflix**, **Uber**, and

Pinterest use Spark for large-scale data processing and transformation, as well as running machine learning models.

Installing Spark ML

Spark ML libraries are included as part of the Spark installation. PySpark is the Python API for Spark, and it can be installed like a regular Python package using PIP (`pip install pyspark`). Note that **PySpark** requires Java and Python to be installed on the machine before it can be installed. You can find Spark's installation instructions at <https://spark.apache.org/docs/latest/>.

Core components of the Spark ML library

Similar to the **scikit-learn** library, Spark and Spark ML provide a full range of functionality for building machine learning models, from data preparation to model evaluation and model persistence. The following diagram shows the core components that are available in Spark for building machine learning models:

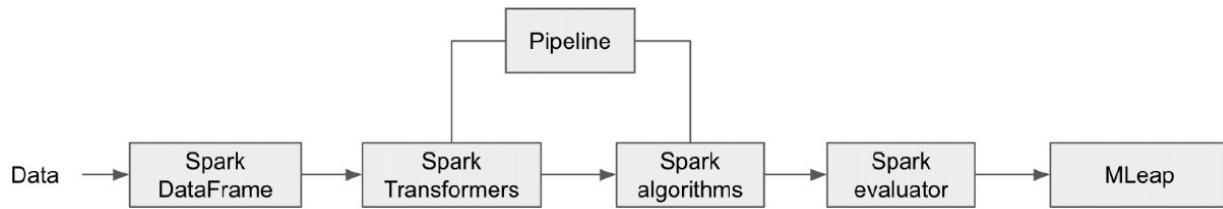


Figure 5.3: Core components of Spark ML

Now, let's take a closer look at the core functions supported by the Spark and Spark ML library packages:

- **Preparing data:** Spark supports Spark DataFrame, a distributed collection of data that can be used for data join, aggregation, filtering, and other data manipulation needs. Conceptually, a Spark DataFrame is equivalent to a table in a relational database. A Spark DataFrame can be distributed (that is, partitioned) across many machines, which allows fast data processing in parallel. Spark DataFrame also operates on a model called the **lazy execution** model. **Lazy execution** defines a set of transformations (for example, adding a column or filtering column) and the transformations are only executed when an action (such as calculating the min/max of a column) is needed. This allows an execution plan for the different transformations and actions to be generated to optimize the execution's performance.

To start using the Spark functionality, you need to create a Spark session. A Spark session creates a **SparkContext** object, which is the entry point to the Spark functionality. The following sample code shows how to create a Spark session:

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('appname').getOrCreate()
```

A Spark DataFrame can be constructed from many different sources, such as structured data files (for example, CSV or JSON) and external databases. The following code sample reads a CSV file into a Spark DataFrame:

```
dataFrame = spark.read.format('csv').load(file_path)
```

There are many transformers for data processing and transformation in Spark, such as **Tokenizer** (breaks text down into individual words) and **StandardScalar** (normalizes a feature into unit deviation and/or zero mean). You can find a list of supported transformers at <https://spark.apache.org/docs/2.1.0/ml-features.xhtml>. To use a transformer, first, you must initiate it with parameters, then call the `fit()` function on the DataFrame that contains the data, and finally call the `transform()` function to transfer the features in the DataFrame:

```
from pyspark.ml.feature import StandardScaler
scaler = StandardScaler(inputCol="features", outputCol="scaledFeatures", withStd=True, withMean=True)
scalerModel = scaler.fit(dataFrame)
scaledData = scalerModel.transform(dataFrame)
```

- **Model training:** Spark ML supports a wide range of machine learning algorithms for classification, regression, clustering, recommendation, and topic modeling. You can find a list of Spark ML algorithms at <https://spark.apache.org/docs/1.4.1/mllib-guide.xhtml>. The following code sample shows how to train a logistic regression model:

```
from pyspark.ml.classification import LogisticRegression
lr_algo = LogisticRegression(maxIter=10, regParam=0.1, elasticNetParam=0.8)
lr_model = lr_algo.fit(dataFrame)
```

- **Model evaluation:** For model selection and evaluation, Spark ML provides utilities for cross-validation, hyperparameter tuning, and model evaluation metrics. You can find the list of evaluators at <https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.evaluation.MulticlassClassificationEvaluator.xhtml>:

```
from pyspark.ml.evaluation import BinaryClassificationEvaluator
dataset = spark.createDataFrame(scoreAndLabels, ["raw", "label"])
evaluator = BinaryClassificationEvaluator()
evaluator.setRawPredictionCol("raw")
evaluator.evaluate(dataset)
evaluator.evaluate(dataset, {evaluator.metricName: "areaUnderPR"})
```

- **Pipeline:** Spark ML also has support for the pipeline concept, similar to that of `scikit-learn`. With the pipeline concept, you can sequence a series of transformation and model training steps as a unified repeatable step:

```
from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import HashingTF, Tokenizer
lr_tokenizer = Tokenizer(inputCol="text", outputCol="tokens")
lr_hashingTF = HashingTF(inputCol="tokens", outputCol="features")
lr_algo = LogisticRegression(maxIter=10, regParam=0.1, elasticNetParam=0.8)
lr_pipeline = Pipeline(stages=[lr_tokenizer, lr_hashingTF, lr_algo])
lr_model = lr_pipeline.fit(training)
```

- **Model saving:** The Spark ML pipeline can be serialized into a serialization format called an **MLeap** bundle, which is an external library from Spark. A serialized **MLeap** bundle can be deserialized back into Spark for batch scoring or a **Mleap** runtime to run real-time APIs. You can find more details about **MLeap** at <https://combust.github.io/mleap-docs/>. The following code shows the syntax for serializing a Spark model into **MLeap** format:

```
import mleap.pyspark
from pyspark.ml import Pipeline, PipelineModel
lr_model.serializeToBundle("saved_file_path", lr_model.transform(dataframe))
```

Spark is a versatile framework that enables large-scale data processing and machine learning. While it excels in traditional machine learning tasks, it also offers limited support for neural network training, including the multilayer perceptron algorithm. However, for more comprehensive deep learning capabilities, we will explore dedicated machine learning libraries in the next section.

Understanding the TensorFlow deep learning library

Initially released in 2015, TensorFlow is a popular open source machine learning library, primarily backed up by Google, that is mainly designed for deep learning. TensorFlow has been used by companies of all sizes for training and building state-of-the-art deep learning models for a range of use cases, including computer vision, speech recognition, question-answering, text summarization, forecasting, and robotics. TensorFlow works based on the

concept of computational graph, where data flows through nodes that represent mathematical operations. The core idea is to construct a graph of operations and tensors, with tensors being n-dimensional arrays that carry data. An example of a tensor could be a scalar value (for example, 1.0), a one-dimensional vector (for example, [1.0, 2.0, 3.0]), a two-dimensional matrix (for example, [[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]]), or even higher dimensional matrices. Operations are performed on these tensors, allowing for mathematical computations like addition or matrix multiplication. The following diagram shows a sample computational graph for performing a sequence of mathematical operations on tensors:

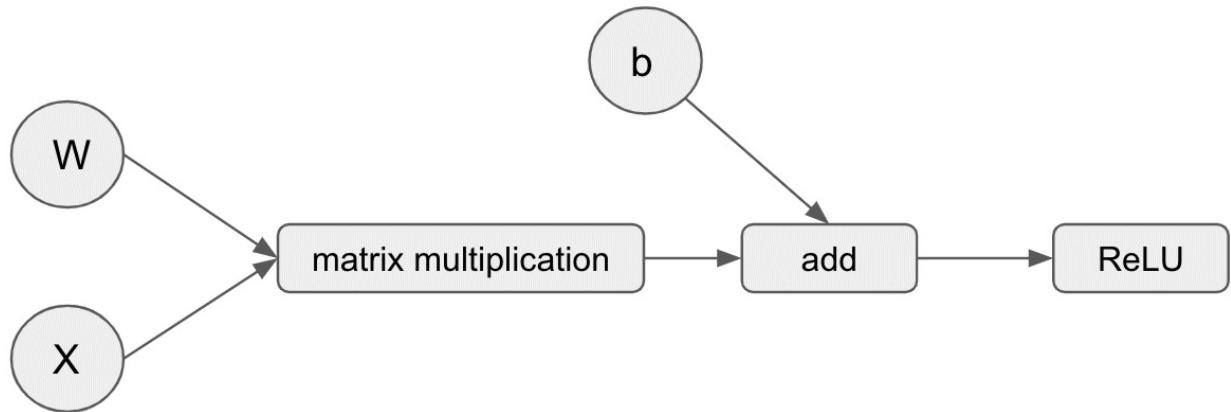


Figure 5.4: Data flow diagram

In the preceding computational diagram, the rectangular nodes are mathematical operations, while the circles represent tensors. This particular diagram shows a computational graph for performing an artificial neuron tensor operation, which is to perform a matrix multiplication of W and X , followed by the addition of b , and, lastly, apply a *ReLU* action function. The equivalent mathematical formula is as follows:

$$f(x) = \text{ReLU}(Wx + b)$$

TensorFlow allows users to define and manipulate the computation graph using its high-level API or by directly working with its lower-level components. This flexibility allows researchers and developers to create complex models and algorithms. Furthermore, TensorFlow supports distributed computing, allowing the graph to be executed across multiple devices or machines, which is crucial for handling large-scale machine learning tasks. This distributed architecture enables TensorFlow to leverage the power of clusters or GPUs to accelerate the training and inference of deep learning models.

Installing Tensorflow

TensorFlow can be installed using the `pip install --upgrade tensorflow` command in a Python-based environment. After installation, TensorFlow can be used just like any other Python library package.

Core components of TensorFlow

The TensorFlow library provides a rich set of features for different machine learning steps, from data preparation to model serving. The following diagram illustrates the core building blocks of the TensorFlow library:

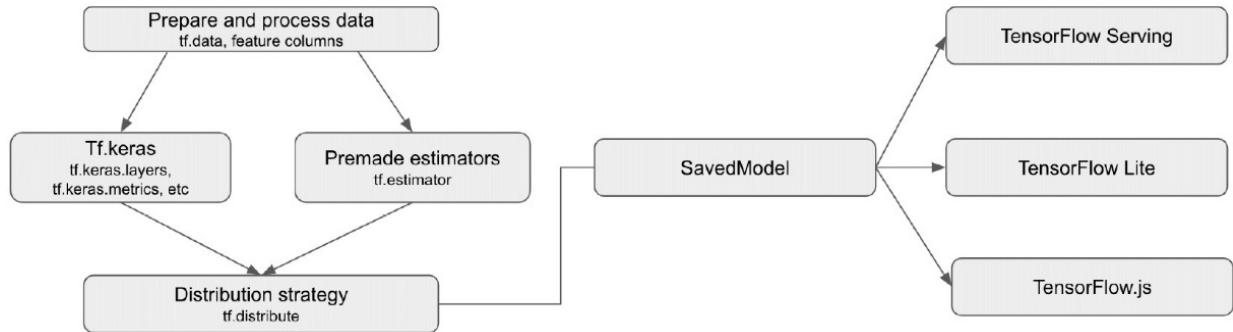


Figure 5.5: TensorFlow components

Training a machine learning model using TensorFlow 2.x involves the following main steps:

- **Preparing the dataset:** TensorFlow 2.x provides a `tf.data` library for efficiently loading data from sources (such as files), transforming data (such as changing the values of the dataset), and setting up the dataset for training (such as configuring batch size or data prefetching). These data classes provide efficient ways to pass data to the training algorithms for optimized model training. The TensorFlow **Keras** API also provides a list of built-in classes (MNIST, CIFAR, IMDB, MNIST Fashion, and Reuters News wires) for building simple deep learning models. You can also feed a NumPy array or Python generator (a function that behaves like an iterator) to a model in TensorFlow for model training, but `tf.data` is the recommended approach.
- **Defining the neural network:** TensorFlow 2.x provides multiple ways to use or build a neural network for model training. You can use the premade estimators (the `tf.estimator` class) such as `DNNRegressor` and `DNNClassifier` to train models. Or, you can create custom neural networks using the `tf.keras` class, which provides a list of primitives such as `tf.keras.layers` for constructing neural network layers and `tf.keras.activation` such as `ReLU`, `Sigmoid`, and `Softmax` for building neural networks. `Softmax` is usually used as the last output of a neural network for a multiclass problem. It takes a vector of real numbers (positive and negative) as input and normalizes the vector as a probability distribution to represent the probabilities of different class labels, such as the different types of hand-written digits. For binary classification problems, `Sigmoid` is normally used and it returns a value between 0 and 1.
- **Defining the loss function:** TensorFlow 2.x provides a list of built-in loss functions such as **mean squared error (MSE)** and **mean absolute error (MAE)** for regression tasks and cross-entropy loss for classification tasks. You can find more details about MSE and MAE at https://en.wikipedia.org/wiki/Mean_squared_error and https://en.wikipedia.org/wiki/Mean_absolute_error. You can find a list of supported loss functions in the `tf.keras.losses` class. For more detail about the different losses, refer to <https://keras.io/api/losses/>. There is also the flexibility to define custom loss functions, if the built-in loss functions do not meet the needs.
- **Selecting the optimizer:** TensorFlow 2.x provides a list of built-in optimizers for model training, such as the **Adam** optimizer and the **Stochastic Gradient Descent (SGD)** optimizer for parameters optimization, with its `tf.keras.optimizers` class. You can find more details about the different supported optimizers at <https://keras.io/api/optimizers/>. Adam and SGD are two of the most commonly used optimizers.
- **Selecting the evaluation metrics:** TensorFlow 2.x has a list of built-in model evaluation metrics (for example, accuracy and cross-entropy) for model training evaluations with its `tf.keras.metrics` class. You can also define custom metrics for model evaluation during training.
- **Compiling the network into a model:** This step compiles the defined network, along with the defined loss function, optimizer, and evaluation metrics, into a computational graph that's ready for model training.
- **Fitting the model:** This step kicks off the model training process by feeding the data to the computational graph through batches and multiple epochs to optimize the model parameters.
- **Evaluating the trained model:** Once the model has been trained, you can evaluate the model using the `evaluate()` function against the test data.
- **Saving the model:** The model can be saved in TensorFlow `SavedModel` serialization format or **Hierarchical Data Format (HDF5)** format.

- **Model serving:** TensorFlow comes with a model serving framework called TensorFlow Serving, which we will cover in greater detail in *Chapter 7, Open Source Machine Learning Platform*.

The TensorFlow library is designed for large-scale production-grade data processing and model training. As such, it provides capabilities for large-scale distributed data processing and model training on a cluster of servers against a large dataset. We will cover large-scale distributed data processing and model training in greater detail in *Chapter 10, Advanced ML Engineering*. To support the complete process of building and deploying machine learning pipelines, TensorFlow provides TensorFlow Extended (TFX). TFX integrates multiple components and libraries from the TensorFlow ecosystem, creating a cohesive platform for tasks such as data ingestion, data validation, preprocessing, model training, model evaluation, and model deployment. Its architecture is designed to be modular and scalable, enabling users to tailor and expand the pipeline to meet their specific requirements. You can get more detail on TFX at <https://www.tensorflow.org/tfx>. TensorFlow offers an expanded ecosystem of libraries and extensions for solving a wide range of advanced machine learning problems, including federate learning (training model using decentralized data), model optimization (optimizing model for deployment and execution), and probabilistic reasoning (reasoning under uncertainty using probability theory).

Hands-on exercise – training a TensorFlow model

In this exercise, you will learn how to install the TensorFlow library in your local Jupyter environment and build and train a simple neural network model. Launch a Jupyter notebook that you have previously installed on your machine. If you don't remember how to do this, visit the *Hands-on lab* section of *Chapter 3, Machine Learning Algorithms*. Once the Jupyter notebook is running, create a new folder by selecting the **New** dropdown and then **Folder**. Rename the folder **TensorFlowLab**. Open the **TensorFlowLab** folder, create a new notebook inside this folder, and rename the notebook **Tensorflow-lab1.ipynb**. Now, let's get started:

1. Inside the first cell, run the following code to install TensorFlow:

```
! pip3 install --upgrade tensorflow
```

1. Now, we must import the library and load the sample training data. We will use the built-in `fashion_mnist` dataset that comes with the `keras` library to do so. Next, we must load the data into a `tf.data.Dataset` class and then call its `batch()` function to set up a batch size. Run the following code block in a new cell to load the data and configure the dataset:

```
import numpy as np
import tensorflow as tf
train, test = tf.keras.datasets.fashion_mnist.load_data()
images, labels = train
labels = labels.astype(np.int32)
images = images/256
train_ds = tf.data.Dataset.from_tensor_slices((images, labels))
train_ds = train_ds.batch(32)
```

1. Let's see what the data looks like. Run the following code block in a new cell to view the sample data:

```
from matplotlib import pyplot as plt
print ("label:" + str(labels[0]))
pixels = images[0]
plt.imshow(pixels, cmap='gray')
plt.show()
```

1. Next, we build a simple MLP network with two hidden layers (one with 100 nodes and one with 50 nodes) and an output layer with 10 nodes (each node represents a class label). Then, we must compile the network using the **Adam** optimizer, use the cross-entropy loss as the optimization objective, and use the accuracy as the measuring metric. The Adam optimizer is a variation of **gradient descent (GD)**, and it improves upon GD mainly in the area of the adaptive learning rate for updating the parameters to improve model convergence, whereas GD uses a constant learning rate for parameter updating. Cross-entropy measures the performance of

a classification model, where the output is the probability distribution for the different classes adding up to 1. The cross-entropy error increases when the predicted distribution diverges from the actual class label. To kick off the training process, we must call the `fit()` function. We will run the training for 10 epochs. One epoch is one pass of the entire training dataset:

```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(100, activation="relu"),
    tf.keras.layers.Dense(50, activation="relu"),
    tf.keras.layers.Dense(10),
    tf.keras.layers.Softmax()
])
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=[tf.keras.metrics.SparseCategoricalAccuracy()])
model.fit(train_ds, epochs=10)
```

When the model is training, you should see a loss metric and accuracy metrics are being reported for each epoch.

1. Now that the model has been trained, we need to validate its performance using the test dataset. In the following code, we are creating a `test_ds` for the test data:

```
images_test, labels_test = test
labels_test = labels_test.astype(np.int32)
images_test = images_test/256

test_ds = tf.data.Dataset.from_tensor_slices((images_test, labels_test))
test_ds = train_ds.batch(32)
test_ds = train_ds.shuffle(30)
results = model.evaluate(test_ds)
print("test loss, test acc:", results)
```

1. You can also use the standalone `keras.metrics` to evaluate the model. Here, we are getting the prediction results and using `tf.keras.metrics.Accuracy` to calculate the accuracy of predictions against the true values in `test[1]`:

```
predictions = model.predict(test[0])
predicted_labels = np.argmax(predictions, axis=1)
m = tf.keras.metrics.Accuracy()
m.update_state(predicted_labels, test[1])
m.result().numpy()
```

1. To save the model, run the following code in a new cell. It will save the model in `SavedModel` serialization format:

```
model.save(filepath='model', save_format='tf')
```

1. Open the `model` directory. You should see that several files have been generated, such as `saved_model.pb`, and several files under the `variables` subdirectory.

Great job! You have successfully installed the TensorFlow package in your local Jupyter environment and completed the training of a deep learning model. Through this process, you have gained the basic knowledge about TensorFlow and its capabilities for training deep learning models. Now, let's shift our focus to PyTorch, another widely used and highly regarded deep learning library that excels in both experimental and production-grade ML model training.

Understanding the PyTorch deep learning library

PyTorch is an open source machine learning library that was designed for deep learning using GPUs and CPUs. Initially released in 2016, it is a highly popular machine learning framework with a large following and many

adoptions. Many technology companies, including tech giants such as **Facebook**, **Microsoft**, and **Airbnb**, all use PyTorch heavily for a wide range of deep learning use cases, such as computer vision and natural language processing. PyTorch strikes a good balance of performance (using a C++ backend) with ease of use with default support for dynamic computational graphs and interoperability with the rest of the Python ecosystem. For example, with PyTorch, you can easily convert between **NumPy** arrays and **PyTorch** tensors. To allow for easy backward propagation, PyTorch has built-in support for automatically computing gradients, a vital requirement for gradient-based model optimization. The PyTorch library consists of several key modules, including tensors, **Autograd**, **Optimizer**, and **Neural Network**. Tensors are used to store and operate multidimensional arrays of numbers. You can perform various operations on tensors such as matrix multiplication, transpose, return max number, and dimensionality manipulation. PyTorch supports automatic gradient calculation with its Autograd module. When performing a forward pass, the Autograd module simultaneously builds up a function that computes the gradient. The Optimizer module provides various algorithms such as SGD and Adam for updating model parameters. The Neural Network module provides modules that represent different layers of a neural network such as the linear layer, embedding layer, and dropout layer. It also provides a list of loss functions that are commonly used for training deep learning models.

Installing PyTorch

PyTorch can run on different operating systems, including Linux, macOS, and Windows. You can follow the instructions at <https://pytorch.org/> to install it in your environment. For example, you can use the `pip install torch` command to install it in a Python-based environment.

Core components of PyTorch

Similar to TensorFlow, PyTorch also supports the end-to-end machine learning workflow, from data preparation to model serving. The following diagram shows what different PyTorch modules are used to train and serve a PyTorch model:

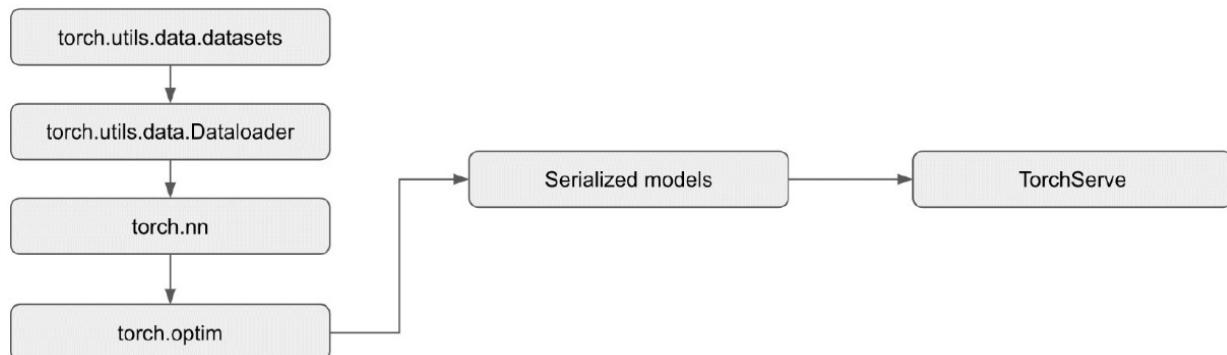


Figure 5.6: PyTorch modules for model training and serving

The steps involved in training a deep learning model are very similar to that of TensorFlow model training. We'll look at the PyTorch-specific details in the following steps:

1. **Preparing the dataset:** PyTorch provides two primitives for dataset and data loading management: `torch.utils.data.Dataset` and `torch.utils.data.DataLoader`. **Dataset** stores data samples and their corresponding labels, while **DataLoader** wraps around the dataset and provides easy and efficient access to the data for model training. **DataLoader** provides functions such as `shuffle`, `batch_size`, and `prefetch_factor` to control how the data is loaded and fed to the training algorithm.
2. As the data in the dataset might need to be transformed before training is performed, **Dataset** allows you to use a user-defined function to transform the data.
3. **Defining the neural network:** PyTorch provides a high-level abstraction for building neural networks with its `torch.nn` class. `torch.nn` provides built-in support for different neural network layers such as linear

layers and convolutional layers, as well as activation layers such as Sigmoid and ReLU. It also has container classes such as `nn.Sequential` for packaging different layers into a complete network. Existing neural networks can also be loaded into PyTorch for training.

4. **Defining the loss function:** PyTorch provides several built-in loss functions in its `torch.nn` class, such as `nn.MSELoss` and `nn.CrossEntropyLoss`.
5. **Selecting the optimizer:** PyTorch provides several optimizers with its `nn.optim` classes. Examples of optimizers include `optim.SGD`, `optim.Adam`, and `optim.RMSProp`. All the optimizers have a `step()` function that updates model parameters with each forward pass. There's also a backward pass that calculates the gradients.
6. **Selecting the evaluation metrics:** The PyTorch `ignite.metrics` class provides several evaluation metrics such as Precision, Recall, and `RootMeanSquaredError` for evaluating model performances. You can learn more about precision and recall at https://en.wikipedia.org/wiki/Precision_and_recall. You can also use the `scikit-learn` metrics libraries to help evaluate models.
7. **Training the model:** Training a model in PyTorch involves three main steps in each training loop: forward pass the training data, backward pass the training data to calculate the gradient, and performing the optimizer step to update the gradient.
8. **Saving/loading the model:** The `torch.save()` function saves a model in a serialized pickle format. The `torch.load()` function loads a serialized model into memory for inference. A common convention is to save the files with the `.pth` or `.pt` extension. You can also save multiple models into a single file.
9. **Model serving:** PyTorch comes with a model serving library called **TorchServe**, which we will cover in more detail *Chapter 7, Open Source Machine Learning Platforms*.

The PyTorch library supports large-scale distributed data processing and model training, which we will cover in more detail in *Chapter 10, Advanced ML Engineering*. Like TensorFlow, PyTorch also offer an ecosystem of library packages for a wide range of ML problems, including ML privacy, adversarial robustness, video understanding, and drug discovery. Now that you have learned about the fundamentals of PyTorch, let's get hands-on through a simple exercise.

Hands-on exercise – building and training a PyTorch model

In this hands-on exercise, you will learn how to install the PyTorch library in your local machine and train a simple deep learning model using PyTorch. Launch a Jupyter notebook that you have previously installed on your machine. If you don't remember how to do this, visit the *Hands-on lab* section of *Chapter 3, Machine Learning Algorithms*. Now, let's get started:

1. Create a new folder called `pytorch-lab` in your Jupyter notebook environment and create a new notebook file called `pytorch-lab1.ipynb`. Run the following command in a cell to install PyTorch and the `torchvision` package. `torchvision` contains a set of computer vision models and datasets. We will use the pre-built MNIST dataset in the `torchvision` package for this exercise:

```
!pip3 install torch
!pip3 install torchvision
```

1. The following sample code shows the previously mentioned main components. Be sure to run each code block in a separate Jupyter notebook cell for easy readability.

First, we must import the necessary library packages and load the MNIST dataset from the `torchvision` dataset class:

```
import numpy as np
import matplotlib.pyplot as plt
import torch
from torchvision import datasets, transforms
from torch import nn, optim
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5,), (0.5,))])
```

```

trainset = datasets.MNIST('pytorch_data/train/', download=True, train=True, transform=transform)
valset = datasets.MNIST('pytorch_data/test/', download=True, train=False, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=64, shuffle=True)

```

1. Next, we must construct an MLP neural network for classification. This MLP network has two hidden layers with ReLU activation for the first and second layers. The MLP model takes an input size of 784, which is the flattened dimension of a 28x28 image. The first hidden layer has 128 nodes (neurons), while the second layer has 64 nodes (neurons). The final layer has 10 nodes because we have 10 class labels:

```

model = nn.Sequential(nn.Linear(784, 128),
                      nn.ReLU(),
                      nn.Linear(128, 64),
                      nn.ReLU(),
                      nn.Linear(64, 10))

```

1. Let's show a sample of the image data:

```

images, labels = next(iter(trainloader))
pixels = images[0][0]
plt.imshow(pixels, cmap='gray')
plt.show()

```

1. Now, we must define a **cross-entropy loss function** for the training process since we want to measure the error in the probability distribution for all the labels. Internally, PyTorch's `CrossEntropyLoss` automatically applies a `softmax` to the network output to calculate the probability distributions for the different classes. For the optimizer, we have chosen the Adam optimizer with a learning rate of 0.003. The `view()` function flattens the two-dimensional input array (28x28) into a one-dimensional vector since our neural network takes one-dimensional vector input:

```

criterion = nn.CrossEntropyLoss()
images = images.view(images.shape[0], -1)
output = model(images)
loss = criterion(output, labels)
optimizer = optim.Adam(model.parameters(), lr=0.003)

```

1. Now, let's start the training process. We are going to run 15 epochs. Unlike the TensorFlow Keras API, where you just call a `fit()` function to start the training, PyTorch requires you to build a training loop and specifically run the forward pass (`model(images)`), run the backward pass to learn (`loss.backward()`), update the model weights (`optimizer.step()`), and then calculate the total loss and the average loss. For each training step, `trainloader` returns one batch (a batch size of 64) of training data samples. Each training sample is flattened into a 784-long vector. The optimizer is reset with zeros for each training step:

```

epochs = 15
for e in range(epochs):
    running_loss = 0
    for images, labels in trainloader:
        images = images.view(images.shape[0], -1)
        optimizer.zero_grad()
        output = model(images)
        loss = criterion(output, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
    else:
        print("Epoch {} - Training loss: {}".format(e, running_loss/len(trainloader)))

```

When the training code runs, it should print out the average loss for each epoch.

1. To test the accuracy using the validation data, we must run the validation dataset through the trained model and use `scikit-learn.metrics.accuracy_score()` to calculate the model's accuracy:

```
valloader = torch.utils.data.DataLoader(valset, batch_size=valset.data.shape[0], shuffle=True)
val_images, val_labels = next(iter(valloader))
val_images = val_images.view(val_images.shape[0], -1)
predictions = model(val_images)
predicted_labels = np.argmax(predictions.detach().numpy(), axis=1)
from sklearn.metrics import accuracy_score
accuracy_score(val_labels.detach().numpy(), predicted_labels)
```

1. Finally, we must save the model to a file:

```
torch.save(model, './model/my_mnist_model.pt')
```

Congratulations! You have successfully installed PyTorch in your local Jupyter environment and trained a deep learning PyTorch model.

Summary

In this chapter, we have explored several popular open source machine learning library packages, including scikit-learn, Spark ML, TensorFlow, and PyTorch. By now, you should have a good understanding of the fundamental components of these libraries and how they can be leveraged to train machine learning models. Additionally, we have delved into the TensorFlow and PyTorch frameworks to construct artificial neural networks, train deep learning models, and save these models to files. These model files can then be utilized in model serving environments to make predictions. In the next chapter, we will delve into Kubernetes and its role as a foundational infrastructure for constructing open source machine learning solutions.

6 Kubernetes Container Orchestration Infrastructure Management

Join our book community on Discord

<https://packt.link/EarlyAccessCommunity>



While establishing a local data science setup for individual use for simple ML tasks may seem straightforward, creating a robust and scalable data science environment for multiple users, catering to diverse machine learning tasks and effectively tracking ML experiments, poses significant challenges. In this chapter, we will explore Kubernetes, an indispensable open source container orchestration platform that serves as a critical foundation for constructing open source ML platforms. Kubernetes offers a wealth of capabilities, enabling the seamless management and orchestration of containers at scale. By leveraging Kubernetes, organizations can efficiently deploy and manage ML workloads, ensuring high availability, scalability, and resource utilization optimization. We will delve into the core concepts of Kubernetes, gaining insights into its networking architecture and essential components. Furthermore, we will explore its robust security features and granular access control mechanisms, crucial for safeguarding ML environments and sensitive data. Through practical exercises, you will have the opportunity to build your own Kubernetes cluster and leverage its power to deploy containerized applications. Specifically, we will cover the following topics:

- Introduction to containers
- Kubernetes overview and core concepts
- Kubernetes networking
- Kubernetes security and access control
- Hands-on lab – building a Kubernetes infrastructure on **AWS**

Technical requirements

In this chapter, you will continue to use services in your AWS account for the hands-on portion of the chapter. We will be using several AWS services, including the **AWS Elastic Kubernetes Service (EKS)**, **AWS CloudShell**, and **AWS EC2**. All code files used in this chapter are located on GitHub: <https://github.com/PacktPublishing/The-Machine-Learning-Solutions-Architect-Handbook/tree/main/Chapter06>.

Introduction to containers

A **container** is a form of operating system virtualization, and it has been a very popular computing platform for software deployment and running modern software based on micro-services architecture. A container allows you to package and run computer software with isolated dependencies. Compared to server virtualization, such as Amazon EC2 or **VMware** virtual machines, containers are more lightweight and portable, as they share the same operating system and do not contain operating system images in each container. Each container has its own filesystem, shares of computing resources, and process space for the custom applications running inside it. The concept of containerization technology traced back to 1970s with the **chroot system** and **Unix Version 7**. However, container technology did not gain much attraction in the software development community for the next two decades and remained dormant. While it picked up some steam and made remarkable advances from 2000 to 2011, it was the introduction of **Docker** in 2013 that started a renaissance of container technology. You can run all kinds of applications inside containers, such as simple programs like data processing scripts or complex systems

like databases. The following diagram illustrates how container deployment is different from other types of deployment. Note that a container runtime can also run in the guest operating system of a virtualized environment to host containerized applications:

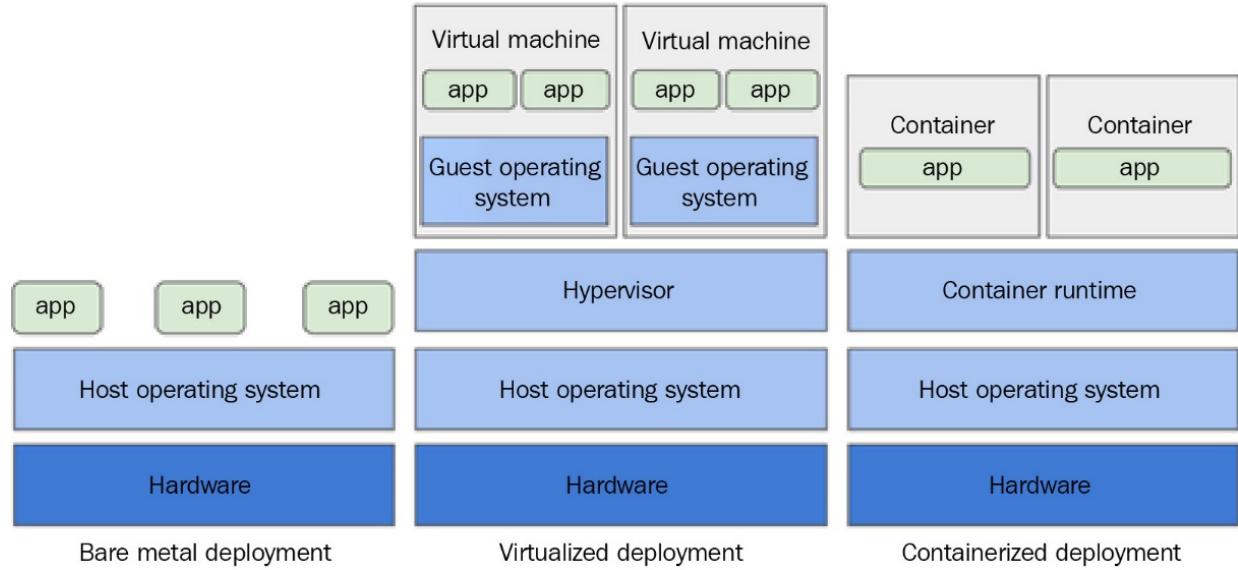


Figure 6.1: The differences between bare metal, virtualized, and container deployment

Containers are packaged as Docker images, which are made of all the files (such as installation, application code, and dependencies) that are essential to run the containers and the applications in them. One way to build a Docker image is the use of a **Dockerfile** – a plain-text file that provides specifications on how to build a Docker image. Once a Docker image is created, it can be executed in a container runtime environment. The following is an example **Dockerfile** for building a Docker image to create a runtime environment based on the **Ubuntu** operating system (the `FROM` instruction) and install various **Python** packages, such as `python3`, `numpy`, `scikit-learn` and `pandas` (the `RUN` instructions):

```
FROM ubuntu:18.04
RUN apt-get -y update && apt-get install -y --no-install-recommends \
    wget \
    python3-pip \
    nginx \
    ca-certificates \
    && rm -rf /var/lib/apt/lists/*
RUN ln -s /usr/bin/python3 /usr/bin/python
RUN ln -s /usr/bin/pip3 /usr/bin/pip
RUN pip --no-cache-dir install numpy==1.16.2 scipy==1.2.1 scikit-learn==0.20.2 pandas flask gunic
```

To build a Docker image from this **Dockerfile**, you can use the Docker **build** command, which is a utility that comes as part of the Docker installation. Now we have an understanding of containers, next, let's dive into Kubernetes.

Kubernetes overview and core concepts

Managing and orchestrating a small number of containers and containerized applications manually within a compute environment can be relatively manageable. However, as the number of containers and servers grows, the task becomes increasingly complex. Enter Kubernetes, a powerful open source system specifically designed to address these challenges. First introduced in 2014, Kubernetes (commonly abbreviated as K8s, derived from replacing "ubernete" with the digit 8) offers a comprehensive solution for efficiently managing containers at scale across clusters of servers. Kubernetes follows a distributed architecture consisting of a master node and multiple

worker nodes within a server cluster. The master node, often referred to as the control plane, assumes the primary role in managing the entire Kubernetes cluster. It comprises four essential components, each playing a distinct role in the overall system:

- **API server:** The API server acts as the central communication hub for all interactions with the Kubernetes cluster. It provides a RESTful interface through which users, administrators, and other components can interact with the cluster. The API server handles authentication, authorization, and validation of requests, ensuring secure and controlled access to the cluster's resources.
- **Scheduler:** The scheduler component is responsible for determining the optimal placement of workloads or pods across the available worker nodes within the cluster.
- **Controller:** The controller manager oversees the cluster's overall state and manages various background tasks to maintain the desired system state.
- **etcd:** etcd is a distributed key-value store that serves as the cluster's reliable data store, ensuring consistent and consistent access to critical configuration and state information. It stores the cluster's current state, configuration details, and other essential data, providing a reliable source of truth for the entire system.

The master node exposes the **API server** layer that allows programmatic control of the cluster. An example of an API call could be the deployment of a web application on the cluster. The control plane also tracks and manages all configuration data in **etcd** that is responsible for storing all the cluster data, such as the desired number of container images to run, compute resource specification, and size of storage volume for a web application running on the cluster. Kubernetes uses **controller** to monitor the current states of Kubernetes resources and take the necessary actions (for example, request the change via the API server) to move the current states to the desired states if there are differences (such as the difference in the number of the running containers) between the two states. The controller manager in the master node is responsible for managing all the Kubernetes controllers. Kubernetes comes with a set of built-in controllers such as **scheduler**, which is responsible for scheduling **Pods** (units of deployment that we will discuss in more detail later) to run on worker nodes when there is a change request. Other examples include **Job controller**, which is responsible for running and stopping one or more Pods for a task, and **Deployment controller**, which is responsible for deploying Pods based on a deployment manifest, such as a deployment manifest for a web application. The following figure (*Figure 6.2*) shows the core architecture components of a Kubernetes cluster:

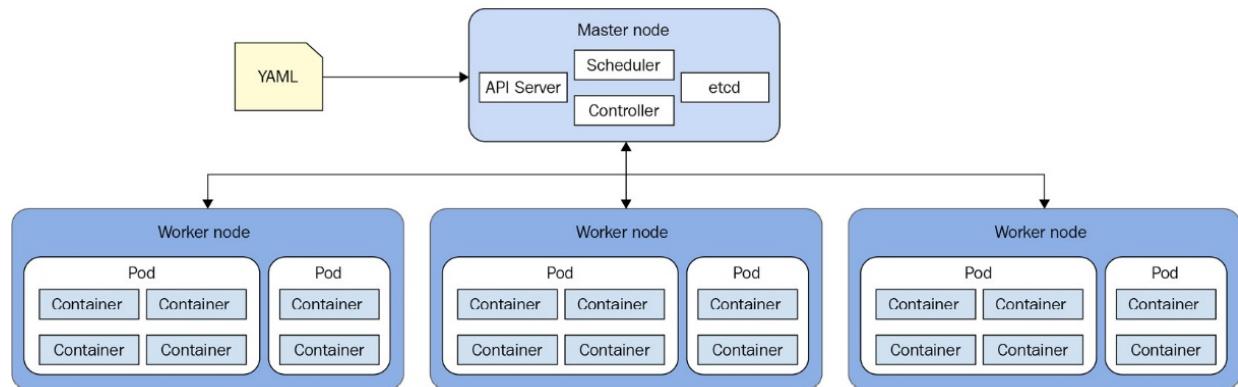


Figure 6.2: Kubernetes architecture

To interact with a Kubernetes cluster control plane, you can use the **kubectl** command-line utility, the Kubernetes Python client (<https://github.com/kubernetes-client/python>), or access directly using the RESTful API. You can get a list of supported **kubectl** commands at <https://kubernetes.io/docs/reference/kubectl/cheatsheet/>. There are several fundamental technical concepts that form the core of the Kubernetes architecture. These concepts are essential to understanding and effectively working with Kubernetes. The following are some of the key concepts:

Namespaces

Namespaces organize clusters of worker machines into virtual sub-clusters. They are used to provide logical separation of resources owned by different teams and projects while still allowing ways for different namespaces to communicate. A namespace can span multiple worker nodes, and it can be used to group a list of permissions under a single name to allow authorized users to access resources in a namespace. Resource usage controls can be enforced to namespaces such as quotas for CPU and memory resources. Namespaces also make it possible to name resources with identical names if the resources reside in the different namespaces to avoid naming conflicts. By default, there is a **default** namespace in Kubernetes. You can create additional namespaces as needed. The default namespace is used if a namespace is not specified.

Pods

Kubernetes deploys computing in a logical unit called a Pod. All Pods must belong to a Kubernetes namespace (either the default namespace or a specified namespace). One or more containers can be grouped into a Pod, and all containers in the Pod are deployed and scaled together as a single unit and share the same context, such as Linux namespaces and filesystems. Each Pod has a unique IP address that's shared by all the containers in a Pod. A Pod is normally created as a workload resource, such as a Kubernetes Deployment or Kubernetes Job.

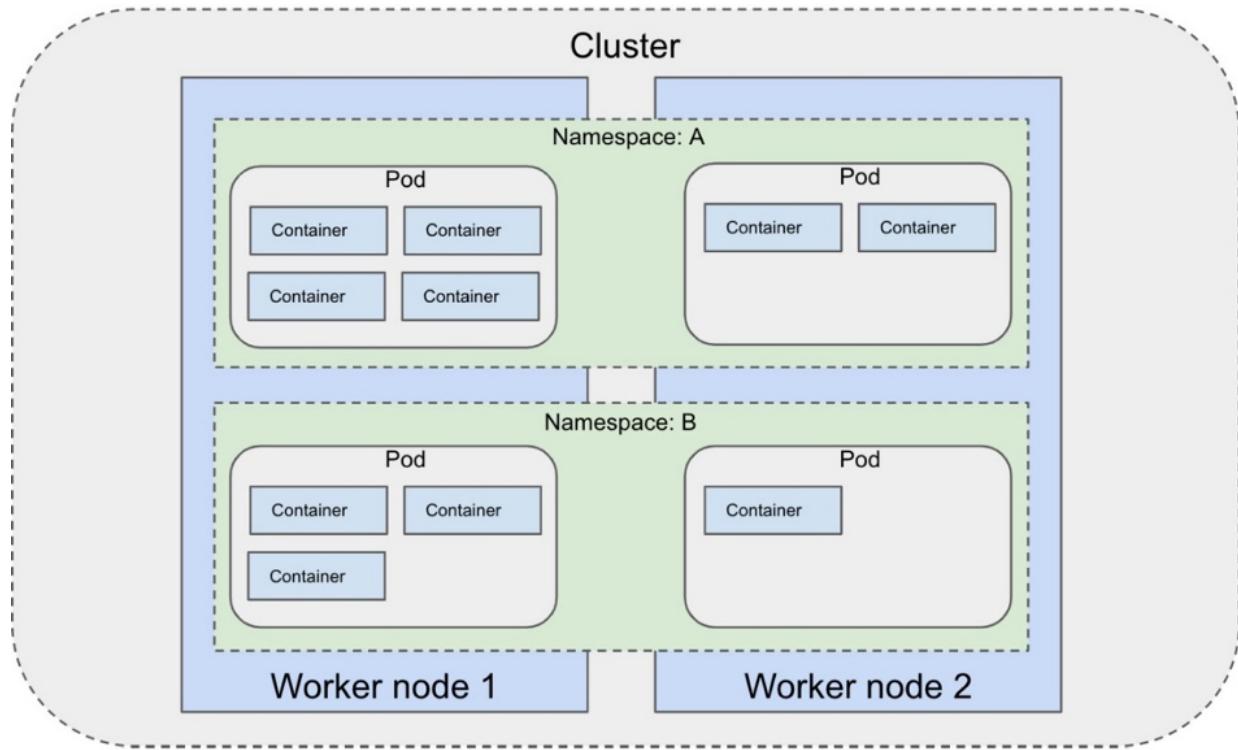


Figure 6.3: Namespaces, Pods, and containers

The preceding figure (Figure 6.3) shows the relationship between namespaces, Pods, and containers in a Kubernetes cluster. In this figure, each namespace contains its own set of Pods and each Pod can contain one or more containers running in it.

Deployment

A deployment is used by Kubernetes to create or modify Pods that run containerized applications. For example, to deploy a containerized application, you create a configuration manifest file (usually in a **YAML** file format) that specifies details, such as the container deployment name, namespaces, container image URI, number of Pod replicas, and the communication port for the application. After the deployment is applied using a Kubernetes client

utility (**kubectl**), the corresponding Pods running the specified container images will be created on the worker nodes. The following example creates a deployment of Pods for an **Nginx** server with the desired specification:

```
apiVersion: apps/v1 # k8s API version used for creating this deployment
kind: Deployment # the type of object. In this case, it is deployment
metadata:
  name: nginx-deployment # name of the deployment
spec:
  selector:
    matchLabels:
      app: nginx # an app label for the deployment. This can be used to look up/select Pods
  replicas: 2 # tells deployment to run 2 Pods matching the template
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.14.2 # Docker container image used for the deployment
          ports:
            - containerPort: 80 # the networking port to communicate with the containers
```

The following figure shows the flow of applying the preceding deployment manifest file to a Kubernetes cluster and creates two Pods to host two copies of the **Nginx** container:

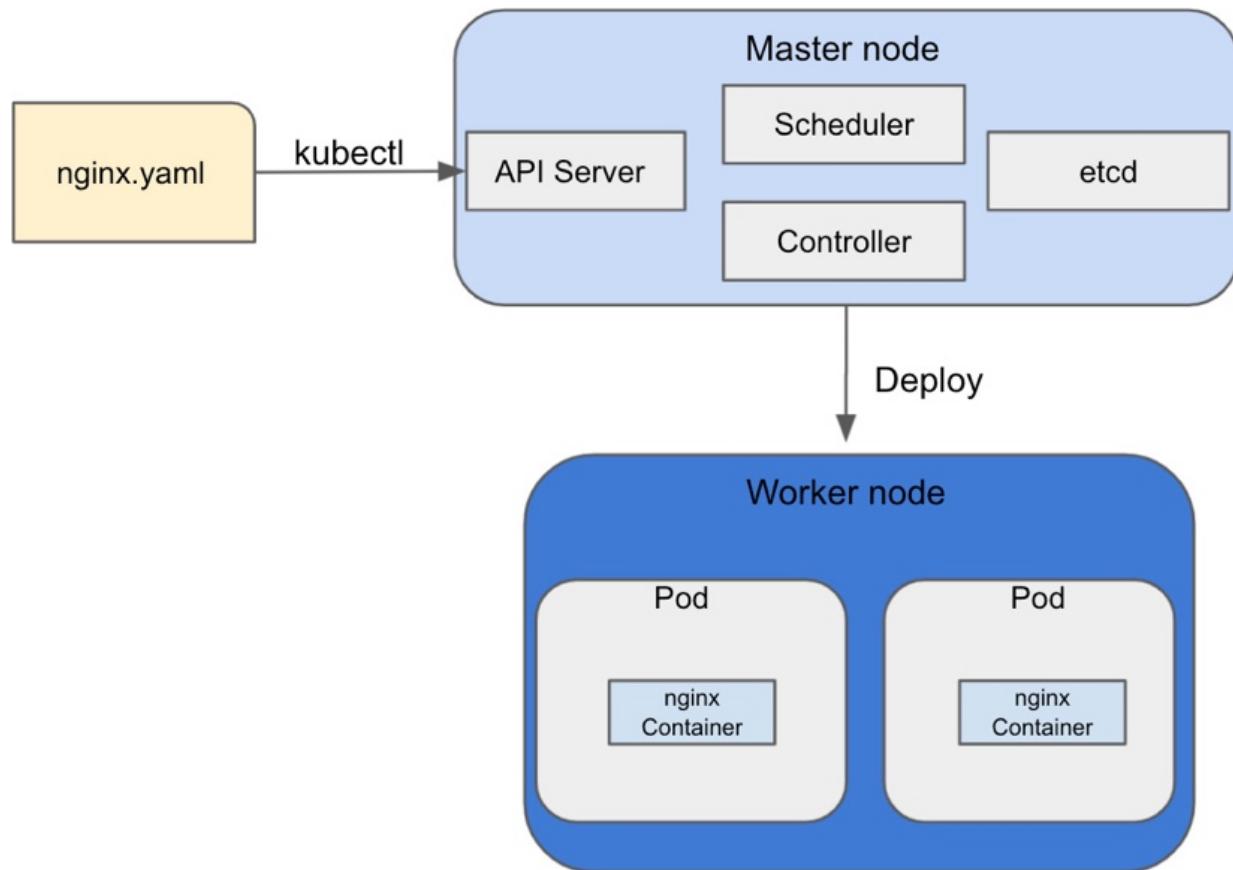


Figure 6.4: Creating an Nginx deployment

After the deployment, a Deployment controller monitors the deployed container instances. If an instance goes down, the controller will replace it with another instance on the worker node.

Kubernetes Job

A Kubernetes Job is a controller that creates one or more Pods to run some tasks, and ensures the job is successfully completed. If a number of Pods fail due to node failure or other system issues, a Kubernetes Job will recreate the Pods to complete the task. A Kubernetes Job can be used to run batch-oriented tasks, such as running batch data processing scripts, ML model training scripts, or ML batch inference scripts on a large number of inference requests. After a job is completed, the Pods are not terminated, so you can access the job logs and inspect the detailed status of the job. The following is an example template for running a training job:

```
apiVersion: batch/v1
kind: Job # indicate that this is the Kubernetes Job resource
metadata:
  name: train-job
spec:
  template:
    spec:
      containers:
        - name: train-container
          imagePullPolicy: Always # tell the job to always pulls a new container image when it is created
          image: <uri to Docker image containing training script>
          command: ["python3", "train.py"] # tell the container to run this command after it is created
          restartPolicy: Never
    backoffLimit: 0
```

Kubernetes custom resources (CRs) and operators

Kubernetes provides a list of built-in resources, such as Pods or deployment for different needs. It also allows you to create CRs and manage them just like the built-in resources, and you can use the same tools (such as **kubectl**) to manage them. When you create the **custom resource (CR)** in Kubernetes, Kubernetes creates a new API (for example, <custom resource name>/<version>) for each version of the resource. This is also known as *extending* the Kubernetes APIs. To create a CR, you create a **custom resource definition (CRD)** YAML file. To register the CRD in Kubernetes, you simply run `kubectl apply -f <name of the CRD yaml file>` to apply the file. And after that, you can use it just like any other Kubernetes resource. For example, to manage a custom model training job on Kubernetes, you can define a CRD with specifications such as algorithm name, data encryption setting, training image, input data sources, number of job failure retries, number of replicas, and job liveness probe frequency. A Kubernetes operator is a controller that operates on a custom resource. The operator watches the CR types and takes specific actions to make the current state match the desired state, just like what a built-in controller does. For example, if you want to create a training job for the training job CRD mentioned previously, you create an operator that monitors training job requests and performs application-specific actions to start up the Pods and run the training job throughout the life cycle. The following figure (*Figure 6.5*) shows the components involved with an operator deployment:

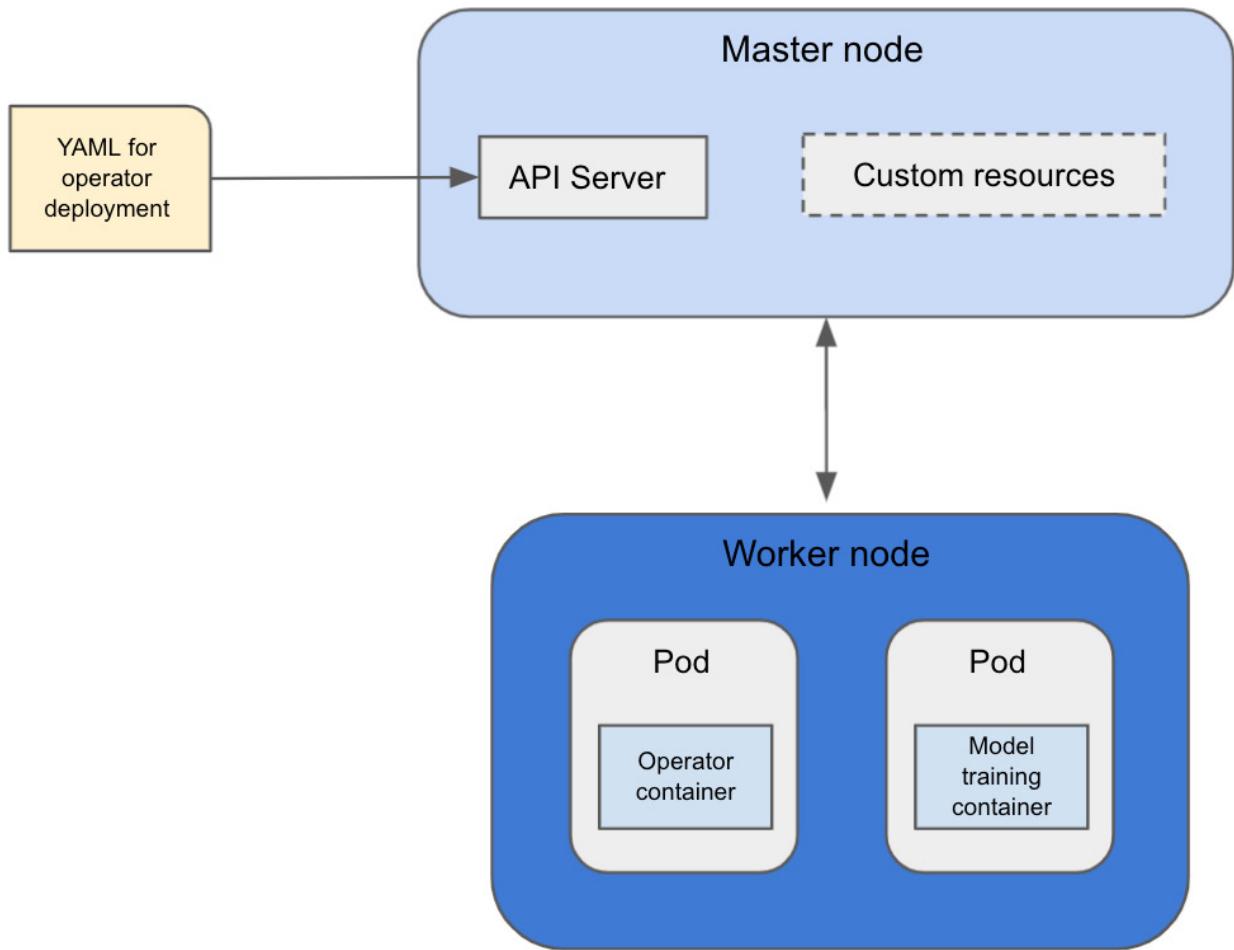


Figure 6.5: A Kubernetes custom resource and its interaction with the operator

The most common way to deploy an operator is to deploy a CR definition and the associated controller. The controller runs outside of the Kubernetes control plane, similar to running a containerized application in a Pod.

Services

Kubernetes Services play a crucial role in enabling reliable and scalable communication between various components and applications within a Kubernetes cluster. As applications in a cluster are often dynamic and can be scaled up or down, Services provide a stable and abstracted endpoint that other components can use to access the running instances of those applications. At its core, a Kubernetes Service is an abstraction layer that exposes a set of pods as a single, well-defined network endpoint. It acts as a load balancer, distributing incoming network traffic to the available pods behind the Service. This abstraction allows applications to interact with the Service without needing to know the specific details of the underlying pods or their IP addresses.

Networking on Kubernetes

Kubernetes operates a flat private network among all the resources in a Kubernetes cluster. Within a cluster, all Pods can communicate with each other cluster-wide without an **network address translation (NAT)**. Kubernetes gives each Pod its own cluster private IP address, and the IP is the same IP seen by the Pod itself and what others see it as. All containers inside a single Pod can reach each container's port on the localhost. All nodes in a cluster have their individually assigned IPs as well and can communicate with all Pods without an NAT. The following

figure (Figure 6.6) shows the different IP assignments for Pods and nodes, and communication flows from different resources:

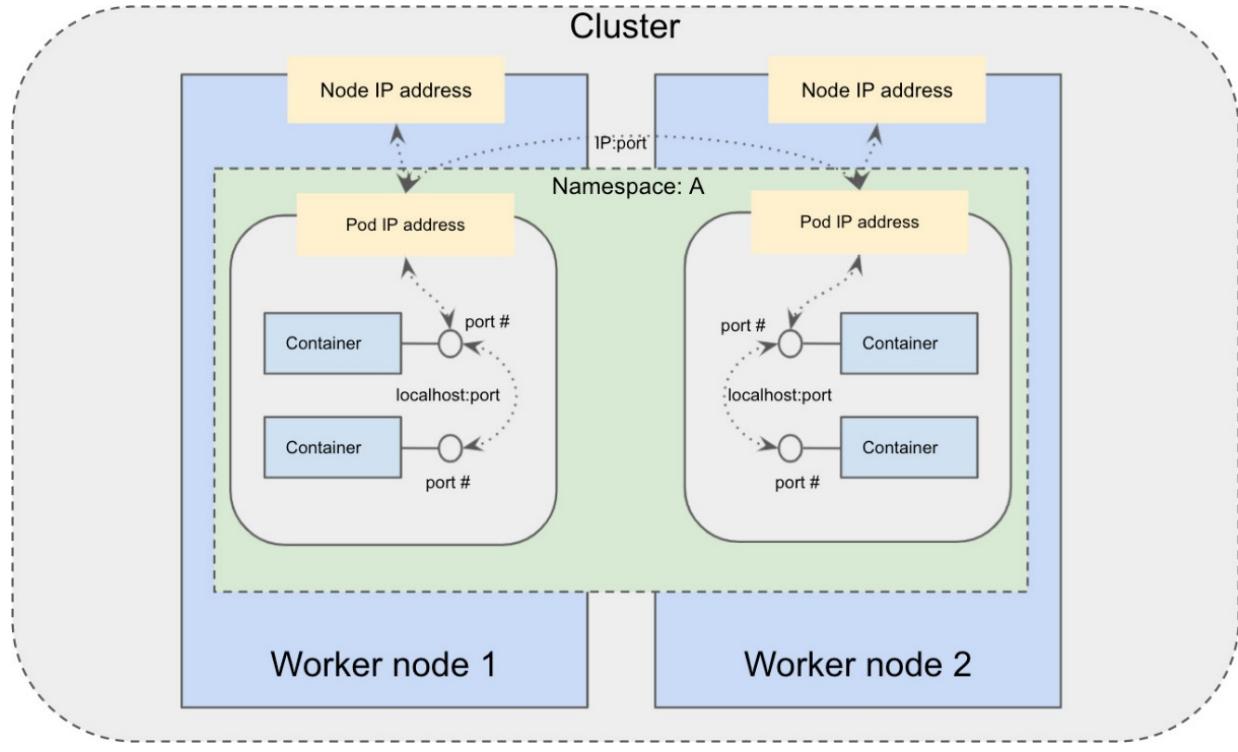


Figure 6.6: IP assignments and communication flow

Sometimes you might need a set of Pods running the same application container (the container for an **Nginx** application) for high availability and load balancing, for example. Instead of calling each Pod by its private IP separately to access the application running in a Pod, you want to call an abstraction layer for this set of Pods, and this abstraction layer can dynamically send traffic to each Pod behind it. In this case, you can create a Kubernetes Service as an abstraction layer for a logical set of Pods. A Kubernetes Service can dynamically select the Pod behind it by matching an **app** label for the Pod using a Kubernetes feature called **selector**. The following example shows the specification that would create a service called **nginx-service** that sends traffic to Pods with the **app nginx** label on port 9376. A service is also assigned with its own cluster private IP address, so it is reachable by other resources inside a cluster:

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

In addition to using **selector** to automatically detect Pods behind the service, you can also manually create an **Endpoint** and map a fixed IP and port to a service, as shown in the following example:

```
apiVersion: v1
kind: Endpoints
metadata:
```

```

name: nginx-service
subsets:
- addresses:
  - ip: 192.0.2.42
  ports:
    - port: 9376

```

While nodes, Pods, and services are all assigned with cluster private IPs, these IPs are not routable from outside of a cluster. To access Pods or services from outside of a cluster, you have the following options:

- **Access from a node or Pod:** You can connect to the shell of a running Pod using the `kubectl exec` command, and access other Pods, nodes, and services from the shell.
- **Kubernetes Proxy:** You can start the Kubernetes Proxy to access services by running the `kubectl proxy --port=<port number>` command on your local machine. Once the proxy is running, you can access nodes, Pods, or services. For example, you can access a service using the following scheme:
- `http://localhost:<port number>/api/v1/proxy/namespaces/<NAMESPACE>/services/<SERVICE NAME>:<PORT>`
- **NodePort:** **NodePort** opens a specific port on all the worker nodes, and any traffic sent to this port on the IP address of any of the nodes is forwarded to the service behind the port. The nodes' IPs need to be routable from external sources. The following figure (*Figure 6.7*) shows the communication flow using **NodePort**:

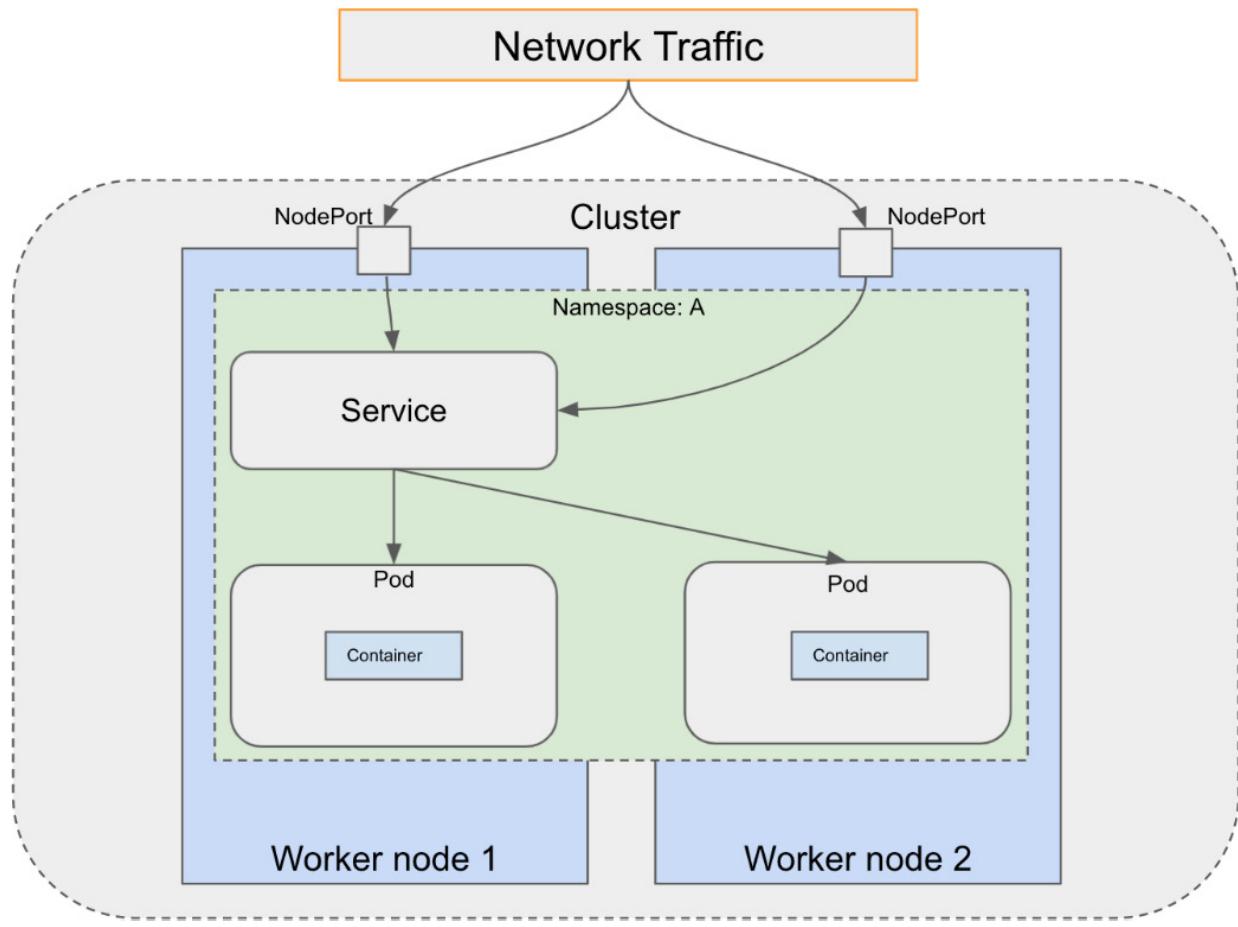


Figure 6.7: Accessing Kubernetes Service via NodePort

NodePort is simple to use, but it has some limitations, such as one service per NodePort, a fixed port range to use (3000 to 32767), and you need to know the IPs of individual worker nodes.

- **Load balancer:** A load balancer is a standard way to expose services to the internet. With a load balancer, you get a public IP address that's accessible to the internet, and all traffic sent to the IP address will be forwarded to the service behind the load balancer. A load balancer is not part of Kubernetes and it is provided by whatever cloud infrastructure a Kubernetes cluster resides on (for example, AWS). The following figure (*Figure 6.8*) shows the communication flow from a load balancer to services and Pods:

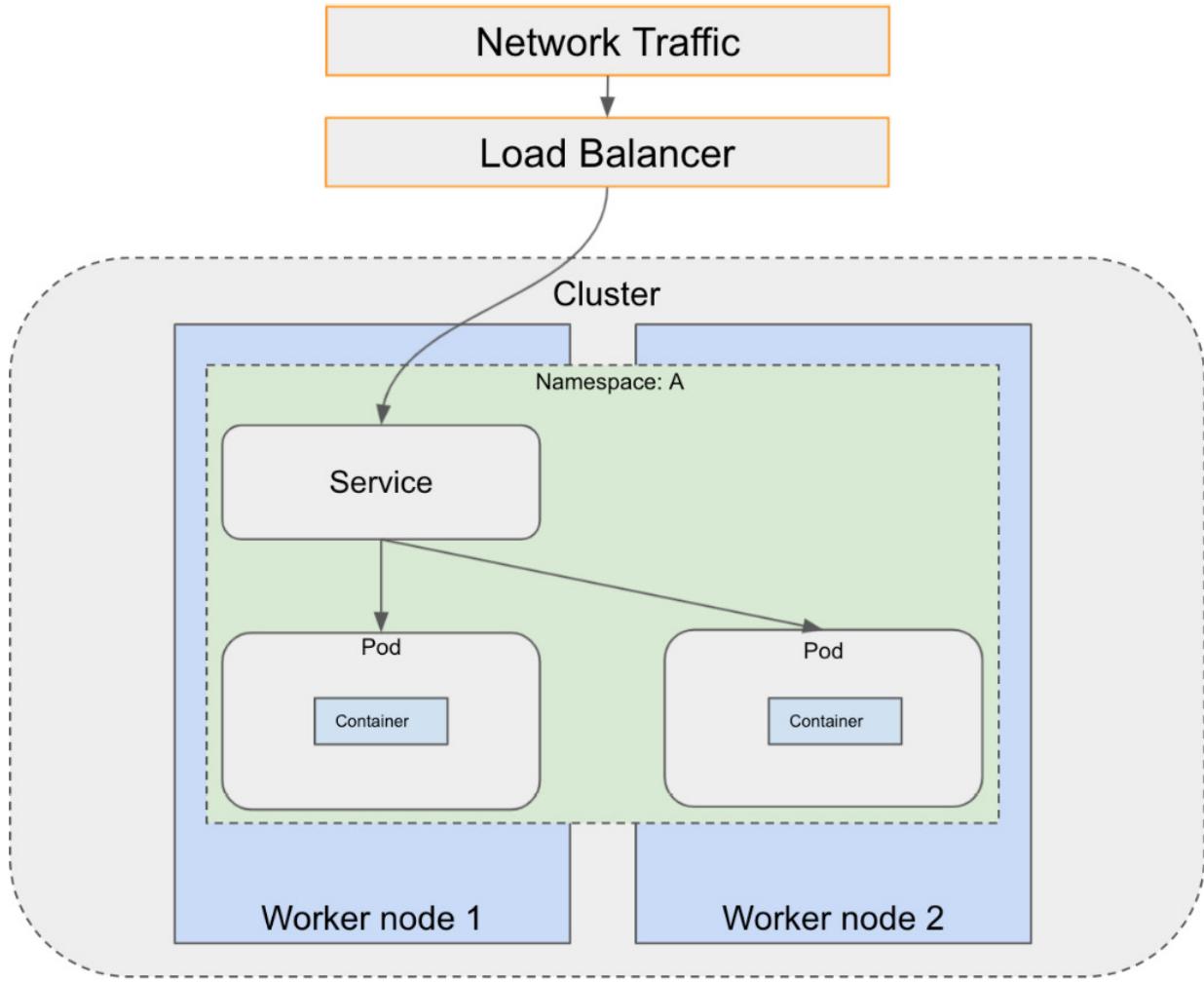


Figure 6.8: Accessing a Kubernetes service via a load balancer

A load balancer allows you to choose the exact port to use and can support multiple ports per service. However, it does require a separate load balancer per service.

- **Ingress:** An Ingress gateway is the entry point to a cluster. It acts as a load balancer and routes incoming traffic to the different services based on routing rules.

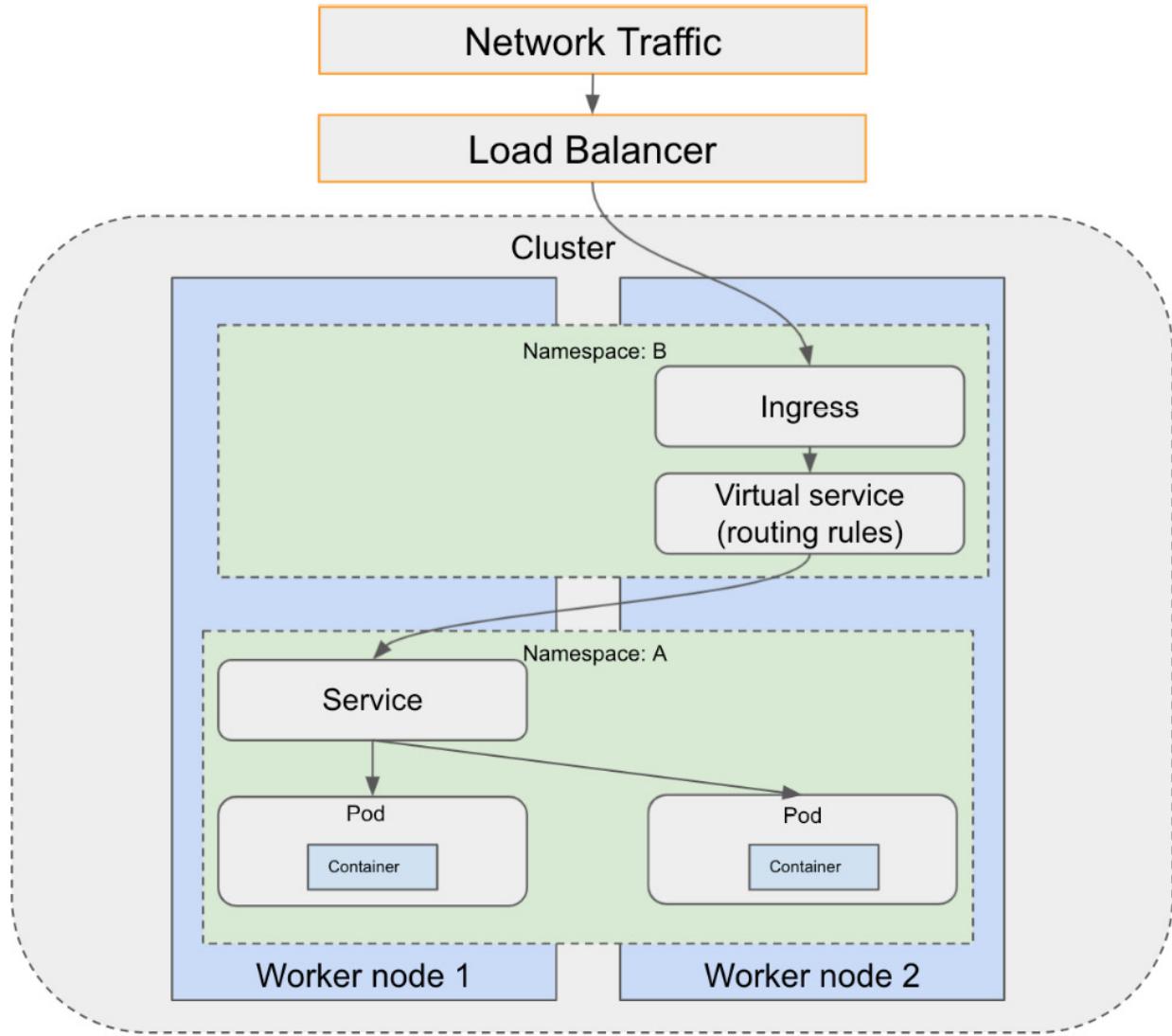


Figure 6.9: Accessing a Kubernetes service via Ingress

An Ingress is different from a load balancer and `NodePort` in that it acts as a proxy to manage traffic to clusters. It works with the `NodePort` and load balancer and routes the traffic to the different services. The Ingress way is becoming more commonly used, especially in combination with a load balancer.

Service mesh

In addition to network traffic management from outside of the cluster, another important aspect of Kubernetes network management is to control the traffic flow between different Pods and services within a cluster. For example, you might want to allow certain traffic to access a Pod or service while denying traffic from other sources. This is especially important for applications built on microservices architecture, as there could be many services or Pods that need to work together. Such a network of microservices is also called a **service mesh**. As the number of services grows large, it becomes challenging to understand and manage the networking requirements, such as *service discovery*, *network routing*, *network metrics*, and *failure recovery*. **Istio** is an open source service mesh management software that makes it easy to manage a large service mesh on Kubernetes, and it provides the following core functions:

- **Ingress:** Istio provides an Ingress gateway that can be used to expose Pods and services inside a service mesh to the internet. It acts as a load balancer that manages the inbound and outbound traffic for the service mesh. A gateway only allows traffic to come in/out of a mesh – it does not do routing of the traffic. To route traffic from the gateway to service inside the service mesh, you create an object called **Virtual Service** to provide routing rules to route incoming traffic to different destinations inside a cluster, and you create a binding between virtual services and the gateway object to connect the two together.
- **Network traffic management:** Istio provides easy rule-based network routing to control the flow of traffic and API calls between different services. When Istio is installed, it automatically detects services and endpoints in a cluster. Istio uses an object called **Virtual Service** to provide routing rules to route incoming traffic to different destinations inside a cluster. Istio uses a load balancer called **gateway** to manage the inbound and outbound traffic for the network mesh. The **gateway** load balancer only allows traffic to come in/out of a mesh – it does not do routing of the traffic. To route traffic from the gateway, you create a binding between virtual services and the **gateway** object.

In order to manage the traffic in and out of a Pod, an Envoy proxy component (aka **sidecar**) is injected into a Pod, and it intercepts and decides how to route all traffic. The Istio component that manages the traffic configurations of the sidecars and service discovery is called the **Pilot**. The **Citadel** component manages authentication for service to service and end user. The **Gallery** component is responsible for insulating other Istio components from the underlying Kubernetes infrastructure. The following figure shows the architecture of Istio on Kubernetes:

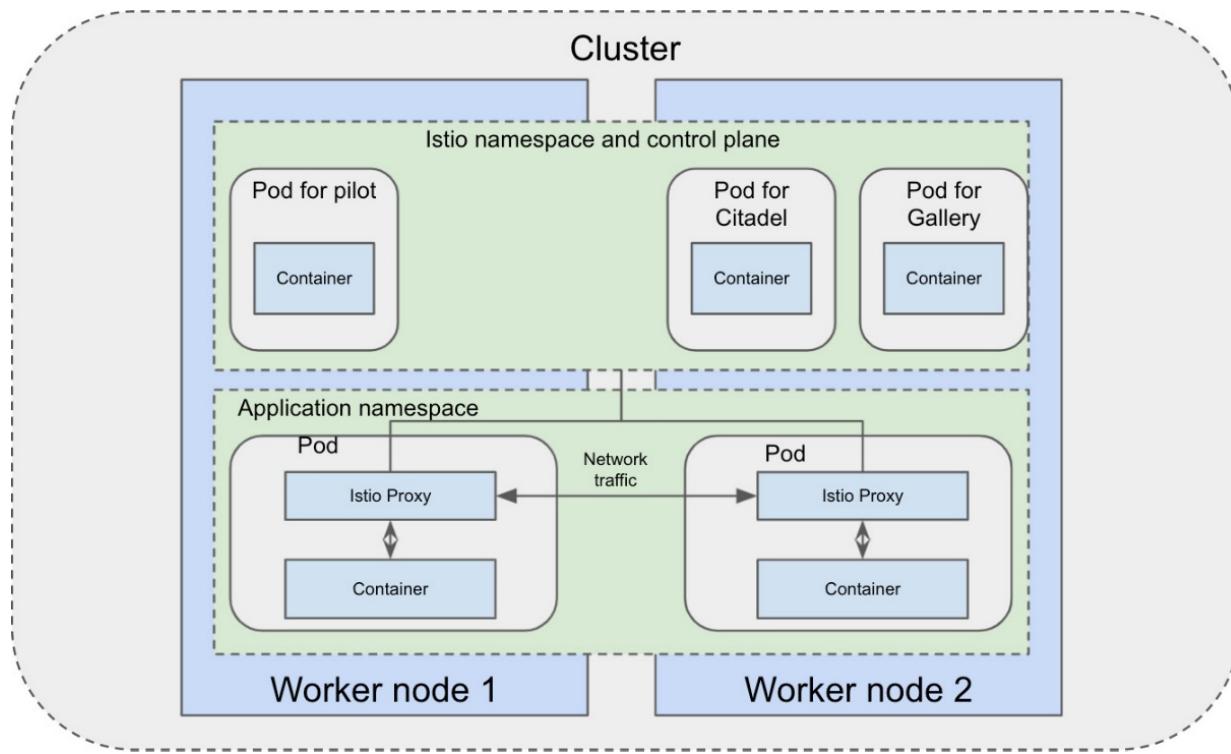


Figure 6.10: Istio architecture

- **Security:** Istio provides authentication and authorization for inter-service communications.
- **Observability:** Istio captures metrics, logs, and traces of all service communications within a cluster. Examples of metrics include network latency, errors, and saturation. Examples of traces include call flows and service dependencies within a mesh.

Istio can handle a wide range of deployment needs, such as load balancing and service-to-service authentication. It can even extend to other clusters.

Security and access management

Kubernetes has many built-in security features. These security features allow you to implement fine-grained network traffic control and access control to different Kubernetes APIs and services. In this section, we will discuss network security, authentication, and authorization.

Network security

By default, Kubernetes allows all Pods in a cluster to communicate with each other. To prevent unintended network traffic among different Pods, **network policies** can be established to specify how Pods can communicate with each other. You can think of a network policy as a network firewall that contains a list of allowed connections. Each network policy has a `podSelector` field, which selects a group of Pods enforced by the network policy and the allowed traffic direction (ingress or egress). The following sample policy denies all ingress traffic to all Pods, as there are no specific ingress policies defined:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny-ingress
spec:
  podSelector: {}
  policyTypes:
    - Ingress
```

Network traffic is allowed if there is at least one policy that allows it.

Authentication and authorization to APIs

Access to Kubernetes APIs can be authenticated and authorized for both users and Kubernetes **service accounts** (a service account provides an identity for processes running in a Pod). Users are handled outside of Kubernetes, and there are a number of user authentication strategies for Kubernetes:

- **X.509 client certificate:** A signed certificate is sent to the API server for authentication. The API server verifies this with the certificate authority to validate the user.
- **Single sign-on with OpenID Connect (OIDC):** The user authenticates with the OIDC provider and receives a bearer token (**JSON Web Token (JWT)**) that contains information about the user. The user passes the bearer token to the API server, which verifies the validity of the token by checking the certificate in the token.
- **HTTP basic authentication:** HTTP basic authentication requires a user ID and password to be sent as part of the API request, and it validates the user ID and password against a password file associated with the API server.
- **Authentication proxy:** The API server extracts the user identity in the HTTP header and verifies the user with the certificate authority.
- **Authentication webhook:** An external service is used for handling the authentication for the API server.

Service accounts are used to provide identity for processes running in a Pod. They are created and managed in Kubernetes. Service accounts need to reside within a namespace, by default. There is also a *default* service account in each namespace. If a Pod is not assigned a service account, the default service account will be assigned to the Pod. A service account has an associated authentication token, saved as a Kubernetes Secret, and used for API authentication. A Kubernetes Secret is used for storing sensitive information such as passwords, authentication tokens, and SSH keys. We will cover secrets in more detail later in this chapter. After a user or service account is authenticated, the request needs to be authorized to perform allowed operations. Kubernetes authorizes authenticated requests using the API server in the control plane, and it has several modes for authorization:

- **Attribute-based access control (ABAC):** Access rights are granted to users through policies. Note that every service account has a corresponding username. The following sample policy allows the **joe** user access to all APIs in all namespaces.

```
{
  "apiVersion": "abac.authorization.kubernetes.io/v1beta1",
  "kind": "Policy",
  "spec": {
    "user": "joe",
    "namespace": "*",
    "resource": "*",
    "apiGroup": "*"
  }
}
```

The following policy allows the `system:serviceaccount:kube-system:default` service account access to all APIs in all namespaces:

```
{
  "apiVersion": "abac.authorization.kubernetes.io/v1beta1",
  "kind": "Policy",
  "spec": {
    "user": "system:serviceaccount:kube-system:default",
    "namespace": "*",
    "resource": "*",
    "apiGroup": "*"
  }
}
```

- **Role-based access control (RBAC):** Access rights are granted based on the role of a user. RBAC authorizes using the `rbac.authorization.k8s.io` API group. The RBAC API works with four Kubernetes objects: `Role`, `ClusterRole`, `RoleBinding`, and `ClusterRoleBinding`.

`Role` and `ClusterRole` contain a set of permissions. The permissions are *additive*, meaning there are no deny permissions, and you need to explicitly add permission to resources. The `Role` object is namespaced and is used to specify permissions within a namespace. The `ClusterRole` object is non-namespaced, but can be used for granting permission for a given namespace or cluster-scoped permissions. The following `yaml` file provides get, watch, and list access to all `Pods` resources in the default namespace for the core API group:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: pod-reader
rules:
- apiGroups: []
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```

The following policy allows get, watch, and list access for all Kubernetes nodes across the cluster:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: nodes-reader
rules:
- apiGroups: []
  resources: ["nodes"]
  verbs: ["get", "watch", "list"]
```

`RoleBinding` and `ClusterRoleBinding` grant permissions defined in a `Role` or `ClusterRole` object to a user or set of users with reference to a `Role` or `ClusterRole` object. The following policy binds the `joe` user to the `pod-reader` role:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
```

```

name: read-pods
namespace: default
subjects:
- kind: User
  name: joe
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io

```

The following `RoleBinding` object binds a service account, `SA-name`, to the `ClusterRole` `nodes-reader`:

```

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: read-secrets-global
subjects:
- kind: ServiceAccount
  name: SA-name
  namespace: default
roleRef:
  kind: ClusterRole
  name: secret-reader
  apiGroup: rbac.authorization.k8s.io

```

Kubernetes has a built-in feature for storing and managing sensitive information such as passwords. Instead of storing this sensitive information directly in plain text in a Pod, you can store this information as Kubernetes Secrets, and provide specific access to them using Kubernetes RBAC to create and/or read these secrets. By default, secrets are stored as unencrypted plain-text Base64-encoded strings, and data encryption at rest can be enabled for the secrets. The following policy shows how to create a secret for storing AWS access credentials:

```

apiVersion: v1
kind: Secret
metadata:
  name: aws-secret
type: Opaque
data:
  AWS_ACCESS_KEY_ID: XXXX
  AWS_SECRET_ACCESS_KEY: XXXX

```

There are several ways to use a secret in a Pod:

- As environment variables in the Pod specification template:

```

apiVersion: v1
kind: Pod
metadata:
  name: secret-env-pod
spec:
  containers:
  - name: mycontainer
    image: redis
    env:
    - name: SECRET_AWS_ACCESS_KEY
      valueFrom:
        secretKeyRef:
          name: aws-secret
          key: AWS_ACCESS_KEY_ID
    - name: SECRET_AWS_SECRET_ACCESS_KEY
      valueFrom:
        secretKeyRef:
          name: aws-secret
          key: AWS_SECRET_ACCESS_KEY
  restartPolicy: Never

```

The application code inside the container can access the secrets just like other environment variables.

- As a file in a volume mounted on a Pod:

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-ml
spec:
  containers:
    - name: pod-ml
      image: <Docker image uri>
      volumeMounts:
        - name: vol-ml
          mountPath: "/etc/aws"
          readOnly: true
  volumes:
    - name: vol-ml
      Secret:
        secretName: aws-secret
```

In the previous examples, you will see files in the `/etc/aws` folder for each corresponding secret name (such as `SECRET_AWS_ACCESS_KEY`) that contains the values for the secrets.

Running ML workloads on Kubernetes

We now know what containers are and how they can be deployed on a Kubernetes cluster. We also understand how to configure networking on Kubernetes to allow Pods to communicate with each other and how to expose a Kubernetes container for external access outside of the cluster using different networking options. Kubernetes can serve as the foundational infrastructure for running ML workloads. For example, you can run a **Jupyter Notebook** as a containerized application on Kubernetes as your data science environment for experimentation and model building. You can also run a model training job as a Kubernetes Job if you need additional resources, and then serve the model as a containerized web service application or run batch inferences on trained models as a Kubernetes Job. In the following hands-on exercise, you will learn how to use Kubernetes as the foundational infrastructure for running ML workloads.

Hands-on – creating a Kubernetes infrastructure on AWS

In this section, you will create a Kubernetes environment using the Amazon EKS. Let's first look at the problem statement in the following section.

Problem statement

As a ML solutions architect, you have been tasked to evaluate Kubernetes as a potential infrastructure platform for building an ML platform for one business unit in your bank. You need to build a sandbox environment on AWS and demonstrate that you can deploy a Jupyter Notebook as a containerized application for your data scientists to use.

Lab instruction

In this hands-on exercise, you are going to create a Kubernetes environment using the Amazon EKS. The EKS is a managed service for Kubernetes on AWS that creates and configures a Kubernetes cluster with both master and worker nodes automatically. The EKS provisions and scales the control plane, including the API server and backend persistent layer. The EKS runs the open source Kubernetes and is compatible with all Kubernetes-based applications. After the EKS cluster is created, you will explore the EKS environment to inspect some of its core components, and then you will learn to deploy a containerized Jupyter Notebook application and make it accessible from the internet. Let's complete the following steps to get started:

1. Launch the AWS CloudShell service.

Log on to your AWS account, select the **Oregon** region, and launch the AWS CloudShell. CloudShell is an AWS service that provides a browser-based **Linux** terminal environment to interact with AWS resources. With CloudShell, you authenticate using your AWS console credential and can easily run **AWS CLI**, **AWS SDK**, and other tools.

1. Install the **eksctl** utility.

Follow the installation instructions for Unix at

<https://github.com/weaveworks/eksctl/blob/main/README.md#installation> to install eksctl. The **eksctl** utility is a command-line utility for managing the EKS cluster. We will use the **eksctl** utility to create a Kubernetes cluster on Amazon EKS in *Step 3*:

1. Run the following command to start creating an EKS cluster in the **Oregon** region inside your AWS account. It will take about 15 minutes to complete running the setup:

```
eksctl create cluster --name <cluster name> --region us-west-2
```

The command will launch a `CloudFormation` template and this will create the following resources:

1. An Amazon EKS cluster with two worker nodes inside a new Amazon **virtual private cloud (VPC)**. Amazon EKS provides fully managed Kubernetes master nodes, so you won't see the master nodes inside your private VPC.
2. An EKS cluster configuration file saved in the `/home/cloudshell-user/.kube/config` directory on CloudShell. The `config` file contains details such as the API server `url` address, the name of the admin user for managing the cluster, and the client certificate for authenticating to the Kubernetes cluster. The `kubectl` utility uses information in the `config` file to connect and authenticate to the Kubernetes API server.
3. EKS organizes worker nodes into logical groups called **nodegroup**. Run the following command to look up the **nodegroup** name. You can look up the name of the cluster in the EKS management console. The name of the node group should look something like `ng-xxxxxxxx`.

```
eksctl get nodegroup --cluster=<cluster name>
```

1. Install the **kubectl** utility.

Following the instructions at <https://docs.aws.amazon.com/eks/latest/userguide/install-kubectl.xhtml> to install kubectl for Linux. You will need to know the version of the Kubernetes server to install the matching kubectl version. You can find the version of Kubernetes server using the `kubectl version --short` command.

1. Explore the cluster.

Now the cluster is up, let's explore it a bit. Try running the following commands in the CloudShell terminal and see what is returned:

Items to explore	kubectl command syntax
Get cluster info.	kubectl cluster-info
List all API resources.	kubectl api-resources
List available namespaces.	kubectl get namespaces
List worker nodes.	kubectl get nodes
List Pods in all namespaces.	kubectl get pods -A
List services in all namespaces.	kubectl get services -A
List all clusterroles.	kubectl get clusterroles
List all roles in all namespaces.	kubectl get roles -A
List all service accounts in all namespaces.	kubectl get sa -A
Describe admin clusterrole.	kubectl describe clusterrole admin
List all secrets.	kubectl get secret -A
List all deployments.	kubectl get deployments -A
List networkpolicies.	kubectl get networkpolicies -A

Table 6.1: kubectl commands

1. Deploy a Jupyter Notebook.

Let's deploy a Jupyter Notebook server as a containerized application. Copy and run the following code block. It should create a file called `deploy_Jupyter_notebook.yaml`. We will use a container image from the Docker Hub image repository:

```
cat << EOF > deploy_Jupyter_notebook.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: jupyter-notebook
  labels:
    app: jupyter-notebook
spec:
  replicas: 1
  selector:
    matchLabels:
      app: jupyter-notebook
  template:
    metadata:
      labels:
        app: jupyter-notebook
    spec:
      containers:
        - name: minimal-notebook
          image: jupyter/minimal-notebook:latest
          ports:
            - containerPort: 8888
EOF
```

Now, let's create a deployment by running the following:

```
kubectl apply -f deploy_Jupyter_notebook.yaml.
```

Check to make sure the Pod is running by running `kubectl get pods`. Check the logs of the Jupyter server Pod by running `kubectl logs <name of notebook pod>`. Find the section in the logs that contains `http://jupyter-notebook-598f56bf4b-spqn4:8888/?token=xxxxxx...`, and copy the token (`xxxxxx...`) portion. We will use the token for *Step 8*. You can also access the pod using an interactive shell by running `kubectl exec --stdin --tty <name of notebook pod> -- /bin/sh`. Run `ps aux` to see a list of running processes. You will see a process related to the Jupyter Notebook.

1. Expose the Jupyter Notebook to the internet.

At this point we have a Jupyter server running in a Docker container in a Kubernetes Pod on top of two EC2 instances in an AWS VPC but we can't get to it because the Kubernetes cluster doesn't expose a route to the container. We will create a Kubernetes service to expose the Jupyter Notebook server to the internet so it can be accessed from a browser. Run the following code block to create a specification file for a new Service. It should create a file called `jupyter_svc.yaml`:

```
cat << EOF > jupyter_svc.yaml
apiVersion: v1
kind: Service
metadata:
  name: jupyter-service
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-type: alb
spec:
  selector:
    app: jupyter-notebook
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8888
  type: LoadBalancer
EOF
```

After the file is created, run `kubectl apply -f jupyter_svc.yaml` to create the service. A new Kubernetes Service called `jupyter-service`, as well as a new `LoadBalancer` object should be created. You can verify the service by running `kubectl get service`. Note and copy the `EXTERNAL-IP` address associated with the `jupyter-service` service. Paste the `EXTERNAL-IP` address to a new browser window, and enter the token you copied earlier into the `Password or token` field (*Figure 6.11*) to log in. You should see a Jupyter Notebook window showing up:



Figure 6.11: Jupyter login screen

The following diagram shows the environment that you have created after working through the hands-on exercise.

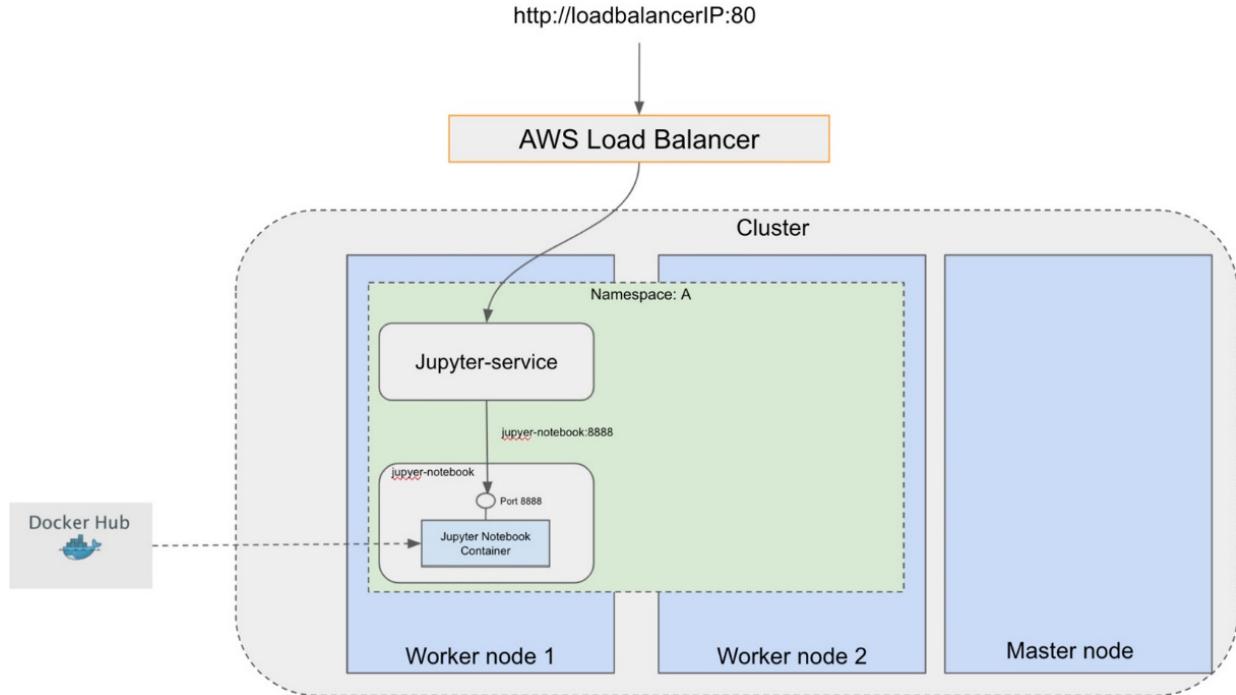


Figure 6.12: Jupyter notebook deployment on the EKS cluster

Congratulations, you have successfully created a new Amazon EKS cluster on AWS and deployed a Jupyter Server instance as a container on the cluster. We will re-use this EKS cluster for the next chapter. However, if you don't plan to use this EKS for a period of time, it is recommended to shut down the cluster to avoid unnecessary costs. To shut down the cluster, run `kubectl delete svc <service name>` to delete the service first. Then run `eksctl delete cluster --name <cluster name>` to delete the cluster.

Summary

In this chapter, we covered Kubernetes, a robust container management platform that forms the infrastructure foundation for constructing open-source ML platforms. Throughout this chapter, you gained an understanding of containers and insights into the functioning of Kubernetes. Moreover, you acquired hands-on experience in establishing a Kubernetes cluster on AWS by leveraging the **AWS Elastic Kubernetes Service (EKS)**. Additionally, you explored the process of deploying a containerized Jupyter Notebook application onto the cluster, thereby creating a fundamental data science environment. In the next chapter, we will shift our focus towards exploring a selection of open-source ML platforms that seamlessly integrate with the Kubernetes infrastructure.

7 Open Source Machine Learning Platforms

Join our book community on Discord

<https://packt.link/EarlyAccessCommunity>



In the previous chapter, we covered how **Kubernetes** can be used as the foundational infrastructure for running **machine learning (ML)** tasks, such as running model training jobs or building data science environments such as **Jupyter notebook** servers. However, to perform these tasks at scale and more efficiently for large organizations, you will need to build ML platforms with the capabilities to support the full data science life cycle. These capabilities include scalable data science environments, model training services, model registries, and model deployment capabilities. In this chapter, we will discuss the core components of an ML platform and explore additional open source technologies that can be used for building ML platforms. We will begin with technologies designed for building a data science environment capable of supporting a large number of users for experimentations. Subsequently, we will delve into various technologies for model training, model registries, model deployment, and ML pipeline automation.

Core components of an ML platform

An ML platform is a complex system, encompassing multiple environments for running distinct tasks and orchestrating complex workflow processes. Furthermore, an ML platform needs to cater a multitude of roles, including data scientists, ML engineers, infrastructure engineers, and operation teams, and security and compliance stakeholders. To construct an ML platform, several components come into play. These components include:

- **Data science environment:** The data science environment provides data analysis and ML tools, such as Jupyter notebooks, code repositories, and ML frameworks. Data scientists and ML engineers use the data science environment to perform data analysis, run data science experiments, and build and tune models. The data science environment also provide collaboration capabilities, allowing data scientists to share and collaborate on code, data, experiments, and models.
- **Model training environment:** The model training environment provides a separate infrastructure tailored to meet specific model training requirements. While data scientists and ML engineers can execute small-scale model training tasks directly within their local

Jupyter environment, they need a separate dedicated infrastructure for large-scale model training. By utilizing a dedicated training infrastructure, organizations can exercise greater control over model training process management and model lineage management processes.

- **Model registry:** Trained models need to be tracked and managed within a model registry. The model registry serves as a centralized repository for inventorying and managing models, ensuring effective lineage management, version control, model discovery, and comprehensive life cycle management. This becomes particularly significant when dealing with a large number of models. Data scientists can register models directly in the registry as they perform experiments in their data science environment. Additionally, models can be registered as part of automated ML model pipeline executions, enabling streamlined and automated integration of models into the registry.
- **Model serving environment:** To serve predictions from trained ML models to client applications, it is necessary to host the models within a dedicated model serving infrastructure that operates behind an API endpoint in real time. This infrastructure should also provide support for batch transform capabilities, allowing predictions to be processed in large batches. Several types of model serving frameworks are available to fulfill these requirements.
- **ML Pipeline development:** To effectively manage the various ML components and stages in the lifecycle, it is crucial to incorporate capabilities that enable pipeline development to orchestrate ML training and prediction workflows. These pipelines play an important role in coordinating different stages such as data preparation, model training, and evaluation.
- **Continuous integration (CI)/continuous deployment (CD) and workflow automation:** Finally, to streamline the data processing, model training, and model deployment processes in an ML platform, it is crucial to establish continuous integration and continuous deployment (CI/CD) practices, along with workflow automation capabilities. These practices and tools significantly contribute to increasing the velocity, consistency, reproducibility, and observability of ML deployments.

Apart from these core components mentioned earlier, there are several other platform architecture factors to consider when building an end-to-end ML platform. These factors include security and authentication, logging and monitoring, version control and reproducibility, data management, and governance. By integrating these additional architectural factors into the ML platform, organizations can enhance security, gain visibility into system operations, and enforce governance policies. In the following sections, we will explore various open source technologies that can be used to build an end-to-end ML platform.

Open source technologies for building ML platforms

Managing ML tasks individually by deploying standalone ML containers in a Kubernetes cluster can become challenging when dealing with a large number of users and workloads. To address this complexity and enable efficient scaling, many open source technologies have emerged as viable solutions. These technologies, including Kubeflow, MLflow, Seldon Core, GitHub, Feast, and Airflow, provide comprehensive support for building data science environments, model training services, model inference services, and ML workflow automation. In the following

sections, we will explore some well-known open source technologies that can be utilized for building ML platforms.

Implementing a data science environment

Kubeflow is an open source machine learning platform built on top of Kubernetes. It offers a set of tools and frameworks specifically designed to simplify the deployment, orchestration, and management of ML workloads. Kubeflow provides features like Jupyter notebooks for interactive data exploration and experimentation, distributed training capabilities, and model serving infrastructure. Core capabilities of Kubeflow include:

- A central UI dashboard
- A Jupyter notebook server for code authoring and model building
- A Kubeflow pipeline for ML pipeline orchestration
- **KFServing** for model serving
- Training operators for model training support

The following figure (*Figure 7.1*) illustrates how Kubeflow can provide the various components needed for a data science environment. Specifically, we will delve into its support for Jupyter notebook servers.

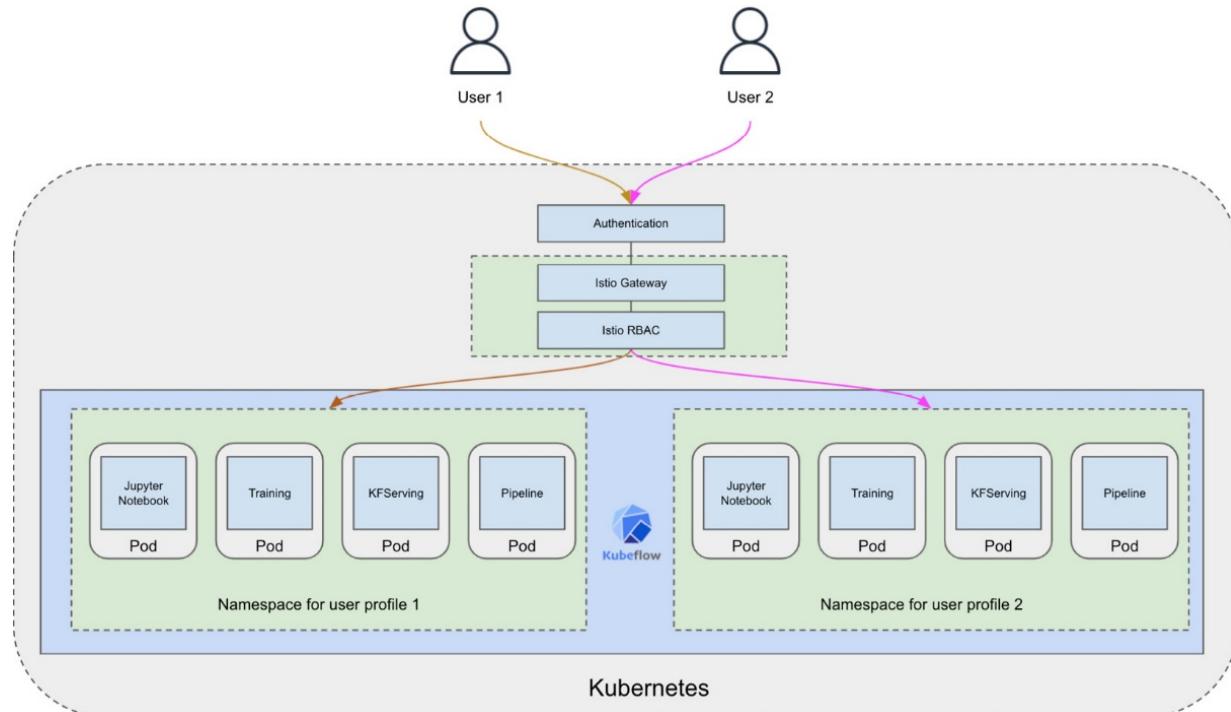


Figure 7.1: A Kubeflow-based data science environment

Kubeflow provides a multi-tenant Jupyter notebook server environment with built-in authentication and authorization support. Let's discuss each of these core components in detail:

- **Jupyter notebook:** As a data scientist, you can take advantage of the Kubeflow Jupyter notebook server, which offers a platform to author and run your **Python** code to explore data and build models inside the Jupyter notebook. With Kubeflow, you can spawn multiple notebook servers, each server associated with a single Kubernetes namespace that corresponds to a team, project, or individual user. Each notebook server runs a container inside a **Kubernetes Pod**. By default, a Kubeflow notebook server provides a list of notebook container images hosted in public container image repositories to choose from. Alternatively, you can create custom notebook container images to tailor to your specific requirements. To ensure standards and consistency, Kubeflow administrators can provide a list of standard images for users to use. When creating a notebook server, you select the namespace to run the notebook server in. Additionally, you specify the **Universal Resource Identifier (URI)** of the container image for the notebook server. You also have the flexibility to specify the resource requirements, such as the number of CPUs/GPUs and memory size.
- **Authentication and authorization:** You access the notebook server through the Kubeflow UI dashboard, which provides an authentication service through the **Dex Open ID Connection (OIDC)** provider. Dex is an identity service that uses OIDC to provide authentication for other applications. Dex can federate with other authentication services such as the **Active Directory** service. Each notebook is associated with a default Kubernetes service account (**default-editor**) that can be used for entitlement purposes (such as granting the notebook permission to access various resources in the Kubernetes cluster). Kubeflow uses **Istio role-based access control (RBAC)** to control in-cluster traffic. The following `YAML` file grants the **default-editor** service account (which is associated with the Kubeflow notebook) access to the Kubeflow pipeline service by attaching the **ml-pipeline-services** service role to it:

```
apiVersion: rbac.istio.io/v1alpha1
kind: ServiceRoleBinding
metadata:
  name: bind-ml-pipeline-nb-admin
  namespace: kubeflow
spec:
  roleRef:
    kind: ServiceRole
    name: ml-pipeline-services
  subjects:
  - properties:
      source.principal: cluster.local/ns/admin/sa/default-editor
```

- **Multi-tenancy:** Kubeflow offers the capability for multiple users to access a shared Kubeflow environment while ensuring resource isolation. This is achieved by creating individual namespaces for each user and leveraging Kubernetes RBAC (Role-Based Access Control) and Istio RBAC to manage access control for these namespaces and their associated resources. For collaborative work within teams, the owner of a namespace has the ability to grant access to other users directly from the Kubeflow dashboard UI. Using the "Manage Contributor" function, the namespace owner can specify which users are granted access to the namespace and its resources.

When onboarding a new Kubeflow user, you create a new user profile, which automatically generates a new namespace for the profile. The following YAML file, once applied using `kubectl`, creates a new user profile called `test-user` with an email of `test-user@kubeflow.org`, and it also creates a new namespace called `test-user`:

```
apiVersion: kubeflow.org/v1beta1
kind: Profile
metadata:
  name: test-user
spec:
  owner:
    kind: User
    name: test-user@kubeflow.org
```

You can run the `kubectl get profiles` and `kubectl get namespaces` commands to verify that the profile and namespaces have been created. After a user is created and added to the Kubeflow Dex authentication service, the new user can log in to the Kubeflow dashboard and access the Kubeflow resources (such as a Jupyter notebook server) under the newly created namespace. Now we reviewed how Kubeflow can be used to provide a multi-tenant Jupyter notebook environment for experimentation and model building. Next, let's see how to build a model training environment.

Building a model training environment

As discussed earlier, within an ML platform, it is common to provide a dedicated model training service and infrastructure to support large-scale and automated model training in an ML pipeline. This dedicated training service should be easily accessible from different components within the platform, such as the experimentation environment (such as a Jupyter notebook) as well as the ML automation pipeline. In a Kubernetes-based environment, there are two main approaches for model training:

- Model training using **Kubernetes Jobs**
- Model training using **Kubeflow training operators**

You can choose which approach to use depending on your training needs. Now, let's take a closer look at each one of them in detail:

- **Model training using Kubernetes Jobs:** As we discussed in *Chapter 6, Kubernetes Container Orchestration Infrastructure Management*, a Kubernetes Job creates one or more containers and runs them through to completion. This pattern is well suited for running certain types of ML model training jobs, as an ML job runs a training loop to completion and does not run forever. For example, you can package a container with a Python training script and all the dependencies that train a model and use the Kubernetes Job to load the container and kick off the training script. When the script completes and exits, the Kubernetes job also ends. The following sample YAML file kicks off a model training job if submitted with the `kubectl apply` command:

```

apiVersion: batch/v1
kind: Job
metadata:
  name: train-churn-job
spec:
  template:
    spec:
      containers:
        - name: train-container
          imagePullPolicy: Always
          image: <model training uri>
          command: ["python", "train.py"]
          restartPolicy: Never
  backoffLimit: 4

```

To query the status of the job and see the detailed training logs, you can run the `kubectl get jobs` command and the `kubectl logs <pod name>` command, respectively.

- **Model training using Kubeflow training operators:** A Kubernetes Job can launch a model training container and run a training script inside the container to completion. Since the controller for a Kubernetes Job does not have application-specific knowledge about the training job, it can only handle generic Pod deployment and management for the running jobs, such as running the container in a Pod, monitoring the Pod, and handling generic Pod failure. However, some model training jobs, such as distributed training job in a cluster, require the special deployment, monitoring, and maintenance of stateful communications among various Pods. This is where the Kubernetes training operator pattern can be applied.

Kubeflow offers a list of pre-built training operators (such as the **TensorFlow**, **Pytorch**, and **XGBoost** operators) for complex model training jobs. Each Kubeflow training operator has a **custom resource (CR)** (for example, **TFJob CR** for TensorFlow jobs) that defines the training job's specific configurations, such as the type of Pod in the training job (for example, **master**, **worker**, or **parameter server**), or runs policies on how to clean up resources and how long a job should run. The controller for the CR is responsible for configuring the training environment, monitoring the training job's specific status, and maintaining the desired training job's specific state. For instance, the controller can set environment variables to make the training cluster specifications (for example, types of Pods and indices) available to the training code running inside the containers. Additionally, the controller can inspect the exit code of a training process and fail the training job if the exit code indicates a permanent failure. The following YAML file sample template represents a specification for running training jobs using the TensorFlow operator (`tf-operator`):

```

apiVersion: "kubeflow.org/v1"
kind: "TFJob"
metadata:
  name: "distributed-tensorflow-job"
spec:
  tfReplicaSpecs:
    PS:
      replicas: 1
      restartPolicy: Never
      template:

```

