Chapter 1

AWS AI ML Stack

THE AWS CERTIFIED MACHINE LEARNING (ML) SPECIALTY EXAM OBJECTIVES COVERED IN THIS CHAPTER INCLUDE BUT ARE NOT LIMITED TO THE FOLLOWING:

- ✓ **Domain 3.0: Modeling**
  - 3.1 Frame business problems as machine learning problems
  - 3.2 Select the appropriate model(s) for a given machine learning problem
    - Sample ML architectures for common business workflows such as video analysis, text mining, and others
    - Details about some common algorithms used in solving complex problems involving unstructured data like text and image
- ✓ **Domain 4.0: Machine Learning Implementation and Operations**
  - 4.2 Recommend and implement the appropriate machine learning services and features for a given problem
    - Details about algorithms for different ML use cases
    - Details about when to use the proper AWS AI/ML Service

In this chapter, you will learn about different AWS Services for Machine Learning, starting with the artificial intelligence (AI) services for common machine learning (ML) tasks such as image and video analysis, natural language processing, text-to-speech conversion or vice versa, or building recommendation systems or time-series forecasting into your applications. These services make it easy for you to build ML-powered applications without machine learning experience. You will then learn about Amazon SageMaker, which is a fully managed service for data scientists and machine learning developers to build, train, and deploy ML models in the AWS cloud for various business applications. For reference, the exam guide can be found at `https://d1.awsstatic.com/training-and-certification/docs-ml/AWS-Certified-Machine-Learning-Specialty_Exam-Guide.pdf`.

Amazon Rekognition

Amazon Rekognition is an AI service that makes it easy for users to implement image or video analysis workflows into their applications. Amazon Rekognition aims to leverage Amazon's vast experience in using deep learning for various image-based workloads such as image classification, object detection, detection of text in image, facial recognition, sentiment, and most recently, public safety.

Although there is a vast amount of deep learning research behind developing models for image analytics, training these deep learning models is often computationally expensive and can take several cycles of data scientist or developer time. That's where Amazon Rekognition comes in. With Amazon Rekognition, developers can simply leverage pretrained models or train custom machine learning models without having to worry about writing the algorithm code, or about setting up or managing the infrastructure to train and deploy a deep learning model. More importantly, you don't require any prior machine learning or deep learning knowledge to use this service.

Before diving into Amazon Rekognition, let's quickly grasp the lay of the land on the subject of images and videos in deep learning. Image recognition typically relies on convolutional neural network (CNN) architectures. CNNs are deep learning algorithms consisting of alternating convolutional layers, which apply various filters on the input data to capture different information at different scales, followed by pooling layers, which reduce the number of parameters in the network and also the spatial size of the representation. The initial layers capture low-level features like edges and curves, whereas latter layers build up to higher-level

ones to eventually identify the object. There are many popular architectures for CNNs such as ResNet or Inception V4, but it is important to understand the basic concept.

It is also useful to understand the concept of *transfer learning*. Transfer learning refers to taking a model that was pretrained on one dataset, freezing the initial layers, and letting it relearn the last few layers of the model on a different dataset. The benefits of this are that:

- It is computationally less expensive than training a full neural network from scratch.
- When you don't have a lot of data or data labeling is expensive, using a pretrained model can provide better model performance than training a model from scratch.

Both Inception V4 and ResNet models are popular algorithms for transfer learning in the image classification space. Transfer learning can be used in many deep learning applications—not just image or video data use cases, but also in natural language processing (NLP).

For object detection, the fundamental architecture is similar, but instead of detecting objects such as a cat versus a dog (fixed label), the model aims to detect a bounding box encapsulating the object of interest. Common algorithms used include single-shot detector (SSD), R-CNN or Faster R-CNN, and YOLO v4.

Finally, semantic segmentation actually segments the object of interest in an image by classifying whether or not an object belongs in a given pixel. An example is detecting a tumor in a human tissue. In order to be useful for doctors, it is not just sufficient to draw a bounding box, but you also need to accurately isolate the tumor from healthy tissue.