```

spec:
  containers:
    - name: tensorflow
      image: <model training image uri>
      command:
  worker:
    replicas: 2
    restartPolicy: Never
    template:
      spec:
        containers:
          - name: tensorflow
            image: <model training image uri>
            command:

```

In this example template, the specification will create one copy of the parameter servers (which aggregate model parameters across different containers) and two copies of the workers (which run model training loops and communicate with the parameter servers). The operator will process the `TFJob` object according to the specification, keep the `TFJob` object stored in the system with the actual running services and Pods, and replace the actual state with the desired state. You can submit the training job using `kubectl apply -f <TFJob specs template>` and can get the status of the `TFJob` with the `kubectl get tfjob` command. As a data scientist, you can submit Kubernetes training jobs or Kubeflow training jobs using the `kubectl` utility, or from your Jupyter notebook environment using the **Python (SDK)**. For example, `TFJob` object has a Python SDK called `kubeflow.tfjob`, and Kubernetes has a client SDK called `kubernetes.client` for interacting with the Kubernetes and Kubeflow environments from your Python code. You can also invoke training jobs using the `Kubeflow Pipeline` component, which we will cover later in the *Kubeflow pipeline* section.

Registering models with a model registry

A **model registry** is an important component in model management and governance, and it is a key link between the model training stage and the model deployment stage. There are several open source options for implementing a model registry in an ML platform. In this section, we will explore the MLflow model registry for model management.

The MLflow model registry

MLflow is a popular open source ML platform. It is designed for managing all stages of the ML life cycle, including experiment management, model management, reproducibility, and model deployment. It has the following four main components:

- **Experiment tracking:** Experiment tracking logs parameters, code versions, metrics, and artifacts when running your machine learning code and for later visualizing the results.
- **ML projects:** Projects package data science code in a format to reproduce runs on any platform.
- **Models:** Models provide a standard unit for packaging and deploying machine learning models.

- **Model registry:** Model Registry stores, annotates, discovers, and manages models in a central repository.

The model registry component of MLflow provides a central model repository for saved models. It captures model details such as model lineage, model version, annotation, and description, and also captures model stage transitions from staging to production (so the status of the model state is clearly described). To use the MLflow model registry in a team environment, you need to set up an MLflow tracking server with a database as a backend and storage for the model artifacts. MLflow provides a UI and an API to interact with its core functionality, including its model registry. Once the model is registered in the model registry, you can add, modify, update, transition, or delete the model through the UI or the API. The following figure shows an architecture setup for an MLflow tracking server and its associated model registry:

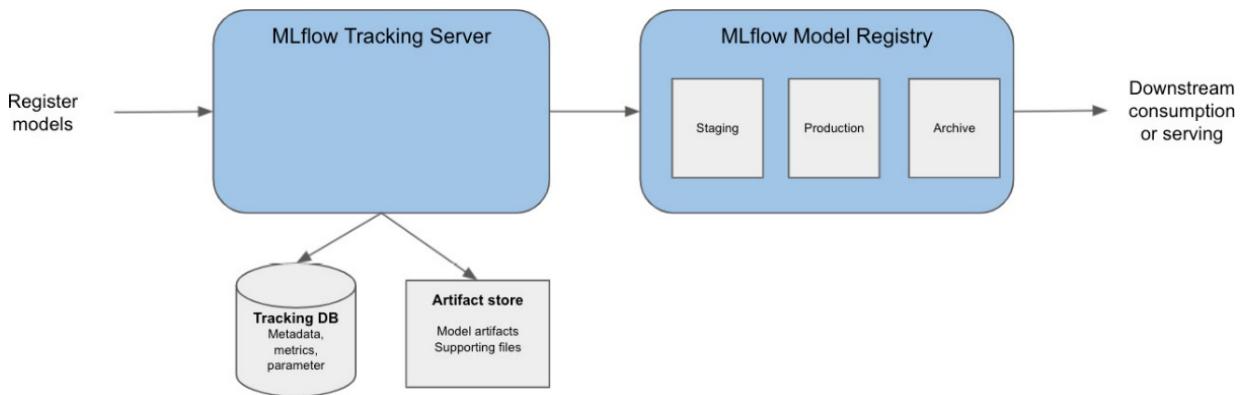


Figure 7.2: The MLflow tracking server and model registry

One limitation of the MLflow model registry is the absence of built-in user permission management. This means that any user with access to the tracking server can potentially access and manipulate all the models stored in the registry. To address this limitation, an external custom entitlement layer should be implemented. This layer enables the management of user-based access to specific resources within the model registry, ensuring that users only have access to the models they are authorized to work with. In addition to the user permission management issue, the MLflow tracking server lacks built-in authentication support. To establish a secure authentication mechanism, an external authentication server, such as **Nginx**, can be utilized. By integrating an authentication server into the MLflow infrastructure, users will be required to authenticate before gaining access to the tracking server and its associated resources. This ensures that only authorized users can interact with the MLflow model registry and perform operations within the system.

Serving models using model serving services

Once a model has been trained and saved, utilizing it to generate predictions is a matter of loading the saved model into an ML package and invoking the appropriate model prediction function provided by the package. However, for large-scale and complex model serving requirements, you will need to consider implementing a dedicated model serving infrastructure

to meet those needs. In the subsequent sections, we will explore a variety of open source model serving frameworks that can assist in addressing such needs.

The Gunicorn and Flask inference engine

Gunicorn and **Flask** are often used for building custom model-serving web frameworks. The following figure (*Figure 7.3*) shows a typical architecture that uses Flask, Gunicorn, and Nginx as the building blocks for a model serving service.

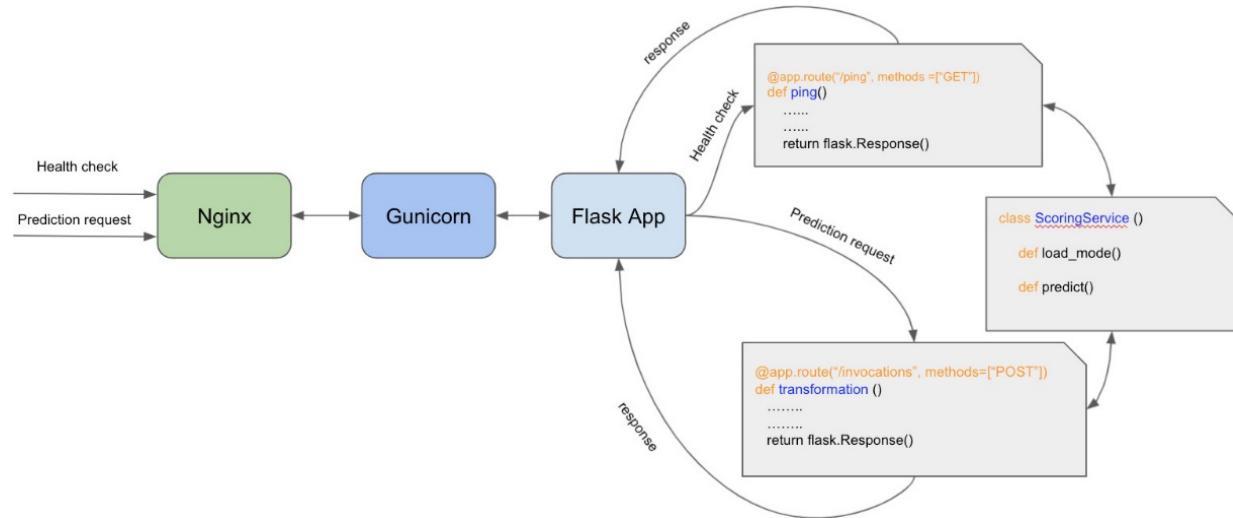


Figure 7.3: A model serving architecture using Flask and Gunicorn

Flask is a Python-based micro web framework for building web apps quickly. It is lightweight and has almost no dependencies on external libraries. With Flask, you can define different invocation routes and associate handler functions to handle different web calls (such as health check calls and model invocation calls). To handle model prediction requests, the Flask app would load the model into memory and call the predict function on the model to generate the prediction. Flask comes with a built-in web server, but it does not scale well as it can only support one request at a time. This is where Gunicorn can help address the scalability gap. Gunicorn is a web server for hosting web apps, including Flask apps. It can handle multiple requests in parallel and distribute the traffic to the hosted web apps efficiently. When it receives a web request, it will invoke the hosted Flask app to handle the request, such as invoking the function to generate model prediction. In addition to serving prediction requests as web requests, an enterprise inference engine also needs to handle secure web traffic (such as SSL/TLS traffic), as well as load balancing when there are multiple web servers. This is where Nginx can play an important role. Nginx can serve as a load balancer for multiple web servers and can handle termination for SSL/TLS traffic more efficiently, so web servers do not have to handle it. A Flask/Gunicorn-based model serving architecture can be a good option for hosting simple model serving patterns. But for more complicated patterns, such as serving different versions of models, A/B testing, or large model serving, this architecture will have limitations. The Flask/Gunicorn architecture pattern also requires custom code (such as the Flask app) to work, as it does not provide built-in support for the different ML models. Next, let's explore some

purpose-built model serving frameworks and see how they are different from the custom Flask-based inference engine.

The TensorFlow Serving framework

TensorFlow Serving is a production-grade, open source model serving framework, and provides out-of-the-box support for serving TensorFlow models behind a RESTful endpoint. It manages the model life cycle for model serving and provides access to versioned and multiple models behind a single endpoint. There is also built-in support for canary deployments. A **canary deployment** allows you to deploy a model to support a subset of traffic. In addition to the real-time inference support, there is also a batch scheduler feature that can batch multiple prediction requests and perform a single joint execution. With TensorFlow Serving, there is no need to write custom code to serve the model. The following figure (*Figure 7.4*) shows the architecture of TensorFlow Serving:

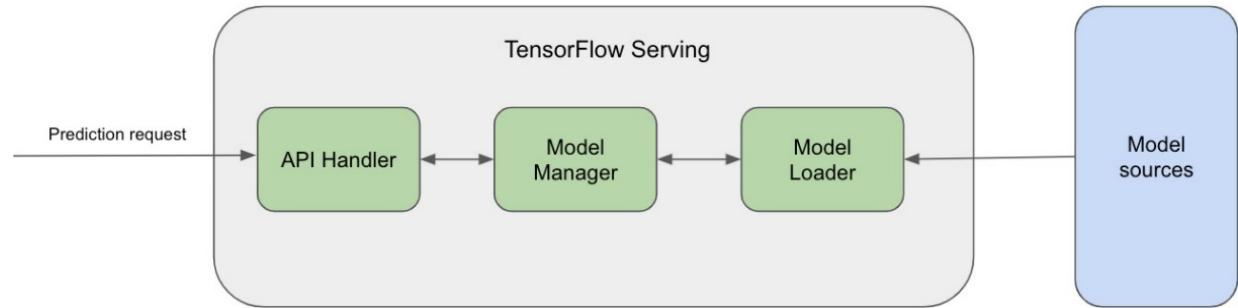


Figure 7.4: TensorFlow Serving architecture

The *API handler* provides APIs for TensorFlow Serving. It comes with a built-in, lightweight HTTP server to serve RESTful-based API requests. It also supports **gRPC** (a **remote procedure call** protocol) traffic. gRPC is a more efficient and fast networking protocol, however it is more complicated to use than the REST protocol. TensorFlow Serving has a concept called a *servable*, which refers to the actual objects that handle a task, such as model inferences or lookup tables. For example, a trained model is represented as a *servable*, and it can contain one or more algorithms and lookup tables or embeddings tables. The API handler uses the servable to fulfill client requests. The *model manager* manages the life cycle of servables, including loading the servables, serving the servables, and unloading the servables. When a servable is needed to perform a task, the model manager provides the client with a handler to access the servable instances. The model manager can manage multiple versions of a servable, allowing gradual rollout of different versions of a model. The *model loader* is responsible for loading models from different sources, such as **Amazon S3**. When a new model is loaded, the model loader notifies the model manager about the availability of the new model, and the model manager will decide what the next step should be, such as unloading the previous version and loading the new version. TensorFlow Serving can be extended to support non-TensorFlow models. For example, models trained in other frameworks can be converted to the **ONNX** format and served using TensorFlow Serving. ONNX is a common format for representing models to support interoperability across different ML frameworks.

The TorchServe serving framework

TorchServe is an open source framework for serving trained **PyTorch** models. Similar to TensorFlow Serving, TorchServe provides a REST API for serving models with its built-in web server. With core features such as multi-model serving, model versioning, server-side request batching, and built-in monitoring, TorchServe can serve production workloads at scale. There is also no need to write custom code to host PyTorch models with TorchServe. In addition, TorchServe comes with a built-in web server for hosting the model. The following figure (Figure 7.5) illustrates the architecture components of the TorchServe framework:

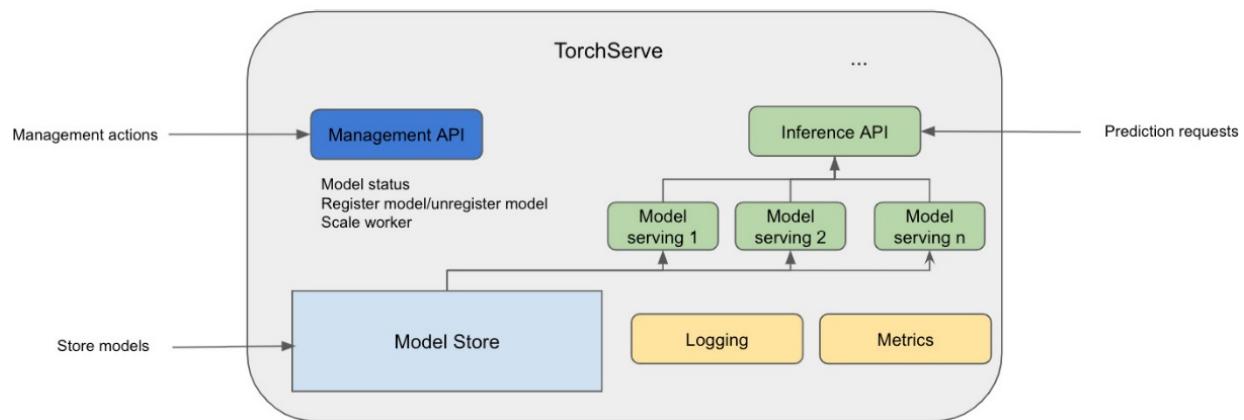


Figure 7.5: TorchServe architecture

The *inference API* is responsible for handling prediction requests from client applications using loaded PyTorch models. It supports the REST protocol and provides a prediction API, as well as other supporting APIs such as health check and model explanation APIs. The inference API can handle prediction requests for multiple models. The model artifacts are packaged into a single archive file and stored in a model store within the TorchServe environment. You use a **command-line interface (CLI)** command called `torch-mode-archive` to package the model. The TorchServe backend loads the archived models from the model store into different worker processes. These worker processes interact with the inference API to process requests and send back responses. The management API is responsible for handling management tasks such as registering and unregistering PyTorch models, checking the model status, and scaling worker process. The management API is normally used by system administrators. TorchServe also provides built-in support for logging and metrics. The logging component logs both access logs and processing logs. The TorchServe metrics collect a list of system metrics, such as CPU/GPU utilization and custom model metrics.

KFServing framework

TensorFlow Serving and TorchServe are standalone model serving frameworks for a specific deep learning framework. In contrast, **KFServing** is a general-purpose multi-framework model serving framework that supports different ML models. KFServing uses standalone model serving frameworks such as TensorFlow Serving and TorchServe as the backend model servers.

It is part of the Kubeflow project and provides pluggable architecture for different model formats:

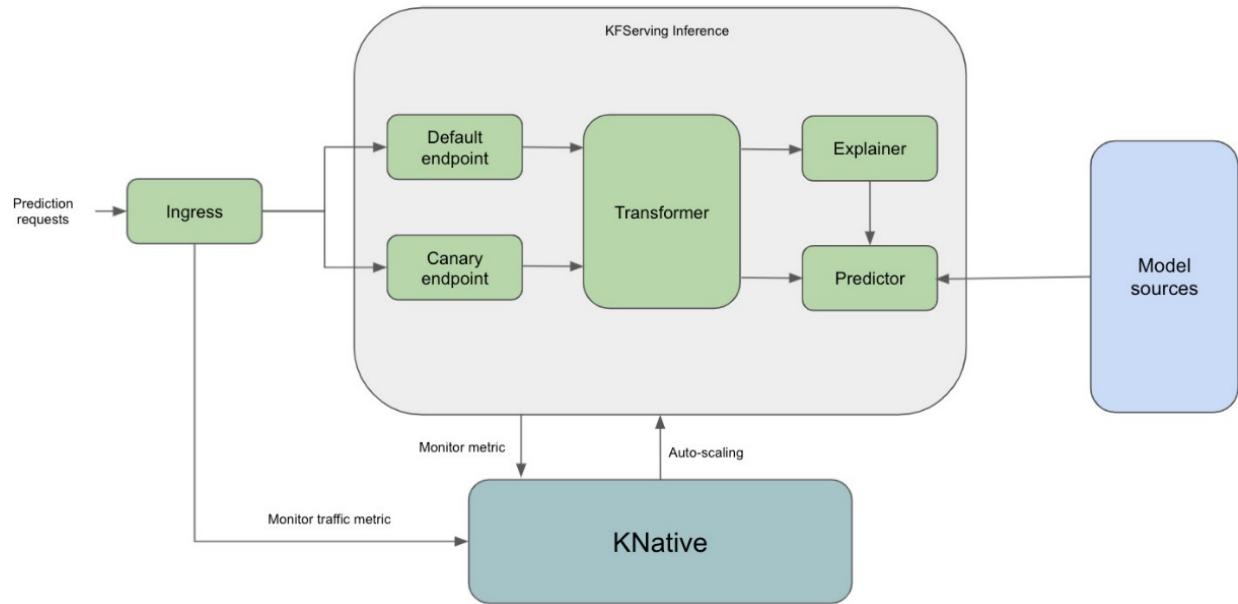


Figure 7.6: KFServing components

As a general-purpose, multi-framework model serving solution, KFServing provides several out-of-the-box model servers (also known as predictors) for different model types, including TensorFlow, PyTorch, XGBoost, scikit-learn, and ONNX. With KFServing, you can serve models using both REST and gRPC protocols. To deploy a supported model type, you simply need to define a YAML specification that points to the model artifact in a data store. Furthermore, you can build your own custom containers to serve models in KFServing. The container needs to provide a model serving implementation as well as a web server. The following code shows a sample YAML specification to deploy a tensorflow model using KFServing:

```
apiVersion: "serving.kubeflow.org/v1alpha2"
kind: "InferenceService"
metadata:
  name: "model-name"
spec:
  default:
    predictor:
      tensorflow:
        storageUri: <uri to model storage such as s3>
```

KFServing has a transformer component that allows the custom processing of the input payload before it is sent to the predictors, and also allows transforming the response from the predictor before it is sent back to the calling client. Sometimes, you need to provide an explanation for the model prediction, such as which features have a stronger influence on the prediction, which we will cover in more details in a later chapter. KFServing is designed for production deployment

and provides a range of production deployment capabilities. Its auto-scaling feature allows the model server to scale up/down based on the amount of request traffic. With KFServing, you can deploy both the default model serving endpoint and the canary endpoint and split the traffic between the two, and specify model revisions behind the endpoint. For operational support, KFServing also has built-in functionality for monitoring (for example, monitoring request data and request latency).

Seldon Core

Seldon Core is another multi-framework model serving framework for deploying models on Kubernetes. Compared to KFServing, Seldon Core provides richer model serving features, for example, model serving inference graphs for use cases such as A/B testing and model ensembles. The following figure shows the core components of the Seldon Core framework:

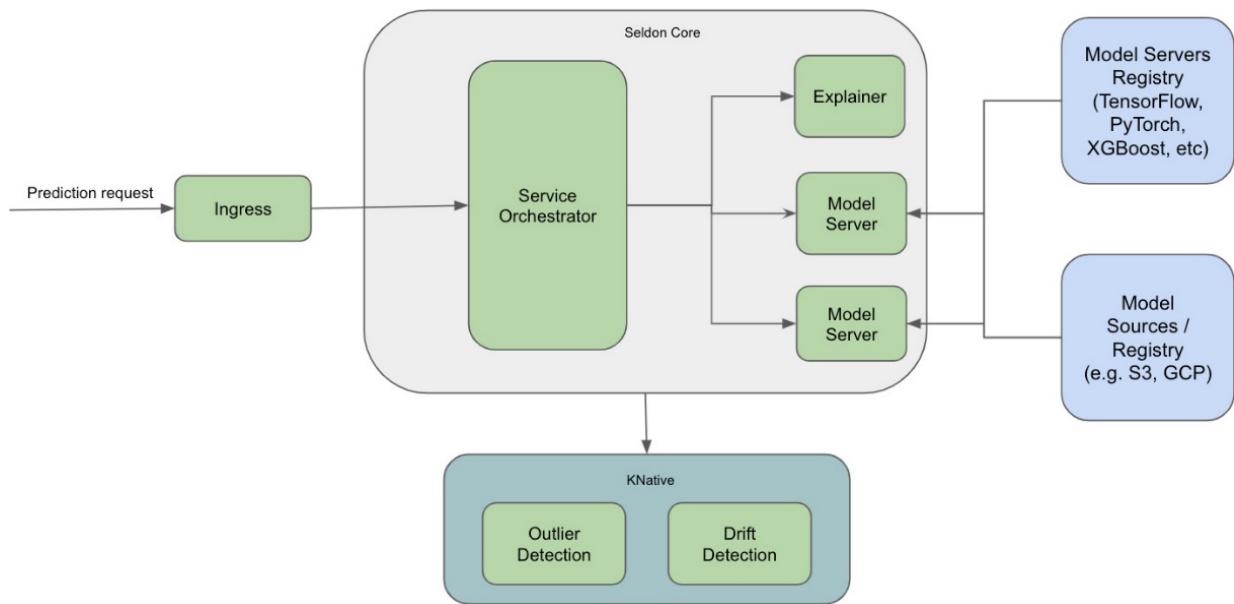


Figure 7.7: The Seldon Core model serving framework architecture

Seldon Core provides packaged model servers for some of the common ML libraries, including the **SKLearn** server for scikit-learn models, the XGBoost server for XGBoost models, TensorFlow Serving for TensorFlow models, and MLflow server-based model serving. You can also build your own custom serving container for specific model serving needs and host it using Seldon Core. The following template shows how to deploy a model using the SKLearn server using Seldon Core. You simply need to change the **modelUri** path to point to a saved model on a cloud object storage provider such as **Google Cloud Storage**, Amazon S3 storage, or **Azure Blob storage**. To test with an example, you can change the following **modelUri** value to an example provided by Seldon Core – `gs://seldon-models/sklearn/iris`:

```
apiVersion: machinelearning.seldon.io/v1alpha2
kind: SeldonDeployment
metadata:
```

```

name: sklearn
spec:
  name: sklearn-model
  predictors:
    - graph:
        children: []
        implementation: SKLEARN_SERVER
        modelUri: <model uri to model artifacts on the cloud storage>
        name: classifier
    name: default
    replicas: 1

```

Seldon Core also supports an advanced workflow, known as an inference graph, for serving models. The *inference graph* feature allows you to have a graph with different models and other components in a single inference pipeline. An inference graph can consist of several components:

- One or more ML models for the different prediction tasks
- Traffic routing management for different usage patterns, such as traffic splitting to different models for A/B testing
- A component for combining results from multiple models, such as a model ensemble component
- Components for transforming the input requests (such as performing feature engineering) or output responses (for example, returning an array format as a **JSON** format)

To build inference graph specifications in YAML, you need the following key components in the `seldondeployment` YAML file:

- A list of predictors, with each predictor having its own `componentSpecs` section that specifies details such as container images
- A graph that describes how the components are linked together for each `componentSpecs` section

The following sample template shows the inference graph for a custom canary deployment to split the traffic into two different versions of a model:

```

apiVersion: machinelearning.seldon.io/v1alpha2
kind: SeldonDeployment
metadata:
  name: canary-deployment
spec:
  name: canary-deployment
  predictors:
    - componentSpecs:
        - spec:
            containers:
              - name: classifier
                image: <container uri to model version 1>
    graph:
      children: []
      endpoint:

```

```

        type: REST
        name: classifier
        type: MODEL
    name: main
    replicas: 1
    traffic: 75
- componentSpecs:
- spec:
    containers:
- name: classifier
    image: <container uri to model version 2>
graph:
    children: []
    endpoint:
        type: REST
        name: classifier
        type: MODEL
    name: canary
    replicas: 1
    traffic: 25

```

Once a deployment manifest is applied, the Seldon Core operator is responsible for creating all the resources needed to serve an ML model. Specifically, the operator will create resources defined in the manifest, add orchestrators to the Pods to manage the orchestration of the inference graph, and configure the traffic using ingress gateways such as Istio.

Triton Inference Server

Triton Inference Server is an open source software designed to streamline the process of AI inferencing. It offers a versatile solution for deploying AI models from various deep learning and machine learning frameworks, including TensorRT, TensorFlow, PyTorch, ONNX, OpenVINO, Python, and more. Triton is compatible with a wide range of devices, supporting inference across cloud environments, data centers, edge devices, and embedded systems. Comparing to Seldon Core, Triton inference server is more focused on performance. It is designed to be highly scalable and efficient, making it a good choice for high-traffic applications. The following figure depicts the core components of the Triton Inference Server:

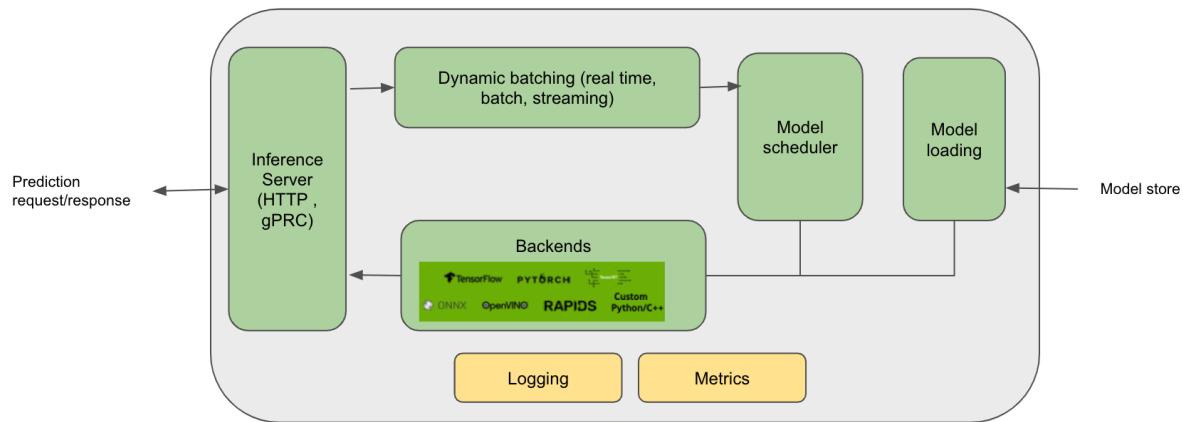


Figure 7.8: Triton Inference Server Architecture

The architecture of Triton Inference Server encompasses several components that work together to enable efficient and scalable inferencing. At its core is the Backend, which represents specific deep learning or machine learning frameworks supported by Triton. Each Backend handles the loading and execution of models trained with its corresponding framework. The Inference Server acts as the central hub, receiving and managing inference requests. It communicates with clients, such as web applications or services, and orchestrates the inferencing process. The Model Repository serves as the storage location for trained models. It contains serialized versions of models compatible with the supported backends. When requested by clients, the server accesses and loads the models into memory for inferencing. Triton supports multiple inference protocols, including HTTP/REST and gRPC, allowing clients to communicate with the server and make inference requests. Clients can specify input data and desired output formats using these protocols. To monitor and optimize performance, Triton provides metrics and monitoring capabilities. These metrics include GPU utilization, server throughput, latency, and other relevant statistics. Monitoring these metrics helps administrators optimize resource utilization and identify potential bottlenecks. Triton also offers dynamic batching capability. This feature allows for efficient processing of multiple inference requests by grouping them together into batches. This batching mechanism optimizes resource utilization and improves overall inferencing performance. Overall, the architecture of Triton Inference Server is designed to facilitate the efficient deployment and execution of AI models across diverse frameworks and hardware platforms. It offers flexibility, scalability, and extensibility, enabling organizations to leverage their preferred frameworks while ensuring high-performance inferencing capabilities.

Monitoring models in production

Model performance can deteriorate over time due to various factors such as changing data patterns, shifts in user behavior, or unforeseen scenarios. To ensure the ongoing effectiveness of deployed ML models, continuous monitoring of their performance and behavior in production is essential. Model monitoring involves actively tracking and analyzing the performance of deployed ML models. This process includes collecting data on different metrics and indicators,

comparing them to predefined thresholds or baselines, and identifying anomalies or deviations from expected behavior. Two critical aspects of model monitoring are data drift and model drift.

- **Data drift:** Data drift refers to the scenarios where the statistical properties of incoming data change over time. This can create a disconnect between the data used to train the model and the data it encounters in the production environment. Data drift significantly impacts the performance and reliability of ML models, as they may struggle to adapt to new and evolving patterns in the data.
- **Model drift:** Model drift refers to the degradation of a machine learning model's performance over time due to changes in underlying patterns or relationships in the data. When the assumptions made during model training no longer hold true in the production environment, model drift occurs. It can lead to decreased accuracy, increased errors, and suboptimal decision-making.

To support model monitoring efforts, there are several open source and commercial products available in the market. These tools provide capabilities for monitoring model performance, detecting data drift, identifying model drift, and generating insights to help organizations take necessary corrective actions. Some popular examples include Evidently AI, Arize AI, Seldon Core, Fiddler, and Author AI.

Managing ML features

As organizations increasingly adopt ML solutions, they recognize the need to standardize and share commonly used data and code throughout the ML lifecycle. One crucial element that organizations seek to manage centrally is ML features, which are commonly used data attributes that serve as inputs to ML models. To enable standardization and reuse of these features, organizations often turn to a feature store. A feature store acts as a centralized repository for storing and managing ML features. It provides a dedicated platform for organizing, validating, and sharing features across different ML projects and teams within an organization. By consolidating features in a single location, the feature store promotes consistency and facilitates collaboration among data scientists and ML practitioners. The concept of a feature store has gained significant attention in the ML community due to its numerous benefits. Firstly, it enhances productivity by eliminating the need to recreate and engineer features for each ML project. Instead, data scientists can readily access precomputed and validated features from the store, saving time and effort. Additionally, a feature store improves model performance by ensuring the consistency and quality of features used in ML models. By centralizing feature management, organizations can enforce data governance practices, perform feature validation, and monitor feature quality, leading to more reliable and accurate ML models. Several open-source feature store frameworks are available in the market, such as Feast Feature Store and Hopsworks Feature Store, offering organizations flexible options for managing their ML features. Let's take a closer look at Feast:

Feast

Feast is an open-source feature store that enables organizations to manage, discover, and serve features for ML applications. Developed by Tecton, Feast is designed to handle large-scale, real-time feature data. It supports feature ingestion from various sources, including batch pipelines and streaming systems like Apache Kafka. Feast integrates well with popular ML frameworks such as TensorFlow and PyTorch, allowing seamless integration into ML workflows. With features like feature versioning, data validation, and online and offline serving capabilities, Feast provides a comprehensive solution for feature management.

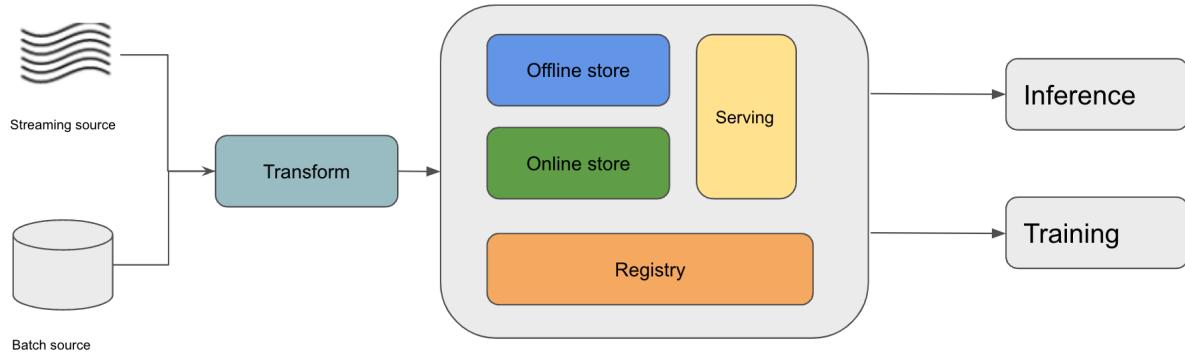


Figure 7.9: Feast Feature Store

At the core of the Feast architecture is the online and offline feature storage, which serves as the centralized storage for feature data. The Feature Repository stores feature data in a distributed storage system, allowing for scalable and efficient storage and retrieval of feature values. Feast employs a decoupled architecture, where the ingestion of feature data and the serving of features are separated. The data ingestion component is responsible for extracting feature data from various sources, such as data warehouses, databases, and streaming platforms. It then transforms and loads the feature data into the feature storage, ensuring data quality and consistency. The feature serving component is responsible for providing low-latency access to feature data for ML models during training and inference. The feature serving component also supports online and offline serving modes, allowing for real-time and batch feature serving. To enable efficient data discovery, Feast employs a feature registry. The feature registry allows for fast lookup and retrieval of feature values based on different feature combinations and time ranges. Feast also integrates with popular ML frameworks, such as TensorFlow and PyTorch, through its SDKs and client libraries. These integrations enable seamless integration of Feast into ML pipelines and workflows, making it easy for data scientists and ML engineers to access and utilize feature data in their models. Overall, the Feast feature store architecture provides a robust and scalable solution for managing and serving ML features. By centralizing feature data management, Feast enables organizations to enhance productivity, improve model performance, and promote collaboration in ML development.

Automating ML pipeline workflows

To automate the core ML platform components we have discussed so far, we need to build pipelines that can orchestrate different steps using these components. Automation not only brings

efficiency, productivity, and consistency, while enabling reproducibility and minimizing human errors. There are several open source technologies available to automate ML workflows, with Apache Airflow and Kubeflow Pipelines being prominent examples.

Apache Airflow

Apache Airflow is an open source software package for programmatically authoring, scheduling, and monitoring multi-step workflows. It is a general-purpose workflow orchestration tool that can be leveraged to define workflows for a wide range of tasks, including ML tasks. First, let's explore some core Airflow concepts:

- **Directed Acyclic Graph (DAG):** A DAG defines independent tasks that are executed independently in a pipeline. The sequences of the execution can be visualized like a graph.
- **Tasks:** Tasks are basic units of execution in Airflow. Tasks have dependencies between them during executions.
- **Operators:** Operators are DAG components that describe a single task in the pipeline. An operator implements the task execution logic. Airflow provides a list of operators for common tasks, such as a Python operator for running Python code, or an Amazon S3 operator to interact with the S3 service. Tasks are created when operators are instantiated.
- **Scheduling:** A DAG can run on demand or on a predetermined schedule.

Airflow can run on a single machine or in a cluster. Additionally, it can be deployed on the Kubernetes infrastructure. The following figure shows a multi-node Airflow deployment:

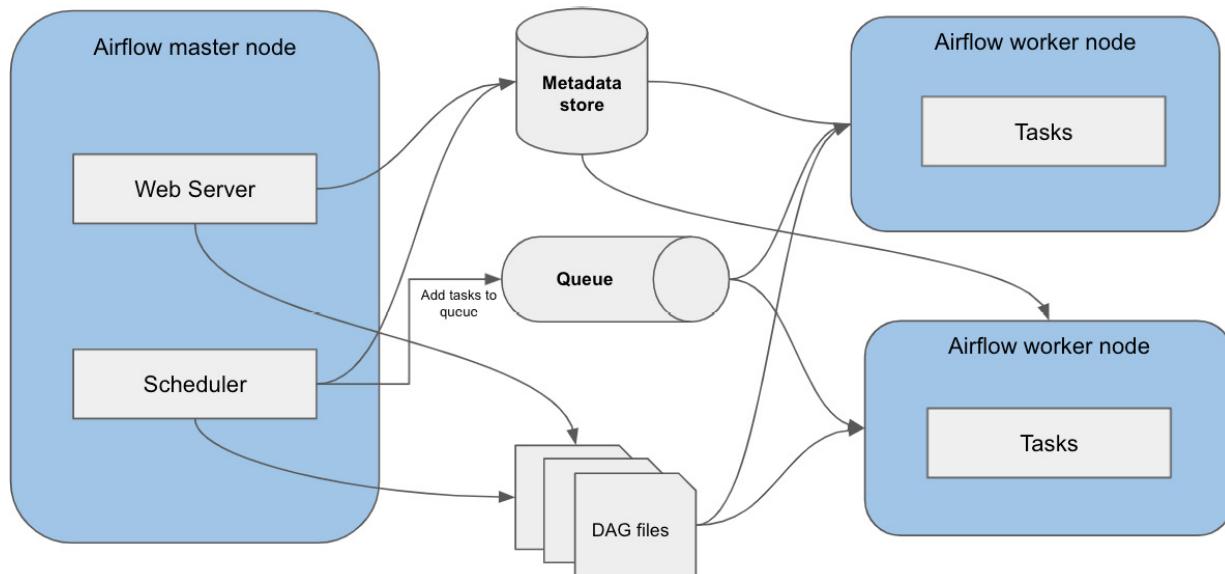


Figure 7.8: Apache Airflow architecture

The *master node* mainly runs the *web server* and *scheduler*. The scheduler is responsible for scheduling the execution of the DAGs. It sends tasks to a queue, and the worker nodes retrieve the tasks from the queue and run them. The metadata store is used to store the metadata of the

Airflow cluster and processes, such as task instance details or user data. You can author the Airflow DAGs using Python. The following sample code shows how to author a basic Airflow DAG in Python with two bash operators in a sequence:

```
from airflow import DAG
from airflow.operators.bash_operator import BashOperator
from datetime import datetime, timedelta
default_args = {
    'owner': 'myname',
}
dag = DAG('test', default_args=default_args, schedule_interval=timedelta(days=1))
t1 = BashOperator(
    task_id='print_date',
    bash_command='date',
    dag=dag)
t2 = BashOperator(
    task_id='sleep',
    bash_command='sleep 5',
    retries=3,
    dag=dag)
t2.set_upstream(t1)
```

Airflow can connect to many different sources and has built-in operators for many external services, such as **AWS EMR** and **Amazon SageMaker**. It has been widely adopted by many enterprises in production environments.

Kubeflow Pipelines

Kubeflow Pipelines is a Kubeflow component, and it is purpose-built for authoring and orchestrating end-to-end ML workflows on Kubernetes. First, let's review some core concepts of Kubeflow Pipelines:

- **Pipeline:** A pipeline describes an ML workflow, all the components in the workflow, and how the components are related to each other in the pipeline.
- **Pipeline components:** A pipeline component performs a task in the pipeline. An example of a pipeline component could be a data processing component or model training component.
- **Experiment:** An experiment organizes different trial runs (model training) for an ML project so you can easily inspect and compare the different runs and their results.
- **Step:** The execution of one component in a pipeline is called a step.
- **Run trigger:** You use a run trigger to kick off the execution of a pipeline. A run trigger can be a periodic trigger (for example, to run every 2 hours), or a scheduled trigger (for example, run at a specific date and time).
- **Output artifacts:** Output artifacts are the outputs from the pipeline components. Examples of output artifacts could be model training metrics or visualizations of datasets.

Kubeflow Pipelines is installed as part of the Kubeflow installation. It comes with its own UI, which is part of the overall Kubeflow dashboard UI. The Pipelines service manages the pipelines and their run status and stores them in a metadata database. There is an orchestration

and workflow controller that manages the actual execution of the pipelines and the components. The following figure (Figure 7.9) illustrates the core architecture components in a Kubeflow pipeline:

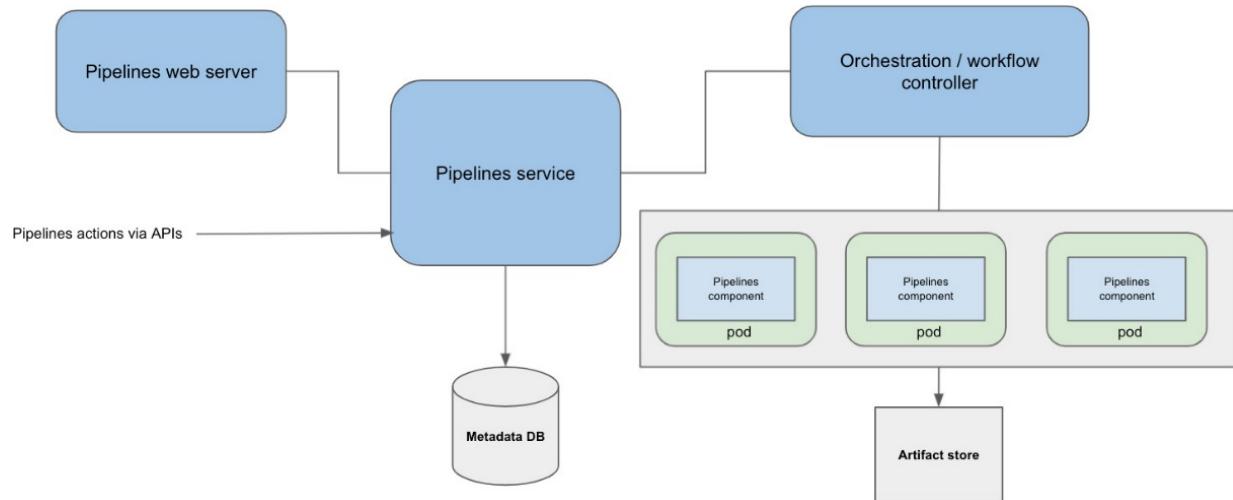


Figure 7.9: Kubeflow Pipelines architecture

You author the pipeline using the pipeline SDK in Python. To create and run a pipeline, follow these steps:

1. Create a pipeline definition using the Kubeflow SDK. The pipeline definition specifies a list of components and how they are joined together in a graph.
2. Compile the definition into a static YAML specification to be executed by the Kubeflow Pipelines service.
3. Register the specification with the Kubeflow Pipelines service and call the pipeline to run from the static definition.
4. The Kubeflow Pipelines service calls the API server to create resources to run the pipeline.
5. Orchestration controllers execute various containers to complete the pipeline run.

Now that we have explored various open source tools for building ML platforms, let's delve into end-to-end architecture using these open source frameworks and components.

Building end-to-end ML platform

Now have discussed several open-source technologies individually, let's delve into their integration and see how these components come together. The architecture patterns and technology stack selection may vary based on specific needs and requirements. The following diagram presents the conceptual building blocks of an ML platform architecture.

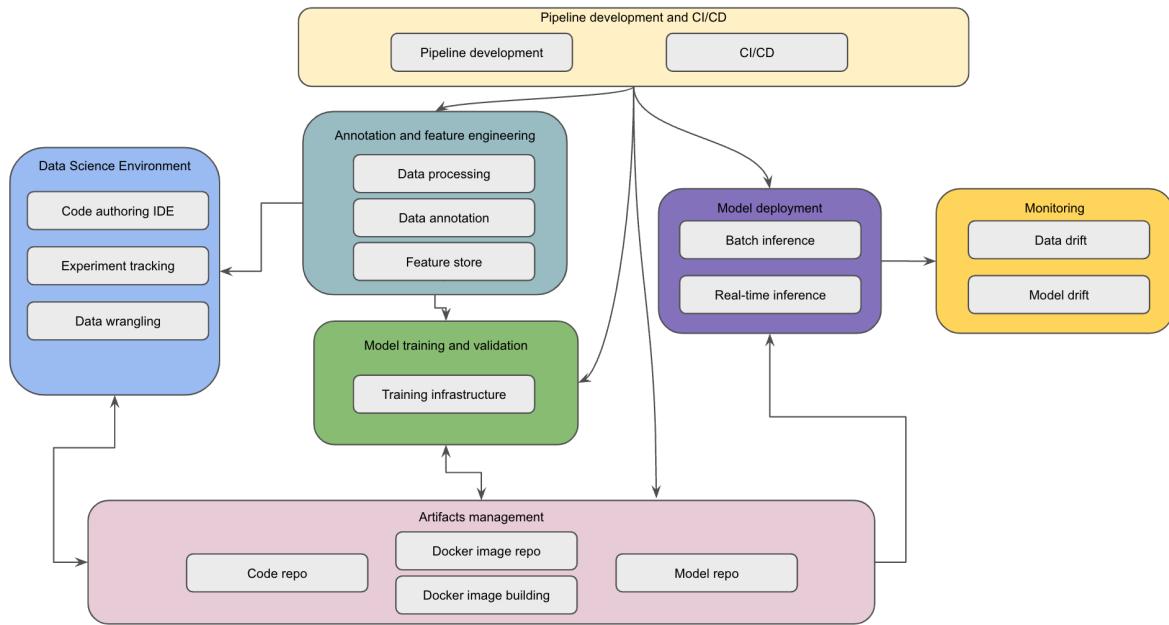


Figure 7.10: ML Platform Architecture

Next, let's delve into different strategies to implement this architecture concept with different combination of open source technologies.

ML platform based strategy

When designing an ML platform using open source technologies, one effective strategy is to utilize a ML platform framework as a base platform and then integrate additional open-source components to address specific requirements. One such ML platform framework is Kubeflow, which provides a robust foundation with its built-in building blocks for an ML platform. By leveraging Kubeflow, you can benefit from its core components while extending the platform's capabilities through the integration of complementary open-source tools. This strategy allows for flexibility and customization by seamlessly integrating a range of open-source ML components into the platform. You would choose this approach if the base ML platform framework meets the most of your requirements, or if you can work within the limitations of the framework. The following table outlines key ML platform components and their corresponding open-source frameworks and tools:

ML Platform Component	Open source framework
Code Repository	Git Hub
Experimentation and model development	Kubeflow Jupyter Notebook
Experiment Tracking	MLFlow Experiment Tracker
Feature Store	Feast Feature Stote
Data Annotation	Computer Vision Annotation Tool (CVAT)

Training	Kubeflow Training Operators
Data and model testing	DeepChecks
Model repository	MLFlow Model Repository
ML Pipeline development	Kubeflow Pipeline
Model inference	Kubeflow KFServing (Seldon Core, TF Serving, Triton)
Docker Image Repository	Docker Hub
CI/CD	Github Actions
Drift monitoring	DeepChecks

By incorporating these frameworks and tools into the architectural conceptual diagram, we can visualize the resulting diagram as follows.

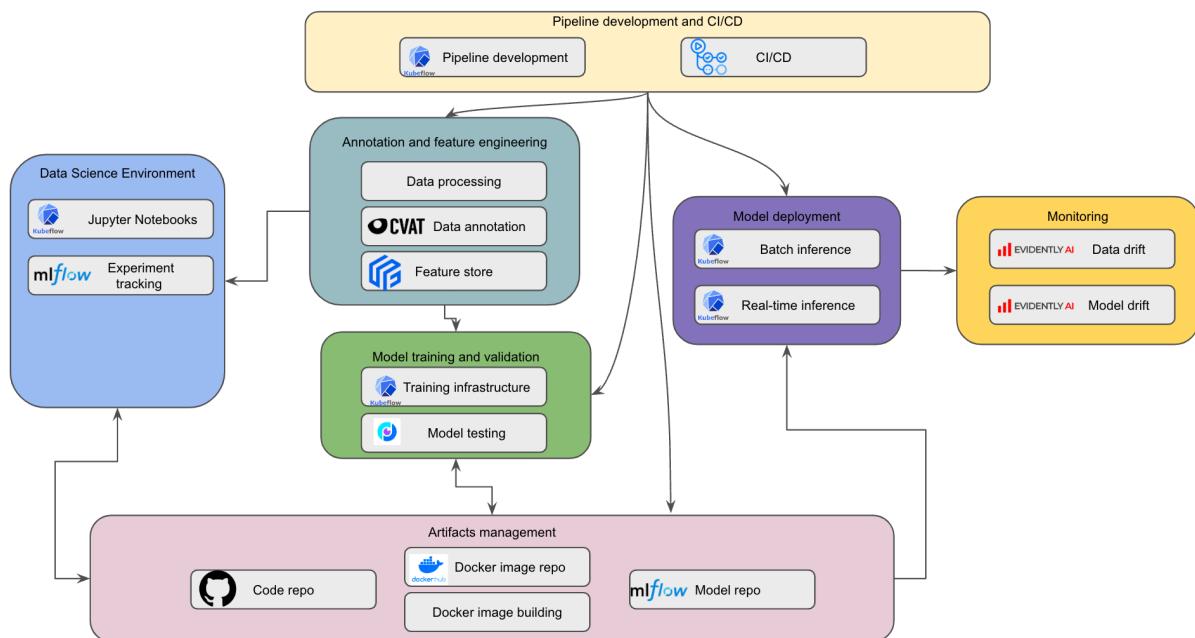


Figure 7.11: Kubeflow based ML platform

Using this architecture, data scientists utilize Kubeflow Jupyter Notebook for conducting experiments and building models. Experiment runs and relevant details, such as data statistics, hyperparameters, and model metrics, are tracked and saved in the MLFlow Experiment Tracking component. Common ML features are stored in the Feast Feature Store. When there is a need for image data annotation, data annotators can employ open-source data annotation tools like the Computer Vision Annotation Tool (CVAT) to label images for model training. Data scientists can utilize features from the feature store and the labeled dataset as part of their experimentation and model building. GitHub serves as the code repository for data scientists. They save all source code, including training scripts, algorithm code, and data transformation scripts, in the code repository. Model training and inference docker images are stored in the Docker Hub. A docker image build process can be deployed to create new docker images for training and inference.

purposes. For formal model training, the training script is pulled from the GitHub repository, and the training docker image is pulled from Docker Hub into the Kubeflow training operator to initiate the model training, along with the training dataset. DeepChecks can be utilized to perform data validation and model performance checks. Once the model is trained, the model artifacts, along with any metadata such as model metrics and evaluation graphs, are stored in the MLFlow Model Registry. When it is time to deploy the model, models are fetched from the MLFlow model registry and loaded into KFServing for model inference, along with model inference docker image and inference script. KFServing offers the flexibility to choose different inference servers, including Seldon Core, TFServing, and Triton. Prediction logs can be sent to the model monitoring component for detecting data drift and model drift. Open-source software tools like Evidently AI can be employed for data drift and model drift detection. To orchestrate various tasks such as data processing, feature engineering, model training, and model validation, Kubeflow Pipeline can be developed. For CI/CD (Continuous Integration/Continuous Deployment), GitHub Actions can be used as a triggering mechanism to initiate different pipelines. Overall, this approach allows you to combine the benefits of a base ML platform framework with the flexibility provided by a wide range of open-source components.

ML component based strategy

An alternative approach is to build the ML platform using individual components rather than relying on a base ML platform framework. This strategy offers the advantage of selecting the best-in-class components for each aspect of the platform, allowing organizations to adhere to their existing open-source standards for core components like pipeline development or notebook IDE. The architectural pattern depicted below illustrates this approach.

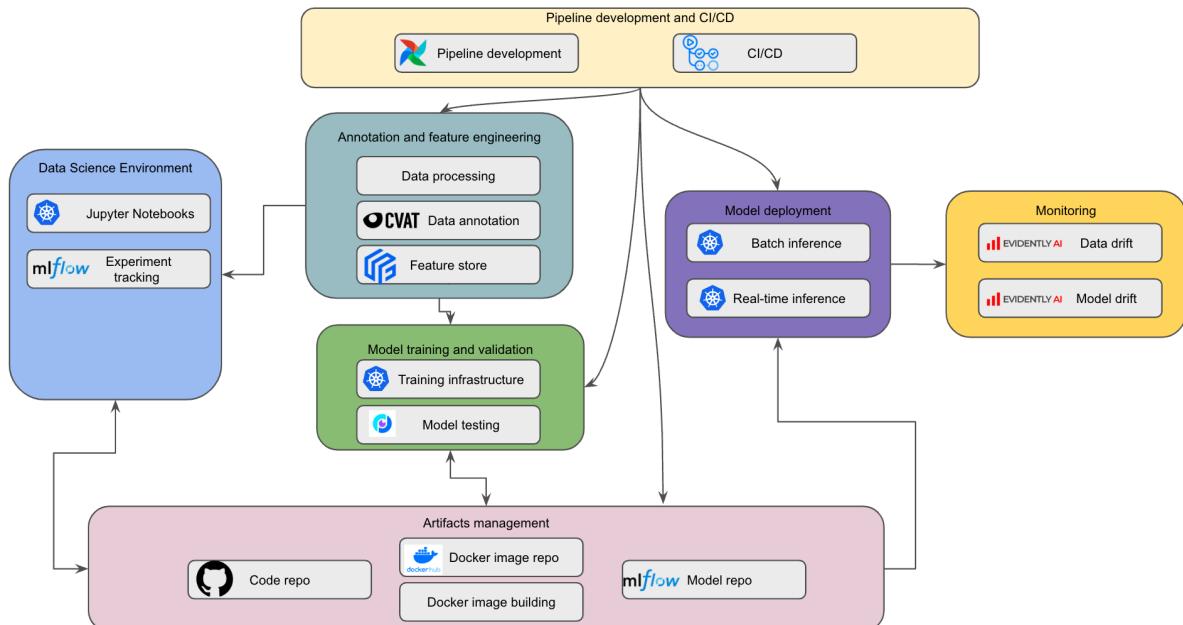


Figure 7.12: Component based architecture

In this architecture pattern, alternative technologies and tools are utilized for pipeline development, notebook environments, and training infrastructure management. Additionally, Kubernetes serves as the infrastructure management framework. This approach allows organizations to leverage specific technologies and tools that align with their unique requirements and preferences. One notable aspect is the use of Airflow as the standard orchestration and pipeline tool across various technical disciplines, including machine learning and data management. Airflow's widespread adoption within organizations enables it to serve as a unifying pipeline management tool, facilitating seamless integration between different components and workflows. Moreover, this architecture pattern emphasizes the building of custom training and inference infrastructure on top of Kubernetes. By leveraging Kubernetes, organizations gain the flexibility to create customized training and inference environments tailored to their specific needs. In addition to the availability of free open-source tools that meet ML platform requirements, it is important to consider the integration of commercial components into the open-source architecture. These commercial offerings can enhance specific aspects of the ML platform and provide additional capabilities. For instance, when deploying this architecture pattern on AWS, it is advisable to explore the use of AWS Elastic Container Registry (ECR) as the docker image repository. AWS ECR provides a managed and secure solution for storing and managing container images, integrating seamlessly with other AWS services. When it comes to monitoring, there are commercial products like Fiddler and Author AI that can offer advanced features and insights. These tools can enhance the monitoring capabilities of the ML platform, providing in-depth analysis, explainability, and visualization of model behavior and performance. Overall, the advantages of this architecture pattern include the ability to choose alternative technologies and tools for different aspects of the ML platform, leveraging Airflow as a unifying pipeline management tool, and building custom training and inference infrastructure on Kubernetes. These choices enable organizations to create a tailored and optimized ML platform that aligns precisely with their requirements and allows for highly customized training and inference processes.

Summary

In this chapter, you have gained an understanding of the core architecture components of a typical ML platform and their capabilities. We have explored various open-source technologies such as Kubeflow, MLflow, TensorFlow Serving, Seldon Core, Triton Inference Server, Apache Airflow, and Kubeflow Pipelines. Additionally, we have discussed different strategies for approaching the design of an ML platform using open-source frameworks and tools. While these open-source technologies offer powerful features for building sophisticated ML platforms, it is important to acknowledge that constructing and maintaining such environments requires substantial engineering effort and expertise, especially when dealing with large-scale ML platforms. In the next chapter, we will delve into fully managed, purpose-built ML solutions that are specifically designed to facilitate the development and operation of ML environments. These managed solutions aim to simplify the complexities of building and managing ML platforms, providing pre-configured and scalable infrastructure, as well as additional features tailored for machine learning workflows.

8 Building a Data Science Environment Using AWS ML Services

Join our book community on Discord

<https://packt.link/EarlyAccessCommunity>



While some organizations opt to build their own **machine learning (ML)** platforms using open source technologies, many other organizations prefer to leverage fully managed ML services as the foundation for their ML platforms. In this chapter, we will delve into the fully managed ML services offered by AWS. Specifically, you will learn about **Amazon SageMaker**, a fully managed ML service, and other related services for building a data science environment for data scientists. We will examine various components of SageMaker, such as SageMaker Studio, SageMaker Training Service, and SageMaker Hosting Service. Additionally, we will delve into the architectural framework for constructing a data science environment and provide a hands-on exercise to guide you through the process.

Technical requirements

In this chapter, you will need access to an AWS account and have the following AWS services for the hands-on lab:

- Amazon S3
- Amazon SageMaker
- Amazon ECR

You will also need to download the dataset from <https://www.kaggle.com/ankurzing/sentiment-analysis-for-financial-news>. The sample source code used in this chapter can be found at <https://github.com/PacktPublishing/The-Machine-Learning-Solutions-Architect-Handbook/tree/main/Chapter08>.

SageMaker overview

Amazon SageMaker offers machine learning (ML) functionalities that cover the entire ML lifecycle, spanning from initial experimentation to production deployment and ongoing monitoring. It caters to various roles, such as data scientists, data analysts, and MLOps engineers. The following diagram showcases the key SageMaker features that support the complete data science journey for different personas.

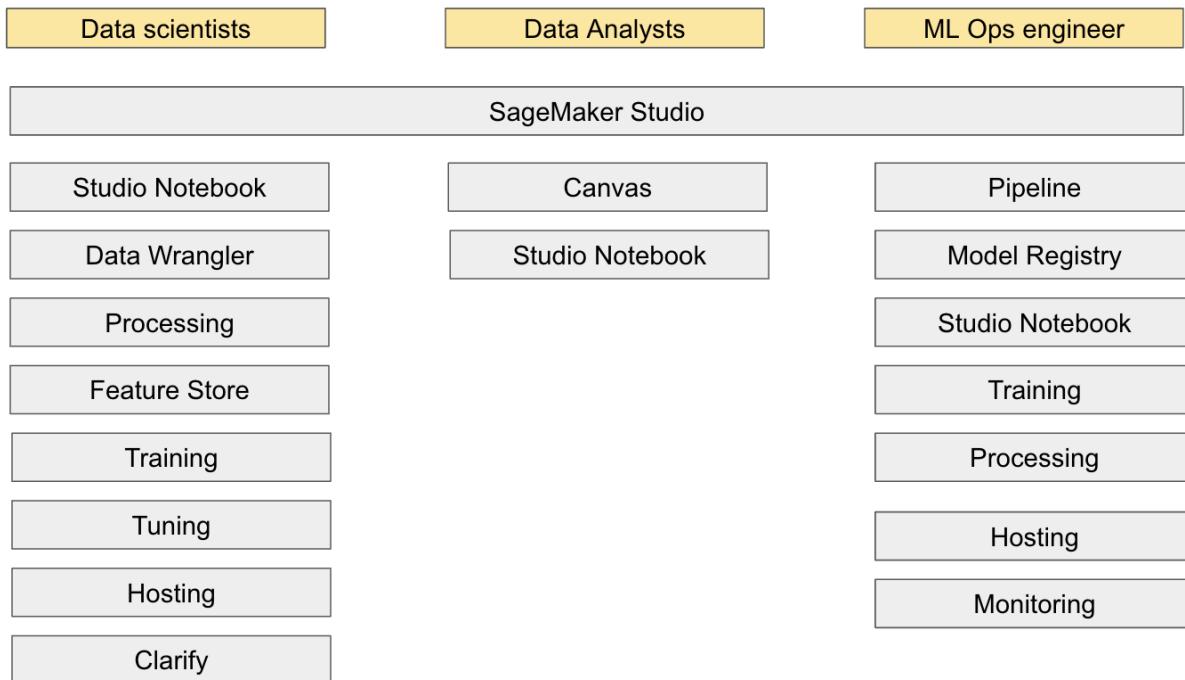


Figure 8.1: SageMaker capabilities

Within SageMaker, data scientists have access to an array of features and services to support different machine learning tasks. These include Studio Notebook for model building, Data Wrangler for visual data preparation, Processing service for large scale data processing and transformation, Training service, Tuning service for model tuning, and Hosting service for model hosting. With these tools, data scientists can handle various ML responsibilities such as data preparation, model building and training, model tuning, and conducting model integration testing. On the other hand, data analysts can utilize SageMaker Canvas, a user-friendly model building service that requires little to no coding. This visual interface empowers analysts to train models effortlessly. Additionally, they can use Studio Notebook for lightweight data analysis and processing. MLOps engineers play a crucial role in managing and governing the ML environment. They are responsible for automating ML workflows and can leverage SageMaker Pipeline, Model Registry, and Endpoint monitoring to achieve this. Furthermore, MLOps engineers configure the processing, training, and hosting infrastructure to ensure smooth operations for both interactive usage by the data scientists and automated operations. In this particular chapter, our focus will center around data science environments catered specifically to data scientists. Subsequently, in the following chapter, we will delve into the administration, governance, and automation of ML infrastructure.

Data science environment architecture using SageMaker

Data scientists use data science environments to iterate different data science experiments with various datasets and algorithms. These environments require essential tools like Jupyter Notebook to author and execute code, data processing engines for handling large-scale data processing and feature engineering, and model training services for training models at scale. Additionally, an effective data science environment should include utilities for managing and tracking different experimentation runs, enabling researchers to organize and monitor their experiments effectively. To manage artifacts such as source code and Docker images, the data scientists also need a code repository and Docker container repository. The following diagram illustrates a basic data science environment architecture that's using Amazon SageMaker and other supporting services:

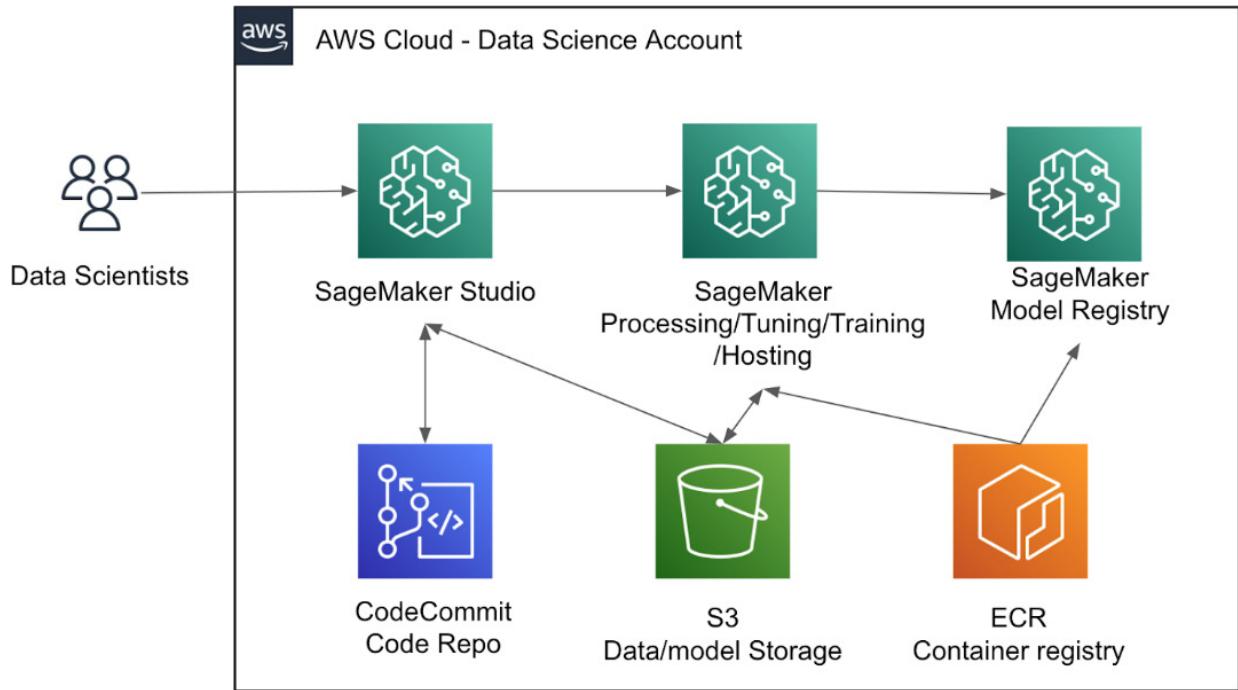


Figure 8.2: Data science environment architecture

SageMaker has multiple data science environments, including Studio which is the primary data science development environments for data scientists, RStudio for R users, and Canvas for users who want a no-code/low-code development environment for building machine learning models. You also have access to TensorBoard, a popular tool for monitoring and visualizing model metrics such as loss and accuracy. Now, let's explore how data scientists can leverage the different components of SageMaker to accomplish data science tasks.

Onboarding SageMaker users

The primary SageMaker user interface for data scientists is the SageMaker Studio, a data science **integrated development environment (IDE)**. It provides core features such as hosted notebooks for running experiments, as well as access to different backend services such as data wrangling, model training, and model hosting services from a single user interface. It is the main interface for data scientists to interact with most of SageMaker's functionality. It also provides a Python SDK for interacting with its backend services programmatically from Python notebooks or scripts. The following diagram shows the key components of SageMaker Studio:

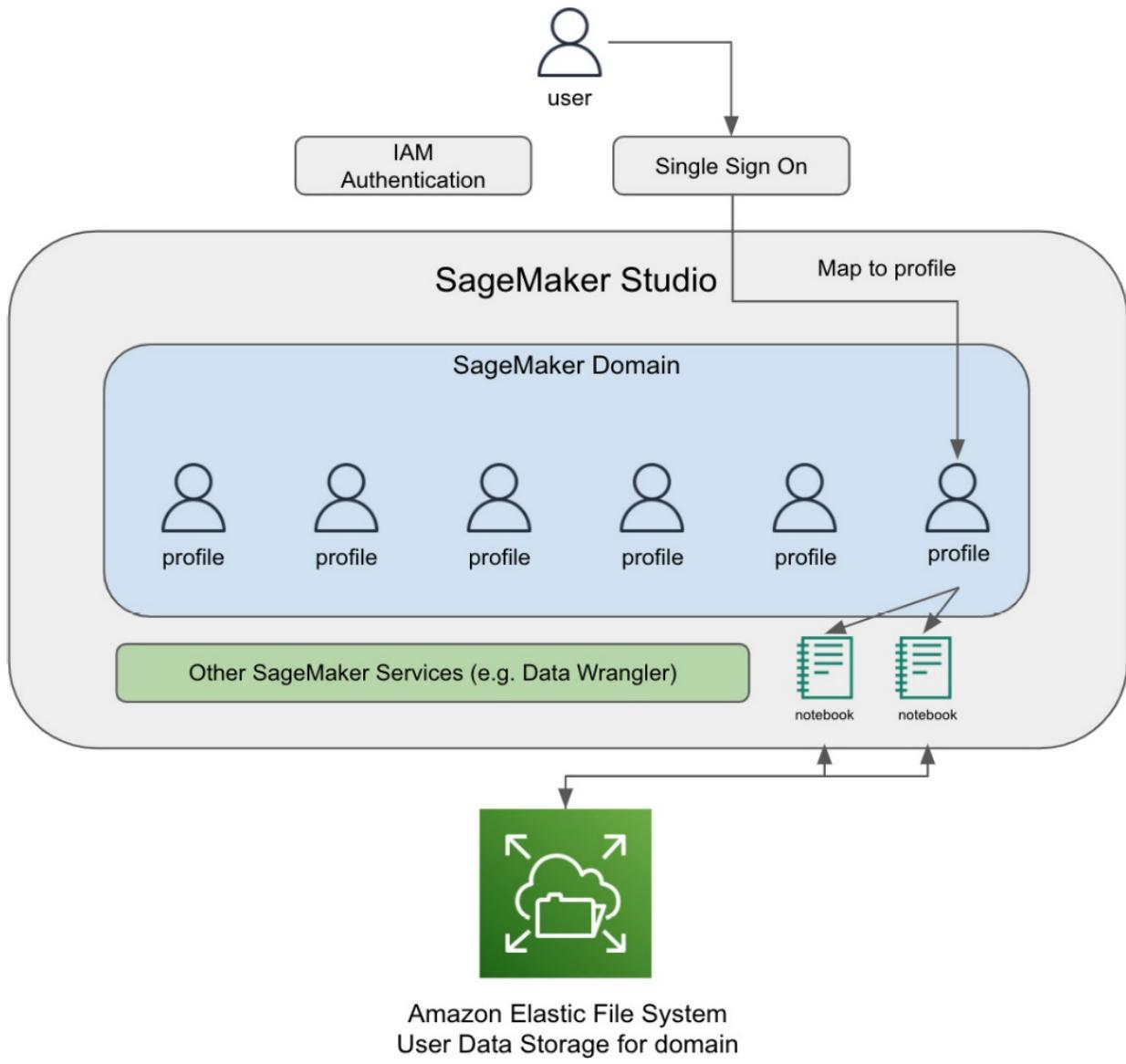


Figure 8.3: SageMaker Studio architecture

SageMaker Studio implements the concept of Domains to segregate user environments. A Domain represents a collection of user profiles, each equipped with specific configurations such as AWS IAM roles for Jupyter Notebook, settings for notebook sharing, and network connectivity options for connecting to AWS. Furthermore, each Domain encompasses one or more shared spaces, which consist of a shared JupyterServer application and a shared directory. All users within a Domain have access to this shared space, and all user profiles within the Domain can access all shared spaces. In your AWS account, you have the flexibility to create multiple domains in each AWS region. For every SageMaker domain, an Amazon Elastic File System (EFS) volume is generated and attached to the domain. This EFS volume serves as the central data storage location for all user data within the domain. Additionally, each user profile is mapped to a specific directory within the EFS volume, facilitating individualized data management. To begin using SageMaker, the initial step is to onboard users into SageMaker Studio. The onboarding process starts by creating a SageMaker Domain, if one doesn't already exist. Following that, user profiles are created within the Domain, and users are granted access to these profiles. Once the user profiles are set up, users can launch a Studio environment by selecting their respective user profile. This allows them to start working within SageMaker Studio.

Launching a notebook

Once you are within the SageMaker Studio environment, launching a notebook can be done by navigating to the **File -> Notebook** menu. Upon selecting this option, you will be prompted to choose a docker image for your notebook. This docker image contains pre-installed and pre-configured library packages such as Python, PyTorch, and TensorFlow. SageMaker has a selection of built-in images, stored in Amazon Elastic Container Registry (ECR). Alternatively, you have the flexibility to build a custom image that you import into the SageMaker environment. For detailed instructions on how to bring your own custom image, please refer to the documentation at <https://docs.aws.amazon.com/sagemaker/latest/dg/studio-byoi.xhtml>. When a user interacts with a notebook in SageMaker, they are essentially accessing a JupyterServer that runs on an EC2 instance within the SageMaker environment. This JupyterServer provides the user with an interactive notebook interface. The notebook interface then establishes connections to one or more docker images, also known as notebook kernels, running on separate EC2 hosts via a component called KernelGateway. KernelGateway also runs on the same remote EC2 hosts. The architecture can be visualized as follows:

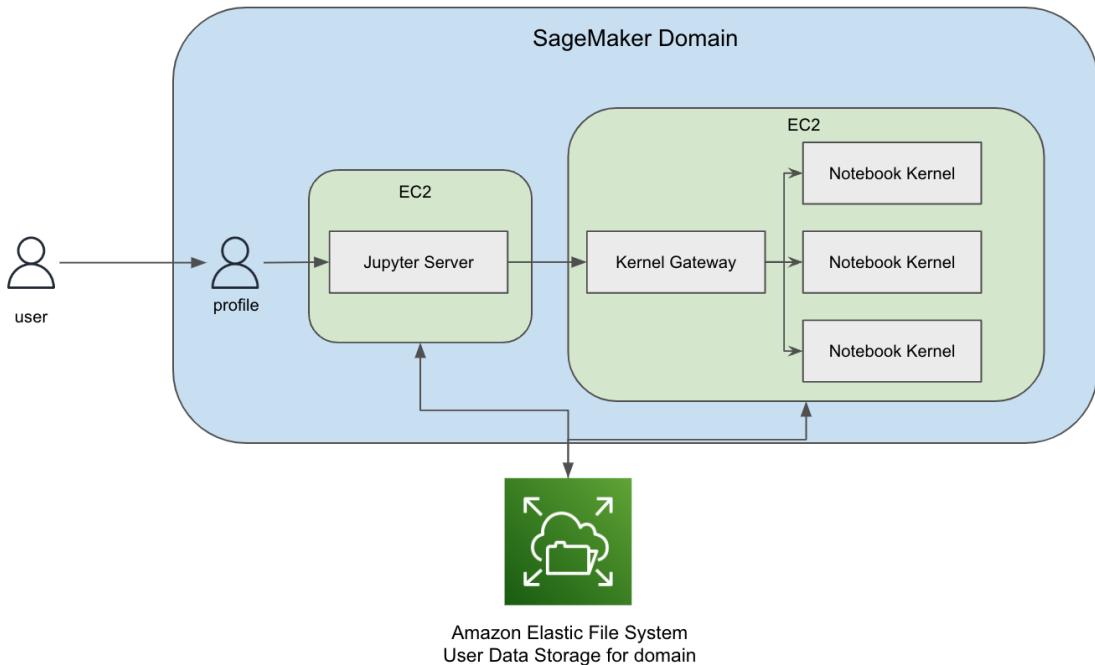


Figure 8.4: SageMaker Studio Notebook Architecture

This architecture enables the notebook interface to establish connections with multiple notebook kernels that run different images. These notebook kernels can be hosted on a single or multiple EC2 instances, which can have varying instance types to cater to different hardware requirements. Both the JupyterServer EC2 host and the notebook kernel hosts are linked to the EFS storage when the user directory is located. This allows for seamless data sharing and access between the notebook interface and the notebook kernels, regardless of the specific EC2 instances they are running on.

Preparing data

To prepare your data for feature engineering and model building, you can install and use your preferred library packages directly inside your Studio notebook. SageMaker also provides several data wrangling and processing services to assist with data preparation.

Preparing data interactively with SageMaker Data Wranglers

SageMaker Data Wrangler is a fully managed service that helps data scientists and engineers prepare and analyze their data for machine learning. With a graphical user interface, it facilitates data preparation tasks, such as data cleaning, feature engineering, feature selection, and visualization. To use Data Wrangler, you construct data flow, a pipeline that connects dataset, transformation, and analysis. When you create and run a data flow, Data Wrangler spins up an EC2 instance to run the transformation and analysis. Each data flow is associated with an EC2 instance. A data flow normally starts with a data import step. Data Wrangler allows you to import data from multiple data sources, including Amazon S3, Athena, Amazon Redshift, Amazon EMR, DataBricks, Snowflake, as well as Software-as-a-Service (SaaS) platforms such as Datadog, Githubs, and Stripe. After the data is imported, you can use data wrangler to clean and explore data, and perform feature engineering with built-in transform. You can also use the preconfigured visualization templates to understand data and detect outliers. You can also assess the data quality with built-in reports. The following diagram shows the flow and architecture of Data Wrangler:

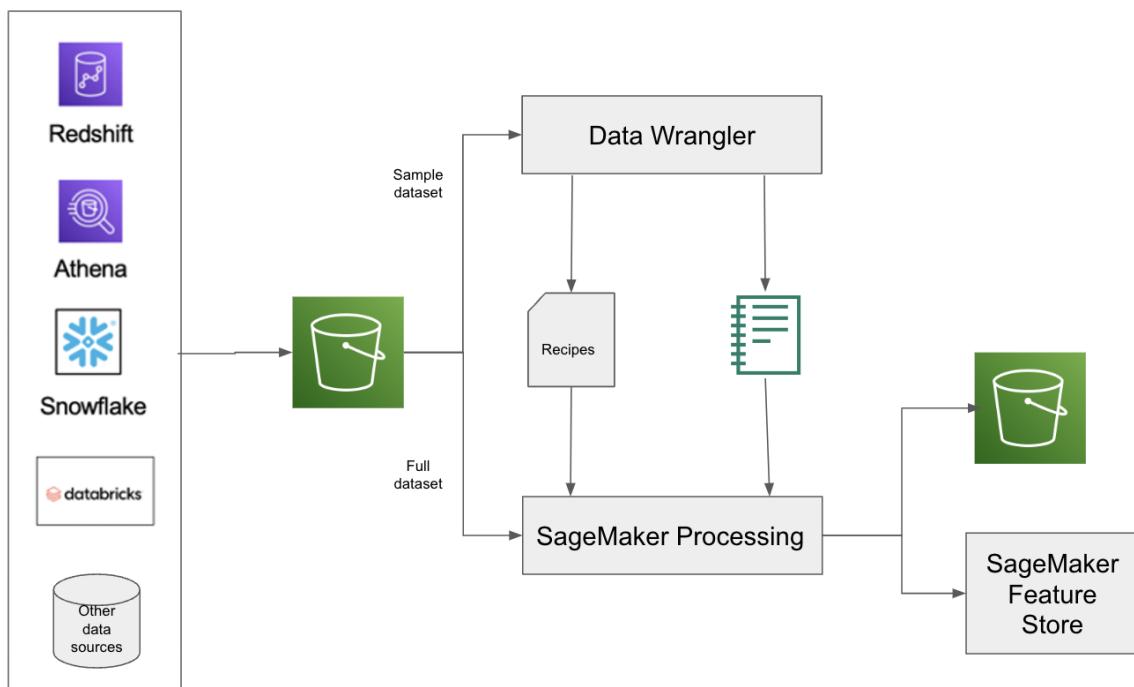


Figure 8.5: SageMaker Data Wrangler Architecture

When utilizing Data Wrangler, the first step involves importing sample data from various data sources to perform data processing and transformations. After completing the necessary transformation steps, you have the option to export a recipe. This recipe can then be executed by SageMaker Processing, which processes the entire dataset based on the defined transformations. Additionally, you have the option to export the transformation steps into a notebook file. This notebook file can be used to initiate a SageMaker processing job, allowing for the data to be processed and transformed. The resulting output can be directed to either the SageMaker FeatureStore or stored in Amazon S3 for further usage and analysis.

Preparing data at scale interactively

When dealing with large scale data analysis, transformation, and preparation tasks, SageMaker offers built-in integration with Amazon EMR and AWS Glue. This built-in integration allows you to manage and handle large-scale interactive data preparation. By leveraging Amazon EMR, you can process and analyze massive datasets, while AWS Glue provides a serverless capability to prepare data at scale as well as providing easy access to Glue

data catalogs. Within the Studio notebook environment, you have the capability to discover and establish connections with their existing Amazon EMR clusters. This enables them to interactively explore, visualize, and prepare large-scale data for machine learning tasks, leveraging powerful tools such as Apache Spark, Apache Hive, and Presto. The following diagram shows how SageMaker integration with EMR works.

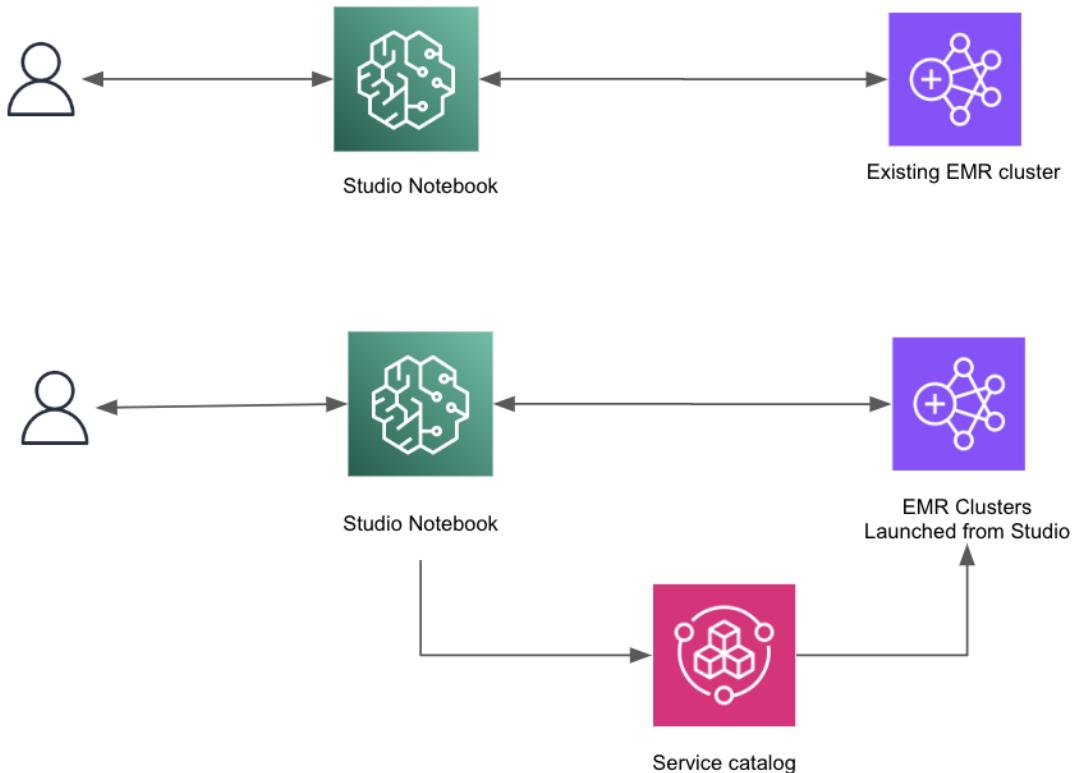


Figure 8.6: SageMaker integration with EMR

As a user, you connect to existing EMR cluster from your Studio notebooks. If there is no EMR cluster available, you can self-provision one directly from the Studio environment by choosing a pre-defined template created by system administrators. System administrators use AWS Service Catalog to define parameterized templates for data scientists to use. Once your notebook is connected to an EMR cluster, you can run a Spark commands or code inside your notebook cells. AWS Glue Interactive Sessions is a serverless service that provides you with the tools to collect, transform, cleanse, and prepare the data. The built-in integration between SageMaker and Glue Interactive Sessions allows you to run interactive sessions for data preparation interactively using the Glue as the backend.

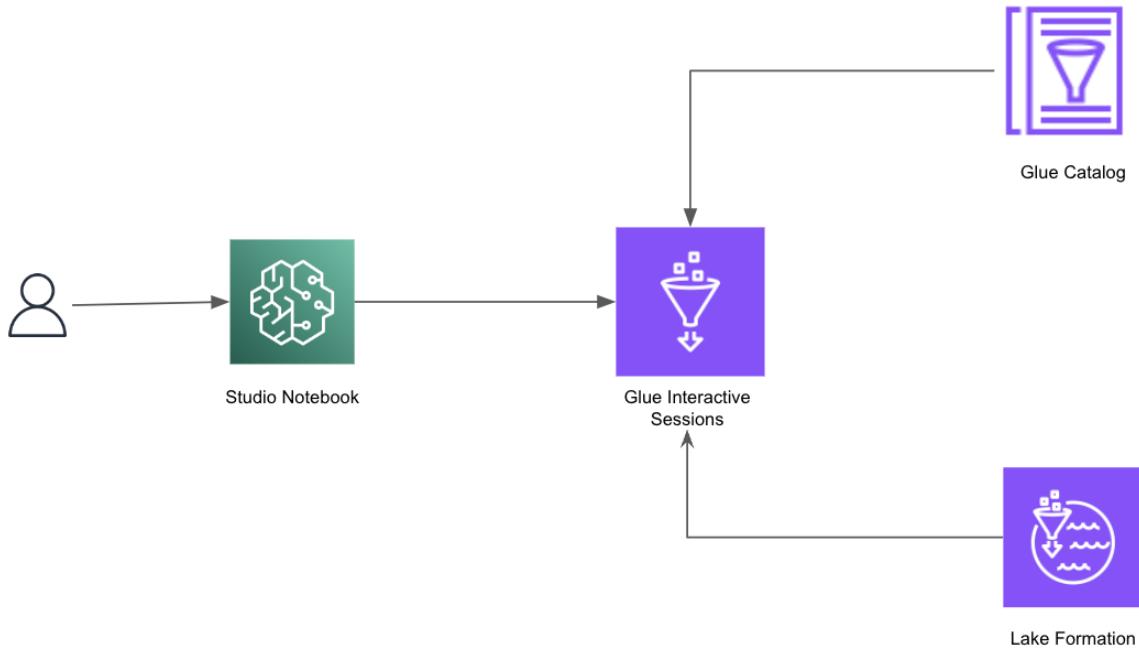


Figure 8.7: SageMaker integration with AWS Glue Interactive Session

To use Glue interactive sessions inside the Studio notebook, you choose the built-in Glue PySpark or Glue Spark kernel when you create your Studio notebook. After initialization, you can browse the Glue data catalog, run large queries, and interactively analyze and prepare data using Spark, all within your Studio notebook. Both EMR and Glue offer similar capabilities for large scale interactive data processing. Glue interactive session is a good choice if you are looking for quick and serverless way to run Spark sessions. EMR provides more robust capabilities and the flexibility to configure your compute cluster for optimization.

Processing data as separate jobs

SageMaker Processing provides a separate infrastructure for large-scale data processing such as data cleaning and feature engineering for large datasets as separate backend jobs. It can be accessed directly from a notebook environment via the SageMaker Python SDK or Boto3 SDK. SageMaker Processing uses Docker container images to run data processing jobs. Several built-in containers, such as scikit-learn containers and Spark containers, are provided out of the box. You also have the option to use your custom containers for processing. The following diagram shows the SageMaker Processing architecture:

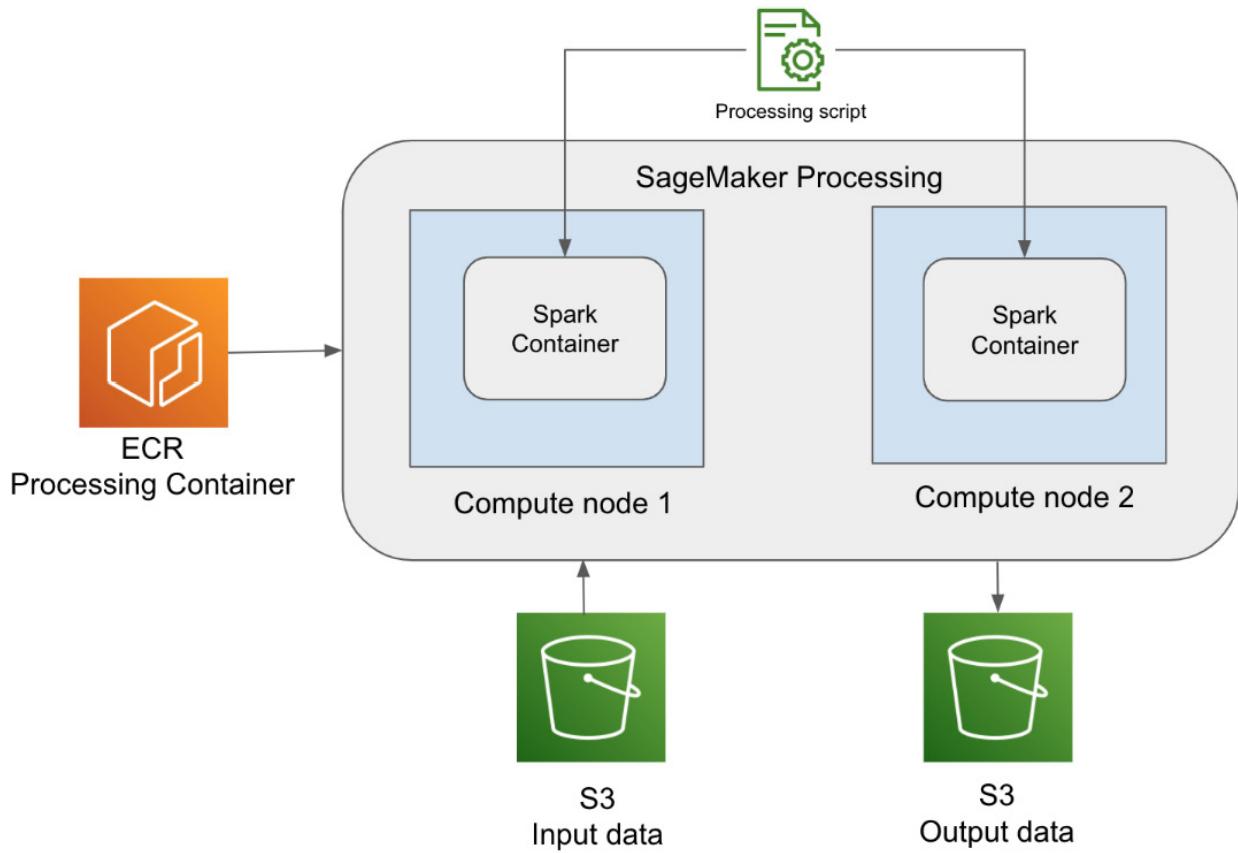


Figure 8.8: SageMaker Processing architecture

When a SageMaker Processing job is initiated, the processing container is pulled from Amazon ECR and loaded into the EC2 compute cluster. The data in S3 is copied over to the storage attached to the compute nodes for the data processing scripts to access and process. Once the processing procedure is completed, the output data is copied back to the S3 output location. SageMaker Processing provides several processors for data processing, including Spark processor, Scikit-learn processor, and your own customer processor by bringing your containers. SageMaker Processing also supports processors for the different machine learning frameworks, including PyTorch, TensorFlow, MXNet, Hugging Face, and XGBoost. You can use one of the processors if you need to use library packages as part of processing script.

Creating, storing, and sharing features

When data scientists perform feature engineering for training data preparation, they often need to reuse the same features for different model tasks. Further, features generated could be used for both training and inference to help reduce the training-serving skew. Amazon SageMaker Feature Store is a service for sharing and managing ML features for ML development and serving. Feature Store is a centralized store for features and associated metadata so features can be discovered and reused. It has an online component as well as an offline component. The online store is used for low latency real-time inference use cases, and the offline store is used for training and batch inference. The following diagram shows how feature store works. As you can see, it is quite similar to the architecture of other open-source alternatives such as Feast, with the key difference being fully managed.

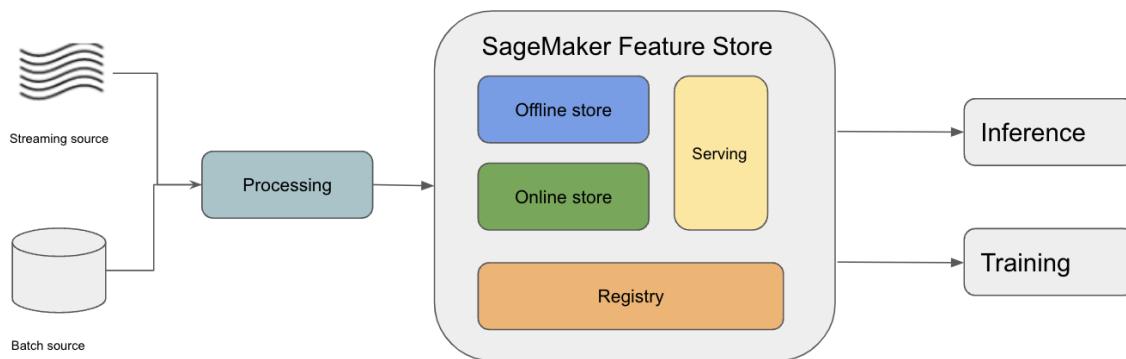


Figure 8.8: SageMaker Feature Store

To begin, you first read and process the raw data. The data is then ingested into online and offline store via streaming, or directly to the offline store via batch. Feature Store uses the concept called FeatureGroup to store and manage features. A FeatureGroup is a collection of features that is defined via a schema in Feature Store, describing the structure and metadata associate with a record. You can visualize a feature group as a table in which each column is a feature, with a unique identifier for each row. The online store is specifically optimized for real-time predictions, offering low-latency reads and high-throughput writes. It is ideal for scenarios that require quick access to feature data for real-time inference. On the other hand, the offline store is designed for batch predictions and model training purposes. It operates as an append-only store, allowing historical feature data to be stored and accessed. The offline store is particularly useful for storing and serving features during exploration and model training processes.

Training machine learning models

Once you have prepared the training data, you are all set to train the model. The SageMaker Training Service offers dedicated infrastructure for model training. Data scientists can use SageMaker Training service to handle model training that require dedicated training infrastructure and instance types. Training Service is also ideal for large scale distributed training using multiple nodes. To start training, you first store your training data in a storage such as Amazon S3, Amazon EFS, or Amazon FSx. You choose different storage option based your specific requirements, such as cost and latency. S3 is the most common one, suitable for the majority of the model training needs. With S3, you have multiple modes to ingest data from S3 to the training infrastructure:

- **File mode:** This the default input mode whereby SageMaker downloads the training data from the S3 to a local directory in the training instance. The training starts when the full dataset has been downloaded. With this mode, the training instance must have sufficient local storage space to fit the entire dataset.
- **Pipe mode:** With this mode, data is streamed directly from a S3 data source. This can provide faster start times and better throughput than the file mode. This model also reduce the size of the storage volumes attached to the training instances. It only needs enough space to store the final model artifacts.
- **Fast file mode:** This mode is the newer and simple-to-use replacement of pipe mode. At start of the training, the data files are identified but not downloaded. Training can start without waiting for the entire dataset to download. Fast file mode exposes S3 objects using a POSIX-compliant file system interface, as if the files are available on the local disk of the training instances. It streams the S3 data on demand as the training script consumes them, meaning that you don't need fit the training data into the training instance storage.

In addition to S3, you can also store your training data in Amazon FSx for Lustre. You want to use FSx when you need high throughput and low latency data retrieval requirements for your training data. With FSx, you can scale to hundreds of gigabytes of throughput and millions of Inputs/Outputs Operations Per Second (IOPS) with low-

latency file retrieval. When a training job is started, it mounts the FXs to the training instance file system as local drive. FSx allows the training job to run faster as it takes less time to read the files and it does not need to copy data to training instance local storage like S3 file mode. EFS provides similar functionality as FSx, but at a lower throughput and higher latency. If you already have data in your EFS system, you can directly launch a training job using data in the EFS without data movement. This reduces the training start time. Comparing to FSx, EFS is less costly but has lower throughput and higher latency. Once the training data is ready in the storage system of your choice, you can kick off the training job using the AWS Boto3 SDK or the SageMaker Python SDK. To run the training job, you need to provide configuration details such as the training Docker image's URL from the ECR, the training script location, framework version, training dataset location, and S3 model output location, as well as infrastructure details such as the compute's instance type and number, as well as networking details. The following sample code shows how to configure and kick off a training job using the SageMaker SDK. SageMaker Training service use containers as a core technology for training management. All training jobs are executed inside containers hosted on SageMaker training infrastructure. The following diagram shows the architecture of SageMaker Training Service:

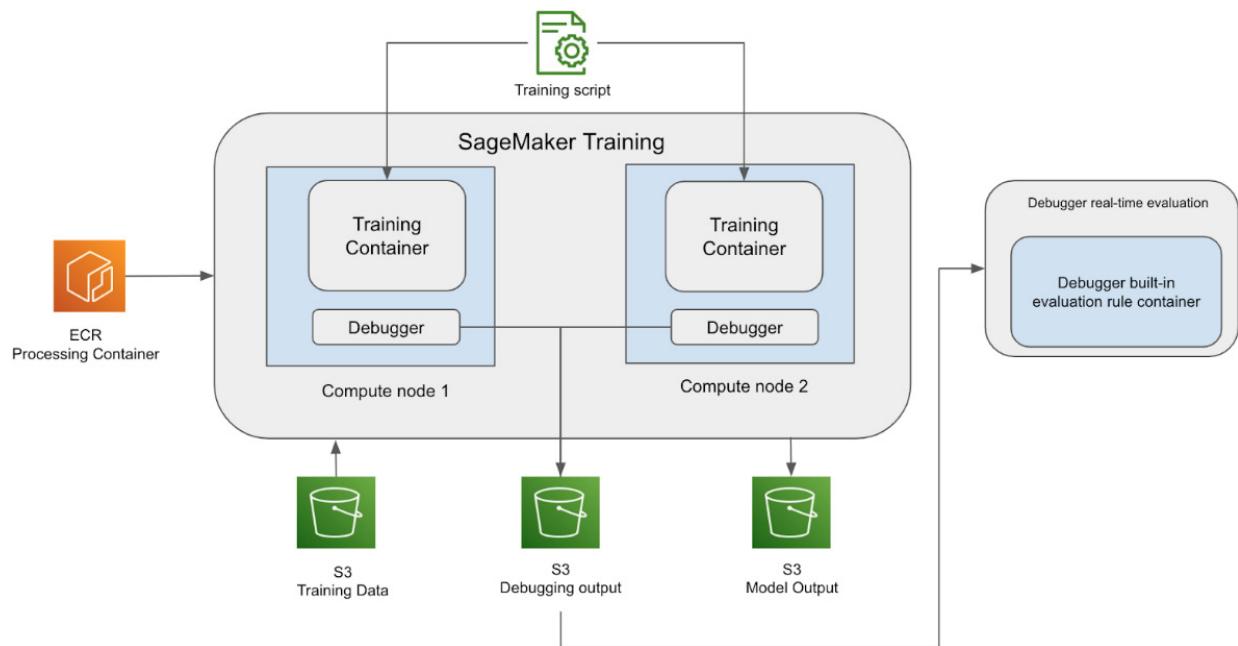


Figure 8.9: SageMaker Training Service architecture

SageMaker provides various managed containers for model training using different machine learning algorithms and machine learning framework. Firstly, there is a list of built-in containerized algorithms for different machine learning tasks such as computer vision, NLP, forecasting, and common tabular regression and classifications. With these built-in algorithms, you only need to provide training data location. SageMaker also provides a list of managed framework containers such as containers for scikit-learn, TensorFlow, and PyTorch. With a managed framework container, in addition to providing data sources and infrastructure specifications, you also need to provide a training script that runs the model training loop. If the built-in algorithms and framework containers do not meet your need, you can bring your own custom container for model training. This container needs to contain the model training scripts, as well as all the dependencies required to run the training loop. By default, SageMaker tracks all training jobs and their associated metadata, such as algorithms, input training dataset URLs, hyperparameters, and model output locations. Training jobs also emit system metrics and algorithm metrics to AWS CloudWatch for monitoring. Training logs are also sent to CloudWatch logs for inspection and analysis needs. These metadata are critical for lineage tracking and reproducibility.

Tuning machine learning models

To optimize the model's performance, you also need to try different hyperparameters, such as a learning rate for gradient descent and model training. An algorithm can contain a large number of hyperparameters, and tuning them manually would be a highly labor-intensive task. The SageMaker Tuning service works with SageMaker training jobs to tune model training hyperparameters automatically. The following 4 types of hyperparameter tuning strategies are supported by the service:

- **Grid search:** Grid Search is an exhaustive search technique that systematically explores a predefined set of hyperparameter values over a specified range for each hyperparameter. It creates a grid of all possible combinations and evaluates the model's performance for each configuration using cross-validation or a validation set. Grid search is highly focused, but it can be highly inefficient due to large number of combinations, especially when hyperparameter dimension is high.
- **Random search:** Random Search is a popular and effective hyperparameter optimization technique that offers an alternative to exhaustive methods like Grid Search. Unlike Grid Search, which evaluates all possible combinations of hyperparameters within predefined ranges, Random Search takes a more stochastic approach. Instead of covering the entire search space systematically, it randomly samples hyperparameter values from defined distributions for each hyperparameter. Random search can be more efficient, compared to grid search, however it might not always find the best combination of hyperparameters.
- **Bayesian search:** This is where the hyperparameter search is treated like a regression problem, where the inputs for regression are the values of the hyperparameters and the output is the model's performance metric once the model has been trained using the input values. The Tuning service uses the values that have been collected from the training jobs to predict the next set of values that would produce model improvement. Comparing to Random search, Bayesian search is more efficient as it uses a model to focus on the most promising search spaces of hyperparameters.
- **Hyperband:** Hyperband leverages the concept of bandit algorithms and successive halving to make the search process more effective and resource-efficient. It begins by randomly sampling a large number of hyperparameter configurations and then divides them into multiple "bands" or sets. In each band, the configurations are evaluated through a predefined number of iterations, eliminating poorly performing ones at regular intervals. The surviving configurations are then promoted to the next band, where they are given additional iterations to fine-tune their performance. This process continues, gradually increasing the resources allocated to promising configurations while efficiently discarding underperforming ones. Hyperband is known to be more efficient than other methods, and it can find good combinations of hyperparameters with fewer iterations.

The SageMaker Tuning service works with SageMaker training jobs to optimize the hyperparameters. It works by sending different input hyperparameter values to the training jobs and picking the hyperparameter values that return the best model metrics. The following diagram shows how SageMaker Tuning Service works:

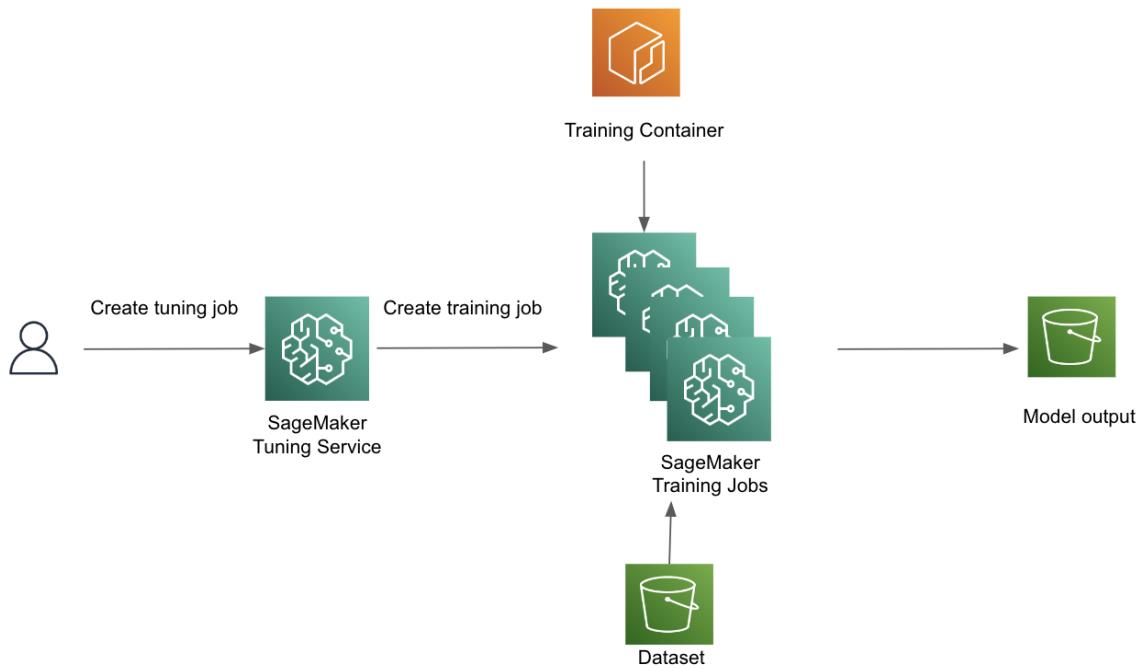


Figure 8.10: SageMaker Tuning Architecture

To use the SageMaker Tuning Service, you create a tuning job and specify configuration details such as tuning strategy, objective metric to optimize, hyperparameters to tune and their ranges, and max number of training jobs to run and the number of jobs to run in parallel. Subsequently, the tuning job will kick off a number of training jobs. Depending on the tuning strategy, the tuning job will pass different hyperparameters to the training jobs to execute. The training metrics from the training jobs will be used by the tuning job to determine what hyperparameters to use in order to optimize the model performance.

Deploying machine learning models for testing

Data scientists normally do not deploy models for client application consumption directly. However, data scientists sometimes need to test the performance of models trained with SageMaker training service, and they will need to deploy these models to an API endpoint for testing. This is especially needed for models of large sizes where they can not be evaluated in a notebook instance. SageMaker provides dedicated service for model hosting and its architecture is illustrated below:

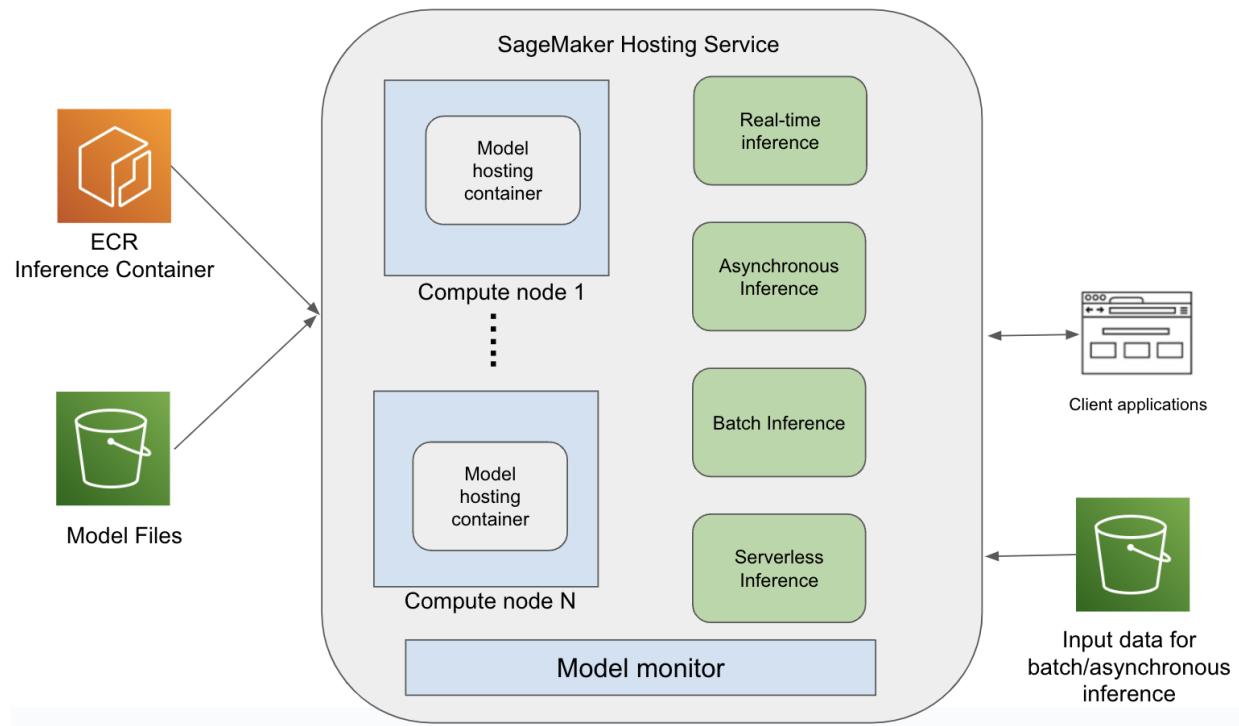


Figure 8.11: SageMaker hosting architecture.

SageMaker Hosting service offers multiple model inference options for different needs, from real-time inference to batch inference. The following are some options available for data scientists to use for model hosting evaluation and testing. One of the most common model serving use case is low-latency prediction in real-time on a sustained basis. For this use case, you should consider real-time inference option from SageMaker hosting service. SageMaker real-time inference provides multiple options for model hosting including single model hosting, multiple models hosting in a single container behind one endpoint, multiple models hosting using different containers behind one endpoint. Sometimes, you have models that used for intermittent predictions with idle periods between traffic burst. For this type of requirement, you can consider the Serverless option from SageMaker hosting service. The key benefits of using Serverless Inference is the removal of infrastructure configuration and management overhead. It is also more cost effective since they don't need to pay for infrastructure when it is not used, unlike the real-time inference which you need to pay for the infrastructure whether there is inference traffic or not. However, there might a delay caused by cold-start with Serverless Inference. Sometimes, the payload size of inference can be very large, and it takes long time to generate the prediction. In this case, real-time endpoint can not work due to large payload size and extended time for the inference. For this type of use case, you can consider the asynchronous inference option of SageMaker hosting. Asynchronous inference queues the incoming request and process them asynchronously. When using Asynchronous inference, the input data and prediction output is stored in S3 instead of sending the payload and getting the response directly from the endpoint API. When prediction is complete, you get a notification from the AWS SNS service. Another model inference pattern is batch inference. This is when you have large number of inference to do, and they don't need individual prediction to be generated and returned. For example, you might need to run a propensity-to-buy models for large number of users and store the output in a database for downstream consumption. For this usage pattern, you can consider the SageMaker Batch Transform feature for model inference. Batch inference is also more cost effective as you only need to spin up the infrastructure when running the batch job.

Automating experimentation and model building

Data scientists frequently engage in iterative experimentation and model development, involving different datasets, new features, and various training scripts. Keeping track of configurations and model metrics for each run becomes essential. To streamline and automate these repetitive tasks, automation pipelines can be constructed, supporting various ML processes like data processing, model training, and model testing. There are several tools available for automation and orchestration, including SageMaker's Pipeline feature. It allows the creation of a Directed Acyclic Graph (DAG) to efficiently orchestrate and automate ML workflows. SageMaker Pipeline shares similarities with Airflow, an open-source workflow orchestration tool discussed in Chapter 7, Open Source ML Platform. Additionally, AWS Step Functions serve as an alternative option for building automated workflow orchestration, providing flexibility and scalability to accommodate diverse ML tasks. By leveraging these tools, data scientists can enhance efficiency, reproducibility, and organization in their ML workflows.

Best practices for building data science environment

The data science environments are meant for data scientists to perform quick experimentations using a wide range of ML framework and libraries. The followings are some best practices to follow when providing such an environment for your data scientists.

- **Run large scale model training using the SageMaker Training Service instead of Studio Notebook:** SageMaker Studio Notebook is meant for quick experimentations with small dataset. While it is possible to provisioning large EC2 instances for certain large model training job, it is not cost effective to always keep a large EC2 running for a Noteboook all the time.
- **Abstract infrastructure configuration details from data scientists:** There are many infrastructure configurations to consider when using SageMaker, such as networking configuration, IAM roles, encryption keys, EC2 instance types, and storage options. To make the life of data scientists easier, abstract these details away from the data scientists. For example, instead of having the data scientists enter specific networking configuration, having those details stored as environment variables or custom SDK options for data scientists to choose from.
- **Create self-service provisioning:** To prevent provisioning bottleneck, consider building self-service provisioning capability to streamline user onboarding. For example, using AWS Service Catalog to create ML product for automated user onboarding.
- **Use Studio Notebook local mode for quick model training job testing:** SageMaker has support for local mode, meaning you can mimic running training job locally in your Studio Notebook. With SageMaker Training Jobs, there is an overhead of spinning up separate infrastructure. Running these tests locally can help speed up experimentation.
- **Setting up guardrails:** To prevent data scientists from making mistakes such as using wrong instance types for model training, or forgot to use data encryption keys. You can use AWS Service Control Policies to help with guardrail management.
- **Clean up unused resources:** Regularly review and clean up unused notebooks, endpoints, and other resources to avoid unnecessary cost.
- **Use Spot instances:** For cost optimization, consider using Amazon EC2 Spot Instances for training jobs if possible. Spot Instances can significantly reduce training costs while maintaining high performance.
- **Use built-in algorithms and managed containers for training:** SageMaker provides a list of built-in algorithms for different ML tasks and managed training containers for different ML frameworks. Taking advantage of these pre-existing resources can substantially reduce the engineering efforts required, eliminating the need to build your own algorithms from scratch.
- **Build automated pipelines for repeatable ML experimentation, model building, and model testing:** Having an automate pipeline can greatly reduce the manual effort and improve the tracking of different experiments. Consider different orchestration technology options based on your technology standards and preferences.

By adhering to these best practices, you can make the most of SageMaker Studio's capabilities, streamline your machine learning workflows, and ensure cost-effective and secure usage of the platform.

Hands-on exercise – building a data science environment using AWS services

In this hands-on exercise, you will create a data science environment using SageMaker with AWS CodeCommit as the source control.

Problem statement

As an ML Solutions Architect, you have been tasked with building a data science environment on AWS for the data scientists in the Equity Research department. The data scientists in the Equity Research department have several NLP problems, such as detecting the sentiment of financial phrases. Once you have created the environment for the data scientists, you also need to build a proof of concept to show the data scientists how to build and train an NLP model using the environment.

Dataset

The data scientists have indicated that they like to use the BERT model to solve sentiment analysis problems, and they plan to use the financial phrase dataset to establish some initial benchmarks for the model:

<https://www.kaggle.com/ankurzing/sentiment-analysis-for-financial-news>.

Lab instructions

Follow these steps to create a data science environment using SageMaker with AWS CodeCommit as the source control.

Setting up SageMaker Studio

Follow these steps to set up a SageMaker Studio environment:

- To create a SageMaker Studio environment, we need to set up a domain and a user profile in the respective AWS region. Navigate to the SageMaker management console once you've logged into AWS Management Console and click on the **Studio** link on the left.
- On the right-hand side of the screen, click on **Create a SageMaker Domain** button. Provide a domain name of your choice, accept the default profile name or enter a new name. For execution role, select **Create a new role** from the dropdown. For the purpose of this lab, select **Any S3 bucket** for the purpose of this lab on the next screen and create the role.
- Click on the **Submit** button to set up the domain and user. Select a VPC and Subnets on the next screen. It will take a few minutes for the domain to be set up.

Additionally, the following resources are also created behind the scenes:

- **An S3 bucket for the domain:** The name of the bucket should look something like `sagemaker-studio-<AWS account number>-xxxx`. This bucket can be used for storing datasets and model artifacts.
- **Elastic File System volume:** This volume is used by the Studio domain for storing user data. If you navigate to the EFS management console, you should see that a new filesystem has been created.

To start the Studio environment for the newly created user, click on the Studio link again, select the user profile you just created and click on the Open Studio to launch Studio. It will take a few minutes for the Studio environment to appear. Once everything is ready, you will see a screen that is similar to the following:

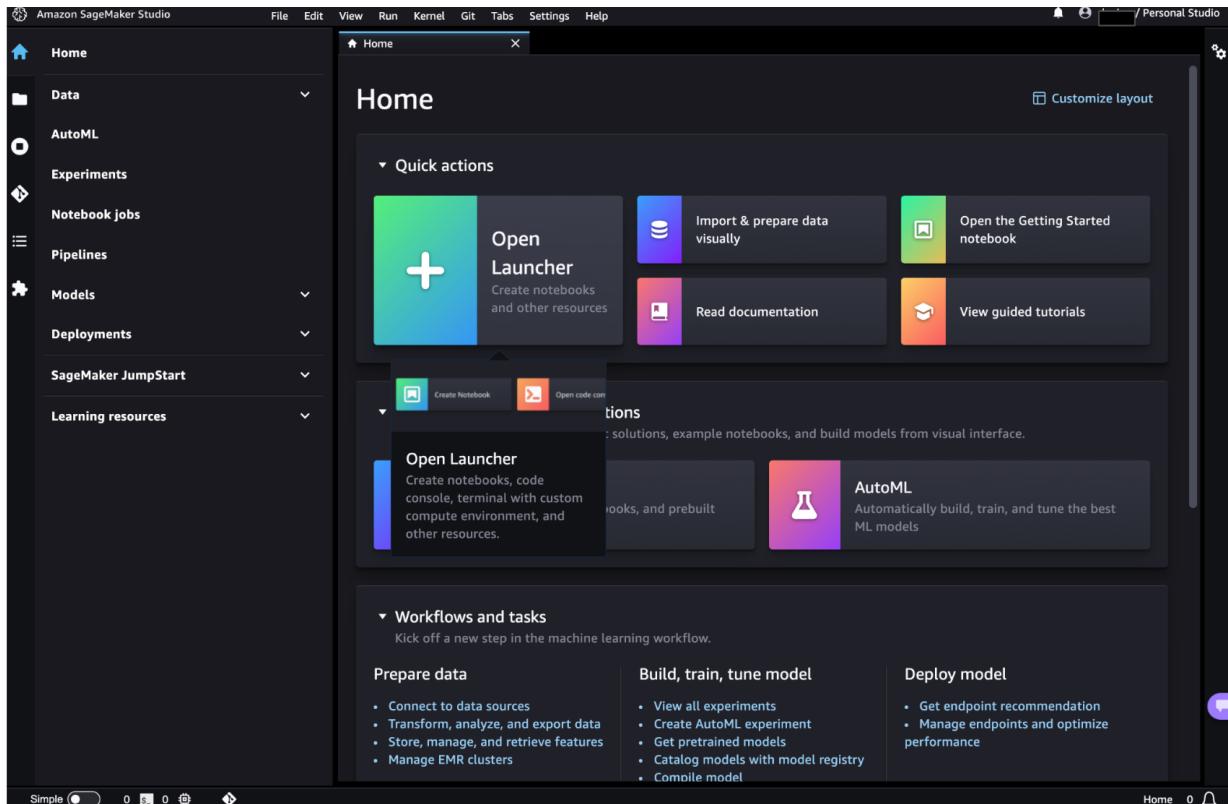


Figure 8.6: Studio UI

Training the BERT model in the Jupyter Notebook

In this part of the hands-on exercise, we will train a financial sentiment analysis NLP model using the BERT transformer, which we learned about in *Chapter 3, Machine Learning Algorithms*. To get started, double-click on the newly cloned folder in the folder view and create a new notebook to author our code by selecting **File > New > Notebook** from the menu dropdown in the folder. When prompted to select a kernel, pick **PyTorch 2.0.0 Python 3.10 GPU Optimized**. You can rename the file so that it has a more meaningful name by selecting **File > Rename Notebook** from the menu. It will take a few minutes for the notebook to be available the first time since it needs to provision a new EC2 instance for the notebook. We will use the financial news sentiment dataset for model training. Download the dataset from Kaggle at <https://www.kaggle.com/ankurzing/sentiment-analysis-for-financial-news>. Note that you will need a Kaggle account to download it. Once it's been downloaded, you should see an `archive.zip` file. Next, let's upload the `archive.zip` file to the Studio notebook. Create a new folder called `data` in the same folder where the new notebook is located and upload it to the `data` directory using the **File Upload** utility (the up arrow icon) in Studio UI. You should see several files inside the folder. Select `all_data.csv` to upload. Now, let's install some additional packages for our exercise. Run the following code block inside the notebook cell to install the transformer package. The transformer package provides a list of pre-trained transformers such as BERT. You will use these transformers to fine-tune an ML task. Note that some of the code block samples are not complete. You can find the complete code samples at <https://github.com/PacktPublishing/The-Machine-Learning-Solutions-Architect-Handbook/blob/main/Chapter08/bert-financial-sentiment.ipynb>:

```
!pip install transformers
!pip install ipywidgets
```

Restart the kernel of the notebook after installing `ipywidgets`. Next, import some libraries into the notebook and set up the logger for logging purposes:

```

import logging
import os
import sys
import numpy as np
import pandas as pd
import torch
from torch.utils.data import DataLoader, TensorDataset
from transformers import AdamW, BertForSequenceClassification, BertTokenizer
from sklearn.preprocessing import OrdinalEncoder
from sklearn.model_selection import train_test_split
from types import SimpleNamespace
logger = logging.getLogger(__name__)
logger.setLevel(logging.DEBUG)
logger.addHandler(logging.StreamHandler(sys.stdout))

```

Now, we are ready to load the **data** file and process it. The following code block loads the **data** file and splits the data into train and test datasets. We will select the first two columns from the file and name them `sentiment` and `article`. The `sentiment` column is the label column. It contains three different unique values (`negative` , `neutral` , and `positive`). Since they are string values, we will convert them into integers (0,1,2) using `OrdinalEncoder` from the scikit-learn library. We also need to determine the max length of the article column. The max length is used to prepare the input for the transformer since the transformer requires a fixed length:

```

filepath = './data/all-data.csv'
data = pd.read_csv(filepath, encoding="ISO-8859-1",
header=None, usecols=[0, 1],
names=["sentiment", "article"])
ord_enc = OrdinalEncoder()
data["sentiment"] = ord_enc.fit_transform(data[["sentiment"]])
data = data.astype({'sentiment':'int'})
train, test = train_test_split(data)
train.to_csv("./data/train.csv", index=False)
test.to_csv("./data/test.csv", index=False)
MAX_LEN = data.article.str.len().max() # this is the max length of the sentence

```

Next, we will build a list of utility functions to support the data loading and model training. We need to feed data to the transformer model in batches. The following `get_data_loader()` function loads the dataset into the PyTorch `DataLoader` class with a specified batch size. Note that we also encode the articles into tokens with the `BertTokenizer` class:

```

def get_data_loader(batch_size, training_dir, filename):
    logger.info("Get data loader")
    tokenizer = BertTokenizer.from_pretrained("bert-base-uncased", do_lower_case=True)
    dataset = pd.read_csv(os.path.join(training_dir, filename))
    articles = dataset.article.values
    sentiments = dataset.sentiment.values      input_ids = []
    for sent in articles:
        encoded_articles = tokenizer.encode(sent, add_special_tokens=True)
        input_ids.append(encoded_articles)
    ...
    return tensor_dataloader

```

The following `train()` function will run the training loop using the `BertForSequenceClassification` class. We will use the pre-trained BERT model for fine-tuning, instead of training from scratch. We will feed one batch of data to the BERT model at a time. Note that we will also check if there is a GPU device on the server. If there is one, we will use the `cuda` device for GPU training, instead of `cpu` for CPU training. We need to manually move the data and BERT model to the same target device using the `.to(device)` function so that the training can happen on the target device with the data residing in memory on the same device. The optimizer we're using here is AdamW, which is a variant of the gradient descent optimization algorithm. The training loop will run through the number of epochs specified. One epoch runs through the entire training dataset once:

```

def train(args):
    use_cuda = args.num_gpus > 0

```

```

device = torch.device("cuda" if use_cuda else "cpu")
# set the seed for generating random numbers
torch.manual_seed(args.seed)
if use_cuda:
    torch.cuda.manual_seed(args.seed)
train_loader = get_data_loader(args.batch_size, args.data_dir, args.train_file)
test_loader = get_data_loader(args.test_batch_size, args.data_dir, args.test_file)
model = BertForSequenceClassification.from_pretrained(
    "bert-base-uncased",
    num_labels=args.num_labels,
    output_attentions=False,
    output_hidden_states=False, )
...
return model

```

We also want to test the model's performance using a separate test dataset during training. To do this, we will implement the following `test()` function, which is called by the `train()` function:

```

def test(model, test_loader, device):
    def get_correct_count(preds, labels):
        pred_flat = np.argmax(preds, axis=1).flatten()
        labels_flat = labels.flatten()
        return np.sum(pred_flat == labels_flat), len(labels_flat)

    model.eval()
    _, eval_accuracy = 0, 0
    total_correct = 0
    total_count = 0
    ...
    logger.info("Test set: Accuracy: %f\n", total_correct/total_count)

```

Now, we have all the functions needed to load and process data, run the training loop, and measure the model metrics using a test dataset. With that, we can kick off the training process. We will use the `args` variable to set up various values, such as batch size, data location, and learning rate, to be used by the training loop and the testing loop:

```

args = SimpleNamespace(num_labels=3, batch_size=16, test_batch_size=10, epochs=3, lr=2e-5, seed=42)
model = train(args)

```

Once you have run the preceding code, you should see training stats for each batch and epoch. The model will also be saved in the specified directory. Next, let's see how the trained model can be used for making predictions directly. To do this, we must implement several utility functions. The following `input_fn()` function takes input in JSON format and outputs an input vector that represents the string input and its associated mask. The output will be sent to the model for prediction:

```

def input_fn(request_body, request_content_type):
    if request_content_type == "application/json":
        data = json.loads(request_body)
        if isinstance(data, str):
            data = [data]
        elif isinstance(data, list) and len(data) > 0 and isinstance(data[0], str):
            pass
        else:
            raise ValueError("Unsupported input type. Input type can be a string or a non-empty list. I got {}".format(data))

    tokenizer = BertTokenizer.from_pretrained("bert-base-uncased", do_lower_case=True)

    input_ids = [tokenizer.encode(x, add_special_tokens=True) for x in data]

    # pad shorter sentence
    padded = torch.zeros(len(input_ids), MAX_LEN)
    for i, p in enumerate(input_ids):
        padded[i, :len(p)] = torch.tensor(p)

```

```

# create mask
mask = (padded != 0)

return padded.long(), mask.long()
raise ValueError("Unsupported content type: {}".format(request_content_type))

```

The following `predict_fn()` function takes `input_data` returned by `input_fn()` and uses the trained model to generate the prediction. Note that we will also use a GPU if a GPU device is available on the server:

```

def predict_fn(input_data, model):
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model.to(device)
    model.eval()
    input_id, input_mask = input_data
    input_id = input_id.to(device)
    input_mask = input_mask.to(device)
    with torch.no_grad():
        y = model(input_id, attention_mask=input_mask)[0]
    return y

```

Now, run the following code to generate a prediction. Replace the value of the article with different financial text to see the result:

```

import json
print("sentiment label : " + str(np.argmax(preds)))
article = "Operating profit outpaced the industry average"
request_body = json.dumps(article)
enc_data, mask = input_fn(request_body, 'application/json')
output = predict_fn((enc_data, mask), model)
preds = output.detach().cpu().numpy()
print("sentiment label : " + str(np.argmax(preds)))

```

Now, let us look at an alternative way to train the BERT model.

Training the BERT model with SageMaker Training Service

In the previous section, you trained the BERT model directly inside a GPU-based Jupyter Notebook. Instead of provisioning a GPU-based notebook instance, you can provision a less costly CPU-based instance and send the model training task to SageMaker Training Service. To use SageMaker Training Service, you need to make some minor changes to the training script and create a separate launcher script to kick off the training. As we discussed in the *SageMaker Training Service* section, there are three main approaches to training a model in SageMaker. Since SageMaker provides a managed container for PyTorch, we will use the managed container approach to train the model. With this approach, you will need to provide the following inputs:

- A training script as the entry point, as well as dependencies
- An IAM role to be used by the training job
- Infrastructures such as the instance type and number
- A data (training/validation/testing) location in S3
- A model output location in S3
- Hyperparameters for training the model

When a training job is started, SageMaker Training Service will perform the following tasks in sequence:

1. Launch the EC2 instances needed for the training job.
2. Download the data from S3 to the training host.
3. Download the appropriate managed container from the SageMaker ECR registry and run the container.
4. Copy the training script and dependencies to the training container.
5. Run the training script and pass the hyperparameters as command-line arguments to the training script. The training script will load the training/validation/testing data from specific directories in the container, run the

training loop, and save the model to a specific directory in the container. Several environment variables will be set in the container to provide configuration details, such as directories for the data and model output, to the training script.

Once the training script exits with success, SageMaker Training Service will copy the saved model artifacts from the container to the model output location in S3. Now, let's create the following training script, name it **train.py**, and save it in a new directory called `code`. Note that the training script is almost the same as the code in *Training the BERT model in the Jupyter Notebook* section. We have also added an `if __name__ == "__main__":` section at the end. This section contains the code for reading the values of the command-line arguments and the values of the system environment variables such as SageMaker's data directory (`SM_CHANNEL_TRAINING`), the model output directory (`SM_MODEL_DIR`), and the number of GPUs (`SM_NUM_GPUS`) available on the host. Note that the following code sample is not complete. You can find the complete code sample at <https://github.com/PacktPublishing/The-Machine-Learning-Solutions-Architect-Handbook/blob/main/Chapter08/code/train.py>:

```
import argparse
import logging
import os
import sys
import numpy as np
import pandas as pd
import torch
from torch.utils.data import DataLoader, TensorDataset
from transformers import AdamW, BertForSequenceClassification, BertTokenizer
logger = logging.getLogger(__name__)
logger.setLevel(logging.DEBUG)
logger.addHandler(logging.StreamHandler(sys.stdout))
...
    train(parser.parse_args())
```

The preceding script requires library packages that are not available in the managed training container. You can install custom library packages using the `requirement.txt` file. Create a `requirement.txt` file with the following code and save it in the `code` directory:

```
transformers==2.3.0
```

Next, let's create a launcher notebook for kicking off the training job using SageMaker Training Service. The launcher notebook will do the following:

- Upload the training and test dataset to the S3 bucket and folders.
- Set up SageMaker PyTorch Estimator using the SageMaker SDK to configure the training job.
- Kick off the SageMaker training job.

Create a new notebook called `bert-financial-sentiment-launcher.ipynb` in the folder where the `code` folder is located and copy the following code block into the notebook one cell at a time. When you're prompted to choose a kernel, pick the `Python 3` kernel and `Data Science` image. The following code specifies the S3 bucket to be used for saving the training and testing dataset, as well as the model artifacts. You can use the bucket that was created earlier in *Setting up SageMaker Studio* section, when the Studio domain was configured. The training and test dataset we created earlier will be uploaded to the bucket. The `get_execution_role()` function returns the IAM role associated with the notebook, which we will use to run the training job later:

```
import os
import numpy as np
import pandas as pd
import sagemaker
sagemaker_session = sagemaker.Session()
bucket = <bucket name>
prefix = "sagemaker/pytorch-bert-financetext"
role = sagemaker.get_execution_role()
inputs_train = sagemaker_session.upload_data("./data/train.csv", bucket=bucket, key_prefix=prefix)
inputs_test = sagemaker_session.upload_data("./data/test.csv", bucket=bucket, key_prefix=prefix)
```

Finally, we must set up the SageMaker PyTorch estimator and kick off the training job. Note that you can also specify the PyTorch framework version and Python version to set up the container. For simplicity, we are passing the name of the training file and test file, as well as max length, as hyperparameters. The `train.py` file can also be modified to look them up dynamically:

```
from sagemaker.pytorch import PyTorch
output_path = f"s3://{bucket}/{prefix}"
estimator = PyTorch(
    entry_point="train.py",
    source_dir="code",
    role=role,
    framework_version="1.6",
    py_version="py3",
    instance_count=1,
    instance_type="ml.p3.2xlarge",
    output_path=output_path,
    hyperparameters={
        "epochs": 4,
        "lr" : 5e-5,
        "num_labels": 3,
        "train_file": "train.csv",
        "test_file" : "test.csv",
        "MAX_LEN" : 315,
        "batch-size" : 16,
        "test-batch-size" : 10
    }
)
estimator.fit({"training": inputs_train, "testing": inputs_test})
```

Once the training job has been completed, you can go to the SageMaker management console to access the training job's details and metadata. Training jobs also send outputs to CloudWatch logs and CloudWatch metrics. You can navigate to these logs by clicking on the respective links on the training job detail page.

Deploying the model

In this step, we will deploy the trained model to a SageMaker RESTful endpoint so that it can be integrated with downstream applications. We will use the managed PyTorch serving container to host the model. With the managed PyTorch serving container, you can provide an inference script to process the request data before it is sent to the model for inference, as well as control how to call the model for inference. Let's create a new script called `inference.py` in the `code` folder that contains the following code block. As you have probably noticed, we have used the same functions that we used in *Training the BERT model in the Jupyter Notebook* section for the predictions. Note that you need to use the same function signatures for these two functions as SageMaker will be looking for the exact function name and parameter lists. You can find the complete source code at <https://github.com/PacktPublishing/The-Machine-Learning-Solutions-Architect-Handbook/blob/main/Chapter08/code/inference.py>:

```
import logging
import os
import sys
import json
import numpy as np
import pandas as pd
import torch
from torch.utils.data import DataLoader, TensorDataset
from transformers import BertForSequenceClassification, BertTokenizer
...
def model_fn(model_dir):
    ...
    loaded_model = BertForSequenceClassification.from_pretrained(model_dir)
    return loaded_model.to(device)
def input_fn(request_body, request_content_type):
    ...
```

```

def predict_fn(input_data, model):
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model.to(device)
    model.eval()
    ...
    return y

```

Next, we need to modify the `bert-financial-sentiment-launcher.ipynb` file to create the endpoint. You can deploy trained models from the SageMaker estimator class directly. Here, however, we want to show you how to deploy a model that's been trained previously, as this is the most likely deployment scenario:

```

from sagemaker.pytorch.model import PyTorchModel
model_data = estimator.model_data
pytorch_model = PyTorchModel(model_data=model_data,
                             role=role,
                             framework_version="1.6",
                             source_dir="code",
                             py_version="py3",
                             entry_point="inference.py")
predictor = pytorch_model.deploy(initial_instance_count=1, instance_type="ml.m4.xlarge")

```

After the model has been deployed, we can call the model endpoint to generate some predictions:

```

predictor.serializer = sagemaker.serializers.JSONSerializer()
predictor.deserializer = sagemaker.deserializers.JSONDeserializer()
result = predictor.predict("The market is doing better than last year")
print("predicted class: ", np.argmax(result, axis=1))

```

Try out different phrases and see if the model predicts the sentiment correctly. You can also access the endpoint's details by navigating to the SageMaker management console and clicking on the endpoint. To avoid any ongoing costs for the endpoint, let's delete it. Run the following command in a new cell to delete the endpoint:

```

predictor.delete_endpoint()

```

With that, you have finished building the model and finalized your source code. Now, let's persist the source code to the CodeCommit repository. Congratulations – you have finished building a basic data science environment and used it to train and deploy an NLP model to detect its sentiment. If you don't want to keep this environment to avoid any associated costs, make sure that you shut down any instances of the SageMaker Studio notebooks.

Summary

In this chapter, we explored how a data science environment can provide a scalable infrastructure for experimentation, model training, and model deployment for testing purposes. You learned about the core architecture components for building a fully managed data science environment using AWS services such as Amazon SageMaker, Amazon ECR, and Amazon S3. You also practiced setting up a data science environment and trained and deployed an NLP model using both SageMaker Studio Notebook and SageMaker Training Service. At this point, you should be able to talk about the key components of a data science environment, as well as how to build one using AWS services and use it for model building, training, and deployment. In the next chapter, we will talk about how to build an enterprise ML platform for scale through automation.

9 Building an Enterprise ML Architecture with AWS ML Services

Join our book community on Discord

<https://packt.link/EarlyAccessCommunity>



Many organizations opt to build enterprise ML platforms to support numerous fast-moving initiatives. These platforms are designed to facilitate the entire ML life cycle and accommodate various usage patterns, all while emphasizing automation and scalability. As a practitioner, I often get asked to provide architectural guidance for creating such enterprise ML platforms. In this chapter, we will explore the fundamental requirements for designing enterprise ML platforms. We will cover a range of topics such as workflow automation, infrastructure scalability, and system monitoring. Throughout the discussion, you will gain insights into architecture patterns that enable the development of technology solutions automating the end-to-end ML workflow and ensuring seamless deployment at a large scale. Additionally, we will delve deep into essential components of enterprise ML architecture, such as model training, model hosting, the feature store, and the model registry, all tailored to meet the demands of enterprise-level operations. AI risk, governance and security are another important consideration for enterprise ML platforms, and we will cover them in greater detail in Chapter 12, AI Risk Management and Chapter 13, Privacy, fairness, and adversarial.

Technical requirements

We will continue to use the AWS environment for the hands-on portion of this chapter. All the source code mentioned in this chapter can be found at <https://github.com/PacktPublishing/The-Machine-Learning-Solutions-Architect-Handbook/tree/main/Chapter09>.

The personas of ML platforms and their requirement

In chapter 8, we talked about building a data science environment for the data scientists and ML engineers who mainly focus on experimentation and model development. In an enterprise setting where an ML platform is needed, there are other personas involved, each with their own specific requirements. At a high level, there are 2 types of personas associated with ML platform: ML platform builder and ML platform users.

ML platform builder

ML platform builders have the crucial responsibility of constructing the infrastructure for data and machine learning platforms. Below are some essential builder types required to build a cloud-based ML platform:

- **Cloud Infrastructure architect/engineer:** These experts design the overall cloud infrastructure, selecting appropriate cloud services and setting up the foundation for the ML platform.
- **Security engineer:** Security engineers ensure that the ML platform adheres to industry-standard security practices, safeguarding sensitive data and protecting against potential threats.
- **ML platform product manager:** ML platform product manager are responsible for understand functional user requirements and non-functional requirements, define ML platform capabilities and implementation roadmap.

- **ML platform engineers:** ML platform engineers are responsible for designing, building, and maintaining the infrastructure and systems that support the end-to-end machine learning (ML) lifecycle within an organization. ML platform engineers play a crucial role in ensuring that data scientists and ML practitioners can efficiently develop, deploy, and manage ML models on the organization's ML platform. They are responsible for the platform design covering key functional areas such as training and hosting, taking into account scalability, performance, security, and integration with existing systems.
- **Data engineers:** Data engineers are responsible for building data pipelines, data storage solutions, and data processing frameworks to ensure seamless data access and processing for ML tasks.
- **ML platform tester:** ML platform testers are responsible for testing the core capabilities of platforms to meet the desired functional and non-functional requirements.

Platform user and operator

Platform users and operators are the actual users of a ML platform. They use ML platform to perform full lifecycle ML tasks from data exploration to model monitoring. The following are some of the key platform user and operator types:

- **Data scientist/ML engineers:** Data scientists and ML engineers are the primary users of an ML platform. They use the platform to explore data, build and train machine learning models, perform feature engineering, and evaluate model performance. They partner with ML platform engineers and ops engineers to integrate trained models into production systems, optimizing model inference performance, and ensure the models are scalable and reliable in real-world environments.
- **Model tester and validator:** primary responsibilities for model tester involve assessing the performance and reliability of machine learning models developed by data scientists using the ML platform. Specifically, model testers are responsible for model testing using different datasets, model performance metrics calculation and evaluation, overfitting/underfitting detection, and testing of edge cases. Model validators are responsible for validating models against business objectives, risk assessment, and other issues such as ethics considerations.
- **Model approver:** This individual or team is responsible for reviewing and approving the deployment of ML models into production or other critical environments. The model approver's primary role is to ensure that the developed ML models meet the organization's standards, business requirements, and compliance policies before they are deployed. They also help ensure all the required testing, post-deployment operations and policies are in place.
- **Operations and support engineers:** ensuring the smooth operation, maintenance, and ongoing support of the ML platform within an organization. Their responsibilities encompass various technical and operational aspects to keep the ML platform running efficiently and to provide assistance to users. Some of the key functions include platform maintenance and upgrades, performance monitoring and optimization, incident management, infrastructure management, security and access control, and platform documentation.
- **AI risk/governance manager:** The primary responsibility of an AI risk/governance manager is to manage and mitigate the potential risks associated with the use of AI/ML systems. Their role is essential in ensuring that AI technologies are developed, deployed, and used responsibly, ethically, and in compliance with relevant regulations. They help ensure appropriate processes, policies, and technology standards are created and adhered to.

You might wonder where the ML solution architect fits into this overall picture. The ML solution architect plays a pivotal role that spans across builders, users, and operators. They serve as a bridge between these groups, offering valuable insights and guidance. Firstly, ML solution architects collaborate with builders, understanding user requirements, and assisting in the end-to-end architecture design. They ensure that the ML platform aligns with the specific needs of users and operators. Secondly, ML solution architects advise users and operators on effectively utilizing the ML platform. They educate them on best practices for configuring and leveraging the platform to meet diverse needs and use cases.

Common workflow of an ML initiative

Different organizations have diverse workflows and governance processes when running ML initiatives. However, most of these workflows typically consist of the following key steps:

- Collecting and processing data from different sources and make them available for data scientists.
- Performing data exploratory analysis, form hypothesis, creating ML features, perform experimentation, and build different ML models using different techniques and ML algorithms using subset of data.
- Data labelling workflow is sometimes needed for labelling training data for ML tasks such as document classification, or object detection.
- Conducting full model training and tuning using full dataset.
- Candidate models trained using the full dataset are promoted into a testing environment for formal quality assurance. Testers document testing details for all the tester and verify if the models meet desired performance metrics and other evaluation criteria such as latency and scalability. Model validators evaluate the ML techniques, perform analysis of the model, and check the alignment with the business outcome.
- Model risk assessment is performed to ensure risk items are assessed, mitigated, or accepted.
- After model passes the testing and validation step, the model is sent to model approver for final review and approval for production deployment.
- The model is deployed into production with approval. Model is registered into model registry, dataset is versioned and retained, any code artifacts are also versioned and stored, detailed training configuration details are also documented.
- The model is monitored in production for model performance, data drift, system issues, and security exposure and attacks. Incident management process is followed to address issues identified.
- At required schedules, auditors perform end-to-end audit to ensure all processes and policies are followed, artifacts are stored, access to systems and models are properly logged, documentation meets the required standards, and any violations are flagged and escalated.

It is worth noting that these steps are not exhaustive. Depending on the organizational, risk, and regulatory requirements, organizations can have more steps to address these requirements.

Platform capability requirements

ML platforms involve a diverse array of potential players and users. The following table outlines the essential needs for ML platforms for both users and operator of the platform. Note that the table does not include builder of the platform.

User/operator	Tools/capability requirements
Data scientist	Access to various ML libraries, tools, and frameworks for model development and experimentation
Model tester & validator	Access to different data sets to perform different ML tasks
	Capabilities to perform data exploration and model training using different hardware
	Workflow automation and model versioning for reproducibility.
Model tester & validator	Access to different test datasets for model testing and validation.
	Access to various libraries and tools for data visualization, model evaluation, ML testing framework,

	bias detection tools, model interpretability tools, statistical testing tools
Model approvers	Access to model documentation, model evaluation metrics, compliance check list, model explainability report.
	Access to approval workflow management tool.
Ops and support engineers	Access to all infrastructure components within the ML platform, including code and container repositories, library packages, training, hosting, pipeline, logging, monitoring and alerting, security and access control, backup and discovery, performance testing, incident management tools.
	Access to tools for platform automation and management
AI risk officer	Access to AI risk assessment tool, governance platform, model Explainability and Interpretability tools, bias detection and fairness assessment tools, AI risk reporting and dashboards, AI regulation and policy monitoring

Table 9.1: ML platform requirements by personas

In summary, the success of a ML platform relies heavily on meeting the distinct tools and capability requirements of its user/operators. By addressing these distinct needs, the ML platform can effectively support its users and operators in building, deploying, and managing AI solutions with confidence.

Key requirements for an enterprise ML platform

To deliver business benefits through ML at scale, organizations must have the capability to rapidly experiment with diverse scientific approaches, ML technologies, and extensive datasets. Once ML models are trained and validated, they need to seamlessly transition to production deployment. While some similarities exist between a traditional enterprise software system and an ML platform, such as scalability and security concerns, an enterprise ML platform presents distinctive challenges. These include the need to integrate with the data platform and high-performance computing infrastructure to facilitate large-scale model training. Now, let's delve into some specific core requirements of an enterprise ML platform to meet the needs of different users and operators:

- **Support for the end-to-end ML life cycle:** An enterprise ML platform must cater to both data science experimentation and production-grade operations and deployments. In Chapter 8, we explored the essential architecture components required to construct a data science experimentation environment using AWS ML services. However, to facilitate seamless production-grade operations and deployment, the enterprise ML platform should also include specific architecture components dedicated to large-scale model training, model management, feature management, and highly available and scalable model hosting.
- **Support for continuous integration (CI), continuous training (CT), and continuous deployment (CD):** In addition to testing and validating code and components, an enterprise ML platform extends its CI capabilities to include data and models. The CD capability for ML goes beyond merely deploying a single software piece; it involves managing both ML models and inference engines in conjunction. CT is a unique aspect of ML, wherein a model is continuously monitored, and automated model retraining can be triggered upon detecting data drift, model drift, or changes in the training data. Data drift refers to a change in the data wherein the

statistical characteristics of the production data differ from the data used for model training. On the other hand, model drift signifies a decline in model performance compared to the performance achieved during the model training phase.

- **Operations support:** An enterprise ML platform should provide capabilities to monitor the statuses, errors, and metrics of different pipeline workflows, processing/training jobs, model behaviors changes, data drift, and model serving engines. Additionally, infrastructure-level statistics and resource usage are continuously monitored to ensure efficient operations. An automated alert mechanism is a crucial component of operations, promptly notifying relevant stakeholders of any issues or anomalies. Moreover, implementing automated failure recovery mechanisms wherever possible further enhances the platform's robustness and minimizes downtime, ensuring smooth and reliable ML operations.
- **Support for different languages and ML frameworks:** An enterprise ML platform empowers data scientists and ML engineers to use their preferred programming languages and ML libraries. It should accommodate popular languages like Python and R, along with well-known ML packages such as TensorFlow, PyTorch, and scikit-learn. This flexibility ensures that teams can leverage their expertise and utilize the most suitable tools for efficient and effective model development within the platform.
- **Computing hardware resource management:** An enterprise ML platform should cater to diverse model training and inference requirements, taking into account cost considerations. This entails providing support for various types of computing hardware, such as CPUs and GPUs, to optimize performance and cost-effectiveness. Furthermore, the platform should be equipped to handle specialized ML hardware, like AWS's Inferentia and Tranium chips, wherever relevant, to leverage the benefits of specialized hardware accelerators for specific ML workloads.
- **Integration with other third-party systems and software:** An enterprise ML platform rarely operates in isolation. It must offer robust integration capabilities with various third-party software and platforms, including workflow orchestration tools, container registries, and code repositories. This seamless integration enables smooth collaboration and interoperability, allowing teams to leverage existing tools and workflows while benefiting from the advanced features and capabilities of the ML platform.
- **Authentication and authorization:** For an enterprise ML platform, ensuring secure access to data, artifacts, and ML platform resources is essential. This requires offering various levels of authentication and authorization control. The platform may include built-in authentication and authorization capabilities, or it can integrate with an external authentication and authorization service.
- **Data encryption:** In regulated industries like financial services and healthcare, data encryption is a critical requirement. An enterprise ML platform must offer robust capabilities for encrypting data both at rest and in transit, often allowing customers to manage their encryption keys. This level of data protection ensures that sensitive information remains secure and compliant with industry regulations, providing the necessary reassurance for handling confidential data within these sectors.
- **Artifacts management:** In the ML life cycle, an enterprise ML platform handles datasets and generates various artifacts at different stages. These artifacts can be features, code, models, and containers. To ensure reproducibility and adhere to governance and compliance standards, the platform must possess the capability to track, manage, and version-control these artifacts. By effectively managing and recording the changes made throughout the ML process, the platform maintains a clear and organized record, facilitating reproducibility of results and providing a reliable audit trail for compliance purposes.
- **ML library package management:** Standardizing and approving ML library packages used by data scientists is crucial for many organizations. By establishing a central library that contains pre-approved packages, it becomes possible to enforce consistent standards and policies across the usage of library packages. This approach ensures that data scientists work with vetted and authorized libraries, promoting reliability, security, and adherence to organizational guidelines when developing ML solutions.
- **Access to different data store:** An essential feature of an enterprise ML platform is to offer seamless access to various data stores, simplifying model development and training processes. This accessibility to diverse data sources streamlines the workflow for data scientists and ML engineers, enabling them to efficiently access and utilize the necessary data for their tasks within the platform.
- **Self-service capability:** To enhance operational efficiency and reduce reliance on central teams, an enterprise ML platform should incorporate self-service capabilities for tasks like user onboarding, environment setup, and pipeline provisioning. By enabling users to perform these tasks independently, the platform streamlines operations, empowering data scientists and ML engineers to work more autonomously and efficiently.

- **Model testing and validation:** An enterprise ML platform should provide comprehensive model testing and validation features to support thorough assessments of machine learning models. This can include features such as A/B testing infrastructure, model robustness testing packages, automated testing pipelines, performance metrics tracking and error analysis tool, and visualization.

Having covered the essential requirements of an enterprise ML platform, let's now explore how AWS ML and DevOps services, such as SageMaker, CodePipeline, and Step Functions, can be effectively utilized to construct a robust, enterprise-grade ML platform.

Enterprise ML architecture pattern overview

Building an enterprise ML platform on AWS starts with creating different environments to enable different data science and operations functions. The following diagram shows the core environments that normally make up an enterprise ML platform. From an isolation perspective, in the context of the AWS cloud, each environment in the following diagram is a separate AWS account:

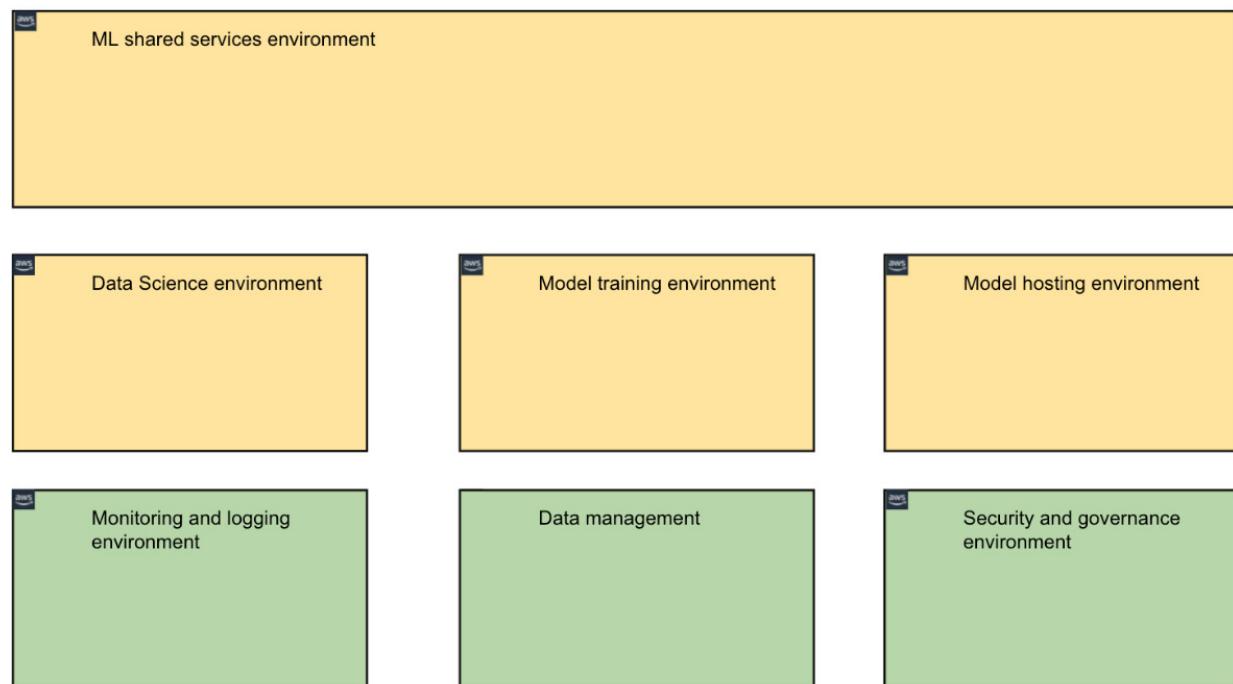


Figure 9.1: Enterprise ML architecture environments

As we discussed in *Chapter 8, Building a Data Science Environment Using AWS ML Services*, data scientists utilize the data science environment for experimentation, model building, and tuning. Once these experiments are completed, the data scientists commit their work to the proper code and data repositories. The next step is to train and tune the ML models in a controlled and automated environment using the algorithms, data, and training scripts that were created by the data scientists. This controlled and automated model training process will help ensure consistency, reproducibility, and traceability for scalable model building. The following are the core functionalities and technology options provided by the training, hosting, and shared services environments:

- **Model training environment:** This environment manages the full life cycle of model training, from computing and storage infrastructure resource provisioning to training job monitoring and model persistence. For this purpose, the SageMaker training service offers a suitable technology option to construct the training infrastructure.

- **Model hosting environment:** This environment is used for serving the trained models behind web service endpoints or in batch inference mode. For this purpose, you can use SageMaker hosting service for this environment. Other supporting services such as the online feature store and API management service can also run in the model hosting environment. There can be multiple model hosting environments for different stages. For example, you can have a testing hosting environment designated for model testing, and a production hosting environment for production model deployment serving real-world traffic. Model testers can perform different tests such as model performance, robustness, bias, explainability analysis, and model hosting testing.
- **Shared service environment:** The shared services environment hosts common services tooling such as workflow orchestration tools, CI/CD tools, code repositories, Docker image repositories, and private library package tools. A central model registry can also run in the shared services environment for model registration and model life cycle management. Service provisioning capabilities, such as creating resources in different environments through **Infrastructure as Code (IaC)** or APIs, also run out of this environment. Any service ticketing tools, such as ServiceNow, and service provisioning tools, such as Service Catalog, can also be hosted in this environment.

In addition to the core ML environments, there are other supporting environments, such as security, governance, monitoring, and logging, that are required in the enterprise ML platform:

- **Security and governance environment:** The security and governance environment centrally manages authentication services, user credentials, and data encryption keys. Security audit and reporting processes also run in this environment. Native AWS services, such as AWS IAM, AWS KMS, and AWS Config, can be used for various security and governance functions. Any custom-built risk and governance tools can also be hosted in this environment to service AI risk/governance manager.
- **Monitoring and logging environment:** The monitoring and logging environment centrally aggregates monitoring and logging data from other environments for further processing and reporting. Custom dashboarding and alerting mechanisms are normally developed to provide easy access to key metrics and alerts from the underlying monitoring and logging data.

Now that you have a comprehensive overview of the fundamental elements that constitute an enterprise ML platform, let's delve deeper into specific core areas. It is important to recognize that there are various patterns and services available for constructing an ML platform on AWS. Moreover, while Figure 9.1 showcases distinct AWS environments (i.e., AWS accounts) to host different ML platform environments, organizations can also choose to combine some of environments in a single AWS account as long as there are proper boundaries between different environments to ensure isolation of infrastructure, process flow, and security control. Furthermore, organizations can create separate AWS accounts for hosting a specific environment for different users or groups. For instance, a large enterprise can choose to create one data science environment for each of the LoBs, or separate production hosting environment based on either organizational structure or workload separation. In this chapter, we will focus on exploring one of the enterprise patterns to build an efficient and scalable ML platform.

Model training environment

Within an enterprise, a model training environment is a controlled environment with well-defined processes and policies on how it is used and who can use them. Normally, it should be an automated environment that's managed by an ML operations team, though it can be self-service enabled for direct usage by data scientists. Automated model training and tuning are the core capabilities of the model training environment. To support a broad range of use cases, a model training environment needs to support different ML and **deep learning** frameworks, training patterns (single-node and distributed training), and hardware (different CPUs, GPUs, and custom silicon chips). The model training environment manages the life cycle of the model training process. This can include authentication and authorization, infrastructure provisioning, data movement, data pre-processing, ML library deployment, training loop management and monitoring, model persistence and registry, training job management, and lineage tracking. From a security perspective, the training environment needs to provide security capabilities for different isolation requirements, such as network isolation, job isolation, and artifacts isolation. To assist with operational support, a model training environment also needs to be able to support training status logging, metrics reporting, and training job monitoring and alerting.

Model training engine using SageMaker

The SageMaker training service provides built-in modeling training capabilities for a range of ML/DL libraries. In addition, you can bring your own Docker containers for customized model training needs. The following are a subset of supported options for the SageMaker Python SDK:

- **Training TensorFlow models:** SageMaker provides a built-in training container for TensorFlow models. The following code sample shows how to train a TensorFlow model using the built-in container through the TensorFlow estimator API:

```
from sagemaker.tensorflow import TensorFlow
tf_estimator = TensorFlow(
    entry_point=<Training script name>,
    role= "<AWS IAM role>",
    instance_count=<Number of instances>,
    instance_type=<Instance type>,
    framework_version=<TensorFlow version>,
    py_version=<Python version>, )
tf_estimator.fit("<Training data location>")
```

- **Training PyTorch models:** SageMaker provides a built-in training container for PyTorch models. The following code sample shows how to train a PyTorch model using the PyTorch estimator:

```
from sagemaker.pytorch import PyTorch
pytorch_estimator = PyTorch(
    entry_point=<Training script name>,
    role= "<AWS IAM role>",
    instance_count=<Number of instances>,
    instance_type=<Instance type>,
    framework_version=<PyTorch version>,
    py_version=<Python version>, )
pytorch_estimator.fit("<Training data location>")
```

- **Training XGBoost models:** XGBoost training is also supported via a built-in container. The following code shows the syntax for training a XGBoost model using the XGBoost estimator:

```
from sagemaker.xgboost.estimator import XGBoost
xgb_estimator = XGBoost(
    entry_point=" <Training script name>",
    hyperparameters=<List of hyperparameters>,
    role=<AWS IAM role>,
    instance_count=<Number of instances>,
    instance_type=<Instance type>,
    framework_version=<Xgboost version>)
xgb_estimator.fit("<train data location>")
```

- **Training scikit-learn models:** The following code sample shows how to train a scikit-learn model using the built-in container:

```
from sagemaker.sklearn.estimator import SKLearn
sklearn_estimator = SKLearn(
    entry_point=" <Training script name>",
    hyperparameters=<List of hyperparameters>,
    role=<AWS IAM role>,
    instance_count=<Number of instances>,
    instance_type=<Instance type>,
    framework_version=<sklearn version>)
sklearn_estimator.fit("<training data>")
```

- **Training models using custom containers:** You can also build a custom training container and use the SageMaker training service for model training. See the following code for an example:

```

from sagemaker.estimator import Estimator
custom_estimator = Estimator (
    Custom_training_img,
    role=<AWS IAM role>,
    instance_count=<Number of instances>,
    instance_type=<Instance type>)
custom_estimator.fit("<training data location>")

```

In addition to using the SageMaker Python SDK to kick off training, you can also use the **boto3** library and SageMaker CLI commands to start training jobs.

Automation support

The SageMaker training service is exposed through a set of APIs and can be automated by integrating with external applications or workflow tools, such as SageMaker Pipeline, Airflow and AWS Step Functions. For example, it can be one of the steps in an Airflow-based pipeline for an end-to-end ML workflow. Some workflow tools, such as Airflow and AWS Step Functions, also provide SageMaker-specific connectors to interact with the SageMaker training service more seamlessly. The SageMaker training service also provides Kubernetes operators, so it can be integrated and automated as part of the Kubernetes application flow. The following sample code shows how to kick off a training job using the low-level API via the AWS **boto3** SDK:

```

import boto3
client = boto3.client('sagemaker')
response = client.create_training_job(
    TrainingJobName='<job name>',
    HyperParameters={<list of parameters and value>},
    AlgorithmSpecification={...},
    RoleArn='<AWS IAM Role>',
    InputDataConfig=[...],
    OutputDataConfig={...},
    ResourceConfig={...},
    ...
)

```

Regarding using Airflow as the workflow tool, the following sample shows how to use the Airflow SageMaker operator as part of the workflow definition. Here, **train_config** contains training configuration details, such as the training estimator, training instance type and number, and training data location:

```

import airflow
from airflow import DAG
from airflow.contrib.operators.sagemaker_training_operator import SageMakerTrainingOperator
default_args = {
    'owner': 'myflow',
    'start_date': '2021-01-01'
}
dag = DAG('tensorflow_training', default_args=default_args,
           schedule_interval='@once')
train_op = SageMakerTrainingOperator(
    task_id='tf_training',
    config=train_config,
    wait_for_completion=True,
    dag=dag)

```

SageMaker also has a built-in workflow automation tool called **SageMaker Pipelines**. A training step can be created using the SageMaker **Training Step** API and become part of the larger SageMaker Pipelines workflow.

Model training life cycle management

SageMaker training manages the life cycle of the model training process. It uses AWS IAM as the mechanism to authenticate and authorize access to its functions. Once authorized, it provides the desired infrastructure, deploys the software stacks for the different model training requirements, moves the data from sources to training nodes,

and kicks off the training job. Once the training job has been completed, the model artifacts are saved into a S3 output bucket and the infrastructure is torn down. For lineage tracing, model training metadata such as source datasets, model training containers, hyperparameters, and model output locations are captured. Any logging from the training job runs is saved in CloudWatch Logs, and system metrics such as CPU and GPU utilization are captured in the CloudWatch metrics. Depending on the overall end-to-end ML platform architecture, a model training environment can also host services for data preprocessing, model validation, and model training postprocessing, as those are important steps in an end-to-end ML flow. There are multiple technology options available for this, such as the SageMaker Processing service, AWS Glue, and AWS Lambda.

Model hosting environment deep dive

An enterprise-grade model hosting environment needs to support a broad range of ML frameworks in a secure, performant, and scalable way. It should come with a list of pre-built inference engines that can serve common models out of the box behind a **RESTful API** or via the **gRPC protocol**. It also needs to provide flexibility to host custom-built inference engines for unique requirements. Users should also have access to different hardware devices, such as CPU, GPU, and purpose-built chips, for the different inference needs. Some model inference patterns demand more complex inference graphs, such as traffic split, request transformations, or model ensemble support. A model hosting environment can provide this capability as an out-of-the-box feature or provide technology options for building custom inference graphs. Other common model hosting capabilities include **concept drift detection** and **model performance drift detection**. Concept drift occurs when the statistical characteristics of the production data deviate from the data that's used for model training. An example of concept drift is the mean and standard deviation of a feature changing significantly in production from that of the training dataset. Components in a model hosting environment can participate in an automation workflow through its API, scripting, or IaC deployment (such as AWS CloudFormation). For example, a RESTful endpoint can be deployed using a CloudFormation template or by invoking its API as part of an automated workflow. From a security perspective, the model hosting environment needs to provide authentication and authorization control to manage access to both the **control plane** (management functions) and **data plane** (model endpoints). The accesses and operations that are performed against the hosting environments should be logged for auditing purposes. For operations support, a hosting environment needs to enable status logging and system monitoring to support system observability and problem troubleshooting. The SageMaker hosting service is a fully managed model hosting service. Similar to KFServing and Seldon Core, which we reviewed earlier in this book, the SageMaker hosting service is also a multi-framework model serving service. Next, let's take a closer look at its various capabilities for enterprise-grade model hosting.

Inference engine

SageMaker provides built-in inference engines for multiple ML frameworks, including scikit-learn, XGBoost, TensorFlow, PyTorch, and Spark ML. SageMaker supplies these built-in inference engines as Docker containers. To stand up an API endpoint to serve a model, you just need to provide the model artifacts and infrastructure configuration. The following is a list of model serving options:

- **Serving TensorFlow models:** SageMaker uses TensorFlow Serving as the inference engine for TensorFlow models. The following code sample shows how to deploy a TensorFlow Serving model using the SageMaker hosting service:

```
from sagemaker.tensorflow.serving import Model
tensorflow_model = Model(
    model_data=<S3 location of the Spark ML model artifacts>,
    role=<AWS IAM role>,
    framework_version=<tensorflow version>
)
tensorflow_model.deploy(
    initial_instance_count=<instance count>, instance_type=<instance type>
)
```

- **Serving PyTorch models:** SageMaker hosting uses TorchServe under the hood to serve PyTorch models. The following code sample shows how to deploy a PyTorch model:

```
from sagemaker.pytorch.model import PyTorchModel
pytorch_model = PyTorchModel(
    model_data=<S3 location of the PyTorch model artifacts>,
    role=<AWS IAM role>,
    framework_version=<PyTorch version>
)
pytorch_model.deploy(
    initial_instance_count=<instance count>, instance_type=<instance type>
)
```

- **Serving Spark ML models:** For Spark ML-based models, SageMaker uses MLeap as the backend to serve Spark ML models. These Spark ML models need to be serialized into MLeap format. The following code sample shows how to deploy a Spark ML model using the SageMaker hosting service:

```
import sagemaker
from sagemaker.sparkml.model import SparkMLModel
sparkml_model = SparkMLModel(
    model_data=<S3 location of the Spark ML model artifacts>,
    role=<AWS IAM role>,
    sagemaker_session=sagemaker.Session(),
    name=<Model name>,
    env={"SAGEMAKER_SPARKML_SCHEMA": <schema_json>}
)
sparkml_model.deploy(
    initial_instance_count=<instance count>, instance_type=<instance type>
)
```

- **Serving XGboost models:** SageMaker provides an XGBoost model server for serving trained XGBoost models. Under the hood, it uses Nginx, Gunicorn, and Flask as part of the model serving architecture. The entry Python script loads the trained XGBoost model and can optionally perform pre- and post-data processing:

```
from sagemaker.xgboost.model import XGBoostModel
xgboost_model = XGBoostModel(
    model_data=<S3 location of the Xgboost ML model artifacts>,
    role=<AWS IAM role>,
    entry_point=<entry python script>,
    framework_version=<xgboost version>
)
xgboost_model.deploy(
    instance_type=<instance type>,
    initial_instance_count=<instance count>
)
```

- **Serving scikit-learn models:** SageMaker provides a built-in serving container for serving scikit-learn-based models. The technology stack is similar to the one for the Xgboost model server:

```
from sagemaker.sklearn.model import SKLearnModel
sklearn_model = SKLearnModel(
    model_data=<S3 location of the Xgboost ML model artifacts>,
    role=<AWS IAM role>,
    entry_point=<entry python script>,
    framework_version=<scikit-learn version>
)
sklearn_model.deploy(instance_type=<instance type>, initial_instance_count=<instance count>)
```

- **Serving models with custom containers:** For custom-created inference containers, you can follow similar syntax to deploy the model. The main difference is that a custom inference container image's uri needs to be provided. You can find detailed documentation on building a custom inference container at <https://docs.aws.amazon.com/sagemaker/latest/dg/adapt-inference-container.xhtml>:

```

from sagemaker.model import Model
custom_model = Model(
    Image_uri = <custom model inference container image uri>,
    model_data=<S3 location of the ML model artifacts>,
    role=<AWS IAM role>,
    framework_version=<scikit-learn version>
)
custom_model.deploy(instance_type=<instance type>, initial_instance_count=<instance count>)

```

SageMaker hosting provides an inference pipeline feature that allows you to create a linear sequence of containers to perform custom data processing before and after invoking a model for predictions. SageMaker hosting can support traffic splits between multiple versions of a model for A/B testing. SageMaker hosting can be provisioned using an AWS CloudFormation template. There is also support for the AWS CLI for scripting automation, and it can be integrated into custom applications via its API. The following are some code samples for different endpoint deployment automation methods:

- The following is a CloudFormation code sample for SageMaker endpoint deployment. You can find the complete code at https://github.com/PacktPublishing/The-Machine-Learning-Solutions-Architect-Handbook/blob/main/Chapter09/sagemaker_hosting.yaml:

```

Description: "Model hosting cloudformation template"
Resources:
  Endpoint:
    Type: "AWS::SageMaker::Endpoint"
    Properties:
      EndpointConfigName:
        !GetAtt EndpointConfig.EndpointConfigName
  EndpointConfig:
    Type: "AWS::SageMaker::EndpointConfig"
    Properties:
      ProductionVariants:
        - InitialInstanceCount: 1
          InitialVariantWeight: 1.0
          InstanceType: ml.t2.large
          ModelName: !GetAtt Model.ModelName
          VariantName: !GetAtt Model.ModelName
  Model:
    Type: "AWS::SageMaker::Model"
    Properties:
      PrimaryContainer:
        Image: <container uri>
        ExecutionRoleArn: !GetAtt ExecutionRole.Arn
...

```

- The following is an AWS CLI sample for SageMaker endpoint deployment:

```

aws sagemaker create-model --model-name <value> --execution-role-arn <value>
aws sagemaker Create-endpoint-config --endpoint-config-name <value> --production-variants <value>
aws sagemaker Create-endpoint --endpoint-name <value> --endpoint-config-name <value>

```

If the built-in inference engines do not meet your requirements, you should consider bringing your own Docker container to serve your ML models.

Authentication and security control

The SageMaker hosting service uses AWS IAM as the mechanism to control access to its control plane APIs (for example, an API for creating an endpoint) and data plane APIs (for example, an API for invoking a hosted model endpoint). If you need to support other authentication methods for the data plane API, such as **OpenID Connect (OIDC)**, you can put a proxy service as the frontend to manage user authentication. A common pattern is to use AWS API Gateway to frontend the SageMaker API for custom authentication management, as well as other API management features such as metering and throttling management.

Monitoring and logging

SageMaker provides out-of-the-box monitoring and logging capabilities to assist with support operations. It monitors both system resource metrics (for example, CPU/GPU utilization) and model invocation metrics (for example, the number of invocations, model latencies, and failures). These monitoring metrics and any model processing logs are captured by AWS CloudWatch metrics and CloudWatch Logs.

Adopting MLOps for ML workflows

Similar to the DevOps practice, which has been widely adopted for the traditional software development and deployment process, the MLOps practice is intended to streamline the building and deployment processes of ML pipelines while enhancing the collaborations between data scientists/ML engineers, data engineering, and the operations team. Specifically, the primary objective of MLOps practice is to yield the following main benefits throughout the entire ML life cycle:

- **Process consistency:** The MLOps practice aims to create consistency in the ML model building and deployment process. A consistent process improves the efficiency of the ML workflow and ensures a high degree of certainty in the input and output of the ML workflow.
- **Tooling and process reusability:** One of the core objectives of the MLOps practice is to create reusable technology tooling and templates for faster adoption and deployment of new ML use cases. These can include common tools such as code and library repositories, package and image building tools, pipeline orchestration tools, the model registry, as well as common infrastructure for model training and model deployment. From a reusable template perspective, these can include common reusable scripts for Docker image builds, workflow orchestration definitions, and CloudFormation scripts for model building and model deployment.
- **Model building reproducibility:** ML is highly iterative and can involve a large number of experimentations and model training runs using different datasets, algorithms, and hyperparameters. An MLOps process needs to capture all the data inputs, source code, and artifacts that are used to build an ML model and establish model lineage from this input data, code, and artifacts for the final models. This is important for both experiment tracking as well as governance and control purposes.
- **Delivery scalability:** An MLOps process and the associated tooling enable a large number of ML pipelines to run in parallel for high delivery throughputs. Different ML project teams can use the standard MLOps processes and common tools independently without creating conflicts from a resource contention, environment isolation, and governance perspective.
- **Process and operations audibility:** MLOps enables greater audibility into the process and the audibility of ML pipelines. This includes capturing the details of machine pipeline executions, dependencies, and lineage across different steps, job execution statuses, model training and deployment details, approval tracking, and actions that are performed by human operators.

Now that we are familiar with the intended goals and benefits of the MLOps practice, let's delve into the specific operational process and concrete technology architecture of MLOps on AWS.

Components of the MLOps architecture

One of the most important MLOps concepts is the automation pipeline, which executes a sequence of tasks, such as data processing, model training, and model deployment. This pipeline can be a linear sequence of steps or a more complex DAG with parallel execution for multiple tasks. The following diagram illustrates a sample DAG for a ML pipeline.

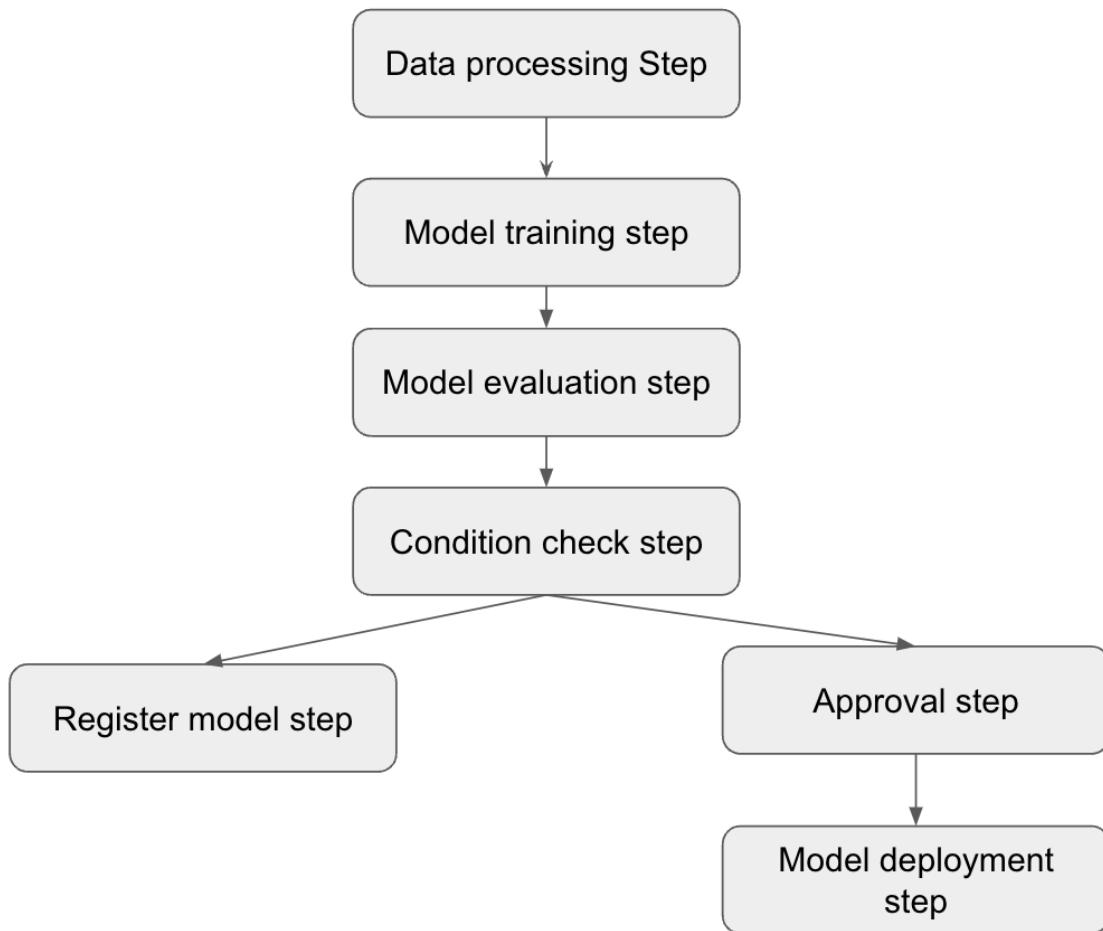


Figure 9.2: Sample ML pipeline flow

An MLOps architecture also has several repositories for storing different assets and metadata as part of pipeline executions. The following diagram lists the core components and tasks involved in an MLOps operation:

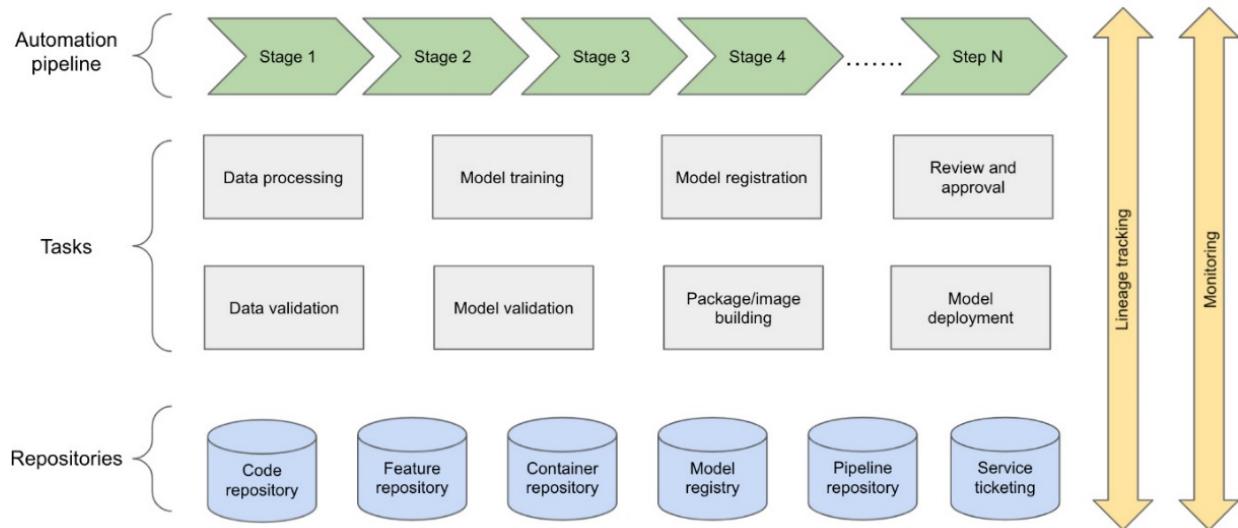


Figure 9.3: MLOps components

A **code repository** in an MLOps architecture component that not only serves as a source code control mechanism for data scientists and engineers – it can also be the triggering mechanism to kick off different pipeline executions. For example, when a data scientist checks an updated training script into the code repository, a model training pipeline execution can be triggered. A **feature repository** stores reusable ML features and can be the target of a data processing/feature engineering job. The features from the feature repository can be a part of the training datasets where applicable. The feature repository is also used as a part of the model inference request. A **container repository** stores the container images that are used for data processing tasks, model training jobs, and model inference engines. It is usually the target of the container building pipeline. A **model registry** keeps an inventory of trained models, along with all the metadata associated with the model, such as its algorithm, hyperparameters, model metrics, and training dataset location. It also maintains the status of the model life cycle, such as its deployment approval status. A **pipeline repository** maintains the definition of automation pipelines and the statuses of different pipeline job executions. In an enterprise setting, a task ticket also needs to be created when different tasks, such as model deployment, are performed, so that these actions can be tracked in a common enterprise ticketing management system. To support audit requirements, the lineage of different pipeline tasks and their associated artifacts need to be tracked. Another critical component of the MLOps architecture is **monitoring**. In general, you want to monitor items such as the pipeline's execution status, model training status, and model endpoint status. Model endpoint monitoring can also include system/resource performance monitoring, model statistical metrics monitoring, drift and outlier monitoring, and model explainability monitoring. Alerts can be triggered on certain execution statuses to invoke human or automation actions that are needed. AWS provides multiple technology options for implementing an MLOps architecture. The following diagram shows where these technology services fit in an enterprise MLOps architecture:

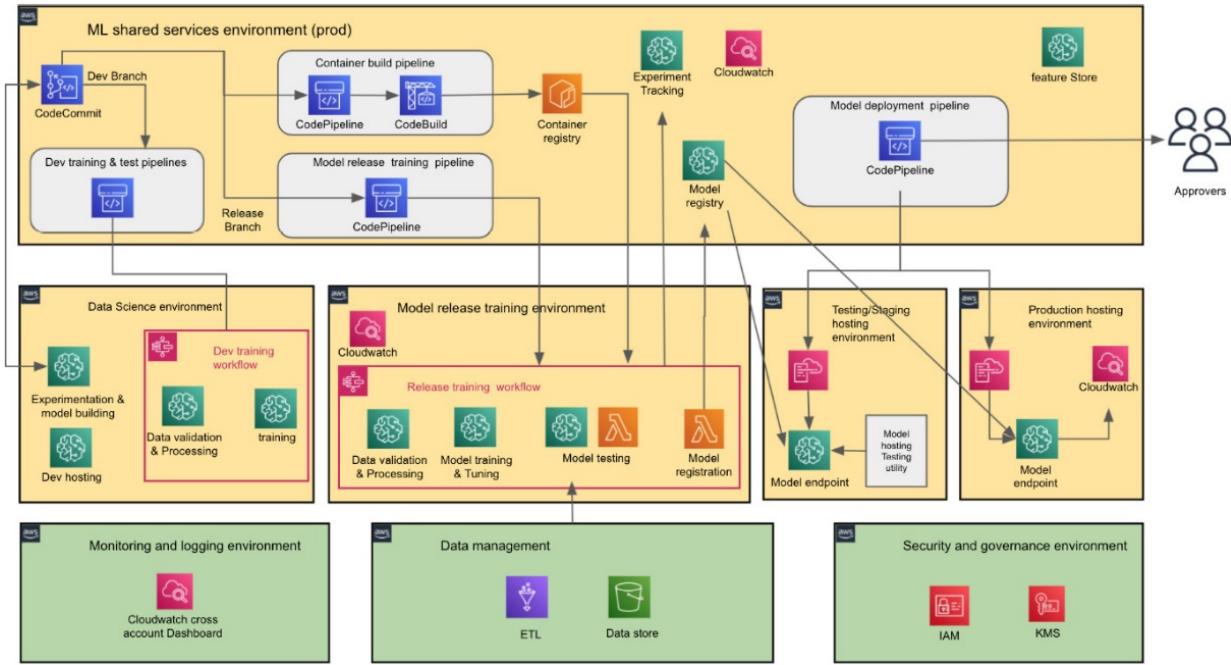


Figure 9.4: MLOps architecture using AWS services

As we mentioned earlier, the shared service environment hosts common tools for pipeline management and execution, as well as common repositories such as code repositories and model registries. Here, we use AWS CodePipeline to orchestrate the overall CI/CD pipeline. AWS CodePipeline is a continuous delivery service that integrates natively with different code repositories such as AWS CodeCommit and Bitbucket. It can source files from the code repository and make them available to downstream tasks such as building containers using the AWS CodeBuild service, or training models in the model training environment. You can create different pipelines to meet different needs. A pipeline can be triggered on-demand via an API or the CodePipeline management console, or it can be triggered by code changes in a code repository. Depending on your requirements, you can create different pipelines. In the proceeding diagram, we can see four example pipelines:

- A container build pipeline for building different container images for training, processing, and inference.
- A model training pipeline for training a model for release.
- A model deployment pipeline for deploying trained models to production en
- A development, training, and testing pipeline for model training and deployment testing in a data science environment.

Note that while this diagram only showcases 4 distinct pipelines, in reality, organizations can have many more pipelines based on specific requirements. Moreover, they can run multiple instances of the same pipeline in parallel to accommodate various ML projects. For instance, different instances of training job pipeline may run independently and concurrently, each dedicated to training distinct ML models using different datasets and configurations. A code repository is one of the most essential components in an MLOps environment. It is not only used by data scientists/ML engineers and other engineers to persist code artifacts, but it also serves as a triggering mechanism for a CI/CD pipeline. This means that when a data scientist/ML engineer commits a code change, it can automatically kick off a CI/CD pipeline. For example, if the data scientist makes a change to the model training script and wants to test the automated training pipeline in the development environment, he/she can commit the code to a development branch to kick off a model training pipeline in the dev environment. When it is ready for production release deployment, the data scientist can commit/merge the code to a release branch to kick off the production release pipelines. In this MLOps architecture, we use AWS **Elastic Container Registry (ECR)** as the central container registry service. ECR is used to store containers for data processing, model training, and model inference. You can tag the container images to indicate different life cycle statuses, such as development or

production. The **SageMaker model registry** is used as the central model repository. The central model repository can reside in the shared service environment, so it can be accessed by different projects. All the models that go through the formal training and deployment cycles should be managed and tracked in the central model repository. **SageMaker Feature Store** provides a common feature repository for reusable features to be used by different projects. It can reside in the shared services environment or be part of the data platform. Features are normally pre-calculated in a data management environment and sent to SageMaker Feature Store for offline model training in the model training environment, as well as online inferences by the different model hosting environments. **SageMaker Experiments** is used to track experiments and trials. The metadata and artifacts that are generated by the different components in a pipeline execution can be tracked in SageMaker Experiments. For example, the processing step in a pipeline can contain metadata such as the locations of input data and processed data, while the model training step can contain metadata such as the algorithm and hyperparameters for training, model metrics, and the location of the model artifact. This metadata can be used to compare the different runs of model training, and they can also be used to establish model lineage.

Monitoring and logging

The ML platform presents some unique challenges in terms of monitoring. In addition to monitoring common software system-related metrics and statuses, such as infrastructure utilization and processing status, an ML platform also needs to monitor model and data-specific metrics and performances. Also, unlike traditional system-level monitoring, which is fairly straightforward to understand, the opaqueness of ML models makes it inherently difficult to understand the system. Now, let's take a closer look at the three main areas of monitoring for an ML platform.

Model training monitoring

Model training monitoring provides visibility into the training progress and helps identify training bottlenecks and error conditions during the training process. It enables operational processes such as training job progress reporting and response, model training performance progress evaluation and response, training problem troubleshooting, and data and model bias detection and model interpretability and response. Specifically, we want to monitor the following key metrics and conditions during model training:

- **General system and resource utilization and error metrics:** These provide visibility into how the infrastructure resources (such as CPU, GPU, disk I/O, and memory) are utilized for model training. These can help with making decisions on provisioning infrastructure for the different model training needs.
- **Training job events and status:** This provides visibility into the progress of a training job, such as job starting, running, completion, and failure details.
- **Model training metrics:** These are model training metrics such as loss curve and accuracy reports to help you understand the model's performance.
- **Bias detection metrics and model explainability reporting:** These metrics help you understand if there is any bias in the training datasets or machine learning models. Model explainability can also be monitored and reported to help you understand high-importance features versus low-importance features.
- **Model training bottlenecks and training issues:** These provide visibility into training issues such as vanishing gradients, poor weights initialization, and overfitting to help determine the required data, algorithmic, and training configuration changes. Metrics such as CPU and I/O bottlenecks, uneven load balancing, and low GPU utilization can help determine infrastructure configuration changes for more efficient model training.

There are multiple native AWS services for building out a model training architecture on AWS. The following diagram shows an example architecture for building a monitoring solution for a SageMaker-based model training environment:

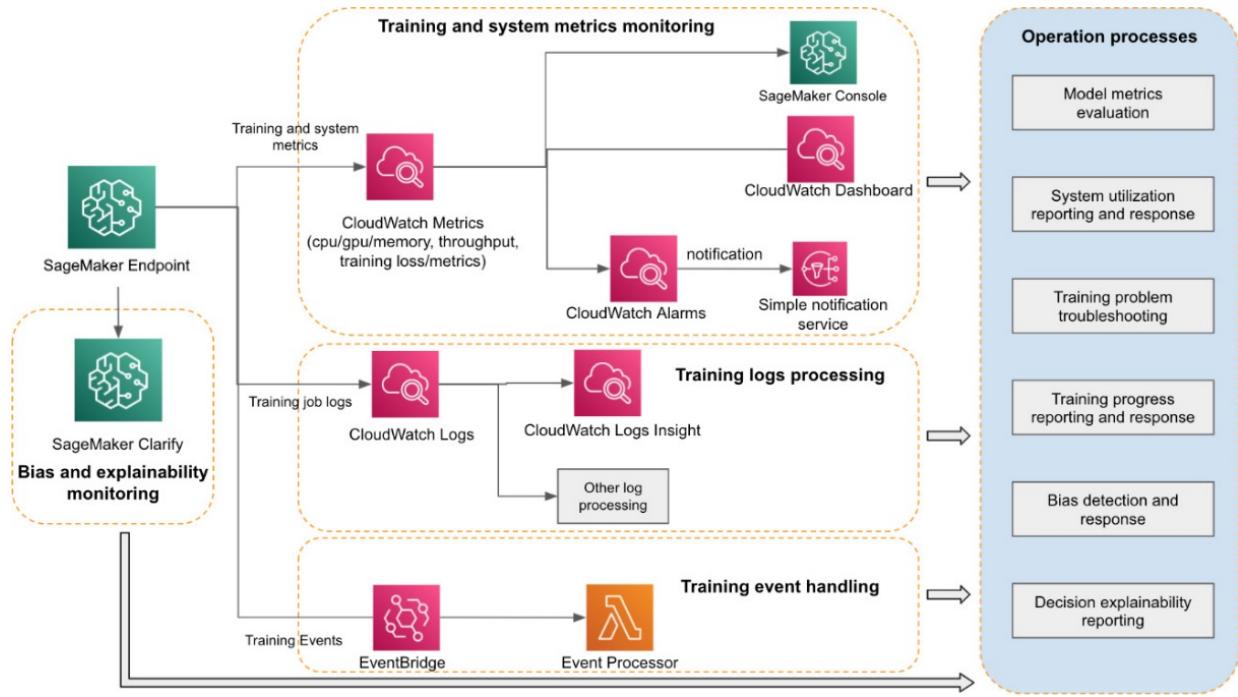


Figure 9.5: Model training monitoring architecture

This architecture lets you monitor training and system metrics and perform log capture and processing, training event capture and processing, and model training bias and explainability reporting. It helps enable operation processes, such as training progress and status reporting, model metric evaluation, system resource utilization reporting and response, training problem troubleshooting, bias detection, and model decision explainability. During model training, SageMaker can emit model training metrics, such as training loss and accuracy, to AWS CloudWatch to help with model training evaluation. AWS CloudWatch is the AWS monitoring and observability service. It collects metrics and logs from other AWS services and provides dashboards for visualizing and analyzing these metrics and logs. System utilization metrics (such as CPU/GPU/memory utilization) are also reported to CloudWatch for analysis to help you understand any infrastructure constraints or under-utilization. CloudWatch alarms can be created for a single metric or composite metrics to automate notifications or responses. For example, you can create alarms on low CPU/GPU utilization to help proactively identify sub-optimal hardware configuration for the training job. And when an alarm is triggered, it can send automated notifications (such as SMS and emails) to support for review via AWS **Simple Notification Service (SNS)**. You can use CloudWatch Logs to collect, monitor, and analyze the logs that are emitted by your training jobs. You can use these captured logs to understand the progress of your training jobs and identify errors and patterns to help troubleshoot any model training problems. For example, the CloudWatch Logs logs might contain errors such as insufficient GPU memory to run model training or permission issues when accessing specific resources to help you troubleshoot model training problems. By default, CloudWatch Logs provides a UI tool called *CloudWatch Logs Insights* for interactively analyzing logs using a purpose-built query language. Alternatively, these logs can also be forwarded to an Elasticsearch cluster for analysis and querying. These logs can be aggregated in a designated logging and monitoring account to centrally manage log access and analysis. SageMaker training jobs can also send events, such as a training job status changing from running to complete. You can create automated notification and response mechanisms based on these different events. For example, you can send out notifications to data scientists when a training job is either completed successfully or failed, along with a failure reason. You can also automate responses to these failures to the different statuses, such as model retraining on a particular failure condition. The **SageMaker Clarify** component can detect data and model bias and provide model explainability reporting on the trained model. You can access bias and model explainability reports inside the SageMaker Studio UI or SageMaker APIs. The **SageMaker Debugger** component can detect model training issues such as non-

converging conditions, resource utilization bottlenecks, overfitting, vanishing gradients, or conditions where the gradients become too small for efficient parameter updates. Alerts can be sent when training anomalies are found.

Model endpoint monitoring

Model endpoint monitoring provides visibility into the performance of the modeling serving infrastructure, as well as model-specific metrics such as data drift, model drift, and inference explainability. The following are some of the key metrics for model endpoint monitoring:

- **General system and resource utilization and error metrics:** These provide visibility into how the infrastructure resources (such as CPU, GPU, and memory) are utilized for model servicing. These can help with making decisions on provisioning infrastructure for the different model serving needs.
- **Data statistics monitoring metrics:** The statistical nature of data could change over time, which can result in degraded ML model performance from the original benchmarks. These metrics can include basic statistics deviations such as mean and standard changes, as well as data distribution changes.
- **Model quality monitoring metrics:** These model quality metrics provide visibility into model performance deviation from the original benchmark. These metrics can include regression metrics (such as MAE and RMSE) and classification metrics (such as confusion matrix, F1, precision, recall, and accuracy).
- **Model inference explainability:** This provides model explainability on a per prediction basis to help you understand what features had the most influence on the decision that was made by the prediction.
- **Model bias monitoring metrics:** Similar to bias detection for training, the bias metrics help us understand model bias at inference time.

The model monitoring architecture relies on many of the same AWS services, including CloudWatch, EventBridge, and SNS. The following diagram shows an architecture pattern for a SageMaker-based model monitoring solution:

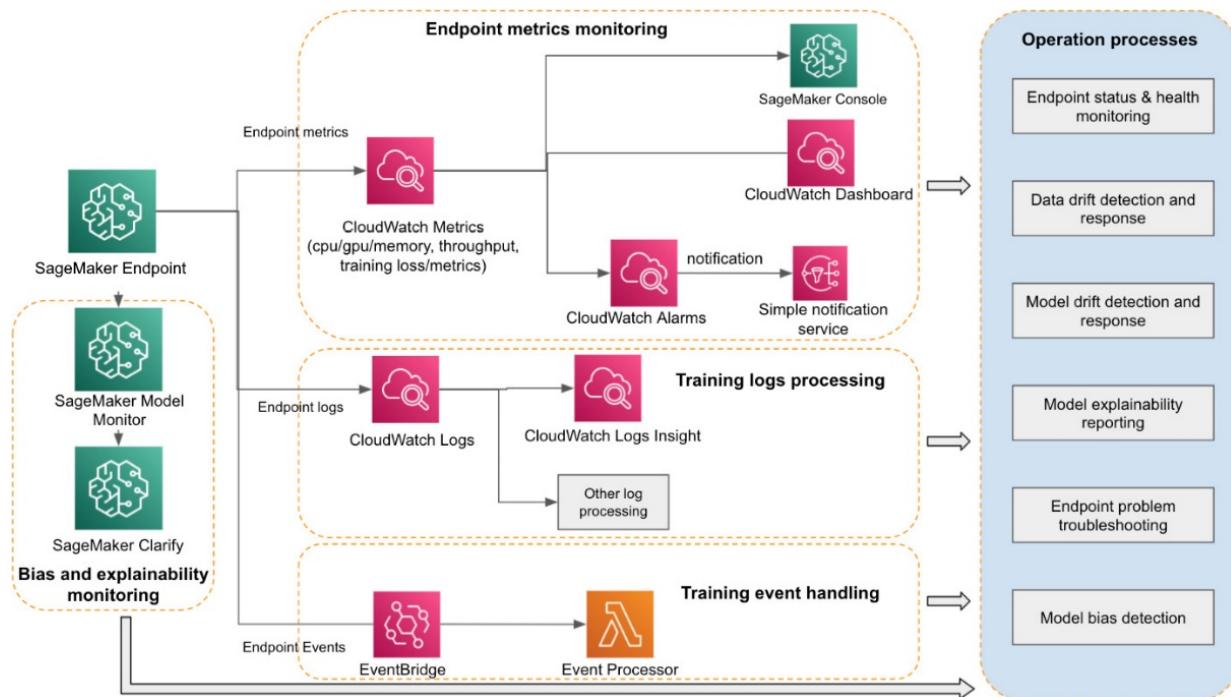


Figure 9.6: Model endpoint monitoring architecture

This architecture works similarly to the model training architecture. **CloudWatch metrics** capture endpoint metrics such as CPU/GPU utilization and model invocation metrics (number of invocations and errors) and model latencies. These metrics help with operations such as hardware optimization and endpoint scaling. **CloudWatch**

Logs captures logs that are emitted by the model serving endpoint to help us understand the status and troubleshoot technical problems. Similarly, endpoint events, such as the status changing from **Creating** to **InService**, can help you build automated notification pipelines to kick off corrective actions or provide status updates. In addition to system and status-related monitoring, this architecture also supports data and model-specific monitoring through a combination of SageMaker Model Monitor and SageMaker Clarify. Specifically, SageMaker Model Monitor can help you monitor data drift and model quality. For data drift, SageMaker Monitor can use the training dataset to create baseline statistics metrics such as standard deviation, mean, max, min, and data distribution for the dataset features. It uses these metrics and other data characteristics, such as data types and completeness, to establish constraints. Then, it captures the input data in the production environment, calculates the metrics, compares them with the baseline metrics/constraints, and reports baseline drifts. Model Monitor can also report data quality issues such as incorrect data types and missing values. Data drift metrics can be sent to CloudWatch Metrics for visualization and analysis, and CloudWatch Alarms can be configured to trigger a notification or automated response when a metric crosses a predefined threshold. For model quality monitoring, it creates baseline metrics (such as MAE for regression and accuracy for classification) using the baseline dataset, which contains both predictions and true labels. Then, it captures the predictions in production, ingests ground truth labels, and merges the ground truth with the predictions to calculate various regression and classification metrics before comparing those with the baseline metrics. Similar to data drift metrics, model quality metrics can be sent to CloudWatch Metrics for analysis and visualization, and CloudWatch Alarms can be configured for automated notifications and/or responses. The following diagram shows how SageMaker Model Monitor works:

Model Deployment and Monitoring for Drift

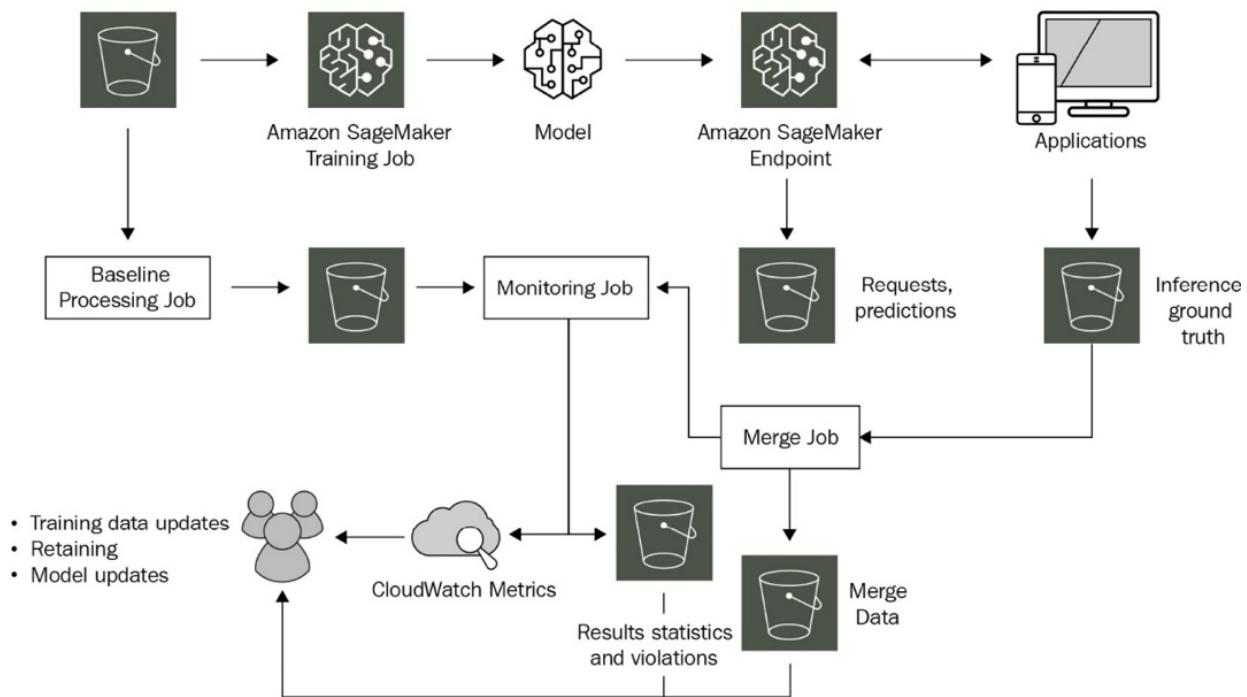


Figure 9.7: SageMaker Model Monitor process flow

For bias detection, SageMaker Clarify can monitor bias metrics for deployed models continuously and raises alerts through CloudWatch when a metric crosses a threshold. We will cover bias detection in detail in *Chapter 13, Privacy, Fairness, and Adversarial*.

ML pipeline monitoring

The ML pipeline's execution needs to be monitored for statuses and errors, so corrective actions can be taken as needed. During a pipeline execution, there are pipeline-level statuses/events as well as stage-level and action-level statuses/events. You can use these events and statuses to understand the progress of each pipeline and stage and get alerted when something is wrong. The following diagram shows how AWS CodePipeline, CodeBuild, and CodeCommit can work with CloudWatch, CloudWatch Logs, and EventBridge for general status monitoring and reporting, as well as problem troubleshooting:

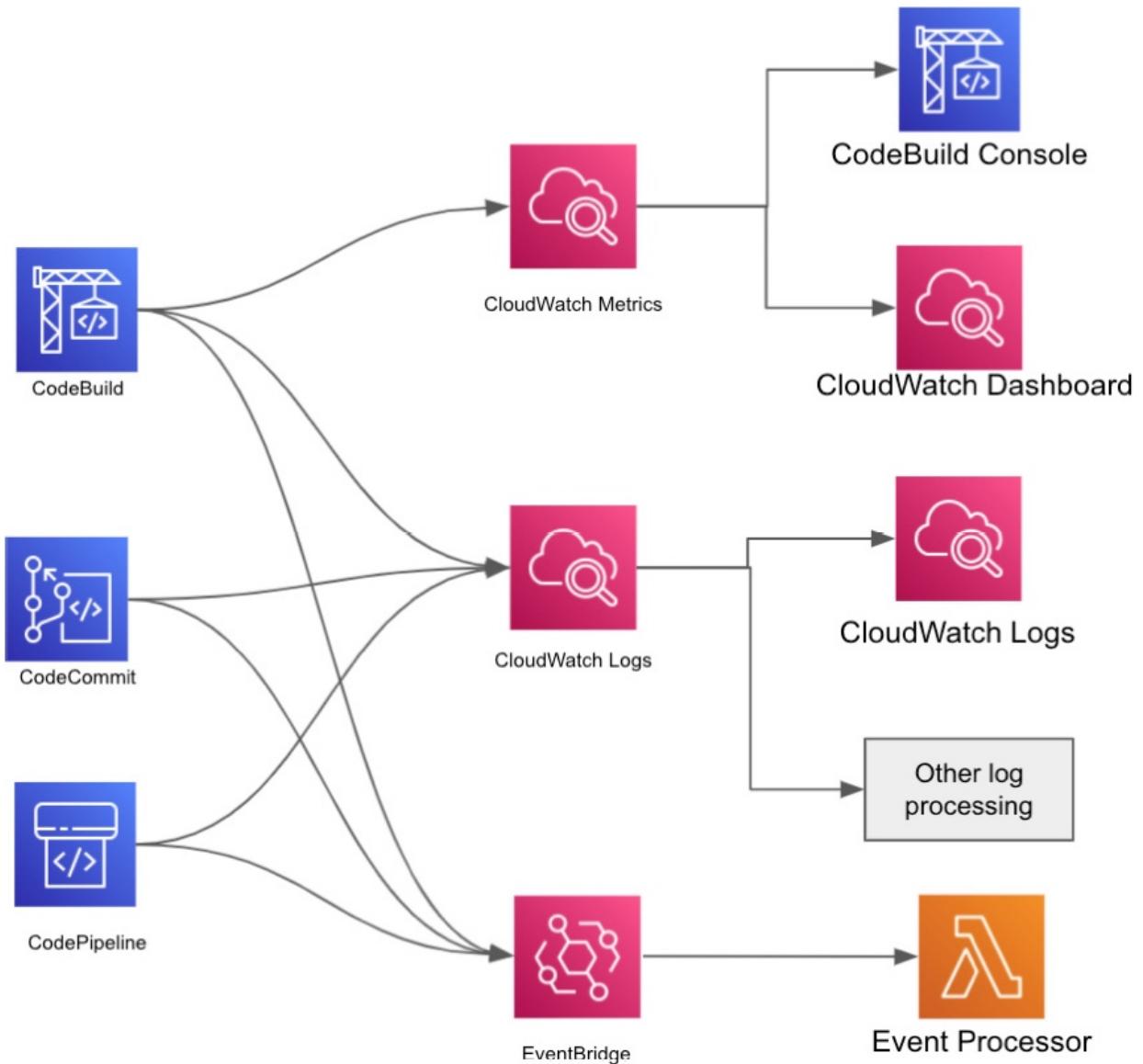


Figure 9.8: ML CI/CD pipeline monitoring architecture

CodeBuild can send metrics, such as SucceededBuilds, FailedBuilds, and Duration metrics. These CodeBuild metrics can be accessed through both the CodeBuild console and the CloudWatch dashboard. CodeBuild, CodeCommit, and CodePipeline can all emit events to EventBridge to report detailed status changes and trigger custom event processing, such as notifications, or log the events to another data repository for event archiving. All three services can send detailed logs to CloudWatch Logs to support operations such as troubleshooting or detailed error reporting. Step Functions also provides a list of monitoring metrics to CloudWatch, such as execution metrics (such as execution failure, success, abort, and timeout) and activity metrics (such as activity started, scheduled, and succeeded). You can view these metrics in the management console and set a threshold to set up alerts.

Service provisioning management

Another key component of enterprise-scale ML platform management is **service provisioning management**. For large-scale service provisioning and deployment, an automated and controlled process should be adopted. Here, we will focus on provisioning the ML platform itself, not provisioning AWS accounts and networking, which should be established as the base environment for ML platform provisioning in advance. For ML platform provisioning, there are the following two main provisioning tasks:

- **Data science environment provisioning:** Provisioning the data science environment for data scientists mainly includes provisioning data science and data management tools, storage for experimentation, as well as access entitlement for data sources and pre-built ML automation pipelines.
- **ML automation pipeline provisioning:** ML automation pipelines need to be provisioned in advance for data scientists and MLOps engineers to use them to automate different tasks such as container build, model training, and model deployment.

There are multiple technical approaches to automating service provisioning on AWS, such as using provisioning shell scripts, CloudFormation scripts, and AWS Service Catalog. With shell scripts, you can sequentially call the different AWS CLI commands in a script to provision different components, such as creating a SageMaker Studio Notebook. CloudFormation is the IaC service for infrastructure deployment on AWS. With CloudFormation, you create templates that describe the desired resources and dependencies that can be launched as a single stack. When the template is executed, all the resources and dependencies specified in the stack will be deployed automatically. The following code shows the template for deploying a SageMaker Studio domain:

```
Type: AWS::SageMaker::Domain
Properties:
  AppNetworkAccessType: String
  AuthMode: String
  DefaultUserSettings:
    UserSettings
  DomainName: String
  KmsKeyId: String
  SubnetIds:
    - String
  Tags:
    - Tag
  VpcId: String
```

AWS Service Catalog allows you to create different IT products to be deployed on AWS. These IT products can include SageMakenotebooks, a CodeCommit repository, and CodePipeline workflow definitions. AWS Service Catalog uses CloudFormation templates to describe IT products. With Service Catalog, administrators create IT products with CloudFormation templates, organize these products by product portfolio, and entitle end users with access. The end users then access the products from the Service Catalog product portfolio. The following diagram shows the flow of creating a Service Catalog product and launching the product from the Service Catalog service:

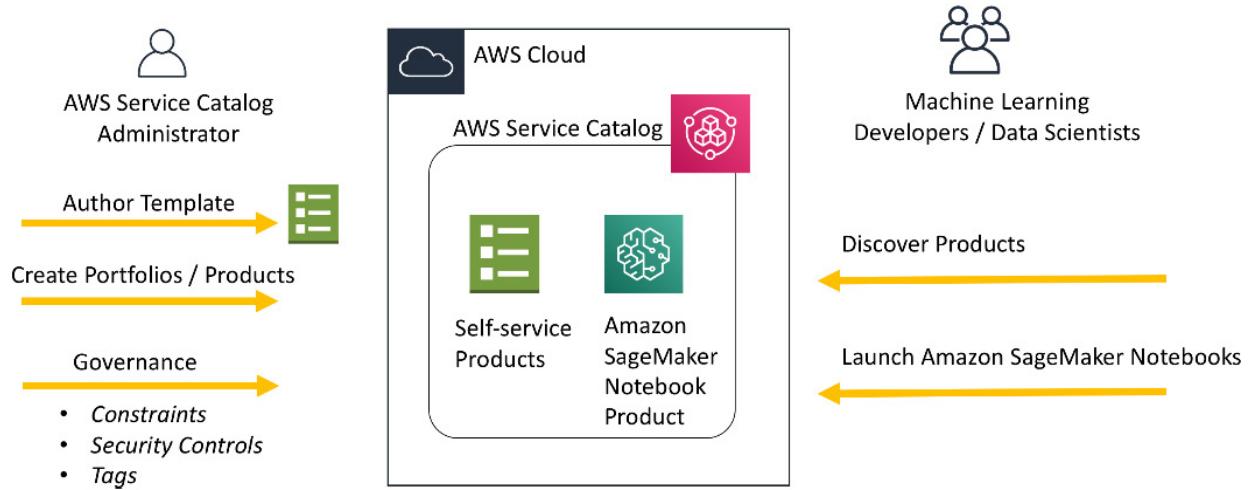


Figure 9.9: Service Catalog workflow

For large-scale and governed IT product management, Service Catalog is the recommended approach. Service Catalog supports multiple deployment options, including single AWS account deployments and hub-and-spoke cross-account deployments. A hub-and-spoke deployment allows you to centrally manage all the products and make them available in different accounts. In our enterprise ML reference architecture, we use the hub-and-spoke architecture to support the provisioning of data science environments and ML pipelines, as shown in the following diagram:

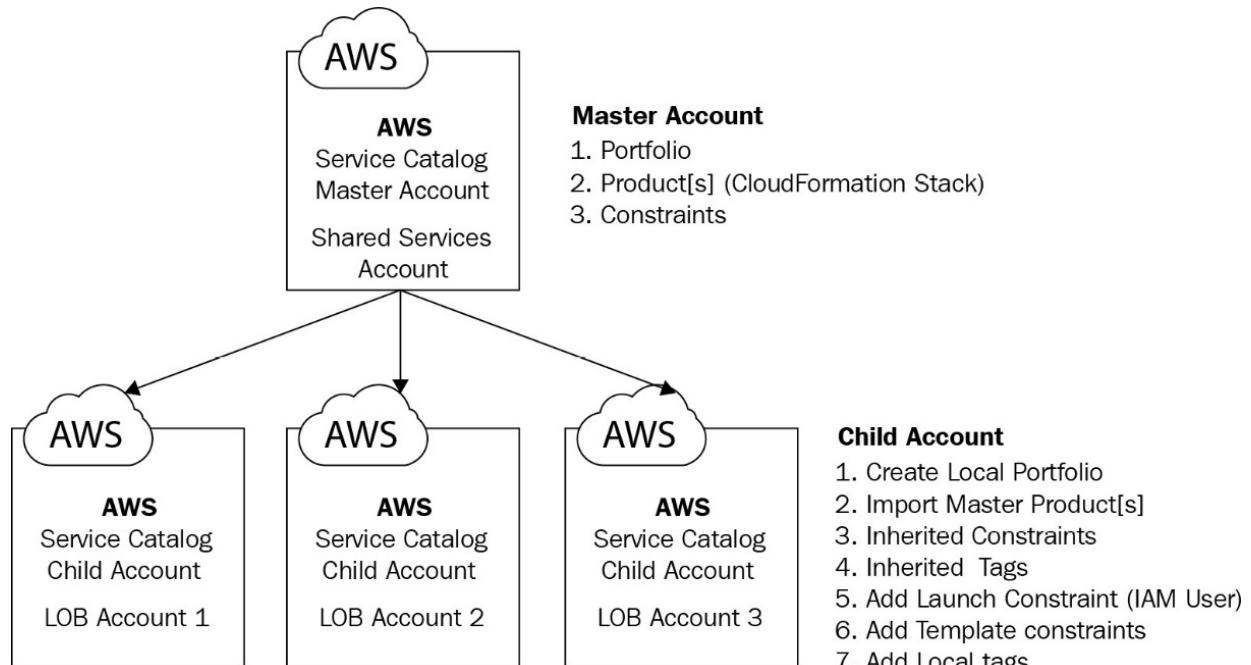


Figure 9.10: The hub-and-spoke Service Catalog architecture for enterprise ML product management

In the preceding architecture, we set up the central portfolio in the shared services account. All the products, such as creating new Studio domains, new Studio user profiles, CodePipeline definitions, and training pipeline definitions, are centrally managed in the central hub account. Some products are shared with the different data science accounts to create data science environments for data scientists and teams. Some other products are shared with model training accounts for standing up ML training pipelines.

Best practices in building and operating ML platform

Constructing an enterprise ML platform is a multifaceted undertaking. It often requires significant time, with organizations taking six months or more to implement the initial phase of their ML platform. Continuous efforts are needed to incorporate new functionalities and enhancements for many years to come. Onboarding users and ML projects onto the new platform is another demanding aspect, involving extensive education for the user base and providing direct technical support. In some cases, platform adjustments might be necessary to ensure smooth onboarding and successful utilization. Having collaborated with many customers in building their enterprise ML platform, I have identified some best practices for the construction and adoption of ML platform.

ML platform project execution best practices

- **Assemble cross-functional teams:** Bring together data engineers, ML researchers, DevOps engineers, application developers, and business domain experts into integrated teams. This diversity of skills and perspectives will enrich the platform design and implementation.
- **Develop governance requirements and processes:** Define processes and requirements early on for model verification, explainability, ethics reviews, and approvals prior to production deployment. This will embed responsible AI practices into the platform.
- **Define key performance indicators (KPIs) to measure success:** Identify relevant business KPIs and implement processes to actively monitor and report on model and platform impact on these KPIs. Share reports with stakeholders.
- **Select pilot ML workloads:** Choose a few pilot ML projects or workloads to implement first on the new platform. Learn from these real-world use cases to validate and improve the platform design and capabilities.
- **Define the target state and execute in phases:** Articulate the long-term vision and target state for the enterprise ML platform. However, strategically execute adoption in incremental phases for faster learning.

ML platform design and implementation best practices

- **Adopt fully managed built-in capabilities:** Leverage SageMaker's managed algorithms, containers, and features as defaults to reduce overhead and simplify integration. Only use custom if needed.
- **Implement infrastructure as code:** Use CloudFormation or Terraform to provision, configure, and manage ML infrastructure through code. Enables consistency and automation.
- **Build CI/CD pipelines:** Implement continuous integration & deployment pipelines leveraging CodePipeline, CodeBuild, CodeDeploy, and SageMaker for automated workflows. Consider GitHub Actions/Jenkins if needed.
- **Automate experiment tracking:** Configure SageMaker or 3rd party tools to automatically log model training metadata like parameters, metrics, and artifacts. Enables debugging, comparisons, and reproducibility.
- **Establish an approved library repository:** Create a centralized, governed repository of approved libraries and packages for training, deployment, and inference code. Ensures consistency.
- **Design for scalability and spikes:** Architect the platform to handle varied usage patterns and traffic spikes via auto-scaling capabilities.
- **Prioritize security from the start:** Implement security best practices including scanning, patching, encryption, and access controls. Have an incident response plan.
- **Build self-service capability:** Develop self-service functionality early and evolve it to empower users while maintaining governance.
- **Centralize model repository:** Use a single, central repository for models to improve collaboration, discovery, compliance, and efficient deployment.
- **Establish a central feature store:** Implement a centralized feature store for sharing, monitoring, and governing feature engineering work and usage.

Platform use and operations best practices

- **Limit production access:** Restrict access to production systems to only essential support and operations staff. This reduces the risk of mistakes or unauthorized changes.
- **Optimize costs:** Leverage auto-scaling, spot instances, availability-based pricing and other capabilities to optimize and reduce cloud costs.
- **Monitoring and observability:** Actively monitor model accuracy, data drift, system performance etc. using CloudWatch, SageMaker Debugger, Model Monitor and other tools.
- **Establish change management:** Define a structured process for managing, reviewing, approving and communicating platform/model changes prior to deployment.
- **Incident management process:** Institute an incident response plan with procedures to detect, escalate and resolve production issues and anomalies in a timely manner.
- **Multi-AZ and region deployments:** Deploy models and platform infrastructure across multiple availability zones and regions to improve resilience and minimize latency.
- **Release management:** Implement structured release processes for coordinating, reviewing and planning changes and new model/platform versions before deployment.
- **Capacity planning:** Proactively assess and project infrastructure capacity needs based on roadmaps and workloads. Scale appropriately.
- **Resource tagging:** A properly designed tagging strategy provides organization, discovery, security, automation, compliance and improved visibility in an ML platform.

Implementing a robust enterprise machine learning platform requires thoughtful strategy and orchestration across people, processes, and technology. By bringing together cross-functional teams, instituting responsible AI governance, monitoring business impact, and designing for scalability, security, and collaboration, organizations can accelerate their AI journey. Adopting modern infrastructure-as-code, CI/CD pipelines, and cloud services lays a solid technology foundation. However, to realize value, platforms must be tightly integrated with line-of-business priorities and continuously provide trustworthy AI. With deliberate planning and phased execution centered on business goals and users, companies can transform into AI-driven enterprises. The key is to balance innovation with governance, move fast through automation while maintaining control, and evolve a platform that responsibly democratizes AI capabilities for both experts and business users. This enables embedding reliable and accountable AI throughout operations for a competitive advantage.

Hands-on exercise – building an MLOps pipeline on AWS

In this hands-on exercise, you will get hands on with building a simplified version of the enterprise MLOps pipeline. For simplicity, we will not be using the multi-account architecture for the enterprise pattern. Instead, we will build several core functions in a single AWS account. The following diagram shows what you will be building:

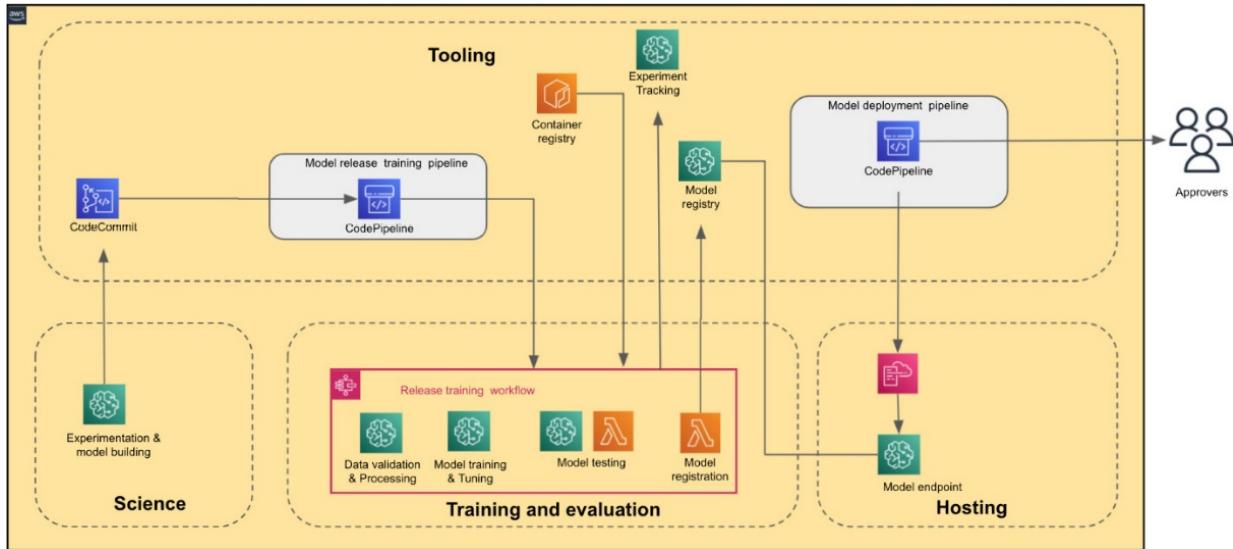


Figure 9.11: Architecture of the hands-on exercise

At a high level, you will create two pipelines using CloudFormation: one for model training and one for model deployment.

Creating a CloudFormation template for the ML training pipeline

In this section, we will create two CloudFormation templates that do the following:

- The first template creates AWS Step Functions for an ML model training workflow that performs data processing, model training, and model registration. This will be a component of the training pipeline.
- The second template creates a CodePipeline ML model training pipeline definition with two stages:
 1. A **source stage**, which listens to changes in a CodeCommit repository to kick off the execution of the Step Functions workflow that we created
 2. A **deployment stage**, which kicks off the execution of the ML model training workflow

Now, let's get started with the CloudFormation template for the Step Functions workflow:

1. Create a Step Functions workflow execution role called

`AmazonSageMaker-StepFunctionsWorkflowExecutionRole`. Then, create and attach the following IAM policy to it. This role will be used by the Step Functions workflow to provide permission to invoke the various SageMaker APIs. Take note of the ARN of the newly created IAM role as you will need it for the next step. You can find the complete code sample at <https://github.com/PacktPublishing/The-Machine-Learning-Solutions-Architect-Handbook/blob/main/Chapter09/AmazonSageMaker-StepFunctionsWorkflowExecutionRole-policy.json>:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sagemaker:CreateModel",
        "sagemaker:DeleteEndpointConfig",
        "sagemaker:DescribeTrainingJob",
        "sagemaker:CreateEndpoint",
        "sagemaker:StopTrainingJob",
        "sagemaker:CreateTrainingJob",
        "sagemaker:ListTrainingJobs",
        "sagemaker:ListEndpointConfigs",
        "sagemaker:ListEndpoints",
        "sagemaker:ListModels",
        "sagemaker:ListTags"
      ]
    }
  ]
}
```

```

        "sagemaker:UpdateEndpoint",
        "sagemaker>CreateEndpointConfig",
        "sagemaker>DeleteEndpoint",
    "sagemaker:AddTags"
    ],
    "Resource": [
        "arn:aws:sagemaker:*:*:*"
    ]
},
...
}

```

1. Copy and save the following code block to a file locally and name it `training_workflow.yaml`. You can find the complete file at https://github.com/PacktPublishing/The-Machine-Learning-Solutions-Architect-Handbook/blob/main/Chapter09/training_workflow.yaml. This CloudFormation template will create a Step Functions state machine with a training step and model registration step. The training step will train the same BERT model we trained in *Chapter 8, Building a Data Science Environment Using AWS ML Services*. For simplicity, we will reuse the same source data and training script as well to demonstrate the MLOps concepts we have learned about in this chapter. Note that we are using CloudFormation here to demonstrate managing IaC. Data scientists also have the option to use the Step Functions Data Science SDK to create the pipeline using a Python script:

```

AWSTemplateFormatVersion: 2010-09-09
Description: 'AWS Step Functions sample project for training a model and save the model'
Parameters:
  StepFunctionExecutionRoleArn:
    Type: String
    Description: Enter the role for Step Function Workflow execution
    ConstraintDescription: requires a valid arn value
    AllowedPattern: 'arn:aws:iam::\w+::role/.*'
Resources:
  TrainingStateMachine2:
    Type: AWS::StepFunctions::StateMachine
    Properties:
      RoleArn: !Ref StepFunctionExecutionRoleArn
      DefinitionString: !Sub |
        {
          "StartAt": "SageMaker Training Step",
          "States": {
            "SageMaker Training Step": {
              "Resource": "arn:aws:states:::sagemaker:createTrainingJob.sync",
...

```

1. Launch the newly created cloud template in the CloudFormation console. Make sure that you provide a value for the `StepFunctionExecutionRoleArn` field when prompted. This is the ARN you took down from the last step. Once the CloudFormation execution is completed, go to the **Step Functions** console to test it.
2. Test the workflow in the **Step Functions** console to make sure it works. Navigate to the newly created **Step Functions state machine** and click on **Start Execution** to kick off the execution. When you're prompted for any input, copy and paste the following JSON as input for the execution. These are the input values that will be used by the Step Functions workflow. Make sure that you replace the actual values with the values for your environment. For the AWS hosting account information for the training images, you can look up the account number at https://github.com/aws/deep-learning-containers/blob/master/available_images.md:

```
{
  "TrainingImage": "<aws hosting account>.dkr.ecr.<aws region>.amazonaws.com/pytorch-training:1.0",
  "S3OutputPath": "s3://<your s3 bucket name>/sagemaker/pytorch-bert-financetext",
  "SageMakerRoleArn": "arn:aws:iam::<your aws account>:role/service-role/<your sagemaker execution role>",
  "S3UriTraining": "s3://<your AWS S3 bucket>/sagemaker/pytorch-bert-financetext/train.csv",
  "S3UriTesting": "s3://<your AWS S3 bucket>/sagemaker/pytorch-bert-financetext/test.csv",
  "InferenceImage": "aws hosting account>.dkr.ecr. <aws region>.amazonaws.com/pytorch-inference",
  "SAGEMAKER_PROGRAM": "train.py",
  "SAGEMAKER_SUBMIT_DIRECTORY": "s3:// <your AWS S3 bucket> /<path to the source code>/sourcedir"
}
```

```

    "SAGEMAKER_REGION": "<your aws region>"  
}

```

1. Check the processing status in the **Step Functions** console and make sure that the model has been trained and registered correctly. Once everything is completed, save the input JSON in *Step 4* to a file called `sf_start_params.json`.
2. Create a CodeCommit code repository called "MLSA-repo" and upload the `sf_start_params.json` file into it. We will use this file in the `CodeCommit` repository for the next section of the lab.

Now, we are ready to create the CloudFormation template for the CodePipeline training pipeline. This pipeline will listen to changes to a `CodeCommit` repository and invoke the Step Functions workflow we just created:

1. Copy and save the following code block to a file called `mlpipeline.yaml`. This is the template for building the training pipeline. You can find the complete file at <https://github.com/PacktPublishing/The-Machine-Learning-Solutions-Architect-Handbook/blob/main/Chapter09/mlpipeline.yaml>:

```

Parameters:  
BranchName:  
  Description: CodeCommit branch name  
  Type: String  
  Default: master  
RepositoryName:  
  Description: CodeCommit repository name  
  Type: String  
  Default: MLSA-repo  
ProjectName:  
  Description: ML project name  
  Type: String  
  Default: FinanceSentiment  
MlopsStepFunctionArn:  
  Description: Step Function Arn  
  Type: String  
  Default: <arn for the step function state machine from step 2>  
Resources:  
CodePipelineArtifactStoreBucket:  
  Type: 'AWS::S3::Bucket'  
  DeletionPolicy: Delete  
Pipeline:  
  Type: 'AWS::CodePipeline::Pipeline'  
...

```

1. Similarly, let's launch this cloud template in the CloudFormation console to create the pipeline definition for execution. Once the CloudFormation template has been executed, navigate to the CodePipeline management console to verify that it has been created. The CloudFormation execution will also execute the newly created pipeline automatically, so you should see that it already ran once. You can test it again by clicking on the **Release changes** button in the **SageMaker management** console.

We want to be able to kick off the CodePipeline execution when a change is made (such as a code commit) in the `CodeCommit` repository. To enable this, we need to create a CloudWatch event that monitors this change and kicks off the pipeline. Let's get started:

1. Add the following code block to the `mlpipeline.yaml` file, just before the **Outputs** section, and save the file as `mlpipeline_1.yaml`. You can find the complete file at https://github.com/PacktPublishing/The-Machine-Learning-Solutions-Architect-Handbook/blob/main/Chapter09/mlpipeline_1.yaml:

```

AmazonCloudWatchEventRole:  
  Type: 'AWS::IAM::Role'  
  Properties:  
    AssumeRolePolicyDocument:  
      Version: 2012-10-17  
      Statement:

```

```

- Effect: Allow
Principal:
  Service:
    - events.amazonaws.com
Action: 'sts:AssumeRole'
Path: /
Policies:
  - PolicyName: cwe-pipeline-execution
  PolicyDocument:
...

```

1. Now, run this CloudFormation template to create a new pipeline. You can delete the previously created pipeline by deleting the CloudFormation stack. This will run the pipeline again automatically. Wait until the pipeline's execution is complete before you start the next step.
2. Now, let's test the automatic execution of the pipeline by committing a change to the code repository. Find a file in your cloned code repository directory. Create a new file called `pipelinetest.txt` and commit the change to the code repository. Navigate to the CodePipeline console; you should see the **codemerge-events-pipeline** pipeline starting to run.

Congratulations! you have successfully used CloudFormation to build a `CodePipeline`-based ML training pipeline that automatically runs when there is a file change in a `CodeCommit` repository. Next, let's build the ML deployment pipeline for the model.

Creating a CloudFormation template for the ML deployment pipeline

To start creating a deployment, perform the following steps:

1. Copy the following code block to create a file called `mldeployment.yaml`. This CloudFormation template will deploy a model using the SageMaker hosting service. Make sure that you enter the correct model name for your environment:

```

Description: Basic Hosting of registered model
Parameters:
  ModelName:
    Description: Model Name
    Type: String
    Default: <model name>
  Resources:
    Endpoint:
      Type: AWS::SageMaker::Endpoint
      Properties:
        EndpointConfigName: !GetAtt EndpointConfig.EndpointConfigName
        EndpointConfig:
          Type: AWS::SageMaker::EndpointConfig
          Properties:
            ProductionVariants:
              ProductionVariant:
                InitialInstanceCount: 1
                InitialVariantWeight: 1.0
                InstanceType: ml.m4.xlarge
                ModelName: !Ref ModelName
                VariantName: !Ref ModelName
            Outputs:
              EndpointId:
                Value: !Ref Endpoint
              EndpointName:
                Value: !GetAtt Endpoint.EndpointName

```

1. Create a CloudFormation stack using this file and verify that a SageMaker endpoint has been created. Now, upload the `mldeployment.yaml` file to the code repository directory and commit the change to `CodeCommit`. Note that this file will be used by the CodePipeline deployment pipeline, which we will create in the following steps.

2. Before we create the deployment pipeline, we need a template config file for passing parameters to the deployment template when it is executed. Here, we need to pass the model name to the pipeline. Copy the following code block, save it to a file called `mldeployment.json`, upload it to the code repository directory in Studio, and commit the change to `codecommit`:

```
{
  "Parameters" : {
    "ModelName" : <name of the financial sentiment model you have trained>
  }
}
```

1. Now, we can create a CodePipeline pipeline CloudFormation template for automatic model deployment. This pipeline has two main stages:

- The first stage fetches source code (such as the configuration file we just created and the `mldeployment.yaml` template) from a `CodeCommit` repository.
- The second stage creates a CloudFormation change set (**a change set** is the difference between a new template and an existing CloudFormation stack) for the `mldeployment.yaml` file we created earlier. It adds a manual approval step and then deploys the CloudFormation template's `mldeployment.yaml` file.

This CloudFormation template also creates supporting resources, including an S3 bucket for storing the CodePipeline artifacts, an IAM role for CodePipeline to run with, and another IAM role for CloudFormation to use to create the stack for `mldeployment.yaml`.

1. Copy the following code block and save the file as `mldeployment-pipeline.yaml`. You can find the complete code sample at <https://github.com/PacktPublishing/The-Machine-Learning-Solutions-Architect-Handbook/blob/main/Chapter09/mldeployment-pipeline.yaml>:

```
Parameters:
  BranchName:
    Description: CodeCommit branch name
    Type: String
    Default: master
  RepositoryName:
    Description: CodeCommit repository name
    Type: String
    Default: MLSA-repo
  ProjectName:
    Description: ML project name
    Type: String
    Default: FinanceSentiment
  CodePipelineSNSTopic:
    Description: SNS topic for NotificationArn
    Default: arn:aws:sns:<aws region>:<AWS account>:<SNS topic name>
    Type: String
  ProdStackConfig:
    Default: mldeploymentconfig.json
    Description: The configuration file name for the production WordPress stack
    Type: String
  ProdStackName:
    Default: FinanceSentimentMLStack1
    Description: A name for the production stack
    Type: String
  TemplateFileName:
    Default: mldeployment.yaml
    Description: The file name of the template
    Type: String
  ChangeSetName:
    Default: FinanceSentimentchangeset
    Description: A name for the production stack change set
    Type: String
Resources:
  CodePipelineArtifactStoreBucket:
    Type: 'AWS::S3::Bucket'
```

```
  DeletionPolicy: Delete
  Pipeline:
  . . .
```

1. Now, let's launch the newly created `mldeployment-pipeline.yaml` template in the **CloudFormation** console to create the deployment pipeline, and then run the pipeline from the **CodePipeline** console.

Congratulations! You have successfully created and run a CodePipeline deployment pipeline to deploy a model from the SageMaker model registry.

Summary

In this chapter, we explored the key requirements and best practices for building an enterprise machine learning platform. We discussed how to design a platform that supports the end-to-end ML lifecycle, process automation, and separation of environments. Architectural patterns were reviewed, including how to leverage AWS services to build a robust ML platform on the cloud. The core capabilities of different ML environments were covered, such as training, hosting, and shared services. Best practices around platform design, operations, governance, and integration were also discussed. Through hands-on exercises, you gained experience in implementing components of an MLOps architecture, automating workflows, and operationalizing models. You should now have a solid understanding of what an enterprise-grade ML platform entails and key considerations for building one on AWS leveraging proven patterns. In the next chapter, we will dive deeper into advanced ML engineering topics. This includes distributed training techniques to scale model development and low-latency serving methods for optimizing inference.

12 AI Risk Management

Join our book community on Discord

<https://packt.link/EarlyAccessCommunity>



As organizations increasingly rely on AI for critical decision-making and incorporate it into different areas of their businesses, effective AI risk management should be a top priority. Ensuring safe and compliant deployment of ML systems is essential to establish trustworthiness in the AI solutions. However, many organizations and individuals have very limited understanding of the risks associated with AI systems, often resulting in outcomes that may negatively impact organizations financially or legally. In this chapter, we will explore key AI risk scenarios, highlight the differences between AI risk management and traditional software risk management, and emphasize the importance of having a robust AI risk management practice. We will present a risk management framework that organizations can consider for managing AI risks. Finally, we will discuss how to manage risks at different stages of ML lifecycle and design ML platforms that support risk management and AI governance. Specifically, we will cover the following key topics:

- Understanding AI risk scenarios
- The regulatory landscape around AI risk management
- Understanding AI risk management
- Applying AI risk management in AI lifecycle

- Designing ML platform with governance and risk management considerations

Understanding AI risk scenarios

Many of the organizations I have worked with have very limited knowledge about the risks presented in their AI systems. They often treat AI risks the same way they deal with risks associated with traditional software. In reality, AI systems present a new set of risks that we do not normally see in traditional software. With traditional software, the risk is mainly about software vulnerability, legacy technology stack, malware, misconfiguration, and unauthorized access to data. AI systems are exposed to many of the same software risks, additionally, AI systems can present new kinds of risk such as bias and misinformation. These risks can have significant negative consequences to organizations and individuals that rely on AI systems for business operations and decision-making. AI risks can manifest in many different ways such as displaying bias behavior or producing unexpected prediction results. Many of the AI risk scenarios are also silent risks that are difficult to detect. The following are some scenarios where AI risk may arise.

Bias and discrimination

One of the most well-known risks associated with AI is the potential for displaying bias and discrimination in AI systems. This can occur when ML algorithms are trained on biased data or when the algorithms themselves are susceptible to bias behaviors. In these cases, the algorithms may learn to discriminate against certain groups, leading to unfair or discriminatory outcomes. For example, a bank can have a ML model that's trained using biased dataset such as including gender and ethnic groups as inputs, resulting in discrimination against certain gender or ethnic groups and potential violation of laws and regulations such as the Equal Credit Opportunity ACT. Nowadays, many organizations use AI to screen resumes and it has been found that some of these AI systems have displayed preferences and bias towards certain types of candidates.

Misinformation and misinterpretation

Another risk associated is the potential for generating misinformation and misinterpretation of facts. This can occur when ML algorithms are used to process large amounts of data, but the data contain errors or inconsistencies that are not easily detected. As a result, the algorithms may generate inaccurate or misleading results, leading to potential wrong decision-making. With the fast rise of generative AI technologies, such as ChatGPT and Stable Diffusion models, it is also becoming more difficult for human to distinguish reality from hallucination.

Lack of interpretability

Many ML algorithms, such as neural network, can be complex and difficult to understand, even for trained experts. This lack of transparency can make it difficult to identify the causes of problems when they arise, making it harder to develop effective mitigation to address the problem. For example, when ChatGPT provides incorrect responses to user prompts, it is often impossible to understand why it made the mistake. For regulated industries, this presents a significant challenge when organizations want to adopt more advanced black-box algorithms such as neural network, as these organizations often need to provide deterministic responses to specific inputs and questions, and how the decisions are made.

Unintended consequences

ML algorithms can sometimes produce unintended consequences or side effects that were not foreseen during the development process. The reason for this is that AI models are often optimized for a specific objective such as increasing company profit, while ignoring other factors such as gender and race. For example, an AI based target marketing system might target a subset of customers with incentives and benefits, while discriminating minority or low-income customers, in its pursuit of profit maximization.

Adversarial Attacks

ML algorithms can be vulnerable to adversarial attacks, which involve deliberately manipulating the input data to produce unexpected or undesirable outcomes, or planting back door access to ML models. For example, an attacker could use an adversarial attack to trick AI-based fraud detection system to classify fake financial transaction as legitimate transaction. ML model can also be compromised to reveal training data by using adversarial techniques such as membership inference attack.

Privacy violation and sensitive data exposure

Nowadays, many of the state-of-the-art models are trained using enormous amount of data from many different sources, and sometimes personal or sensitive information are used in the development of these models. This inherently increases the risk of potential privacy violation and unintended disclosure of sensitive data. For example, to train a medical imaging model for cancer detection, you often need real patient data such as CT scans and other personal medical information. If not handled correctly, this information can be exposed to people who are not authorized for access. Also, as part of the model training, some sensitive data can be memorized by the trained model, and the model can potentially disclose these information when making predictions.

Third party risk

While third party risk also exists with traditional software from third party vendors, AI system elevates this risk in areas that we have not seen before. With the advent of machine learning techniques such as transfer learning and fine-tuning from pre-trained models, more and more organizations are building custom models based on existing pre-trained models. However, given the black-box nature of these pre-trained models, it increases the uncertainty of the model behavior and unknowns around the scientific validity of the models. Since the pre-trained models were outside of security and process controls of the consuming organizations, the consuming organization may inherit risks that may already exist in the pre-trained models such bias or backdoor attack vulnerabilities.

Model testing risk

Comparing to traditional software, the testing standards and tools for AI based software and models are underdeveloped. Traditional software testing mainly focuses on functional components such as user interface flow or business logic that's well defined, and nonfunctional areas such as scalability and latency. With AI/ML testing, in addition to many of the traditional software testing requirements, there are new testing concepts such as error analysis of different failure modes, model sensitivity, model robustness, and adversarial testing, which are more difficult to perform than the traditional software testing. The available testing tool in this domain is also very limited, often data scientists and testers need to manually prepare different testing scenarios and testing data.

The regulatory landscape around AI risk management

With the fast advancement of AI technologies and adoption in critical business decision makings, and the negative impacts that AI systems can potentially cause to individuals, organizations, and societies, many countries and jurisdictions have established policies, guidance, regulations to help manage the risks of AI adoption. It is also expected more and more legislations to be proposed and passed by different countries and jurisdictions at fast rate. In the United States, the Federal Reserve and the Office of Controller and Currency (OCC) have published the Supervisory Guidance on Model Risk Management (OCC 2011-2012 / SR 11-7) as early as 2011. SR 11-7 has become the key regulatory guidance for model risk management in the US. This guidance establishes the main principles for model risk management covering governance, policies and controls, model development, implementation and use, and model validation processes. In the governance and policy area, it provides guidance on model inventory management, risk rating, roles, and responsibilities. In the model development and implementation area, it covers topics such as design process, data assessment, model testing, and documentation. And in the validation area, it provides guidance on validation procedure, monitoring,

and finding resolutions. In Europe, the European Central Bank (ECB) Banking Supervision launched the Targeted Review of Internal Models (TRIM) guideline in 2016 to provide guidance on model risk management framework (MRM). Specifically, the guideline states that an MRM needs to have a model inventory to allow a holistic view of the models and their applications, a guideline for identifying and mitigating known model deficiencies, definitions of roles and responsibilities, and definitions of policies, measurement procure and reporting. More recently in 2021, the European Union introduced EU AI Act to promote the benefits of AI, while also to ensure the safe and responsible use of AI in European Union. The Act takes a risk-based approach to regulate AI, with different requirements based on the level of risks associated with the AI systems. For example, AI systems supporting critical infrastructure would be rated with the highest risk designation and will require the strictest oversight and regulations. The regulation also proposes provisions for AI transparency and accountability in AI-decision making process, such as requirements for explainability and ability to challenge the decision made by AI systems. It also has new rules for AI uses in biometric identification and surveillance.

Understanding AI risk management

To address the various risks associated with AI and to comply with different compliance regulations, many organizations, especially in the regulated industry, have developed and implemented AI risk management programs. In short, AI risk management is the process of identifying, assessing, and mitigating the risk associated with the use of AI in automated decision-making. The ultimate goal of AI risk management is to establish trust in the AI/ML systems and ensure compliance with applicable rules and regulations. Organizationally, AI risk management can sit in various areas such as the Chief Risk Officer (CRO) department, CIO/CTO organization, or directly within business units.

Placing trust in AI systems

Trusting an AI system requires rigorous assessment and considerations of the AI system across many different dimensions and criteria. Functionally, a trusted AI system need to provide valid predictions/responses reliably for its intended use. This means predictions/responses generated are consistently valid and can be trusted for decision-making reliably. Ethically, a trusted AI system needs to be safe to use, explainable, privacy-protected, and fair with bias properly managed and mitigated. From a cybersecurity perspective, a trusted AI system also needs to be secure and resilient against adversarial attacks. Lastly, a trustworthy AI system need to provide transparency such as what and how data, algorithms, and models are used in the system. It is worth noting that it is often a balancing act and trade-off across these different dimensions when building and operating a trustworthy system, based on the needs and objective of an organization. For example, to protect privacy, an organization might need to make sacrifices in model accuracy or prediction speed.

Creating risk management framework for trustworthy AI

Now we understand what it takes to have trust in AI systems, let's explore and dive deep into AI risk management framework and its various components. The following Figure illustrates the key components in AI risk management, which is mainly consisted of applying model risk management, enterprise risk management, and third party risk management across AI lifecycle, governed by a set of AI risk governance principals.

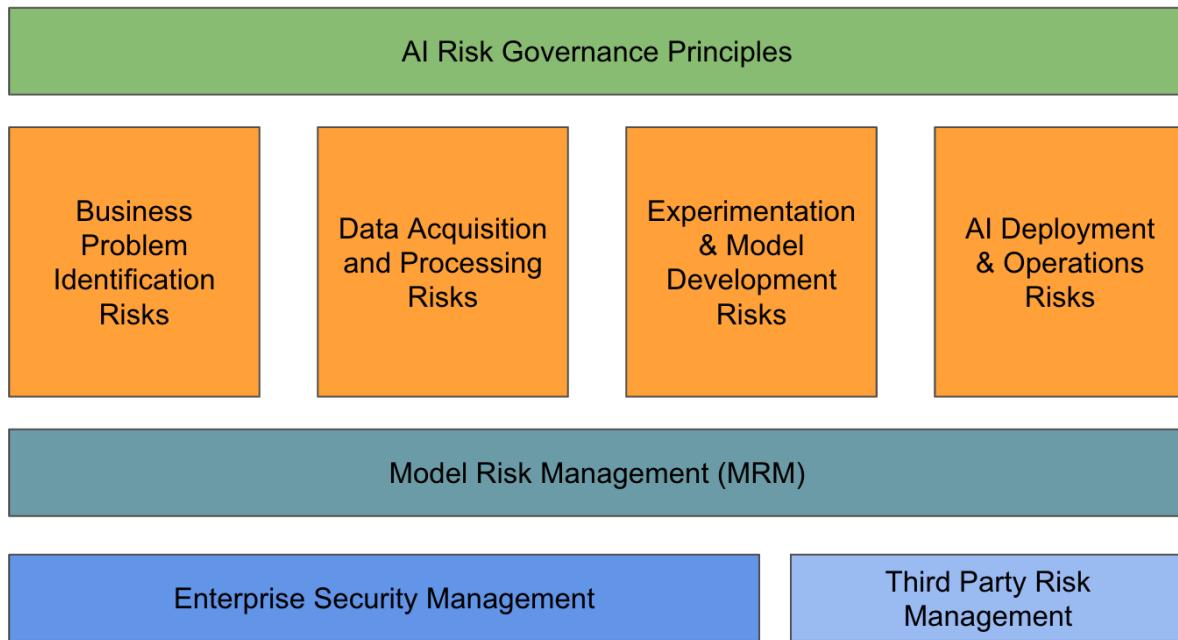


Figure 12.1: AI Risk Management Components

In the following sections, we will delve into the details for some of core components. We will focus mainly on the understanding of AI risk governance and MRM, with the understanding that traditional enterprise security and third-party risk managements are also part of the overall AI risk management considerations.

Governance oversight Principles

Implementing AI risk management starts with establishing key governance principals. The principals clarify the ultimate goals for what need to be accomplished with the risk management programs. Depending on the business and the regulatory environment an organization is in, an organization can make decision on what should be included in its risk management framework. The following are some of key areas for considerations:

- **Transparency:** AI systems should be designed in a way that allows stakeholders to understand how decisions are made and why. This may include ability to explain the ML model prediction, as well as

transparency on what and how the data and algorithms are used and implemented.

- **Accountability:** Organizations should be accountable for the decisions made by the AI systems, including any negative consequences that may arise from the use of AI systems. This accountability will ensure the owning organizations are incentivized to institute relevant policies and processes to govern the ML lifecycle.
- **Data governance:** Organizations should ensure that the data used to train AI systems is accurate, representative, ethical, and unbiased. Without proper data governance, it would highly challenging to trust the AI systems that are built with the ungoverned data.
- **Human oversight:** While AI systems are meant to make decision automatically in most cases without human intervention, organization should have the ability to implement human oversight where it requires, meaning that human should be involved in decision-making where it makes sense and human has the ability to override the decision when necessary.
- **Privacy and security:** Appropriate policies and process should be established to ensure AI systems are designed to protect privacy and security of securities according to laws and regulation. Privacy and security breach can have significant financial and non-finance implications to an organization.
- **Fairness:** AI systems should not discriminate against certain individuals or groups based on attributes such as race and gender.
- **Validity and reliability:** AI systems should be designed to produce reliable and valid results. Appropriate model validation and testing framework and processes need to be implemented to ensure highly reliable and predictable behaviors are displayed by AI systems in production. Mechanisms should be established to monitor systems behaviors with processed for mitigation and rollback when abnormal behaviors are observed.

With governance oversight, organizations should also consider requirements for regulatory compliance, policies and guidelines around roles and responsibility, and standards and process around AI systems and

model inventory and risk classification, as well as how to deal with lifecycle and change management.

AI risk management framework

With AI governance oversight principals defined, organizations can move forward to establish formalized AI risk management framework and established detailed mechanisms for risk identification, risk assessment, risk mitigation across the end-to-end ML lifecycle. One of common frameworks adopted by many organizations is the 3-lines-of-defense Model Risk Management (MRM) that are commonly used in the financial services industry. The MRM should be combined with enterprise technology risk and cyber security management, and third-party risks to ensure comprehensive coverage of AI risks. The MRM framework focuses on establishing policies, roles and responsibilities, and processes designed to identify, assess, mitigate, and audit potential model risks associated with business problem identification, data management, model development, model deployment, model uses:

- The first line of defense is owned by the business operations. This line of defense focuses on the development and use of ML models. The business operations are responsible for creating and retaining all data and model assumptions, model behavior, model performance metrics in structured documents, based on model classification and risk exposure. Models are tested and registered, and the associated artifacts are persisted, and results can be reproduced. Once models are deployed, system issues, model outputs, model bias, and data and model drifts are monitored and addressed according to the established procedures and guidelines.
- The second line of defense is owned by risk management function, and it focuses on model validation. The risk management function is responsible for independently reviewing and validating the documents generated by the first line. This line of defense introduces standards on controls and documentation, makes sure that documents are self-contained, results are reproducible, and the limitations of models are well-understood by stakeholders.

- The internal audit owns the third line of defense. The third line of defense mainly focuses on control and processes and less on model artifacts and theories. Specifically, this line of defense is responsible for auditing the first and second lines of defense to ensure all established processes and guidelines are effectively followed and implemented. This line of defense provides independent validation of internal controls, reviews the documentation, timeliness, frequency, and completeness of the model risk management activities.

The MRM along mainly addresses the risks associated with model development and development lifecycle. A comprehensive AI risk management also need to cover other risks such as system scalability and reliability, unauthorized access of systems, denial of access, and third-party failure risks.

Applying risk management across AI lifecycle

AI risks can exist in any stage of the AI lifecycle, spanning from business problems identification to AI systems uses. In the following sections, we will explore the various risks that can arise at each stage of the AI lifecycle and suggest effective strategies and considerations to mitigate them.

Business problem identification and definition

In this initial stage of the AI lifecycle, organizations develop a comprehensive comprehension of the business problems that AI can address. They also outline the overall solution approach and data prerequisites. It is critical during this phase to verify that the AI solution aligns with governance principles, standards, and requirements while achieving specific business objectives. Some of the potential hazards during this phase include:

- **Regulatory compliance risk:** This risk arises when there is no consideration for potential regulatory compliance requirements. It is important to understand if there are any applicable regulatory requirements around the AI projects and make sure appropriate

measures have been taken to address them problem identification phase. Without it, it can lead to project being out of compliance with known regulatory requirements and detailing the entire effort.

- **Missing ethics risk:** Ethics has direct impact to an organization's value and brand reputation. If not considered as part of business problem identification, the final system can cause misalignment with core value and brand.
- **Risk of unexpected consequences:** AI systems should be designed for intended use. If misused, it could result in unforeseen negative consequences.
- **Risk of missing risk rating and classification:** Proper risk rating and classification can help assess the potential impact when something goes wrong and drive the different levels of risk management. Without it, AI system and its data could be mishandled, resulting in unexpected consequences.
- **Missing security and privacy requirements risk:** Without security and privacy requirements, organizations run the risk of privacy violation and adversarial manipulation of the AI systems.

For each of the potential risks identified at this stage, it is important to conduct assessment to determine the severity and likelihood of the risk happening and the resulting impact. Determine if any mitigation measures should be considered to reduce the risk, based on the risk tolerance level of each individual organization. Only move forward with the project only if the key risks are understood, mitigated, or accepted.