You can use Amazon Rekognition with the following key use cases:

Image Labeling This refers to labeling whether an image consists of certain objects (popular objects in nature), events (party, graduation, etc.), concepts (landscape, nature, evening), or activities.

Custom Image Labeling Imagine that you are a manufacturer and you need to detect whether or not parts on your assembly line are defective. Since your parts do not correspond to common objects found in nature, you may need to train a custom model. We will discuss this in more detail later, but Amazon Rekognition allows you to train a custom model for use cases of this kind.

Face Detection and Search Amazon Rekognition can not only detect faces in images but also search for faces from an existing collection. Imagine you are a company that wants to implement face detection for your employees to access your corporate buildings. You can store pictures of your employees in a collection, and call Amazon Rekognition APIs to recognize employees from that collection.

People Paths Amazon Rekognition can track the movement of people in a video. For example, you may want to track the movement of players on a field during a game for postprocessing, stats, and analytics for fans.

Text Detection Amazon Rekognition can detect text in images and convert it to machine-readable text that you can use for downstream actions.

Celebrity Detection Amazon Rekognition recognizes celebrities from images and stored videos.

Personal Protective Equipment (PPE) Amazon Rekognition can now detect PPE on persons in an image.

- Look out for key phrases like "without any prior machine learning/deep learning knowledge" or "cost effective" or any of the use cases just described to think of Amazon Rekognition as the solution.
- If the question contains a phrase like "custom model," unless it has to do with image labeling, usually Amazon Rekognition is not the answer.

Image and Video Operations

Amazon Rekognition operates on both static images and stored videos. Image operations are synchronous whereas video operations are asynchronous. This means that when you ask Amazon Rekognition to process a video, once the job is completed, Amazon Rekognition will notify you using Amazon SNS by publishing to an SNS topic. You then have to call a `Get*` API to access the outputs. For synchronous API calls, you will get the answer right away.

Amazon Rekognition does not support video for all operations; for example, the PPE detection APIs only support images. Likewise, the people pathing use case is only available for video and not for images.

Let's take a concrete use case example to illustrate this further, such as detecting objects in an image or video.

For object detection in an image, you simply need to pass in the location of the image (in JPEG or PNG) in Amazon S3 or a byte-encoded image input. If you are using `boto3`, which is Amazon's Python SDK, then you need to make the following API call, reproduced from the AWS developer guide at `https://docs.aws.amazon.com/rekognition/latest/dg/labels-detect-labels-image.html`:

```
import boto3
def detect_labels(photo, bucket):
    client=boto3.client('rekognition')
    response =
client.detect_labels(Image={'S3Object':{'Bucket':bucket,'Name':photo}},MaxL
abels=10)
```

The sample output may look like this:

```
{

    {
    "Labels": [
        {
```

```
        "Name": "Vehicle",
        "Confidence": 99.15271759033203,
        "Instances": [],
        "Parents": [
            {
                "Name": "Transportation"
            }
        ]
    },
    {
        "Name": "Transportation",
        "Confidence": 99.15271759033203,
        "Instances": [],
        "Parents": []
    },
    {
        "Name": "Automobile",
        "Confidence": 99.15271759033203,
        "Instances": [],
        "Parents": [
            {
                "Name": "Vehicle"
            },
            {
                "Name": "Transportation"
            }
        ]
    },
    {
        "Name": "Car",
        "Confidence": 99.15271759033203,
        "Instances": [
            {
                "BoundingBox": {
                    "Width": 0.10616336017847061,
                    "Height": 0.18528179824352264,
                    "Left": 0.0037978808395564556,
                    "Top": 0.5039216876029968
                },
                "Confidence": 99.15271759033203
            },
```

```
                {
                    "BoundingBox": {
                        "Width": 0.2429988533258438,
                        "Height": 0.21577216684818268,
                        "Left": 0.7309805154800415,
                        "Top": 0.5251884460449219
                    },
                    "Confidence": 99.1286392211914
                },
            ],
            "Parents": [
                {
                    "Name": "Vehicle"
                },
                {
                    "Name": "Transportation"
                }
            ]
        },
    "LabelModelVersion": "2.0"
}

}
```


By specifying `MaxLabels`, you can limit how many responses you receive and Amazon Rekognition will synchronously return a response showing the bounding boxes and confidence scores of the various objects detected in the image, as shown in the previous example. The confidence score can be used for downstream actions.

By contrast, for a video job, you cannot pass in bytes but must pass in the location of a video stored in Amazon S3. The API is `StartLabelDetection` and you also need to pass in an SNS topic for Amazon Rekognition to push a notification to, once it completes the video labeling task. You can then call a `GetLabelDetection` API to access the outputs.

The quality of Rekognition's output strongly depends on the quality of your image. Refer to the best practices documentation (`https://docs.aws.amazon.com/rekognition/latest/dg/best-practices.html`) for more on this subject. In particular, for object detection, the object must be at least 5 percent in size of the shorter pixel dimension.

A key benefit of Amazon Rekognition Video is that you can work with streaming videos. Amazon Rekognition can ingest streaming videos directly from Amazon Kinesis Video streams, process the videos, and publish the outputs to Amazon Kinesis Data Streams for stream processing.

If you are looking to build a scalable image or video analytics workflow, consider using tools like the AWS Lambda function to make Amazon Rekognition API calls in a serverless manner. You may also consider using Amazon SQS to queue your incoming data to prevent throttling of Amazon Rekognition APIs. See a detailed architecture pattern here: `https://github.com/aws-samples/amazon-rekognition-large-scale-processing`.

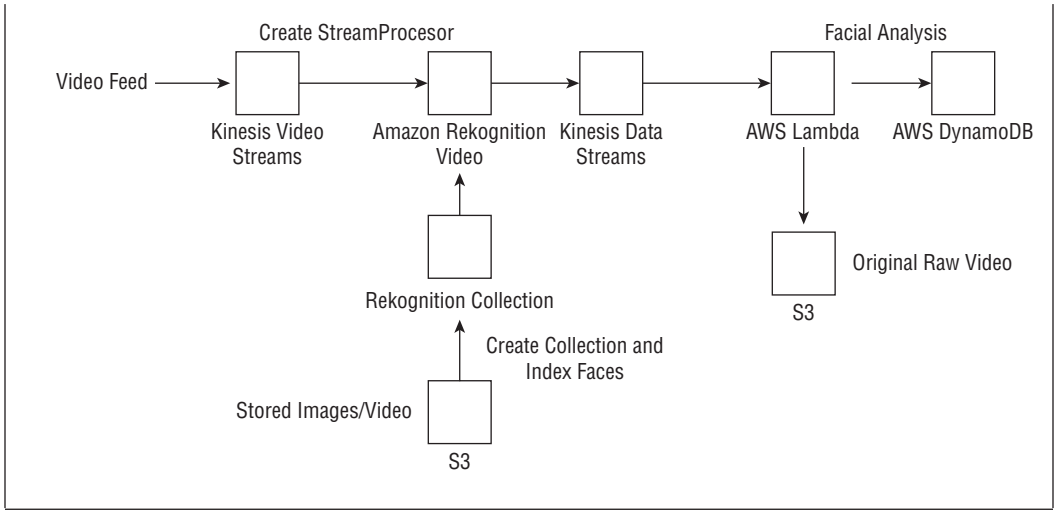
Real World Scenario

Facial Recognition in Video

Imagine you are a solutions architect at a media company who wants to ingest streaming video content as it is aired and detect faces of known people in the videos. However, your executives are concerned that building, training, and maintaining these ML models could be time intensive and challenging given the deep learning expertise required. They want you to design an architecture that can address this use case but also keep costs low.

The key to a scenario like this is to understand that the executives are concerned about the lack of deep learning expertise in their organization being a barrier to developing this use case. Amazon Rekognition Video is ideally suited for this.