Data acquisition and management

During this phase of the project lifecycle, it is crucial for organizations to identify appropriate data sources, establish a data acquisition strategy, and assess their technology capabilities for data processing and management. AI systems present a distinct set of data acquisition and processing risks, in addition to exposure to many common data-related risks. These risks span from selecting the appropriate dataset to managing data end-to-end. The following is a list of key data-related risks that require careful attention:

- **Data set selection or bias risk:** When data set is not correctly selected or sampled for the problem at hand, it may cause significant data relevancy or bias risk, resulting in biased model, or model not solving the problem. For example, during data collection phase for a credit scoring project, if certain groups of people are not adequately represented in the data set, the final model will show bias towards groups that are overly represented.
- **Data quality and missing data risk:** Data quality and missing data issues can pose significant challenges and risks for data scientists to develop high quality ML models.
- **Data labeling risk:** Data labeling is mainly a manual process, which can not only become the bottleneck of the model development, but also cause poor model accuracy from mislabeling mistakes.
- **Missing regulatory and compliance data check:** If an AI project is in scope for specific regulatory compliance requirements, then it is important to enforce regulatory and compliance data check. For example, there might be data sovereignty rules to comply with, and it would be important to check to make sure the data set used for model development does not violate the rules. Without proper checks, it can cause potential fines and lawsuit for any violation, resulting in financial and reputational damages.
- **Data privacy risk:** The capability of AI improves with more data, including personal information for both analysis and model training, potentially infringing upon personal privacy.
- **Adversarial attack risk:** ML model development is subject to new type of adversarial attacks where the training data can be manipulated to cause the resulting model to behave incorrectly in scenarios specifically targeted by the attackers. For example, if the training data labels are manipulated, then models will learn incorrectly and produce wrong results.

To mitigate data-related risks, the strategy and mechanisms need to be comprehensive and effective in addressing issues related to data quality, data bias, and human errors, as well as regulatory compliance requirements. These mechanisms may include implementing data validation methods, establishing consistent definitions and standards for data selection and

sampling, conducting regular reviews of data quality and completeness, and providing guidance on mitigating approaches such as extending data sources, using synthetic data, and employing appropriate sampling techniques.

- For data selection and sampling risks, standards and consistent definitions for different data types and sources should be established to ensure consistency in data selection from different sources. The standards should cover considerations for areas such as relevancy, data gaps, bias, representation, and it should also provide guidance on mitigation approaches such as extending data sources, using synthetic data, sampling techniques, and data validation methods.
- For data quality risks, verify minimum data quality is met to support high quality data processing and model training, including establishing rules and sample data review for accuracy, completeness, consistency, and validity, and appropriate representation of target population.
- To mitigate data labeling risk, establish controls for data labeling process to ensure consistency and avoid subjective bias, and sample test the validity of data set labels.
- For regulatory compliance and privacy checks, include robust set of regulatory compliance check in the MRM process. Establish enhanced controls around data access, ownership, collection, storage, transmission, and data assess to satisfy regulations. Include robust set of regulatory compliance checks for data privacy protection in the MRM process. Link the controls to enterprise access and authentication platform for central governance. Enforce data encryption and data masking where it is needed.

Ultimately, the mitigation strategy and mechanisms for data-related risks should be tailored to the specific needs of the organization and continuously reviewed and updated to keep up with the evolving landscape of AI/ML technologies and their associated risks.

Experimentation and model development

During this phase of the project lifecycle, data scientists utilize various algorithms and data sets to experiment and develop models to address business problems. The risks associated with this phase of the project lifecycle are mostly specific to AI/ML. They encompass a wide range of topics, such as algorithm selection and associated assumptions, limitations, model validation and robustness, model transparency and explainability, model fairness, compliance, and intended model use.

- **Model assumption and limitations risk:**

Incorrect/incomplete/inconsistent model assumptions and unrecognized limitations pose a risk for the model to not fit the situation at hand. For example, linear regression algorithm assumes a linear relationship between the predictor variables and response variables. If there is no such relationship among the variables of the underlying data, then model can result incorrect or biased prediction. Certain algorithms may have limitation on the size of the data sample and may not work well when the size is too small.

- **Model selection risk:** Model selection is the process of deciding which models are best suited to solve the problem at hand. There are a number of risks associated with model selection such as selecting model based on training performance leading to overfitting, selecting model without interpretability while there is a requirement for explainability.

- **Lack of sensitivity and scenarios analysis risk:** Missing/inadequate sensitivity/scenario analysis can lead to a lack of model robustness. For example, if a sensitivity of a credit risk model is not well understood, then it may fail to detect small changes in the input data, leading to wrong prediction. A financial forecast model may only consider a limited range of economic scenarios, and fail to function correctly when expected events occur.

- **Model transparency risk:** The lack of transparency in a model constrains its explainability, which in turn, hinders the ability to verify the decisions made by the model and determine if they align with business objectives. For example, if a lender receives a complain about incorrect lending decision made by an AI system, the lender will need

to explain what factors drove the decision. Failure to do so may potentially leave the lender legally vulnerable.

- **Model fairness risk:** Not only can data introduce bias, but the model itself can also introduce bias, which can raise concerns about the fairness of the AI system. For example, Naïve Bayes algorithms can introduce bias if it assumes all underlying features are independent, leading to incorrect predictions when the features are correlated.
- **Model evaluation risk:** Risk can arise when there is no thorough and independent validation of models, or incorrect model validation methods or metrics are used. For example, a model developer may not thoroughly test their own models, and it could lead to unexpected behavior when there is no independent validation of the models.
- **Model use and impact risk:** Model use and impact risk refers to the potential negative consequences that can arise from the use of machine learning and other AI models. For example, models may not perform as expected when deployed in the real world. Models are often trained on historical data, but the future may be different from the past. If the underlying assumptions of the model no longer hold, it may make inaccurate predictions or decisions.
- **Missing lineage risk:** A lack of understanding about the lineage from the data source to the model artifacts, including all the different transformations and experimentations that occurred during the modeling process, can make it difficult to comprehend the behavior of a model and identify the root cause of any issues that arise.

In order to mitigate these risks, it is essential for an organization to establish extensive MRM standards that encompass model evaluation, validation, selection, and fairness. Additionally, the organization should develop its capabilities and best practices to recognize assumptions and limitations, address known gaps, and ensure model transparency and lineage:

- For model assumption and limitation risk, clearly define and validate the assumptions underlying the algorithms used. Test difficult to validate assumptions with different techniques for completeness, and properly calculate model uncertainty to determine the confidence levels in model outputs.

- To mitigate the potential risks associated with model selection, data scientists need to develop a strong set of candidate models for the problem at hand. The models should be reviewed by a diverse team of technical and business resources, as well as representative from the target audience to help ensure the selected model will correctly address the problems. The modeling approach should be assessed and adjusted as needed to ensure it is fit-for-purpose, explainable, reproducible, and robust. Finally, the decision and supporting evidence, including the techniques used for model selection, should be documented.
- To address the shortcomings of inadequate sensitivity and scenario testing, it is necessary to establish and incorporate standards for model sensitivity and scenario testing within the framework of model risk management (MRM). These procedures should be an integral part of the model development process and should be performed to gain insight into the boundary conditions that impact the robustness of the model, minimize errors, and enhance comprehension of the interplay between model inputs and outputs.
- In order to address any deficiencies in managing model transparency risks, it is essential to establish and incorporate standards for model transparency. This should involve bolstering communication and feedback mechanisms within the development team to promote transparency throughout the model development process. Additionally, documentation of the process, including the techniques used for model validation, should be meticulously recorded to serve as corroborating evidence for model transparency.
- To address issues related to model fairness and improve model risk management (MRM), it is important to define and include model fairness standards. Fairness checks should be embedded within the model lifecycle, and business and technical stakeholders should be involved in the mitigation of discovered issues. Governance techniques and tools should be enhanced to promote fairness, recognizing that fixing discrimination in algorithmic systems is an ongoing process that needs continual improvement. There are different ways to adjudicate injustice in algorithmic systems, including narrative thinking and comparison across cases, and there is controversy regarding whether the systems should satisfy classification parity or calibration.

Impossibility results suggest that these measures of fairness are often incompatible, highlighting the existing gap that cannot currently be addressed.

- The model risk management (MRM) should include model performance evaluation standards as part of validation. These standards should be embedded in the model lifecycle and involve both business and technical stakeholders in the discovery and mitigation of issues. Standardized validation tools and techniques should be employed to promote consistent procedures and uniform execution of validation assessment. Multiple *ex ante* performance measures aligned with the transformation logic and the business purpose should be defined to evaluate the accuracy of the model's representation. An end-to-end evaluation of the model's performance against agreed-upon standards should be conducted to ensure the required level of ethics, performance, transparency, etc. The metrics should be monitored for changes in feature importance while retraining
- The model risk management (MRM) should include evaluation of model use and impact risks. This can be achieved by verifying the understanding of decisions made during model design/deployment/validation and implications of the algorithm's criteria. Additionally, an impact assessment should be conducted to assess risks involved in model development and use against the pre-set threshold. The model outcomes should be verified to achieve the desired level of precision, consistency, relevance, and alignment with trustworthy AI criteria.
- Establish mechanism and technology capability to track the lineage of model from data source to training/testing dataset, to experimentation run, to model training, and model deployment. This may include establishing a comprehensive metadata management mechanism and technical capability to capture all relevant information in the end-to-end pipeline, implementing version controls for all relevant artifacts such as code, data, configuration files, and model artifacts. Model registry is another important component for tracking critical information about models, such as version and performance metrics, and other meta data. Lastly, having an auditing and logging

mechanisms to understand what changes have been made, who made them, and for what purpose.

Overall, a comprehensive MRM framework that includes technical standards, ongoing monitoring and updates, and a culture of ethical decision-making is crucial for organizations to effectively manage the risks associated with model development so they can be used in trustworthy manner.

AI system deployment and operations

During this stage of the lifecycle, organizations design and build technology environments for AI system/model deployment in production to handle real-world business workflows in the broader application ecosystem, and establish operational process and standards to monitor the environments and remediate production issues ranging from basic system failure to model performance degradation. The following are some of the key risks that may arise at this stage:

- **Human supervision risk:** It is essential to subject AI models and systems to human review before their deployment into production, where feasible, to ensure their continued suitability for their intended purpose. Failure to do so may lead to the deployment of inadequately prepared AI systems, resulting in problems during production or unforeseen outcomes.
- **Technology integration risk:** AI systems normally do not operate alone. They are usually integrated into the wider technology ecosystem to support one and more business functions. Issues could arise when integrating with these upstream and downstream systems for data transfer, model integration, or API integration. For example, wrong model version could cause compatibility issues across different systems.
- **Technology scalability risk:** Often times, there might be unexpected spikes in data volumes, business users, and customers, after the AI systems/models are deployed. Failure to handle these scalability scenarios can negatively impact the business and user experiences.

- **Model performance and behavior change risk:** AI systems are developed using historical data. Unexpected changes in real-world environments such as data drift and outlier conditions can cause the model to behave differently from the original assumption.
- **Fallback procedure risk:** When production issues are detected, it is important to have well-established fallback procedures to mitigate the issues. Failure in having robust fallback protocols can put system operation continuity at risks.
- **Adversarial Attacks:** AI solutions offer new opportunities for adversarial attacks, such as feeding AI systems bad data leading to incorrect predictions and faulty decisions downstream.

In order to effectively manage risks during deployment and operations, it is essential to have robust mitigation mechanisms and technological capabilities in place. The primary focus of mitigating these risks is on carrying out rigorous testing and operational checks prior to deployment, setting up integration standards, monitoring model performance, following established issue resolution processes, and integrating adversarial monitoring and remediation throughout the AI lifecycle. Next, let us delve deeper into some of the specific recommendations:

- One of the key components for effective AI risk management is a model management system with model registry. A model management system needs to provide details about the models, performance metrics, uses, and other related metadata. Organizations need to incorporate model risk management standards and embed model operationalization checks for models in the registry. Stress testing and scaling simulation should be performed to understand its behavior under heavy load. In addition, processes and tools should be put place for model owners to monitor, manage, govern, and analyze the model's results.
- To mitigate potential AI integration risks, organizations should incorporate integration standards and requirements in MRM to ensure the AI model's operational sustainability. It is important to ensure interoperability among different platforms, frameworks, and approaches and conducting robust integration testing to meet

compliance and functional standards. Additionally, organizations need to verify proper configuration and integration of the model into the production environment to prevent errors when migrating the model from one programming environment to another or upgrading from one model version to another.

- Establish model deployment review and approval standards in MRM. The model deployment review and approval standards should include a detailed review of the AI model's design and underlying algorithms, as well as its testing results and performance metrics. It should also take into account the potential risks associated with the model's deployment and outline clear steps for mitigating those risks.
- The main approach to address performance and behavior changes for operational continuity is to include model monitoring, performance issue tracking and resolution standards in MRM.
- Three types of metrics (statistical, technical, and from a business perspective) should be continuously monitored to track the model's decay over time and its performance. Real-time circuit breakers should be used to establish performance boundaries and ensure the model is operating as intended. Pre-specifying benchmark or legacy models and using them as fallback options can be helpful when the model's performance boundaries are breached. Track the reasoning behind the model's decisions and monitoring model explainability, including testing input data for outliers and using benchmark models to trigger further investigation if necessary
- Issue logs should be maintained and the severity of the risk should be assessed. Remediation actions should be identified and activated following prescribed MRM procedures to ensure that the model remains fit for purpose given the availability of new data or potential changes in the business, economic, or regulatory environment. A feedback mechanism on issue resolution should also be established.
- Additionally, existing data remediation processes and associated testing infrastructure should be enhanced to address the high volume of structured and unstructured data that AI/ML models typically ingest.
- Organizations should include adversarial attack monitoring standards in model risk management to prevent malicious input from causing a

malfuction in the model. Adversarial AI attacks are enabled by inherent limitations in algorithms themselves and their reliance on data. Effective testing and auditing techniques and certification programs should be developed to address AI models' vulnerabilities. Research on adversarial attacks and model data leakage should be leveraged to test AI models for vulnerabilities and assess their overall robustness and resilience to different forms of attacks. Finally, cyber threat hunting should be instituted to proactively and iteratively search through networks to detect and isolate advanced threats that evade existing security solutions.

What we have covered so far does not include all the risks we might encounter throughout the AI lifecycle and there are new emergent risks coming up regularly. It is also not possible and potentially counterproductive to mitigate all the risks in practices. Organizations should determine the tolerance levels for the different risks, which are going to highly contextual and application and use-case specific. Other factors such policies established by system owners and regulators, organizational priority, and resource considerations can also influence the risk tolerance. It is also worth noting that risk tolerance is likely to change over time as the influencing factors evolve. It is also important to prioritize the risk identified in the AI lifecycle. Organizations should recognize that not all risks are the same, and scarce resources should be allocated appropriately to address the different risks. The assessed risk levels and potential impact of an AI system should be used to prioritize the resource allocation to the mitigations of these risks. Finally, AI risks are not isolated risks and should be considered and incorporated into the broader enterprise risk management strategies and processes. The roles and responsibilities of different players in managing risks will span different functional domains such as engineering, data sciences, cybersecurity, audit, and compliance.

Designing ML platform for governance and risk considerations

ML technology systems play a crucial role in the AI risk management process and activities. To begin with, these systems must be developed and constructed to comply with both internal and external policies and guidelines. Additionally, technology can aid in streamlining and automating ML governance procedures. The following figure illustrates the different ML governance touchpoints in an enterprise ML platform. It is important to know that ML technology alone can only help address a subset of AI risks, other enterprise security technology needs to be incorporated to form a more comprehensive governance and defense mechanism.

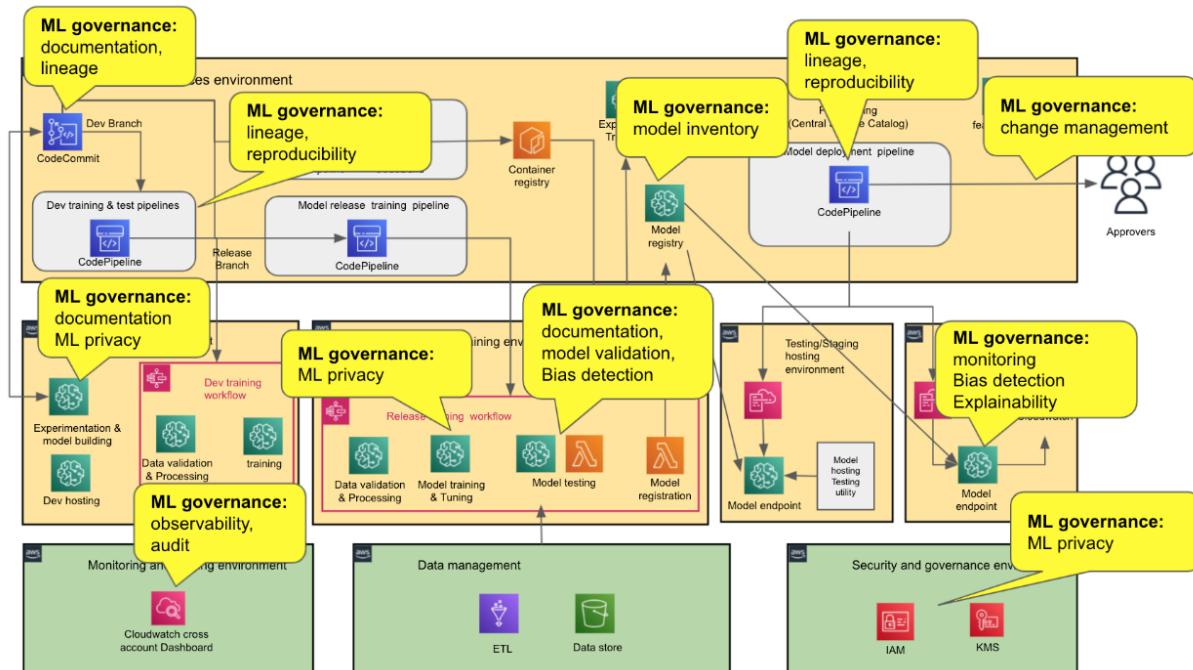


Figure 12.2: ML platform and ML governance

When an ML platform is built with AI risk management and governance in mind, it can gather and furnish the information to support the MRM programs, and while optimizing the risk management workflows. Online data store, workflow applications, document sharing systems, and model inventory databases are among the technology solutions employed for AI governance. In the following section, let's delve deeper into some of the core ML governance components and where a ML platform or technology can fit in.

Data and model documentation

Documentation is one of essential elements of AI governance is documentation. All models used for decision making should be properly documented. The scope of the documentation may include the following:

- Data overview, data quality report on the valuation and assessment of the input data.
- Model development document including methodology and assumption, model usage instruction, performance and validation results, and other qualitative and quantitative analysis.
- Model validation strategy and report by second line and third line of defense.
- Model performance monitoring results. Data drift reports.
- Model implementation, and user acceptance testing reports.

The role of the ML platform in ML governance documentation is usually to provide data points that feed into the formal risk management documentations or generate some of the ready-to-use reports. Specifically, an ML platform that can track, store, and report the following data points:

- Data quality metrics such as data description, statistics, bias, and errors
- Model metrics and validation results in development and testing
- Model bias and explainability reports
- Model performance monitoring results in production
- Model description and intended use
- Risk rating and classification details

Different ML platforms have varying capabilities supporting the AI governance documentation requirements. Here, we will discuss the various capabilities of SageMaker supporting this requirement. SageMaker can produce data and documentations to be incorporated into model risk documentations. This includes tracking and producing information that are relevant for AI governance documentation, such as:

- **Model metrics:** The SageMaker training service tracks the model metrics such as training error and validation error.
- **Data and model bias reports:** SageMaker Clarify is the bias detection component in SageMaker. If you enable SageMaker Clarify, you can get data and model bias reports for the training data and trained model. The data and model bias reports have details such as imbalances in training data and prediction behavior across different age groups and genders.
- **Model explainability reports:** SageMaker Clarify also provides model explainability feature. It uses SHAP to explain the contribution of each input to the final decision. You can get more detail about SHAP at <https://shap.readthedocs.io/en/latest/index.xhtml>.
- **Model card:** SageMaker Model Card can be used to document critical information about ML models, such as intended use of the model, risk rating, detailed description of model training and performance.

Both open-source and managed model registry platforms are available for managing model registries. For example, the MLFlow model registry is an open-source option, while Amazon SageMaker offers a managed model registry service. The SageMaker model registry has several key capabilities that can aid in ML governance activities and processes:

- **Model inventory:** All versions of the different models belong to a respective model group in the SageMaker registry. You can view all the model groups and different versions of a model in the registry. Meta data such as model metrics, training job details, hyper-parameters used for training, and training data sources are important data points for the model reviews and model audit processes.
- Depending on specific business requirements, you can set up a central model registry for a single enterprise view, or distributed model registries if that can meet the governance and audit requirements.
- **Model approval and lifecycle tracking:** You can track the approval of models and model stages directly inside the SageMaker model registry. This detail helps the business operations and audit to ensure the proper processes are followed.

Monitoring models after deployment is crucial to identify any potential failures and take prompt remedial action to mitigate risks. To ensure smooth functioning, models must be monitored for system availability and errors, as well as data and model drift and prediction failure. Amazon SageMaker offers a model monitoring feature that can detect both data drift and model drift. The SageMaker Model Monitor provides the following capabilities:

- Data drift : With SageMaker Model Monitor, you can monitor data quality issues and data distribution skews (aka data drift) in production. To use this feature, you create a baseline using a baseline dataset, such as a model training dataset, to get data statistics, data types, and suggest constraints for monitoring. SageMaker Model Monitor can capture live inference traffic, calculate data statistics, examine data types, and verify them against the constraints and trigger alerts. For example, if a feature's mean and standard deviation changes significantly from the baseline, an alert can be triggered.
- Model performance drift : You can use the Model Monitor to detect model performance changes in production. To use this feature, you create a model performance baseline job using a baseline dataset that contains both the inputs and labels. The baseline job will suggest constraints, which are the metrics thresholds that the Model Monitor will monitor against the metrics to be calculated with ground truth data collected in production. Metrics can be optionally sent to CloudWatch for visualization.
- Feature attribution drift : When enabled with SageMaker Clarify, SageMaker model monitor can be report feature attribution drift. Feature attributions are indicators of feature importance to the prediction output. Similar to data and model drift, you create a SHAP baseline job using baseline data to generate constraint suggestion. The separate monitoring job is then scheduled to monitor predictions in production against the baseline.

Lineage and reproducibility

MRM requires establishing lineage across data and models to ensure reproducibility. Lineage information includes training data sources,

algorithm selection, hyperparameter configurations, and the model training script. SageMaker offers several features that aid in establishing lineage:

- SageMaker training job keeps lineage data such as training data source, training job container (contains algorithm and training script), hyperparameter configuration, and model artifact location. Historical training job data are immutable in the SageMaker environment for record retention purpose.
- SageMaker Experiment and ML Lineage can contain additional component details such as data processing for a more complete lineage tracking.
- SageMaker Hosting has information on the location of the original model artifact and the inference container to trace the lineage from model to endpoint.

These data points are available through calling the SageMaker API, an external application can call the SageMaker API directly to extract these data for review purpose. Alternatively, a data extraction job can be developed to extract these datapoints and load them into purpose-built risk management store for analysis. The significance of ML privacy is growing rapidly in the implementation of ML systems. To comply with data privacy regulations or internal data privacy controls, ML systems must have fundamental infrastructure security features, such as data encryption, network isolation, compute isolation, and private connectivity. By utilizing a SageMaker-based ML platform, you can enable the following essential security controls:

- **Private networking:** As SageMaker is a fully managed service, it runs in the AWS owned account. By default, resources in your own AWS account communicates with SageMaker APIs via public internet. To enable private connectivity to SageMaker components from your own AWS environment, you can attach them to a subnet in your own virtual private cloud (VPC).
- **Storage encryption:** Data-at-rest encryption can be enabled by providing an encryption key when you create a SageMaker notebook, a training job, a processing job, or a hosting endpoint.

- **Disabling internet access:** By default, the SageMaker notebook, training job, and hosting service have access to internet. The internet access can be disabled via configuration.

Observability and auditing

Auditing primarily concentrates on process verification and artifact collection to support audit activities. The platform on which the process takes place usually functions as an information source for collecting artifacts. For instance, suppose there is a model risk management policy that necessitates approval before deploying a model into production. In that case, the audit will require access to the system of record to ensure that the required data is collected and retained. SageMaker and other related services can be a data source in support of the overall audit process. Specifically, it provides the following information that can be relevant for auditing purpose:

- Activity and access audit trail: SageMaker sends all audit trail data to CloudWatch logs, which can be retained and analyzed for audit purpose.
- Model approval tracking: Model deployment approvals are tracked in SageMaker model registry. This can be provided to auditor as evidence that required approval processes are followed.
- Lineage tracking: SageMaker Experiment and ML Lineage tracking components can track and retain model lineages such as data processing, model training, and model deployment. Lineage tracking information helps the auditor to verify that the model can be reproduced using its original data and configuration dependencies.
- **Configuration changes:** System configuration data are captured in AWS CloudTrail as change events. For example, when a SageMaker endpoint is deleted, there will be an entry in the CloudTrail indicating this change.

Scalability and performance

To mitigate the potential scalability risk, AI systems should be designed to handle dynamic and unexpected load. For ML platform, this usually means that the training infrastructure is designed and implemented to support single large training job as well as many training jobs running in parallel. Similarly the model hosting infrastructure should be capable to handling large number of models running in parallel as well running large number of model instances across many nodes. If your platform of choice is SageMaker, then the following capabilities can help mitigate training and hosting scaling challenges:

- **Training infrastructure scaling:** SageMaker has support for large-scale distributed training, with the ability to utilize hundreds of nodes and thousands of CPUs/GPUs. Additionally, SageMaker provides a purpose-built library for running both data-parallel and model-parallel training jobs. For storage scaling, high-performance storage solutions like Elastic File System (EFS) and FSx can be mounted onto SageMaker training nodes to accommodate training jobs that require a large-scale dataset exceeding 1TB. AWS accounts can run multiple training jobs in parallel, and the soft limit can be increased upon request.
- **Hosting infrastructure scaling:** SageMaker offers several options to scale model hosting needs. The Multi-Model Endpoint (MME) capability allows you to host multiple models behind a single endpoint, while reducing the costs. SageMaker's automatic scaling feature enables you to define a scaling policy based on metrics such as the number of invocations per host, which can increase the number of instances running the same model automatically when the traffic increases. Additionally, the serverless inference option allows you to run a single model concurrently up to the maximum number supported by SageMaker.

Data quality

Data quality checks should take places in multiple phases of the lifecycle, including data acquisition and processing, exploratory data analysis and data wrangling, feature engineering, and model inference. The check should

cover many aspects of data quality such as missing data, data accuracy, inconsistency across different sources, incorrect format, incompleteness, imbalanced data, duplication. From an AWS technology perspective, there are several purpose-built tools and features that can help with data quality management:

- AWS Glue DataBrew offers a range of data quality features that can help ensure the accuracy and reliability of data used for analysis or model training. DataBrew is mainly used by data engineers who are responsible for sourcing and cleaning the data during the data acquisition and processing phase for downstream users such as data scientists. Some of these features include:
 - Data profiling: DataBrew can automatically profile datasets to identify data quality issues, such as missing or inconsistent values, outliers, or duplicates.
 - Data cleaning: DataBrew provides a range of data cleaning transformations that can be applied to address common data quality issues, such as filling in missing values, removing duplicates, or standardizing data formats.
 - Data validation: DataBrew can perform data validation checks to ensure that data values fall within expected ranges or conform to predefined standards or formats.
 - Data lineage: DataBrew tracks the lineage of data transformations to help ensure that data is being processed correctly and that any changes can be traced back to their source.
 - Data versioning: DataBrew supports versioning of datasets, making it easy to track changes and roll back to previous versions if necessary.
- SageMaker Data Wrangler offers some of the similar data quality capability, targeting mainly data scientists who are doing data exploratory analysis and feature engineering in the SageMaker environment. The Data Quality and Insights report in Data Wrangler can automatically verifies data quality (such as missing values, duplicate rows, and data types) and helps detect anomalies (such as outliers, class imbalance, and data leakage) in your data.

Summary

This chapter delves into several areas related to AI risk management and the technology platforms that support it. By now, you should have a solid understanding of the key AI-related risk scenarios, why AI risk management is critical, and how to detect and address potential risks throughout the AI lifecycle. Additionally, you should be aware of the significance of ML platforms in supporting AI risk management. It is worth noting that AI risk is vast and complex domain with many unresolved risk challenges with new emergent risks are arising rapidly. Moreover, the fast advancement in AI technology and adoption are also creating new risk exposure that risk management professionals must constantly address. In the next chapter, we will dive deeper into several specific AI risk topics and mitigation techniques, including bias, model explainability, model robustness, and adversarial attacks.

13 Bias, Explainability, Privacy, and Adversarial attacks

Join our book community on Discord

<https://packt.link/EarlyAccessCommunity>



In the previous chapter, we explored the topic of AI risk management framework and discussed its importance in mitigating the risks associated with AI systems. We covered the core concepts of what it is, the importance of identifying and assessing risks, and recommendation for managing those risks. In this chapter, we will take a more in-depth look at specific risk topics and techniques for mitigations. We will explore the essential areas of Bias, Explainability, Privacy, and Adversarial attacks, and how they relate to AI systems. We will examine how bias can lead to unfair and discriminatory outcomes, and how explainability can enhance the transparency and accountability of AI systems. We will also discuss the criticality of privacy in AI systems, as well as the potential risks of adversarial attacks and how to mitigate them.

Understanding bias

Detecting and mitigating bias is a crucial focus area for AI risk management. The presence of bias in ML models can expose an organization to potential legal risks but also lead to negative publicity, causing reputational damage and public relation issues. Specific laws and regulations, such as *Equal Credit Opportunity Act*, also prohibit discrimination in business transactions, like credit transactions, based on race, color, religion, sex, nationality origin, marital status, and age. Some other examples of laws against discrimination include the *Civil Rights Act of 1964* and *Age Discrimination in Employment Act of 1967*. ML bias can result from the underlying prejudice in data. Since ML models are trained using data, if the data has a bias, then the trained model will also exhibit bias behaviors. For example, if you build an ML model to predict loan default rate as part of the loan application review process, and you use race as one of the features in the training data, then the ML algorithm can potentially pick up race-related patterns and favor certain ethnic groups over others. Bias can be introduced in different stages of ML lifecycle. For example, there could be data selection bias as certain groups might have stronger representation in the data collection stage. There could be labeling bias where a human makes an intentional or unintentional mistake in assigning labels to a dataset. Data sources with disinformation can also be a source of bias that results in bias AI solutions. The ability to explain the decisions made by models helps an organization to satisfy compliance and audit requirements from the governance bodies. Furthermore, model explainability helps an organization understand the cause-and-effect relationships between the inputs and the ML prediction to make better business decisions. For example, if you can understand the reasons (such as rewards program) behind strong customer interest in a financial product, you can adjust your business strategy, such as doubling down on rewards programs, to increase revenues. Being able to explain model decisions also helps establish trust with domain experts in the ML models. If domain experts agree with how the predictions are made by the models, they would more likely adopt the models for decision makings. There are various techniques for bias detection and model explainability, and we will take a closer look at some of the techniques next.

Bias detection and mitigation

To detect and mitigate bias, some guiding principles need to be established on what is considered as fair. For example, a bank's loan approval process should treat similar people similarly and the process may be considered fair when applicants with similar qualifications are assessed similarly. The bank also needs to ensure different

demographics subgroups are treated equally for loan approval and measure metrics such as the rate for loan rejection to be approximately similar across different demographics subgroups. Depending on the definition of fairness, bias can be measured using different metrics. Some of the metrics might even contradict with each other. Therefore, you need to choose the metrics that best support the definition of fairness with social and legal consideration and inputs from different demographics groups. In this section, we list some of the bias metrics for consideration:

- **Class imbalance:** This metrics measures the imbalanced representations of different demographics groups, especially disadvantage group, in a dataset.
- **Difference in positive proportion in observed labels:** This metric measures the differences in positive labels across different demographics groups.
- **Kullback and Leibler (KL) divergence:** This metrics compares the probability distribution in features and labels for the different groups, such as advantaged and disadvantaged group.
- **Conditional Demographic Disparity in Labels:** This metrics measure if a group has the bigger proportion in rejected outcome than the proportion in the accepted outcome in the same group.
- **Recall difference:** This metrics measure if ML model is finding more true positives for one group (advantaged group) than other groups (disadvantaged groups).

There are several ways for mitigating bias after they are detected. The following are some examples that can be applied:

- **Removal of features:** This approach helps mitigate bias by removing features that can contribute the bias such as gender and age.
- **Rebalance of training data:** This approach corrects bias in different numbers of representations for the different groups in the training data.
- **Adjust labels in the training data:** This approach brings the proportions of labels closely for the different subgroups

There are a number of open source libraries for fairness and bias management, such as:

- Fairness (<https://github.com/algofairness/fairness-comparison>)
- Aequitas (<https://github.com/dssg/aequitas>)
- Themis (<https://github.com/LASER-UMASS/Themis>)
- Responsibly (<https://github.com/ResponsiblyAI/responsibly>)
- IBM AI Fairness 360 (<https://aif360.mybluemix.net/>)
- There is also a component in SageMaker for bias detection, which we will cover in greater detail in a later section.

Understanding ML explainability

There are two main concepts when it comes to explaining the behaviors of an ML model:

- **Global explainability:** This is the overall behaviors of a model across all data points used for model training and/or prediction. This helps to understand collectively how different input features affect the outcome of model predictions. For example, after training a ML model for credit scoring, it is determined that income is the most important feature in predicting high credit score across data points for all loan applicants.
- **Local explainability:** This is the behavior of a model for a single data point (instance), and which features had the most influence on the prediction for a single data point. For example, when you try to explain which features influenced the decision the most for a single loan applicant, it might turn out that education was the most important feature, even though income was the most important feature at the global level.

Some ML algorithms such as linear regression and decision trees are considered explainable algorithms with built-in ability to explain the model. For example, the coefficients of linear regression models directly represent the relative importance of different input features, and the split points in a decision tree represent the rules used for

decision making. For black-box models such as neural network, it is very hard to explain how the decisions are made in part due to non-linearity and model complexity. One technique for solving this is to use white-box surrogate model to help explain the decisions of a black-box model. For example, you can train a linear regression model in parallel with a black-box neural network model using the same input data. While the linear regression model might not have the same performance as the black-box model, it can be used to explain at a high level how the decision was made. There are various open-source packages, such as **LIME (local interpretable model-agnostic explanations)**, and **SHAP (SHapley Additive exPlanations)**, for model explainability. Both LIME and SHAP adopt the surrogate model approach.

LIME

LIME supports local (instance) explainability as the name suggests. The main idea behind LIME is to perturb the original data points (tweak the data points) and feed them into the black-box model and see the corresponding outputs. The perturbed data points are small changes to the original data point and are weighted based on their proximities to the original data. It then fits a surrogate model, such as linear regression, using the perturbed data points and responses. And finally, the trained linear model is used to explain how the decision was made for the original data point. LIME can be installed as a regular python package and can be used to explain text classifiers, image classifiers, tabular classifiers, and regression models. The following are the explainers available in LIME:

- **Tabular data explainer:** `lime_tabular.LimeTabularExplainer()`
- **Image data explainer:** `lime_image.LimeImageExplainer()`
- **Text data explainer:** `lime_text.LimeTextExplainer()`

LIME has some short comings such as lack of stability and consistency since LIME uses random sampling to generate data points for approximation. Also the linear surrogate might be inaccurate for local data points that cannot be approximated by a linear model.

SHAP

SHAP is a more popular package, and it addresses some of the short comings of LIME. It computes the contribution of each feature to the prediction using the coalition game theory concept, where each feature value of each data instance is a player in the coalition. The basic idea behind the coalition game theory is to form different permutations of coalitions of players when playing a game, then observe the game results from the different permutations, and finally calculate the contribution of each player. For example, if there are 3 features (A, B, C) in the training dataset, then there will be 8 distinct coalitions (2^3). We train one model for each distinct coalition for a total of 8 models. We use all 8 models to generate predictions on the dataset and figure out the marginal contribution of each feature and assign a Shapley value to each feature to indicate the feature importance. For example, if the model that uses a coalition with only features A, B generates an output of 50, and the model that uses features A, B, C generates an output of 60, then feature C has a marginal contribution of 10. This is just a generalization of the concept; the actual calculation and assignments are more involved. SHAP can also be installed like a regular Python package. It can be used to explain tree ensemble models, natural language models (such as transformers), and deep learning models. It has the following main explainers:

- **TreeExplainer:** An implementation for computing SHAP values for trees and ensemble of trees algorithms.
- **DeepExplainer:** An implementation for computing SHAP values for deep learning models
- **GradientExplainer:** An implementation of expected gradients to approximate SHAP values for deep learning models.
- **LinearExplainer:** For explanation of linear model with independent features.
- **KernelExplainer:** A model agnostic method to estimate SHAP values for any model because it makes no assumptions about the model type.

SHAP is widely considered the state-of-art model explainability algorithm, and it has been implemented in commercial offerings such as SageMaker. It can be used for both computing global feature importance as well as

local explainability for a single instance. It does have some shortcomings as well, such as slow computation associated with KernelExplainer.

Understanding security and privacy preserving ML

ML privacy is becoming increasingly important in ML implementation. To ensure compliance with data privacy regulations or even internal data privacy controls, ML systems need to provide foundational infrastructure security features such as data encryption, network isolation, compute isolation, and private connectivity. With a SageMaker based ML platform, you can enable the following key security controls:

- **Private networking** – As SageMaker is a fully managed service, it runs in the AWS owned account. By default, resources in your own AWS account communicates with SageMaker APIs via public internet. To enable private connectivity to SageMaker components from your own AWS environment, you can attach them to a subnet in your own **virtual private cloud (VPC)**.
- **Storage encryption** – Data-at-rest encryption can be enabled by providing an encryption key when you create a SageMaker notebook, a training job, a processing job, or a hosting endpoint.
- **Disabling internet access** – By default, the SageMaker notebook, training job, and hosting service have access to internet. The internet access can be disabled via configuration.

In addition to infrastructure security, you also need to think about data privacy and model privacy to protect sensitive information from adversarial attacks, such as reverse engineering of sensitive data from anonymized data. There are three main techniques for data privacy protection for machine learning:

- **Differential privacy** – Differential privacy allows the sharing of datasets while withholding information about individuals within the dataset. This method works by adding random noises into the computation so that it is hard to reverse engineer the original data (if it is not impossible). For example, you can add noises to the training data or model training gradients to obfuscate the sensitive data.
- **Homomorphic encryption (HE)** - HE is a form of encryption that allows users to perform computation on encrypted data without first decrypting it. This leaves the computation output in an encrypted form that when decrypted is equivalent to the output as if the computation was performed on the unencrypted data. With this approach, the data can be encrypted before it is used for model training. The training algorithm will train the model with the encrypted data, and the output can be decrypted only by the data owner with the secret key.
- **Federated learning** – Federated learning allows model training to take place in edge devices while keeping data locally on the device, instead of sending the data to a central training cluster. This protects individual data as it is not shared in a central location, while the global model can still benefit from individual data.

Each of these topics warrants its own separate book. So, we will not dive into the details of all three. Instead, we will only do an introduction of differential privacy in this book to explain the main intuition and concept behind this method.

Differential privacy

To understand the problem that differential privacy solves, let's take a look at the real-world privacy breach that happened with Netflix. In 2006, Netflix provided 100 million movie ratings submitted by 480K users as the data for the Netflix price competition. Netflix anonymized user names with unique subscribers' ids in the dataset, thinking that this would protect subscribers' identities. Just 16 days later, two university researchers were able to identify some subscribers' true identities by matching their reviews with data from IMDB. This type of attack is called a **linkage attack**, and this exposes the fact that anonymization is not enough in protecting sensitive data. You can find more information about this at https://en.wikipedia.org/wiki/Netflix_Prize. Differential privacy solves this problem by adding noises to the dataset used in the computation on the dataset, so the original data cannot be easily reverse-engineered. In addition to protection against linkage attacks, differential privacy also helps quantify privacy loss as a result of someone running processing against the data. To help understand what this means, let's look at the following example: Suppose your organization is a regional bank, and your customer data repository contains sensitive data about your customers, including name, social security number, zip code, income, gender,

and education. To ensure data privacy, this data cannot be freely shared by all departments, such as the marketing department. However, the aggregate analysis of the customer data, such as the number of customers with income over a threshold, is allowed to be shared. To enable access to the aggregated data, a data query tool was built to return only the aggregate data (such as count, sum, average, min, max) to the marketing department. Separately, another database contains customer churn data with unique customer IDs, and a customer support database contains customer names and unique customer IDs. Both the churn database and customer support database are accessible to the marketing department. An ill-intentioned analyst wanted to find the names of customers whose incomes are above a certain threshold for some personal purpose. This analyst queried the database one day and found out that out of 4000 total customers, there were 30 customers with incomes over \$1 million in a particular zip code. A couple of days later, he queried the customer data again and found out there were only 29 customers with incomes over \$1 million, out of a total of 3999 customers. Since he had access to the churn database and customer support database, he was able to identify the name of the customer who churned and figured out this customer had an income of over \$1 million. To prevent this from happening, the query tool was changed to add a little noise (such as adding or removing records) to the result without losing meaningful information about the original data. For example, instead of returning the actual result of 30 customers out of 4000 customers in the first query, the result of 31 customers out of 4001 customers was returned. And the second query returns 28 out of 3997 instead of the actual 29 out of 3999 figures. This added noise does not significantly change the overall magnitude of the summary result, but it makes reverse-engineering of the original data much more difficult, as now you can pinpoint a specific record. This is the intuition behind how differential privacy works. The *Figure 11.2* shows the concept of differential privacy, where computation is performed on 2 databases, and noises are added to one of the databases. The goal is to ensure **Result 1** and **Result 2** are as close as possible as that's where it becomes harder and harder to tell the difference in distribution between **Result 1** and **Result 2** even though the 2 databases are slightly different. Here the Epsilon (ϵ) value is the privacy loss budget, which is the ceiling of how much probability an output distribution can change when adding/removing a record. The smaller the Epsilon value, the lower the privacy loss.

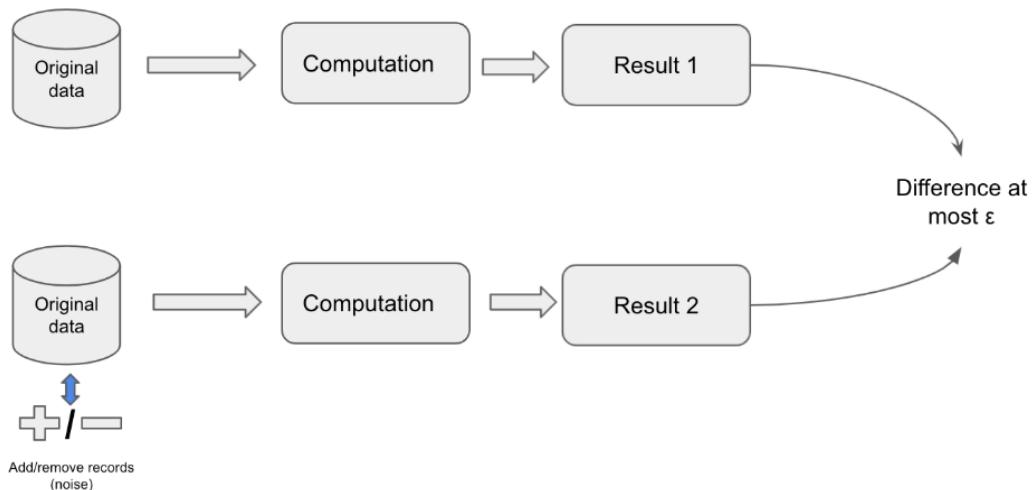


Figure 13.1: Differential privacy concept

ML models are susceptible to privacy attacks. For example, it is possible to extract information from trained models that directly map to the original training data, as deep learning models may have unintended memorization of training data. Also, overfitted models also likely to memorize training data. Differential privacy is one of the techniques that can help minimize the effect of unintended memorization. Since differential privacy can make the computational outputs of two input datasets (one with sensitive data, one with sensitive data removed) almost indistinguishable from a query perspective, the hacker cannot confidently infer if a piece of sensitive data is in the original dataset or not. There are different ways to apply differential privacy to ML model training such as adding noises to the underlying training data or adding noises to the model parameters. Also, it is important to know that

different privacy does not come for free. The higher the privacy protection (smaller epsilon), the lower the model accuracy. Differential privacy is implemented in TensorFlow Privacy. TensorFlow Privacy provides a differentially private optimizer for model training and requires minimum code changes. The following code sample shows the syntax of using the `DPKerasSGDOptimizer` object for differential privacy training. The main steps are as follows:

1. Import tensorflow privacy library package
2. import tensorflow_privacySelect your differentially private optimizer

```
optimizer = tensorflow_privacy.DPKerasSGDOptimizer(  
    l2_norm_clip=l2_norm_clip,  
    noise_multiplier=noise_multiplier,  
    num_microbatches=num_microbatches,  
    learning_rate=learning_rate)
```

1. Select your loss function

```
loss = tf.keras.losses.CategoricalCrossentropy(  
    from_logits=True, reduction=tf.losses.Reduction.NONE)
```

1. Compile your model

```
model.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])
```

PyTorch supports differential privacy with its `opacus` package. It is also fairly straight forward to use the `opacus` package to enable differential privacy training. The following code samples shows how to wrap an optimizer in the `PrivacyEngine` object, and just use the optimizer the same way in a Pytorch training loop.

```
from opacus import PrivacyEngine  
optimizer= torch.optim.SGD(model.parameters(), lr=learning_rate)  
privacy_engine = PrivacyEngine(  
    model,  
    sample_rate=sample_rate,  
    max_grad_norm=max_per_sample_grad_norm,  
    noise_multiplier = noise_multiplier  
)  
privacy_engine.attach(optimizer)
```

Understanding Adversarial Attacks

Adversarial attacks are a type of attacks on machine learning models that exploit their weaknesses and cause them to make incorrect predictions. Imagine you have a machine learning model that can accurately identify pictures of animals. An adversarial attack might manipulate the input image of an animal in such a way that the model misidentifies it as a different animal. These attacks work by making small, often imperceptible changes to the input data that the model is processing. These changes are designed to be undetectable by humans, but can cause the model to make large errors in its predictions. Adversarial attacks can be used to undermine the performance of machine learning models in a variety of settings, including image recognition, speech recognition, and natural language processing. There are two types of adversarial attacks objectives: targeted or untargeted. Targeted objective means to cause the ML systems to predict a specific class determined by the attacker, and untargeted objective simply cause the ML systems to misclassify. Adversarial attacks can take many different forms, including evasion attacks, data poisoning attacks, and model extraction attacks. The following Figure 13.x illustrates the different attacks that an adversary may carry out against an ML systems.

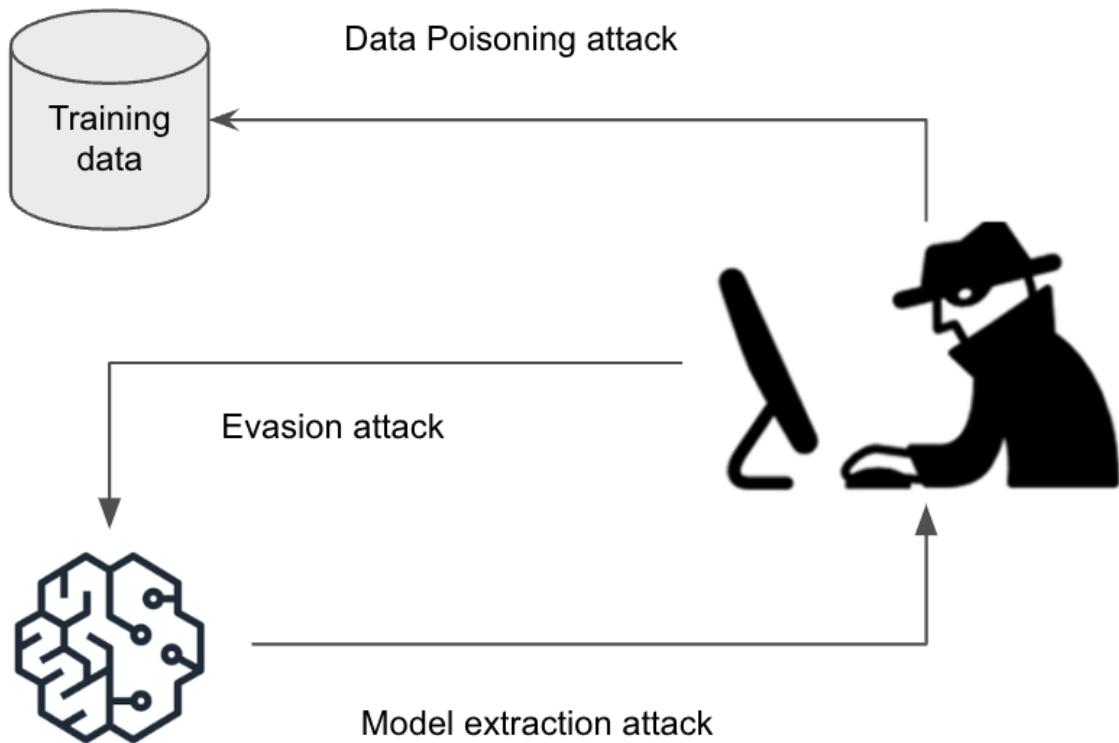


Figure 13.2: Types of adversarial attacks

Depending on attacker's knowledge of the ML systems and their ability to access the model and data, an attack can be a white box attack or a black box attack. The majority of the attacks are white box attacks, meaning this attack assumes you have the total knowledge of the ML models. This means if you want to cause adversarial attacks against a neural network models, you need to know all the weights values and network structure. Opposite of white-box attack is black box attack. With black-box attack, you probe the ML model for a number of trials using different inputs, and record the results from the ML model, and use that information to design an attack against the model.

Evasion attacks

ML evasion attacks are a type of attack where a malicious actor attempts to manipulate the input data to evade the detection or classification of an ML model. In an ML evasion attack, the attacker modifies the input data to generate an adversarial sample that appears legitimate to a human observer but can cause the ML model to produce incorrect output or misclassify the input data. The goal of an ML evasion attack can vary, from causing a malfunction in the system to making it vulnerable to more severe attacks. The following figure shows that by introducing small human unnoticeable noises into the image, the ML model can generate incorrect prediction.

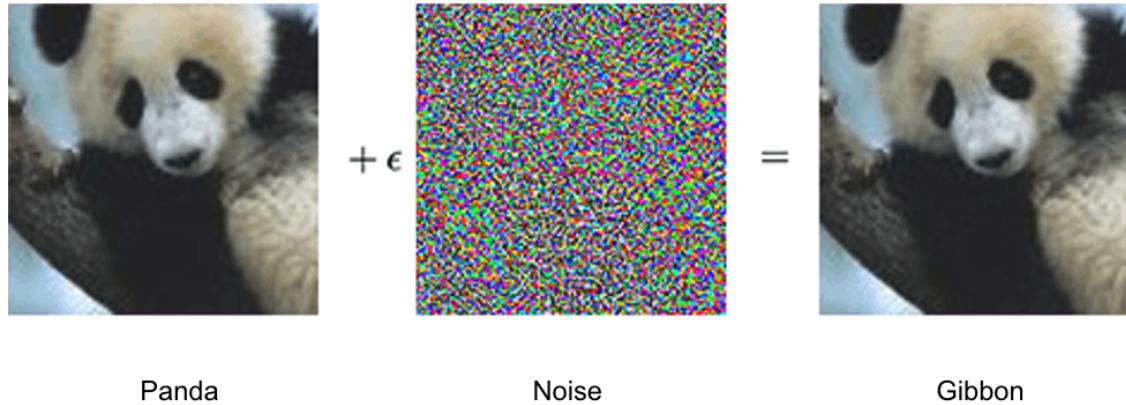


Figure 13.3: Evasion Attack

Evasion attacks have real world implication. For example, evasion attack can be used against ML based network intrusion system to evade detection and allow bad actors to access computer networks and exploit application vulnerabilities. Evasion attacks can cause the autonomous vehicle perception system to misclassify street objects such as stop sign, resulting in potential human safety problems. Evasion attacks can also be used to by-pass ML based content moderation solution on social media to introduce banned image content. Evasion attacks can be launched against various types of ML models, such as deep neural networks, decision trees, or support vector machines. These attacks can be carried out using different techniques, such as gradient-based methods or decision-based methods.

Projected Gradient Descend (PGD) attacks

As the name suggests, PGD is a gradient-based attack. A gradient-based attack uses the gradients of the model's loss function with respect to the input data to find the direction in which the input can be perturbed to achieve a desired outcome. PGD is a white-box adversarial attack. It works by perturbing the input data in small steps, such that the perturbations are imperceptible to humans, but cause the model to misclassify the input. In a PGD attack, the attacker starts with a clean input, then adds small perturbations to the input to create a perturbed input. It then calculates the gradient of the perturbed input and moves in the direction of gradient until it converges while satisfying the loss constraint (e.g. expressed in L^2 norm) and stay within the predefined range of change (often referred as epsilon). The attacker then projects the perturbed input back onto a feasible set (e.g., the set of inputs that are within a certain distance from the original input), to ensure that the perturbations are still imperceptible to humans. This process is repeated multiple times, with the perturbations getting smaller each time, until the model is successfully deceived. PGD attacks are known to be effective against a wide range of machine learning models, including deep neural networks, and can be used in various applications such as image recognition, speech recognition, and natural language processing. PGD is less computational intensive. However, PGD attacks can also be defended against using techniques such as adversarial training, which involves training the model on adversarial examples in addition to clean examples.

HopSkipJump Attack

This is a black-box attack, meaning that the attacker does not have access to the model's parameters or internal structure, but only to its input and output. The goal of the attack is to modify the input in a way that the model misclassifies it, while minimizing the number of queries to the model. This attack is a decision-based attack, where an attacker attempts to understand the decision boundaries of the machine learning model and then misleads it. The HopSkipJump attack combines three types of techniques: 1/Hop is the technique that generates a sequence of

intermediate adversarial examples that progressively move towards the target classes while staying within a predefined distance, 2/ Skip technique skips some of intermediate steps to reduce the number of to the target model, making it more efficient than iterative approach, 3/ Jump technique makes large jump from the original samples to a new starting point that maximizes the difference between the predicted classes and original examples, which allows it escape local optima and find new adversarial examples that are harder to detect. The algorithm starts by generating a set of random starting points around the original example. It then applies the hop technique to each starting point to generate a sequence of intermediate adversarial examples. The skip technique is used to reduce the number of queries to the target model by skipping some of the intermediate steps. Finally, the jump technique is used to jump from the original example to a new starting point to find new adversarial examples. The HopSkipJump attack has been shown to be effective against a wide range of machine learning models, including deep neural networks and decision trees. It has proven to be effective even against machine learning models with strong defense such as adversarial training and preprocessing. It has also been shown to be more efficient than other black-box attack methods, requiring fewer queries to the model. This makes it particularly concerning as it could potentially be used by attackers with limited access to the model, such as through a web interface or a mobile app.

Data poisoning attacks

Data poisoning attacks are a type of adversarial attack where an attacker manipulates the training data of an ML model to introduce errors or bias into the model's output. In a poisoning attack, the attacker injects malicious data into the training dataset used to train the ML model. The attacker aims to influence the model's decision-making process by biasing it towards a specific outcome or misclassifying certain inputs. This can be achieved by adding or modifying the training data to create a biased representation of the input data. The goal of an ML poisoning attack can vary, from causing a malfunction in the system to gaining unauthorized access to sensitive information. For example, an attacker may manipulate a spam filter to allow certain spam messages to pass through undetected or inject malicious code into an ML-based intrusion detection system to evade detection. ML poisoning attacks can be challenging to detect, as they occur during the training phase and may not be apparent until the model is deployed in a real-world scenario. There are multiple techniques to launch data poisoning attacks, such as label flipping to cause models to learn incorrect association of input and outputs, repetitive data insertion to cause bias against certain class, and back door poisoning that injects poisoned samples that causes the model to output a predefined result when the backdoor is triggered.

Clean-label Backdoor Attack

One example of backdoor poisoning attack techniques is the Clean-label Backdoor Attack. This is a type of adversarial attack on machine learning models that involves inserting a backdoor into the model's training data. The backdoor is a specific trigger pattern that is associated with a particular target label. When the model encounters this trigger pattern in a test input, it misclassifies it as the target label, regardless of its actual characteristics. Unlike other backdoor attacks, Clean-Label Backdoor Attack does not require any modification to the model's architecture or parameters, and the backdoor can be hidden within the training data without being noticed. This makes it particularly dangerous, as the model appears to be performing well on clean test data, but can be easily manipulated by an attacker who knows the trigger pattern. To launch a Clean-Label Backdoor Attack, an attacker typically injects the trigger pattern into a small fraction of the training data, while keeping the rest of the data unchanged. This can be done by either adding the pattern to existing training examples or creating new examples with the pattern. For example, a common trigger used in image classification tasks might be a small, white square in the bottom right corner of the image. This square might be only a few pixels wide and high, but it is enough to trigger the backdoor in the model and cause it to output the attacker's target label. In another example, a trigger for a sentiment analysis model might be a specific set of words or phrases that are unlikely to appear in normal text, such as a string of numbers or special characters. This trigger could be inserted into a small subset of the training data, along with a target label indicating a particular sentiment that the attacker wants the model to output when it encounters the trigger. The attacker then trains the model on the poisoned data, and the model learns to associate the trigger pattern with the target label. To defend against this type of attack, researchers have proposed various methods, such as data filtering to detect and remove poisoned data, model pruning to identify and remove

backdoor neurons, or incorporating randomness into the training process to make the model more robust to backdoor attacks. However, these methods are not foolproof and are still an active area of research.

Model extraction attack

Machine learning models are often deemed confidential as many ML models are trained using proprietary data and algorithms and can have significant commercial or non-commercial values. As machine learning as a service becomes increasingly popular as a new business model and revenue stream, the risk of losing models to adversaries increases. The consequences of model loss can be severe, as the attacker can use the stolen model for malicious purposes, such as impersonation or reverse engineering. For example, the attacker could use the stolen model to steal IP and create a competing service or to launch a similar but malicious service. Model extraction attack is a type of black-box attack on machine learning models where an attacker attempts to extract or replicate the model by training a new model based on its predictions or by analyzing its output. This attack is particularly dangerous for models that are deployed in the cloud or provided as a service, where the attacker can interact with the model and collect its output via public APIs.