First, create a collection of faces using Rekognition Image or Video by detecting faces from an existing database of images or stored video. Once your collection is created, consider using a tool like Kinesis Video Streams to first ingest the video stream and a Kinesis Data Stream as the output data stream. Rekognition Video can then process the incoming video stream using the `CreateStreamProcessor` API, passing the Kinesis Video stream as input. The outputs of the analysis will be published to Kinesis Data Streams. From Kinesis Data Streams you can use AWS Lambda as a consumer to publish the outputs to S3 or to a key-value store such as Amazon DynamoDB. The following graphic illustrates the high-level architectural flow.



- While the machine learning specialty will generally not test your ability to memorize service limits, know that to avoid throttling APIs, you can use SQS to queue jobs.
- Similarly, make sure to read the Rekognition FAQs: `https://aws.amazon.com/rekognition/faqs`.

Amazon Textract

Amazon Textract is an AI service that allows you to quickly extract intelligence from documents such as financial reports, medical records, tax forms, and university application forms beyond simple optical character recognition (OCR). With Textract, you don't have to build deep learning computer vision models to extract text, forms, or tables from PDF documents; Amazon Textract will do that for you, so you can focus on using the extracted information for downstream business tasks.

More importantly, Amazon Textract allows you to quickly build automated document processing workflows, which are largely manual today. Most large organizations such as financial institutions have huge amounts of unstructured and semi-structured text data stored in documents as free-form text, tables, or forms. Companies are only now beginning to realize the power of using machine learning to process these documents and derive insights from them.

- Note that Textract is used for extracting forms, tables, and text from PDFs or images. It does not do document classification, sentiment analysis, or entity recognition on those documents themselves. That is done by a different service called Amazon Comprehend, which we will cover later in this chapter.
- For the exam, know the differences between Textract and Comprehend and also how they can work together.

Common use cases for Amazon Textract include the following:

- Creating a search index by storing the outputs of Textract document analysis in a key-value store like DynamoDB.
- Mining text from documents for natural language processing (NLP): Textract can extract words, lines, and tables that you can subsequently use in NLP-based workflows.
- Automating data capture from forms: Textract can extract information from structured documents such as tax forms or application forms.
- Cost effective: As with most AWS services, you pay for what you use, or what documents you analyze in Textract's case.

Sync and Async APIs

Documents can come in many different sizes, varying lengths, scanned images in PNG or JPEG format, or multipage PDFs.

For the synchronous APIs, you have the option of passing a document to Textract for processing either as a byte array or as an Amazon S3 object. You can use a synchronous API such as `DetectDocumentText` or `AnalyzeDocument` to return a JSON output containing the detected or analyzed text. The Analyze API also recognizes the hierarchy in a document such as form data, tables, and lines and words of text. The Detect API only detects text.

Although documents are generally considered unstructured data, there is often a hierarchy in document structure. For example, consider a form such as an application that contains a Name field, with the name John Smith. Now, if the service simply returned the outputs as Name, John Smith, that would not be very useful to someone trying to parse the document downstream.

Amazon Textract instead returns the text as a key-value pair, allowing the user to seamlessly ingest these outputs into a key-value database store that you may use to query later. Similarly, tables and table data are returned as Block and Cell objects, respectively, providing the bounding box information about the table location in the document, followed by information about underlying cells in the table.

For documents in PDF form or documents that are larger than a single page, use the async APIs `StartDocumentAnalysis` and `StartDocumentTextDetection`. Since detecting text in large documents can take some time, Amazon Textract will process your documents behind the scenes and publish the Completion status to an SNS topic. A subscriber