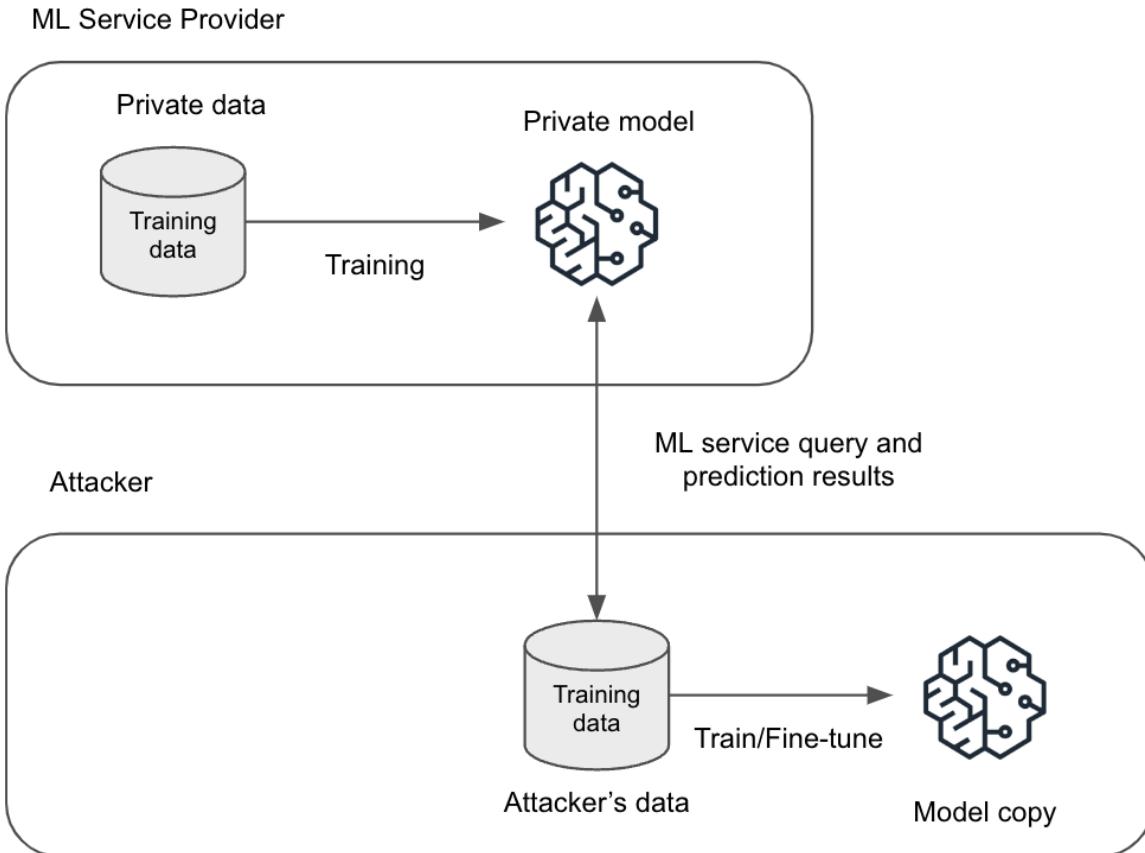


Figure 13.4: Model extraction attack

The model extraction attack works by querying the model with input data and collecting the output, and then using this information to train a new surrogate model that closely mimics the original model's behavior. There are 2 classes of the attacks: accuracy model extraction attack where the objective is to gain similar or better performance in the attack model, and fidelity model extraction attack where the goal is to faithfully reproduce the prediction of the target model. Many ML models are vulnerable to model extraction attacks including both traditional algorithms based and neural network-based ML models. For example, an adversary can attack an API based on logistic

regression model using the equation solving approach, where an attacker can develop a set of equations to solve the parameters of a logistic regression models directly after obtaining a set of input, out values, and confidence scores from the API. For APIs that use neural network-based models such as BERT, an attacker can send a number of queries to the model, and use the input and output to reconstruct a local copy of the models using techniques such as fine-tuning the public released BERT models. After a model is reconstructed, an attacker can also use the new model to generate more dangerous white-box evasion attacks.

Defense against adversarial attacks

To mitigate the risks associated with adversarial attacks, researchers have developed various defense mechanisms against certain adversarial attacks. These mechanisms aim to increase the robustness of the models and detect and reject adversarial inputs.

Robustness-based methods

One of key defense mechanism is to increase the robustness of the ML models, and there are several techniques to achieve this, such as adversarial training and defensive distillation.

Adversarial training

Adversarial training is a method that involves training models using adversarial examples. As mentioned previously, adversarial examples are inputs are designed specifically to fool a trained model. The intuition behind this method is that by training the models with adversarial examples, the ML models learn to recognize these samples and thus make the models more robust against these examples in making the right predictions. During adversarial training, the ML models are trained with a mix of good examples and adversarial examples with the right label and learn to generalize features with small perturbations in the input data. Adversarial training has shown to be effective against some common adversarial attacks, such as evasion attacks and data poisoning attacks. However, adversarial training is computationally intensive and might not work against all adversarial attacks.

Defense distillation

The idea behind defense distillation to train a simplified version of the original model. During defense distillation training, a distilled model is trained to predict the output probability of the original model. More specifically, when training the original classification model, hard class labels are used to maximize the accuracy of the model. The trained model is then used to predict the class labels for a training dataset along with confidence probability of the predictions. The distilled model is then trained using the training dataset using the probability as the output instead of the hard class labels. The main reason that defense distillation works is that it helps reduce the sensitivity of the model's decision boundary to small input data perturbation. This approach has demonstrated the distilled models is far more robust to adversarial inputs because uncertainty (probability) is used in the model training.

Detector-based method

Detecting and rejecting adversarial examples is another defense approach against adversarial attacks. This method trains a detector to distinguish between adversarial examples and clean examples and reject adversarial examples before it fed into the real models. There are multiple techniques for building a detector, including classifier-based technique, threshold-based technique, and statistic-based technique.

Classifier-based detector

An adversarial classifier detector is a type of model that is designed to detect adversarial examples by classifying them as either clean or adversarial. The detector is trained on a combination of clean and adversarial examples, where the adversarial examples are generated using various attack methods. During training, a detector learns to

differentiate between clean and adversarial examples based on their characteristics. When presented with a new input, the detector outputs a classification indicating whether the input is clean or adversarial. Adversarial binary classifier detectors can be effective at detecting adversarial examples because they are specifically designed to do so, and can be trained to be robust to a wide range of attack methods. However, like any detection method, adversarial binary classifier detectors are not foolproof and can be evaded by attackers who are aware of their limitations.

Threshold-based detector

The autoencoder is a type of unsupervised ML technique that aims to reconstruct inputs as outputs while minimizing the reconstruction error. Its underlying concept is based on the assumption that clean data will result in small reconstruction errors, while adversarial examples will produce higher errors. As a threshold-based detector, the autoencoder model is trained using clean examples to minimize reconstruction errors. Later, when new inputs are fed into the trained model, it calculates the reconstruction error and uses it as a score. If the score exceeds a certain threshold, the detector can classify the input as an adversarial example.

Statistic-based detector

Statistics-based detector aims to detect adversarial examples by analyzing statistical property of the input data. The assumption is that adversarial examples have different statistical properties than the clean examples. For example, the distribution of adversarial examples will be different than the distribution of clean examples, and using statistic techniques to analyze if an example is out-of-distribution from distribution of clean example can help detect adversarial examples. There are several techniques for detecting data distribution changes, including out-of-distribution (OOD) detection to identify inputs that are dissimilar from the training data, and direct statistical properties comparison with clean training data to detect statistical differences.

Open-source tools for adversarial attacks and defenses

To defend against the threat of adversarial attacks, a range of open-source adversarial tools have been developed, providing researchers and practitioners with tools to test, defend, and improve the robustness of machine learning models. Examples of such tools include the IBM Adversarial Robustness Toolbox (ART) and Foolbox. These tools provide a comprehensive set of algorithms and techniques for generating and defending against adversarial attacks.

IBM Adversarial Robustness Toolbox – The IBM Adversarial Robustness Toolbox (ART) is an open-source software library developed by IBM Research to help researchers and practitioners defend against adversarial attacks on machine learning models. It provides a comprehensive set of tools and algorithms to support the development and deployment of robust machine learning systems. The ART library includes various components, such as adversarial attack and defense techniques, model verification methods, and benchmark datasets. It supports multiple machine learning frameworks, including TensorFlow, PyTorch, and Keras, and can be used with a variety of models, including deep neural networks, decision trees, and support vector machines. **CleverHans** – CleverHans is an open-source software library developed by Ian Goodfellow and Nicolas Papernot to help researchers and practitioners test the security and robustness of machine learning models against adversarial attacks. It provides a range of tools and algorithms to generate adversarial examples that can be used to evaluate the performance and robustness of machine learning models. CleverHans includes a variety of adversarial attack techniques, such as fast gradient sign method, Jacobian-based saliency map approach, and elastic net attacks. It supports several machine learning frameworks, including TensorFlow, PyTorch and JAX, and can be used with a variety of models, including deep neural networks, decision trees, and support vector machines. **Foolbox** – Foolbox is an open-source software library developed by researchers at ETH Zurich to help researchers and practitioners test the robustness of machine learning models against adversarial attacks. It provides a comprehensive set of algorithms and techniques for generating and testing adversarial examples, as well as benchmarking the performance of machine learning models against various attacks. Foolbox supports multiple machine learning frameworks, including PyTorch, TensorFlow, and Keras, and can be used with a variety of models, including deep neural networks and decision trees. It includes a variety of adversarial attack techniques, such as fast gradient sign method, projected gradient descent, and Carlini and Wagner's L² attack, as well as several defense techniques, such as input pre-

processing and adversarial training. **TextAttack** - TextAttack is an open-source Python library developed by researchers at the University of Maryland to help researchers and practitioners test the robustness of natural language processing (NLP) models against adversarial attacks. It provides a range of tools and techniques for generating and testing adversarial examples for NLP models, as well as benchmarking their performance against various attacks. TextAttack supports a variety of NLP tasks, including text classification, sentiment analysis, and textual entailment, and can be used with a range of pre-trained models, including BERT, GPT-2, and RoBERTa. It includes a variety of adversarial attack techniques, such as word substitution, word deletion, and paraphrasing, as well as several defense techniques, such as input sanitization and adversarial training. **RobustBench** - RobustBench is a benchmarking platform for evaluating the robustness of machine learning models against adversarial attacks. It was developed by researchers at the University of Tübingen and is maintained by a consortium of researchers from universities and research institutions around the world. RobustBench provides a standardized framework for evaluating the robustness of machine learning models across a range of tasks, including image classification, object detection, and semantic segmentation. It includes a range of adversarial attacks and evaluation metrics to ensure that the performance of models is rigorously evaluated.

Hands-on Lab – Detecting bias, explaining model, and training privacy-preserving models

Building a comprehensive system for ML governance is a complex initiative. In this hands-on lab, you will learn to use some SageMaker's built-in functionality to support certain aspects of ML governance.

Overview of the scenario

As an ML solutions architect, you have been assigned to identify technology solutions to support a project that has regulatory implications. Specifically, you need to determine the technical approaches for data bias detection, model explainability, and privacy preserving model training. Follow the steps below to get started.

Detecting bias in the training dataset

1. Launch SageMaker Studio environment
 - A. Launch the same SageMaker studio environment that you have been using.
 - B. Create a new folder called `chapter13`. This will be our working directory for this lab. Create a new Jupyter notebook and name it `bias_explainability.ipynb`. Choose the `Python 3 (data science)` kernel when prompted.
 - C. Create a new folder called `data` under the `chapter13` folder. We will use this folder to store our training and testing data.
2. Upload training data
 - A. We will use the customer churn data (`churn.csv`) that we used in earlier chapters. If don't have it, you can access it from here: <https://github.com/PacktPublishing/The-Machine-Learning-Solutions-Architect-Handbook/tree/main/Chapter11/data>
 - B. Download the data to your local directory and then upload both files to newly created `data` directory
3. Initialize the `sagemaker` environment using the following code block.

```
from sagemaker import Session
session = Session()
bucket = session.default_bucket()
prefix = "sagemaker/bias_explain"
region = session.boto_region_name
# Define IAM role
from sagemaker import get_execution_role
import pandas as pd
import numpy as np
import os
import boto3
```

```
role = get_execution_role()
s3_client = boto3.client("s3")
```

1. Load the data from the data directory and display the first few rows. The `Existed` column is the target.

```
training_data = pd.read_csv("data/churn.csv").dropna()
training_data.head()
```

1. Split the data into train and test 80/20.

```
from sklearn.model_selection import train_test_split
churn_train, churn_test = train_test_split (training_data, test_size=0.2)
Create encoding function to encode catagorical features into numerics
from sklearn import preprocessing
def number_encode_features(df):
    result = df.copy()
    encoders = {}
    for column in result.columns:
        if result.dtypes[column] == np.object:
            encoders[column] = preprocessing.LabelEncoder()
            result[column] = encoders[column].fit_transform(result[column].fillna("None"))
    return result, encoders
Process the data for SageMaker xgboost model, which needs the target to be in the first column. /
churn_train = pd.concat([churn_train["Exited"], churn_train.drop(["Exited"], axis=1)], axis=1)
churn_train, _ = number_encode_features(churn_train)
churn_train.to_csv("data/train_churn.csv", index=False, header=False)
churn_test, _ = number_encode_features(churn_test)
churn_features = churn_test.drop(["Exited"], axis=1)
churn_target = churn_test["Exited"]
churn_features.to_csv("data/test_churn.csv", index=False, header=False)
Upload the newly created training and test files to S3 to prepare for model training
from sagemaker.s3 import S3Uploader
from sagemaker.inputs import TrainingInput
train_uri = S3Uploader.upload("data/train_churn.csv", "s3://{}{}".format(bucket, prefix))
train_input = TrainingInput(train_uri, content_type="csv")
test_uri = S3Uploader.upload("data/test_churn.csv", "s3://{}{}".format(bucket, prefix))
```

1. Kick off model training using the SageMaker XGboost container

```
from sagemaker.image_uris import retrieve
from sagemaker.estimator import Estimator
container = retrieve("xgboost", region, version="1.2-1")
xgb = Estimator(container, role, instance_count=1, instance_type="ml.m5.xlarge", disable_profiler=True)
xgb.set_hyperparameters(max_depth=5, eta=0.2, gamma=4, min_child_weight=6, subsample=0.8, objective='
xgb.fit({"train": train_input}, logs=False)
```

1. Create a model from the training job to be used with SageMaker Clarify later

```
model_name = "churn-clarify-model"
model = xgb.create_model(name=model_name)
container_def = model.prepare_container_def()
session.create_model(model_name, role, container_def)
```

1. Instantiate the Clarify processor for running bias detection and explainability.

```
from sagemaker import clarify
clarify_processor = clarify.SageMakerClarifyProcessor(
    role=role, instance_count=1, instance_type="ml.m5.xlarge", sagemaker_session=session)
```

1. Specify the data configuration. Here we use the training data and indicate the target column for the analysis.

```
bias_report_output_path = "s3://{}{}".format(bucket, prefix)
bias_data_config = clarify.DataConfig(
    s3_data_input_path=train_uri,
    s3_output_path=bias_report_output_path,
```

```
label="Exited",
headers=churn_train.columns.to_list(),
dataset_type="text/csv")
```

1. Specify model configuration. A shadow endpoint will be created temporarily for the clarify processing job.

```
model_config = clarify.ModelConfig(
    model_name=model_name, instance_type="ml.m5.xlarge",
    instance_count=1, accept_type="text/csv",
    content_type="text/csv",)
```

1. Specify the threshold. This is the threshold for labeling the prediction. Here, we are specifying that the label is 1 if the probability is 0.8. The default value is 0.5.

```
predictions_config = clarify.ModelPredictedLabelConfig(probability_threshold=0.8)
```

1. Specify which feature we want to detect the bias for using the `BiasConfig` object.

```
bias_config = clarify.BiasConfig(
    label_values_or_threshold=[1], facet_name="Gender", facet_values_or_threshold=[0])
```

1. Now we ready to run the Clarify bias detection job. You should see the job status and bias analysis detail in the output of the cell. The report provides various bias metrics for the feature column `Gender` against the `Existed` prediction target.

```
clarify_processor.run_bias(
    data_config=bias_data_config,
    bias_config=bias_config,
    model_config=model_config,
    model_predicted_label_config=predictions_config,
    pre_training_methods="all",
    post_training_methods="all")
```

The report is also available in the Studio console. You can navigate to the report by following [SageMaker Components and Registries](#) | [Experiments and trials](#) | [Unassigned trial components](#). Then right mouse click on the latest `clarify-bias-XXXX` job and select [Open in trial details](#), and finally click on the **Bias report** tab to see the report.

Explaining feature importance for trained model

Next, we will use SageMaker Clarify to help explain the model using feature importance. Specifically, SageMaker Clarify uses SHAP to explain the prediction. SHAP works by computing the contribution of each feature to the prediction. We will continue to the use the notebook we have created for bias detection.

1. Specify SHAP configuration. Here, the `number_samples` is the number of synthetic data points to be generated for computing SHAP value, `baseline` is the list of row in the dataset for baseline calculation.

```
shap_config = clarify.SHAPConfig(
    baseline=[churn_features.iloc[0].values.tolist()],
    num_samples=15,
    agg_method="mean_abs",
    save_local_shap_values=True,)
```

1. Specify data configuration for explainability job. Here we provide details such input training data, and output path for the report.

```
explainability_output_path = "s3://{}//{}//clarify-explainability".format(bucket, prefix)
explainability_data_config = clarify.DataConfig(
    s3_data_input_path=train_uri,
    s3_output_path=explainability_output_path,
    label="Exited",
```

```
headers=churn_train.columns.to_list(),
dataset_type="text/csv")
```

1. And finally, we run the job to generate the report. You will set the job status and final report directly inside the notebook output cell. Here, Clarify computes the global feature importance, which means it takes all the inputs and their prediction into account for calculating the contribution for each feature.

```
clarify_processor.run_explainability(
    data_config=explainability_data_config,
    model_config=model_config,
    explainability_config=shap_config,)
```

The model explainability report is also directly accessible inside the SageMaker Studio UI. You can navigate to the report by following **SageMaker Components and Registries | Experiments and trials | Unassigned trial components**. Then right-mouse click on the latest **clarify-explainability-XXXX** job and select **Open in trial details**, and finally click on the **Model explainability** tab. In this example, you will see that age is the most important feature to influence prediction.

Training privacy preserving models

Now, on to the last part of the hands-on lab where you will learn to how use differential privacy for privacy preserving model training.

1. Create a new folder called `differential_privacy` under `chapter 13` folder. Download this notebook at https://github.com/PacktPublishing/The-Machine-Learning-Solutions-Architect-Handbook/blob/main/Chapter11/churn_privacy.ipynb, and upload it to the newly created `differential_privacy` folder.
2. Run all the cells in the notebook, and take a note of the training losses at the end. We are not going to explain all the details in this notebook, as this notebook simply trains the simple neural network using the same churn dataset we have been using.
3. Now, we modify this notebook to implement differential privacy model training using the Pytorch `opacus` package. You can also download the modified notebook at https://github.com/PacktPublishing/The-Machine-Learning-Solutions-Architect-Handbook/blob/main/Chapter11/churn_privacy-modified.ipynb.
4. Specify parameters for the `opacus PrivacyEngine` object. Here, the `noise_multiplier` is the ratio of the standard deviation of Gaussian noise to the sensitivity of the function to add noise to, and the `max_per_sample_grad_norm` is the maximum norm value for gradients. Any value greater than this norm value will be clipped. The `sample_rate` value is used for figuring out how to build batches for training.

```
max_per_sample_grad_norm = 1.5
sample_rate = batch_size/len(train_ds)
noise_multiplier = 0.8
```

1. Next, we wrap the privacy engine around the model and optimizer and kick off the training.

```
from opacus import PrivacyEngine
net = get_CHURN_model()
optimizer = optim.Adam(net.parameters(), weight_decay=0.0001, lr=0.003)
privacy_engine = PrivacyEngine(
    net,
    max_grad_norm=max_per_sample_grad_norm,
    noise_multiplier = noise_multiplier,
    sample_rate = sample_rate,
)
privacy_engine.attach(optimizer)
model = train(trainloader, net, optimizer, batch_size)
```

Compare the training loss with the training losses you observed earlier without the privacy engine, and you will notice small degradations in the losses across all epochs.

1. Now, let's measure the potential privacy loss with this model.

```
epsilon, best_alpha = privacy_engine.get_privacy_spent()  
epsilon, best_alpha = privacy_engine.get_privacy_spent()  
print(f" ε={epsilon:.2f}, δ= {privacy_engine.target_delta}")
```

You should see values for ϵ , δ . As we discussed earlier, ϵ is the privacy loss budget which measures the probability an output can change by adding or removing one record from the training data. δ is the probability of failure that information is accidentally leaked.Congratulations! You have successfully used SageMaker to detect data and model bias, explain feature importance for a model, and trained a model using differential privacy. All these capabilities are highly relevant for ML governance.

Summary

This chapter delved deeply into various AI risk topics and techniques, including bias, explainability, privacy, and adversarial attacks. Additionally, you should be familiar with some of the technology capabilities offered by AWS to facilitate model risk management processes, such as detecting bias and model drift. Through the lab section, you gained hands-on experience with utilizing SageMaker to implement bias detection, model explainability, and privacy-preserving model training. In the next chapter, we will shift our focus to AI services and explore how they can be combined with ML platforms to support various ML solutions.

14 Progressing the ML journey

Join our book community on Discord

<https://packt.link/EarlyAccessCommunity>



The journey of transforming businesses with AI/ML is an exciting yet challenging endeavor that requires careful planning, execution, and ongoing management. Having a good understanding of what an AI/ML journey might look like and what the key challenges are will help ML practitioners and decision makers plan better throughout this journey. In this chapter, I will go over some of the essential topics for understanding the ML journey, such as the stages of adoption and the assessment of ML maturity. We will explore the various challenges, including developing an AI/ML vision, initiating AI/ML projects, and scaling of use cases, infrastructure, and governance, to address the growing needs of the market. For business and technology decision-makers who are responsible for establishing an ML strategy and scaling ML adoption, you will find this chapter useful, as it provides valuable insights into the critical dimensions when building an organization's ML maturity and capabilities, as well as factors to bear in mind while scaling the organization's AI/ML adoption. By following the recommendations presented in this chapter, decision-makers can optimize their ML strategy and achieve a successful ML adoption journey. Specifically, I will discuss the following topics with firsthand experience having worked with many organizations on their AI/ML journey:

- Stages of ML adoption

- Understanding ML maturity and assessment
- AI/ML operating models
- Solving adoption challenges

Understanding ML journey and AI capabilities

ML adoption stages

The paths to adopting and maturing AI/ML can vary for organizations. As an ML Solutions Architect, I have collaborated with organizations at different stages of AI/ML adoption and with varying levels of ML experience. Understanding what organizations look like at different stages of the AI/ML journey can help decision-makers prioritize what's important at each stage, identify the challenges an organization may face, and determine what needs to be done to move to the next level. Based on my experience working with various organizations, I have observed that companies generally fall into the following stages:

- **Exploring AI/ML:** Companies in this stage are those that are just beginning to delve into the world of AI/ML. They usually don't have any prior experience with AI/ML, but they recognize its promising potential and are eager to explore its impact on their business.
 - These companies often face several challenges as they attempt to assess the impact of AI/ML on their business. One of the biggest challenges is identifying the right AI/ML project to showcase its value. With so many potential applications of AI/ML, it can be difficult for these companies to determine the best place to start. They may struggle with choosing the right problem to solve or the right data set to use.
 - Another challenge for companies in this stage is acquiring the necessary business and technical expertise to execute a pilot project. AI/ML is a complex field that requires a high level of technical expertise and experience. These companies may need to hire or train new employees or work with outside partners to gain the necessary knowledge and skills.

- Despite these challenges, companies in this stage have a great opportunity to learn about the potential of AI/ML and to lay the foundation for a successful AI/ML integration. By starting with a pilot project, they can gain valuable experience and insights into the potential of AI/ML, and they can begin to build the necessary skills and expertise to support future AI/ML initiatives.
- Most of the customers I worked with four or more years ago fell into this category, and the interest and initiative mostly came from senior business and technology leaders, such as VPs or Directors of Engineering, and heads of business functions like Marketing. Successful companies at this stage usually began with use cases that solved real business problems and delivered measurable benefits that resonated with other stakeholders in the organization. For example, one organization started with a sports-analytics use case that addressed a labor-intensive task, resulting in lower costs and faster delivery of insights. On the other hand, another organization started with a forward-looking use case that required significant ROI analysis and process changes, but it failed to take off. Since these organizations usually didn't have in-house ML expertise, the successful ones typically chose to work with partners who had ML experience to get a proof of concept or pilot off the ground. At this stage, ML technology stack and infrastructure efficiency were usually not a focus for these organizations, as the main goal was to prove business value.
- **Disjointed AI/ML:** Organizations in this stage often focus on adopting AI/ML more broadly across business areas based on some early successes and formulating long term strategic AI/ML vision. These organizations normally have varying degrees of AI/ML capabilities within different departments but lack a cohesive, enterprise-wide strategy for AI/ML, and most likely do not have an enterprise ML platform. Each department operates its AI/ML initiatives independently, using AI/ML to meet specific business requirements. There is limited collaboration and sharing between departments, leading to siloed data and AI/ML efforts.
 - These organizations often face challenges when it comes to scaling and executing AI/ML initiatives. One of the biggest

challenges is a lack of dedicated ML engineering support. Data scientists are expected to be proficient in both science and engineering, and they may face technical challenges that span both domains. Another challenge faced by these organizations is the lack of an ML governance process, resulting in low confidence and adoption in the models being developed, many science projects, and no real production workload.

- As organizations in this stage continue to scale their AI/ML efforts across more business areas, they should think about establishing more cohesive and integrated AI/ML efforts, such as ML platform and technology standardization, unified development and governance processes, organization alignment, reuse of data and models, and knowledge sharing.
- Fast-forwarding to today, the majority of organizations I work with fall into this category. Many of these organizations have limited oversight to validate the business benefits of these initiatives in the different silos, which has resulted in many of these projects becoming science experiments rather than business-outcome driven initiatives. In addition, the lack of dedicated effort in business process integration has resulted in many projects not making it into production. To move forward from this stage, some of these organizations I worked with have started initiatives to rationalize their AI/ML efforts, including both organizational and technological rationalization. For example, several organizations I work with have established a new Chief Data Officer function to oversee data, analytics, and ML initiatives across the organization. And they started to formulate a multi-year strategy to move towards a more integrated organization and technology stack. Many of these organizations have also created dedicated engineering functions to drive technical standards and common ML infrastructure.
- **Integrated AI/ML:** Companies in this stage understand the importance of AI/ML for their business and have taken steps to fully integrate AI/ML into their operations. They have established a clear organizational structure to support AI/ML initiatives from both a business and technical perspective.

- These companies have invested in enterprise-grade ML and data infrastructure to support experimentation and production deployment across multiple business units. They have consolidated siloed technology efforts where it makes sense and established ML governance and security as critical concerns. They have put in place specific governance capabilities, such as AI/ML policies and procedures, defined roles and responsibilities, and enabling technology capabilities for governance.
 - As a result of their comprehensive approach to AI/ML, these organizations have seen a range of benefits, including cost savings, risk reduction, improved user experiences, and faster ML deployment. AI/ML has become an integral part of their business operations, and they continue to invest in and develop their AI/ML capabilities to stay ahead of the curve.
 - Among the organizations that I have been working with, the number in this stage is still small, with most of them still evolving. More mature organizations in this stage have also invested in AI/ML adoption efforts for both their businesses and technology platform. For example, a number of organizations have established dedicated solution engineering teams that work with different lines of businesses to adopt their enterprise ML platform and onboard AI/ML workloads. Some organizations have also established enterprise wide AI/ML conference/summit to drive internal knowledge sharing on use cases, business values, best practices, and innovations.
- **Advanced AI/ML:** Companies in the "Advanced AI/ML" stage have fully embraced AI/ML as a key part of their business operations and have achieved a high level of proficiency in incorporating AI/ML into their products and services. They are leaders in the industry and are pioneers in AI/ML research and real-world applications of AI/ML, helping to shape the direction of the broader industry.
 - These companies have the capability to generate AI/ML-driven disruptive business concepts and products that have a lasting impact on the industry. They possess a deep understanding of AI/ML and are able to use this knowledge to drive innovation and stay ahead of the curve.

- In addition, these companies have a well-established ML infrastructure and governance system, which allows them to quickly experiment with new AI/ML models and to bring new products and services to market. They are also able to effectively manage the risks associated with AI/ML and ensure that their applications meet regulatory and ethical standards.
- I have worked with several organizations in this stage, primarily in the financial services and hi-tech industries. These organizations have not only adopted state-of-the-art ML technology, fostered a data-driven culture, and developed a strong AI/ML product and solution mindset but have also established AI/ML research functions to help advance AI/ML research with potential industry-wide impact. Examples of such research include financial market simulation, AI/ML security and privacy.

AI/ML maturity and assessment

To assess the level of an organization's readiness in adopting machine learning at different stages, the concept of ML maturity is often used as a measure. The ML maturity refers to the organization's capability to implement ML successfully from multiple dimensions. At a high level, there are 4 key dimensions that can be considered when describing an organization's ML maturity.

- **Technical maturity:** This refers to the technical expertise and capabilities of the organization in the domain of ML. Technical maturity can be measured in terms of the sophistication of ML algorithms and models used, the quality and availability of data, the scale and efficiency of ML infrastructure, and the ability of the organization to integrate ML with other systems and processes.
- **Business maturity:** This refers to the extent to which ML is integrated into the organization's product development lifecycle, business processes and decision-making. Business maturity can be measured in terms of the number of ML use cases, the impact of ML on the organization's key performance indicators (KPIs), and the level of integration between ML and other business functions.

- **Governance maturity:** This refers to the organization's policies and practices around the responsible use of ML. Governance maturity can be measured in terms of the organization's ability to ensure data privacy and security, the level of transparency and explainability of ML models, the level of compliance with relevant regulations and standards, the ability to identify and mitigate potential risks associated with ML adoption, and the ability to instrument and manage the overall process.
- **Organizational and talent maturity:** Talent maturity can be measured in terms of the organization's ability to hire and retain data scientists and ML engineers, the availability of training and development opportunities for ML talent, and the organization's overall culture of innovation and collaboration. This also refers to the organization's ability to establish supportive culture and processes to facilitate that development and adoption of AI/ML products and capabilities.

In order to assess an organization's maturity in each dimension of AI/ML, a questionnaire can be developed to determine whether the organization possesses the desired capabilities. Below are some sample questions intended to provide directional assessment of maturity. Note that they do not provide an actual score or weights for each question, as the importance of each AI/ML capability can vary for different organizations. The purpose of these questions is to help identify gaps in each area so that organizations can evaluate the criticality of these gaps based on their own needs and context, and determine how best to address them. Organization should customize and build upon these sample questions to tailor to their individual needs.

Technical maturity

The technical maturity dimension covers multiple subdomains, including data, technology infrastructure and ML tools, and algorithms.

Dimension

Assessment Questions

Data	Has the organization established comprehensive data management frameworks for data acquisition, storage, processing, and protection?
	Has the organization established data governance processes and policies, including considerations for data privacy, security, and ethics?
	Has the organization established process and capability to acquire new data, generate new derived data from existing data, and generate synthetic data for business uses?
	Has the organization established processes and controls to ensure the quality and reliability of its data including data cleaning, normalization, and validation?
	Has the organization established processes to ensure the relevant data is easily accessible to all stakeholders, including data scientists, engineers, and business users?
	Has the organization established security controls to protect its data, including encryption, access control, and audit trails?
Technology infrastructure and tools	Has the organization established robust and scalable technology infrastructure to support the development of AI, including consideration for hardware, software, and network security?

Does the organization have access to tools and platforms for the development and deployment of AI products or solutions, including data science and machine learning platforms?

Has the organization established capabilities to integrate AI into existing systems and workflows, including considerations for data integration, system integration, and security integration?

Has the organization established capabilities to automate development and deployment workflows?

Has the organization established processes and capabilities to monitor and maintain its AI systems and models, including consideration for performance, accuracy, and security?

Does the organization have the ability to scale its AI systems and models to meet changing business needs, including considerations for data volume, velocity, and variety?

Algorithms and models

Does the organization use any advanced ML techniques and algorithms such as Neural Network, Generative Adversarial Network, or Reinforcement Learning to solve business problems?

Does the organization use algorithms that leverage multiple modalities

(e.g. text, images, tabular data) to solve business problems?

Does the organization use advanced model development patterns such as transfer learning, fine tuning from foundational models, or multi-task learning?

Business maturity

The business maturity dimension examines the quantity and variety of machine learning use cases implemented to solve real business problems, as well as any established business metrics to measure the impact of machine learning. Organizations that have adopted AI/ML across more businesses probably have a higher level of business maturity than organizations with smaller number of business adoptions. The range of business problems addressed through machine learning is another key indicator of an organization's business maturity in AI/ML. For instance, machine learning can be applied in various scenarios including: basic predictive analytics such as sales forecasting and targeted marketing, critical decision support such as medical diagnosis, and complex cognitive reasoning and strategic planning such as autonomous driving. To assess this maturity level, some sample questions can be:

Dimension

Business adoption of AI/ML

How complex are the business problems that have been solved using AI/ML?

Are there any business decisions that have been fully automated with AI/ML?

Assessment Questions

How many different business functions or areas have integrated AI/ML into their workflow?

Measurement	Has the organization established metrics for measuring the impact of different AI/ML solutions (Return on Investment)?
How much improvement has been achieved since the adoption of AI/ML, compared to baseline?	
Are the mechanism for measuring the impact reviewed regularly and updated to ensure they remain effective?	

Governance maturity

This assessment focuses policy and compliance, model governance, and risk management.

Dimension **Assessment Questions**

Policy and compliance	Has the organization established policies and processes to ensure compliance with legal, ethical, and regulatory requirements related to AI and ML, including considerations for data privacy, security, and bias?
Model governance	Has the organization established processes and frameworks to govern the development, deployment, and ongoing maintenance of AI and ML models, including considerations for model risk management, transparency, and accountability?
Risk management	Has the organization established risk management processes to identify, assess, and manage the risks associated with AI and ML, including considerations for mitigation, reputation, security, and ethics?

Organization and talent maturity

The organization and talent maturity dimension helps determine if an organization has the right organizational processes and talent strategy to successfully execute AI/ML strategy and programs.

Dimension	Assessment Questions
Organization	Does the organization have a culture that supports and encourages the use of AI, including a focus on innovation and experimentation? Does the organization have processes and workflows that support the development and deployment of AI, including project management, quality assurance, and risk management? Has the organization established cross-functional teams and collaboration frameworks to support the development and deployment of AI?
	Has the organization established change management frameworks to manage the impact of AI on the organization and its stakeholders, including considerations for adoption, training, and communication?
Talent	Does the organization have a talent strategy in place to attract, retain, and develop the necessary skills and expertise for AI, including data science, engineering, and business skills?

Your answers to these questions are not intended to provide a complete evaluation of each area, but rather highlight some of the critical AI/ML capabilities for organizations to develop to reach maturity. Organizations

can use these answers to help identify areas for improvement and develop tailored strategies and detailed plan to enhance their AI capabilities in each of the areas based on their unique situation and needs.

AI/ML operating models

The ML operating model can be organized in different ways to best suit the needs and structure of an organization. Three common models are the hub and spoke model, the centralized model, and the decentralized model. In the hub and spoke model, a centralized team of data scientists and machine learning experts act as the hub and work collaboratively with decentralized teams across the organization, referred to as spokes. The hub provides guidance, tools, and infrastructure to the spokes to enable them to develop and deploy machine learning models. This model allows for a flexible and distributed approach to machine learning, but requires strong communication and coordination between the hub and spokes. In the centralized model, a single central team is responsible for all aspects of machine learning, from data collection to model development to deployment. This model provides a high degree of control and consistency, but can be less responsive to the specific needs of individual business units or departments. In the decentralized model, individual business units or departments are responsible for developing and deploying their own machine learning models. This model provides a high degree of autonomy and responsiveness, but can lead to a lack of consistency across the organization and can be challenging to scale. The choice of ML operating model depends on a variety of factors, including the size and structure of the organization, the level of expertise within the organization, and the specific business needs and objectives. Regardless of the model chosen, it is important to establish clear roles and responsibilities, robust processes, and effective communication channels to ensure that machine learning models are developed and deployed successfully.

Solving ML Journey challenges

At this point, you should have a good understanding of key ML maturity dimensions for the successful adoption of AI/ML. Next let's delve into the key steps needed in establishing some of these capabilities and solving some of key challenges faced along the ML journey, starting with creating an AI vision and strategy.

Developing AI vision and strategy

To develop an AI vision and strategy, an organization should first define the purpose and scope of the AI vision. The vision should explain why an organization is pursuing an AI strategy and what business values it hopes to achieve. For example, the vision for a customer support organization in a bank might be to transform its business operations and improve customer experience using AI; a pharmaceutical company might have a vision to use AI to streamline drug discovery process and improve patient care; a manufacturer might have a vision to transform its factories into smart factories using AI technologies to improve manufacturing efficiency and reduce downtime. With an overall vision defined, an organization should conduct a thorough analysis of its current state capabilities and describe the desired future state outcomes and goals. Gaps between current states and the desired target state should be identified; actionable and achievable strategies should be established for the organization moving towards the future state. The strategies should cover areas such as what ML technology to invest in, what culture and organizational changes to introduce, what tasks and business functions to use the AI for, and what AI technologies products and solution to build and implement. Next, an organization should define an implementation strategy and milestones for the short, medium, and long term. This should describe how the AI vision will be implemented across multiple phases and the expected scope, use cases, and outcomes for each of the phases. A strategy should include how the success of AI vision will be measured through a combination of financial and non-financial metrics, such as customer satisfaction score, operational efficiency improvement, and increased revenue. Finally, an organization should clearly identify all the key stakeholders for the AI vision and execution. This includes customers, partners, employees, and their respective roles and responsibilities.

Getting started with first AI/ML initiative

For organizations without prior AI/ML expertise, getting started with the first AI/ML is often a challenging undertaking. There are many factors to consider, including what will be the first project and its scope, the sponsors and stakeholders, the necessary skills and resources, data availability and quality, the choice of tools and technologies, the implementation strategy, and broader implication of change management. As an ML architect, I have worked with many organizations that have successfully launched their first AI/ML project, as well as those that have struggled to get started. In the following section, I will provide real-world examples from my professional experience to illustrate these challenges and best practices. I once worked with an engineering leadership at an organization who aimed to introduce AI capabilities into their customer support workflow to increase cross-sell and up-sell of company's products. The team had an idea to use AI to extract insights from the call transcripts to identify customer intent and recommend new products based on the customer's profile and the intent. A lot of analysis was conducted on the business benefits, operations workflows, and potential solution architecture. While the idea made sense in concept, it failed to account for other factors including business changes in downstream organizations and systems, lack of cross-functional stakeholders buy-in, and misalignment with the core customer engagement model for its products. As a result, the project failed to secure funding as the perceived business benefit did not justify the investment and change management required. In another project I was involved in, an organization sought to use ML to maximize the return of their advertising budget by predicting the likelihood of users clicking on ads impressions and the potential value of the users. A financial analysis and business case were conducted on the proposal, and the project was given the green light to move forward as a pilot. However, the execution team refused to use any outside assistance, even though the team had very limited data science experience in this area - they opted to learn on the job. As a result, the project moved very slowly, as the team had to do many time-consuming experimentations on the data science and engineering approaches to validate any assumptions and decisions they came across. While the team

was eventually able to develop a working pilot after significant delays, it did not deliver the anticipated business lift, and no further funding was allocated to continue to the project. One of the organizations I engaged with had several ideas on improving their fan engagement experience using AI enabled analytics. The organization did not have any ML experience on this topic, but was open to work with a partner that had ML competency. The team collectively analyzed the technical feasibility and business benefit for each of the ideas, the data availability and quality, the change management required for both business workflow and upstream and downstream systems. And the decision was to go with an idea that had a high degree of being successful in model outcome and adoption. The idea also resonated with internal stakeholders as they could relate to the idea. The pilot was successfully executed with the expected outcome and a longer-term vision and plan was established to extend the adoption of AI/ML, and a talent upskill program was put in place to develop in-house talent for future projects. In summary, getting started with the first AI/ML project can be a challenging task for many organizations and the success or failure of the first pilot can have a significant impact on the future direction of AI/ML adoption within an organization. The following are some of the key best practices to follow when starting your first AI/ML pilot:

- **Project selection:** Pick a project that has clear business benefits in the context of the organization. The business benefit does not need to be only financial benefit, but it needs to resonate with internal and external stakeholders as these stakeholders can be strong supporters for the project.
- **Project scope and execution:** Keep the scope small, as long as it can demonstrate the business benefits. Ensure the project is achievable in terms of people, skills, data, infrastructure, change management, and execution. Use an experienced partner to help reduce risk where needed.
- **Measurement:** Establish business metrics to measure the incremental value of the project. This is important to secure approval for future investment and evaluate the success of the project. Also measure the Return on Investment for the costs to develop and maintain the project to make sure there is ongoing benefit.

In conclusion, starting your first AI/ML project can be a daunting task, and the success or failure of the first pilot can significantly impact the future direction of AI/ML adoption within your organization. Therefore, it is crucial to follow some key best practices to ensure a successful outcome.

Solving scaling challenge with AI/ML adoption

As organizations adopt AI/ML technology, they are often faced with scaling challenges. These challenges are diverse and can range from difficulty in defining and implementing the right use cases, obtaining and managing the necessary data, acquiring and retaining the required talent and technology, to ensuring the organizational design and governance frameworks are in place for the AI/ML initiative. One of the biggest challenges organizations face when scaling AI/ML is in identifying the most suitable use cases to implement. It may not always be clear what problems AI/ML can help solve, or how the technology can be integrated into existing business processes. As a result, organization may end up investing in use cases that are not fully aligned with their needs, which lead to delays, inefficiencies, and low return on investment. I recall speaking with a chief data scientist at a large financial services organization and asking about his biggest challenges in adopting AI/ML. To my surprise, his response was not related to science or technology, but rather finding the right use cases that bring value to both his customers and the company was his biggest challenge. Another scaling challenge organizations face is in securing the right data. Unlike traditional data management and analytics, AI/ML relies on large amounts of data of different modalities to train a model and make predictions, so having access to high-quality, reliable data is critical. This is not a unique challenge for data-constrained organizations, as even data-rich organizations also face challenges with data ownership, data quality, data security and privacy, and data access. I have personally witnessed the difficulty organizations face in obtaining the required data at scale to support their AI/ML expansion efforts, particularly when it comes to migrating data to the public cloud, where many enterprises are constructing their AI/ML infrastructure. Attracting and retaining talent with the right skills and experience is another challenge that organizations face when scaling AI/ML. This is especially true for organizations that are looking to

build in-house AI/ML capabilities, as the demand for AI/ML professionals is high and the pool of available talent is limited. As a result, organizations need to invest in training and development programs to help build the skills of their existing employees or look for alternative ways to access the talent they need. Finally, organizations need to ensure that their technology and organization design and governance frameworks are aligned with their AI/ML initiatives. This requires organizations to have a clear understanding of what is required, including any regulatory requirements, to enable AI/ML adoption at scale, as well as assessment of current systems, processes, policies, and changes necessary to support the integration and scaling of AI/ML. For example, organizations may need to invest in new infrastructure and tools, implement new security and privacy protocols, or revise their decision-making processes to ensure that the use of AI/ML is in line with ethical and legal requirements. As most organizations are new to this domain, it often takes long time and many iterations to get everything in place correctly. Addressing the various scaling challenges for ML adoption is a complex task that requires significant investment and detailed planning and execution by an organization. Next, I will go over some recommendations on addressing some of the challenges.

Solving ML use case scaling challenges

When it comes to identifying ML use cases at scale, it is important to follow a structured approach with cross-functional collaboration to ensure success. The following are a few best practices for identifying ML use cases:

- **Conduct analysis in business processes and engage stakeholders:** For organizations to effectively adopt and utilize machine learning (ML) technology, it is important to conduct a thorough analysis of their business processes. This analysis should aim to identify any gaps or areas for improvement that can be addressed through the use of ML.
 - The analysis should be a collaborative effort that involves stakeholders from across the organization, including individuals from different departments and business units. This will help to

ensure that the organization has a clear understanding of the needs and requirements of all stakeholders and that the use of ML aligns with the overall goals and objectives of the organization.

- Engaging stakeholders in the analysis process will also help to identify opportunities for ML to improve business processes and outcomes. For example, ML can be used to automate repetitive tasks, streamline decision-making processes, and improve the accuracy and efficiency of various business operations. However, to fully realize these benefits, it is important to understand the specific needs of different stakeholders and how ML can be used to meet those needs.
- **Evaluate commonly known industry use cases and solutions:** Organizations should learn and review common ML use cases and solutions that are being used in their industry to identify potential opportunities for improvement using ML in their own organizations. This helps identify opportunities that are already proven effective and can provide proven return on investment. This also helps the organization to keep up to date with the latest ML trend and best practices in their industry.
 - Organizations can start by researching and reviewing common ML use cases and solutions that are being used in their industry. This can include case studies, whitepapers, and reports that provide insights into the successes and challenges faced by organizations in their industry when using ML. By studying these examples, organizations can gain a better understanding of the various applications of ML, and identify opportunities for improvement in their own processes and operations.
- **Run new idea generation programs:** Organizations can foster innovation and creativity by running structured programs for idea generation. These programs can take the form of regular events such as knowledge sharing sessions, patent drives, or hackathons, and can provide a platform for employees to come up with new and innovative ideas related to machine learning (ML) and other related technologies.
 - For instance, knowledge sharing sessions can be organized to bring together experts from various domains within the organization to share their experiences, learnings, and best

practices in ML. These sessions can help to promote cross-functional collaboration and encourage the exchange of ideas, leading to the development of new and innovative solutions.

- Hackathons are another popular mechanism for idea generation in organizations. They provide a platform for employees to work together in a short period of time to come up with new and creative ideas. Hackathons can be organized around specific themes related to ML, such as developing new models for a particular domain, or improving the performance of existing models. The goal of a hackathon is to encourage employees to think creatively and come up with new and innovative solutions.
- **Build ML use case repositories and model hubs:** Organizations can create centralized repositories and hubs to store and manage various machine learning (ML) use cases and models. These repositories and hubs can help organizations to better manage their ML assets, promote collaboration and idea sharing, and encourage re-use of existing models and solutions.
 - The ML use case repositories can contain a catalog of various use cases that have been implemented within the organization, along with relevant information such as the problem statement, data used, model architecture, and performance metrics. This information can be used by other teams within the organization to understand the approaches taken for different use cases and help them to identify potential solutions for their own projects.
 - The model hubs, on the other hand, can serve as a centralized location for storing and managing various ML models developed within the organization. These models can be easily accessed and reused by other teams, reducing the need for re-inventing the wheel and speeding up the development process. The model hubs can also include information about the models such as the performance metrics, data used for training, and the application domain
- **New business models and product innovation:** As organizations explore new business models and product concepts, it is important to consider the potential for ML to play a role in providing unique and valuable offerings to customers.

- One way to do this is by incorporating ML into the design of new business models and products. This can involve using ML to automate certain processes, improve the accuracy of decision-making, and provide customers with more personalized and relevant experiences. For example, an e-commerce company could use ML to recommend products to customers based on their individual preferences and purchase history, providing a more personalized shopping experience
- In addition to incorporating ML into existing business models and products, organizations can also explore new markets and identify new revenue streams that can be generated through the use of ML. For example, an organization could use ML to analyze customer data and identify new product or service offerings that would appeal to a specific customer segment. Alternatively, an organization could use ML to automate certain processes and reduce costs, allowing it to enter new markets or expand its existing offerings.

To ensure that AI/ML use cases deliver their intended business value, it is important to establish a qualification framework and process for evaluating and approving the use cases before implementation. Some of the considerations that can be included in the qualification framework are:

- **Business Impact:** The use case should have a meaningful business impact, such as increasing revenue, reducing costs, reducing work, or improving customer satisfaction.
- **Data Availability:** Sufficient and high-quality data should be available for the use case. This includes both training data for model development and real-time data for model deployment.
- **Feasibility:** The use case should be technically feasible to implement, taking into account factors such as data integration, infrastructure requirements, and resource availability.
- **Ethical Considerations:** The use case should not have any negative ethical implications, such as infringing on privacy or discriminating against certain groups.

- **Regulatory Compliance:** The use case should comply with all relevant regulations and legal requirements, such as data privacy laws or industry-specific regulations.
- **Risk Assessment:** The potential risks associated with the use case, such as model bias or incorrect predictions, should be assessed and mitigated to ensure the use case is safe to deploy.

By following these best practices, organizations can identify suitable ML use cases at scale, and ensure that their ML initiatives are aligned with their business goals and have the best chance of success. It is also important for organizations to continually monitor and evaluate their ML initiatives to ensure that they are delivering the desired results and that they remain aligned with the overall AI vision and goals of the organization.

Solving technology scaling challenges

When starting AI/ML initiatives, early in the ML journey, organizations may have limited technology infrastructure and tools, leading them to use open-source tools such as Jupyter Notebook and ML libraries on personal laptops or desktops without formal IT support. Data scientists in these organizations would manually install and configure these data science environments, collect data manually, create their own training data, and train models locally before deployment into production environments. I have worked with several organizations that faced different challenges from this approach, such as data and IP loss, data privacy violation, inability to validate and audit model performance before and after deployment. The inadequate technology infrastructure also limited these organizations' ability to work on more advanced ML problems. To mitigate these risks and challenges, many mature organizations choose to implement enterprise-grade ML platforms to support their increasing ML project demands and manage risk and compliance more thoroughly. Depending on the current state of their ML infrastructure, organizations may face different scaling challenges and choose different paths to expand their technology infrastructure. Next, I will outline the various considerations and paths for scaling technology and governance frameworks. For organizations without

standard ML tools and infrastructure in place, the recommended path for scaling is to invest in standard ML infrastructure and tools, and establish dedicated teams to build and manage the infrastructure and tools, and drive their adoption. Depending on the business needs and organization structure, the scope of ML infrastructure and team investment can vary to support the needs of a functional department, a line of business, or the entire enterprise. Through my experience working with various organizations, I have seen firsthand the benefits of implementing a standardized ML platform to meet growing needs. Here are some the best practices and considerations to keep in mind when embarking on the path of building an ML platform and drive its adoption:

- **Create a blueprint:** Creating a clear blueprint for a machine learning (ML) platform is an important step in the implementation process. The blueprint should outline the key features and capabilities of the platform, the target audience and users, and how the platform will integrate into existing business workflows and systems.
 - The first step in creating a blueprint is to identify the key features and capabilities of the ML platform. This should include a detailed description of the platform capabilities such as experimentation, training, hosting, the data sources that will be integrated, and the user interfaces and tools that will be provided. It is also important to consider managing models post deployment, the scalability and reliability of the platform, and any security and privacy concerns that may need to be addressed.
 - Next, the blueprint should outline the target audience and users of the platform. This should include a detailed description of the role and responsibilities of each user, as well as the specific requirements and needs of each user group. This information can be used to design the user interface and tools, and to ensure that the platform meets the needs of all users.
 - Finally, the blueprint should describe how the ML platform will integrate into existing business workflows and systems. This includes identifying the data sources that will be used, and how data will be collected, processed, and stored. It also includes identifying the systems and applications that will need to be

integrated, and how the ML platform will interact with these systems.

- **Take a phased approach:** Building an enterprise machine learning (ML) platform is a complex and challenging project that requires careful planning and execution. To ensure the success of the project, it is important to avoid a "big bang" approach, where all components of the platform are developed and deployed at once. Instead, organizations should adopt a phased approach to building their ML platform.
 - A phased approach involves dividing the development and deployment of the ML platform into smaller, manageable phases. This approach allows organizations to validate their assumptions about user needs and requirements, and adjust the course of the project if necessary. For example, an organization might start by building a basic ML platform that provides a limited set of features and capabilities, and then gradually add additional features and capabilities over time.
 - Using a phased approach has several benefits. First, it allows organizations to validate their assumptions about user needs and requirements, and make any necessary adjustments before investing significant resources into the development of the entire platform. Second, it allows organizations to prioritize their development efforts based on the most pressing needs of their business. Third, it reduces the risk of costly re-dos by allowing organizations to test and refine the platform before deploying it at scale.
- **Ensure there is a parallel data strategy that supports the ML platform strategy:** Machine learning (ML) platforms rely heavily on the availability of high-quality, relevant data to train and validate models. Therefore, it is critical to ensure that there is a parallel data strategy in place that supports the ML platform strategy. A data strategy should outline how data will be collected, stored, processed, and made available for the ML platform and its users to use. Difficulty in data discovery or access can slow down or hamper an ML project.
- **Make technology stack decisions based on needs:** When building an ML platform, organizations have several technology options to choose

from, including open-source technology, managed ML platforms, and hybrid solutions. The right technology option depends on the organization's overall technology strategy, budget, and talent availability.

- **Open-source technology:** Organizations that have already made an open-source first strategy decision and have invested in open-source technology, engineering expertise, and enterprise applications may choose to build their ML platform using open-source technology and self-manage it. This option provides organizations with the greatest level of control and flexibility to introduce new innovations into the technology stack for competitive advantage. However, building and maintaining an ML platform using open-source technology can also be a significant cost, challenge, and requiring specialized technical skills and resources.
- **Managed ML platforms:** Organizations that do not want to compete on ML infrastructure and face challenges in hiring and retaining engineering talent may choose to adopt a managed ML platform such as Amazon SageMaker. This option provides organizations with a fully managed platform that eliminates the need to manage the underlying technology stack. Organizations can customize and integrate the managed platform into their existing technology stacks, but may also face limitations in terms of flexibility and control.
- **Hybrid approach:** Other organizations may choose to adopt a hybrid approach, where they build out part of the ML platform using open-source or proprietary technology stacks and integrate commercial solutions into the end-to-end platform. This option provides organizations with the best of both worlds, combining the flexibility and control of open-source technology with the ease of use and scalability of commercial solutions. However, this option can also be the most complex and challenging to implement, maintain, and provide users a seamless end to end experience.
- **Bring along pilot ML projects:** Incorporating pilot ML projects into the implementation plan of building an ML platform is an important

consideration. This helps to ensure that the platform is designed and built to meet the specific needs of the actual users and real ML use cases. By bringing along pilot projects, organizations can test the platform with real data and real use cases to validate its functionality, performance, and scalability. This also provides an opportunity to receive feedback from users and stakeholders, and make any necessary adjustments to the platform before its wider deployment. This approach helps to ensure that the ML platform is fit-for-purpose and meets the needs of the organization, which can ultimately lead to faster adoption and more successful outcomes from the platform.

- **Invest in an adoption program:** Investing in an adoption program is crucial to the success of an ML platform. Building the platform is only half the battle, organizations also need to ensure that their users are aware of the platform and are equipped with the skills and knowledge to use it effectively. This is why it is important to invest in enablement and knowledge-sharing programs that aim to increase awareness and understanding of the platform and the new technologies that are used in it.
 - One way to drive adoption is to provide self-service capabilities, such as self-service provisioning, which allows users to easily access the platform and its resources. This can help to reduce friction and increase user adoption, as users are not required to wait for approval or assistance from IT or other teams.
 - Another effective approach is to form a solution engagement team, whose role is to help different lines of business (LOBs) and users to onboard their workloads onto the platform. This solution team can provide support and guidance to users, helping to ensure a smooth and successful transition to the new platform.

In conclusion, building and scaling an ML platform is a complex and challenging project that requires careful planning and execution. Organizations should invest in standard ML infrastructure and tools, establish dedicated teams to build and manage the infrastructure and tools, and drive their adoption. They should create a clear blueprint for the ML platform, take a phased approach to building it, ensure there is a parallel

data strategy that supports it, and make technology stack decisions based on their needs.

Solving governance scaling challenges

Machine Learning (ML) governance encompasses the policies, processes, and standards that organizations establish to govern the development, deployment, and usage of ML models. Its purpose is to ensure that the deployment of ML models aligns with the organization's objectives, values, and ethical standards. ML governance also manages the risks involved in deploying ML models, such as incorrect predictions, discriminatory outcomes, and privacy breaches. A well-designed governance framework provides peace of mind to both the organization and its stakeholders and increases trust in the organization's AI/ML capabilities. On the contrary, the lack of governance can lead to various problems, such as privacy violations, biased model consequences, model drift, and non-compliance with regulations. Thus, it is essential for organizations to invest in building and maintaining a robust governance framework as they embark on their AI/ML journey. However, many organizations find it challenging to implement an effective ML governance framework that balances responsibility and ethics with the need for innovation and fast-paced deployment. This requires finding a delicate balance between control and flexibility, having the appropriate processes in place to regulate and monitor the deployment of ML models, and considering the organization's specific needs, resources, and goals. It is also essential for organizations to periodically review and update their governance framework as the field of ML continues to change and evolve. Next, let's take a look at some of the best practices in establishing a governance framework:

- 1. Clearly Defined Objectives:** Having clear and well-defined objectives for the ML governance framework is crucial to ensure that it is focused on the right issues and will not impede the pace of innovation. These objectives should be aligned with the overall goals and values of the organization and should reflect the organization's priorities for responsible and ethical deployment of ML. Examples of objectives for

a ML governance framework may include ensuring compliance with regulations and ethical standards, promoting transparency in decision making, and protecting sensitive data.

2. **Clear Responsibility and Accountability:** Defining clear responsibility and accountability is a critical component of a successful ML governance framework. This means that the framework should clearly outline the responsibilities of different teams and individuals involved in the ML deployment process. This will help to ensure that the framework is properly implemented and that any issues or concerns are addressed in a timely manner.
3. **Data Management and Privacy:** A well-designed ML governance framework should address data management and privacy concerns to ensure that personal data is protected and used responsibly. This is particularly important in industries such as healthcare, finance, and retail, where large amounts of sensitive personal data are collected and used.
4. The framework should include policies and procedures for data collection, storage, and use, as well as security measures to prevent unauthorized access to sensitive data. This may include data privacy policies, data protection regulations, and data management protocols that are designed to ensure that data is collected, stored, and used in a responsible and ethical manner, and in compliance with various regulations, such as General Data Protection Regulation (GDPR) or California Privacy Rights Act (CPRA).
5. **Governance automation:** Governance automation refers to the use of technology and automated processes to support and manage the ML governance framework. Automating certain aspects of the framework can significantly reduce the overhead and manual operations involved in managing and monitoring the deployment of ML models.
6. For example, automating data collection processes can help to ensure that data is collected accurately and consistently, without the need for manual data entry or verification. Automated evidence validation and compliance analysis can also help to streamline the compliance review process, reducing the time and effort required to manually review and validate evidence.

7. Additionally, automation can be used to support on-going monitoring and reporting of the deployment of ML models. For example, automated monitoring and reporting systems can be used to track and report on the performance and impact of the models, and to identify any potential risks or issues that may arise

Establishing a governance framework for machine learning is essential for ensuring responsible and ethical deployment of ML models. By following these guidelines and best practices, organizations can ensure that their ML models are deployed in a responsible and ethical manner that aligns with their overall goals and values. This will not only help to prevent potential risks and issues, but also promote transparency and trust with customers, stakeholders, and regulators.

Solving user onboarding and business integration challenges

Simply having a lot of ML models does not guarantee successful business integration. Integrating ML models into the business requires a significant amount of effort and investment, sometimes even more than the initial efforts of identifying ML use cases and implementing the ML models and infrastructure. This is because integrating ML models into the business process requires changes to the way the business operates, as well as investment in communication, training, and onboarding. One key challenge when integrating ML models into the business is ensuring that everyone in the organization understands the changes and is on the same page.

Communication is critical, and it is important to keep all stakeholders informed and engaged throughout the process. This includes not only the technical teams responsible for building the models, but also business stakeholders who will be affected by the changes. Another challenge is redesigning business workflows to accommodate the new ML capabilities. This involves identifying where ML can be integrated most effectively, rethinking existing business processes, and identifying new workflows that take advantage of the new capabilities. For example, if you have developed ML models to identify customers with high cross-sell and upsell potentials, you need to implement the required business process changes to act on

these insights such as building outbound engagement teams or mechanisms to reach out to these customers. To enhance the effectiveness of AI/ML integration, organizations must not only implement the ML solution but also take actions and make decisions based on the insights provided by these systems. For instance, if an organization has implemented a credit decision system that automatically scores users' eligibility for loans, but continues to rely on humans to make the final decision for the majority of cases, the value of the ML system will be diminished. This undermines the potential for sustained adoption of ML across the organization. Therefore, it is crucial for organizations to trust and utilize the insights provided by the ML solutions to fully realize the benefits of the technology. Finally, investing in training and onboarding is essential to ensure that end users and teams are able to ramp up on the new ML capabilities and workflows. This includes not only technical training for those responsible for using and maintaining the models, but also training for business stakeholders who may not have technical expertise but will be using the models in their day-to-day work.

Summary

In this Chapter, we explored the different phases of ML adoption and AI/ML capabilities. You were introduced to the assessment of ML adoption maturity through a set of questions aimed at identifying key areas for developing AI/ML maturity. We also discussed the best practices in establishing an AI/ML vision, initiating an AI/ML initiative, and scaling your AI/ML adoption across different ML use cases, ML infrastructure, and ML governance.