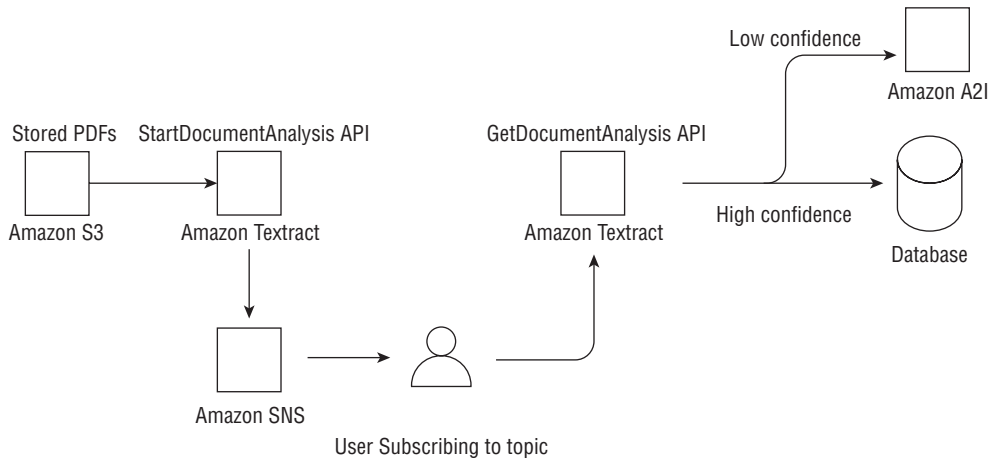
to this topic will be subsequently notified that the job is complete and can view the outputs by calling the `GetDocumentAnalysis` or `GetDocumentTextDetection` API. The outputs of the job can then be stored in a DynamoDB table, an Amazon S3 bucket, or another data store.

Repeatedly calling Textract APIs can result in a throttling exception called `ProvisionedThroughputExceededException` if the transactions per second (TPS) exceed the maximally allowed value. In that case, specify an automatic retry using the Config utility with the AWS SDK. Amazon Textract will automatically retry jobs a certain number of times before failing. Generally, you can set this value to 5. For more information on the AWS service limits for Textract, refer to the documentation here: `https://docs.aws.amazon.com/general/latest/gr/aws_service_limits.html#limits_textract`.

Before concluding our discussion of Textract, let's consider a use case where a medical company wants to extract text from patient forms for downstream processing such as improving the overall patient experience using machine learning. The company has millions of PDF documents currently stored in Amazon S3. Since this is a medical company, protecting patient data is a paramount concern and Health Insurance Portability and Accountability Act (HIPAA) eligibility is a requirement for any cloud service. Finally, the company executives don't trust entirely ML-based solutions and want humans to review some of the ML outputs. The company has reached out to AWS to see if they can help develop a cost-effective solution for them.

First, Amazon Textract can be a potential solution here since it is HIPAA eligible and has async APIs to extract text from large numbers of PDF documents with a pay-as-you-go pricing model. Furthermore, Amazon has a service called Amazon Augmented AI (Amazon A2I) that can directly integrate with the Textract document analysis API to send documents for human review based on a particular threshold condition such as low confidence on the detected text (Figure 1.1). We will discuss A2I in more detail later in this chapter.

FIGURE 1.1 Document analysis with human review flow



Amazon Transcribe

Imagine that you work for a global hotel chain that has a large volume of incoming customer call voice data that needs both real-time/streaming and batch transcription. Due to your global brand, the calls take place in multiple languages such as Arabic, Mandarin, English, French, Spanish, Japanese, Korean, German, and some others. Your manager has asked you to find a solution that leverages machine learning but is both scalable and cost-effective and does not require significant setup.

Traditionally, human speech recordings were stored simply as waveforms. By matching these waveforms with those of common sounds, a computer could transcribe speech to text.

The modern approach for this is called Automatic Speech Recognition (ASR) and powers technologies such as Amazon Alexa. These technologies use neural networks to take an input audio sequence as input and produce an output sequence consisting of text, using models known as sequence-to-sequence models. However, building these models accurately requires massive amounts of data that not all companies have.

Enter Amazon Transcribe. Amazon Transcribe leverages the same technologies powering Amazon Alexa but is available as a transcription service that allows you to transcribe your voice data without any prior machine learning knowledge. Let's dive into some of the features.

Transcribe Features

Here is a list of some the features provided by Transcribe:

Stream and Batch Mode Transcribe supports both streaming and batch transcription modes. For streaming transcription, audio is directly streamed via the HTTP/2 protocol. Transcribe provides a streaming client, or you have the option to use your own client with the WebSocket protocol. For existing audio files stored in S3, you can run a batch transcription job using the `StartTranscriptionJob` API.

Multiple Language Support You can transcribe speech in an ever-growing list of languages. Although the exam will not test you on specific languages, it is helpful to know that many of the popular ones are supported.

Multiple Language Transcription Transcribe does not require your audio to contain a single language. If you know whether your audio will include additional languages, you can pass the language code as part of your API call by specifying `LanguageOptions`. See `https://docs.aws.amazon.com/transcribe/latest/dg/transcribe-lang-id.html` for the full list of supported languages.

Job Queuing Transcribe provides options for you to send jobs to a queue so that you don't have to build custom logic to prevent API throttling.

Custom Vocabulary and Filtering Transcribe provides a custom vocabulary that includes a list of words you want Transcribe to recognize, such as proper nouns or

domain-specific language. Additionally, you can filter unwanted words such as any profane or offensive language.

Automatic Content Redaction If your audio includes personally identifiable information (PII), Transcribe gives you the option to redact it from the transcribed output or provide both unredacted and redacted scripts. This information may include entities such as account numbers, credit card numbers, names, U.S. phone numbers, and U.S. Social Security numbers. Note that this feature is only available in English.

Language Identification Transcribe will identify the dominant language in your transcription.

Speaker Identification This feature allows you to identify different speakers in a transcription for English audio.

Going back to the hotel chain scenario, you have researched the aforementioned options and have secured buy-in from your management to develop a proof-of-concept (POC) with Amazon Transcribe to transcribe audio output. Since customer data security is extremely important, your manager has asked whether the service can be used to remove PII to be stored separately. On researching, you learn that you can use the PII detection capability within Transcribe to produce an output transcription with the original and the redacted versions of the transcript.

Having started your POC, you notice that although Transcribe is doing well overall, there are some domain-specific situations where the transcriptions are not as accurate. Furthermore, this domain has highly specialized vocabulary that exceeds the 50 KB limit for custom vocabulary. Your AWS Solutions Architect recommends that you build a custom model to improve the transcription accuracy.

Amazon Transcribe now lets you build a custom language model simply by providing your text as an input. Transcribe will build the model, and then you can use this model instead for your domain-specific transcriptions. Furthermore, you can provide a training dataset consisting of your text, and a test dataset containing a sample of audio transcripts.

Know the difference between a custom model with Amazon Comprehend (which we will cover later) versus Amazon Transcribe. The Transcribe custom language model is only applicable for audio transcription use cases. Comprehend Custom is for document classification and entity extraction, among other uses.

Transcribe Medical

The medical domain is highly specialized with a vast vocabulary. As a result, most common deep learning transcription models do not work well in this field, unless they have been

specifically trained on medical data. Amazon Transcribe Medical is an ASR service that enables you to transcribe medical audio such as physician dictation, patient-to-physician conversations, and telemedicine. Transcribe Medical is available both in streaming and batch mode (only for Primary Care) and allows you to build custom vocabularies and redact personal health information (PHI) from your streaming transcriptions. For more information on Transcribe Medical, see the following document: `https://docs.aws.amazon.com/transcribe/latest/dg/transcribe-medical.html`.

- Remember, not all AI services have a medical specialty. Among the ones that do are Comprehend and Transcribe. You may get a question on the test that requires custom transcription but the answers may include non-existent services like Translate Medical or Textract Medical. Those are immediately incorrect, allowing you to narrow down your answers.
- Know that the batch transcriptions for Transcribe Medical is only available with Primary Care use cases. For medical use cases in cardiology, neurology, oncology, urology, and radiology, only streaming transcriptions are supported.

Amazon Translate

Once again, you work for a global hotel chain that operates in several different countries all over the world and you want to aggregate and collect customer service chat data in order to improve customer experience. The only problem is, the calls happen in different languages, and you know that building neural-network-based translation models in different languages is hard and expensive. You are also concerned with the quality of existing third-party translation tools and prefer a pay-as-you-go pricing model without any vendor lock-in.

Enter Amazon Translate, a text translation service that uses advanced state-of-the-art deep learning to provide high-quality translations to customers without any deep learning experience, with a pay-for-what-you-use pricing structure. As the name suggests, the main use case for Amazon Translate is to translate text from various languages such as Spanish, Arabic, Hindi, Greek, German, French, Chinese, and many others.

As always, know what a service is and what it is not. Amazon Translate does not translate directly from voice, such as calls. You can, however, chain Amazon Transcribe with Amazon Translate to make this flow work.

Having told your boss about Amazon Translate, you have secured their buy-in to experiment with this new tool. Next, you meet with the teams that have been storing all the chat data, and they have posed two problems:

- There are a few applications they have built using AWS Lambda that contain small amounts of text to translate, but the bulk of the incoming chat data is stored in S3 buckets and requires asynchronous processing.
- For certain countries where your chain operates, you have some custom terminology, corresponding to your hotel names, that need to be accounted for during translation instead of being translated into the local language.

Amazon Translate Features

Here is a list of features offered by Amazon Translate:

- Sync and Async APIs: Amazon Translate allows customers to asynchronously process large numbers of documents using a batch job (in 5 GB batches) with the `StartTextTranslationJob` API. This API is helpful when the individual documents comprising the collection are small, such as social media postings or user reviews. For smaller documents you can run a translation operation in real time using the `TranslateText` API. Please refer to the Developer Guide for more details on this synchronous API: `https://docs.aws.amazon.com/translate/latest/dg/sync.html`.

  To run a batch job, you need to provide the path to your chat data in Amazon S3, an output location for the translated chats, and the source and target language for translating the chats.
- Custom terminology and parallel data: Furthermore, you can customize the outputs of your translation by supplying a custom terminology as a CSV file, which provides the custom terms in the source language and the target terminology that you want. You can also pass in parallel data that shows the service how you want segments of text to be translated. Note that not all languages are suitable for custom terminologies; you can find the list at the following site: `https://docs.aws.amazon.com/translate/latest/dg/permissible-language-pairs.html`.

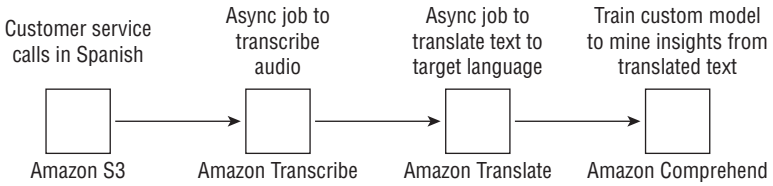
Having reviewed these features, your boss is now super happy as it appears that Amazon Translate will fully meet her requirements. Her final concern is whether Amazon will use any client data to improve its own machine learning algorithms.

Remember that Translate does not let you build your own custom translation models; it uses models trained by Amazon. As a result, Amazon may use customer data to improve the quality of its algorithms and models. To opt out of this for Amazon Translate and other AI services, refer to the following documentation: `https://docs.aws.amazon.com/organizations/latest/userguide/orgs_manage_policies_ai-opt-out.html`.

- In addition to individual features of a service, understand how services play together to form a coherent end-to-end architecture.
- For example, a text-related service like Translate can be combined with Amazon Transcribe to transcribe calls before translation, Amazon Polly to convert translated text to speech or audio after translation, or even Amazon Comprehend to extract sentiment from translated text.
- You might get a question or two on the exam on combining different services together.
- You don't need to know all the supported languages for Translate, but it is helpful to know the popular ones.

Figure 1.2 shows an entirely batch-based flow to process stored customer service calls from one language to English and then run a custom entity or label detection model using Amazon Comprehend.

FIGURE 1.2 Flow showing how to translate customer service calls followed by entity or label detection.



Amazon Polly

Amazon Transcribe converts speech to text, whereas Amazon Polly does the opposite—converts text to speech (TTS). How it works is that a user provides some text either as plain text or using a syntax called Speech Synthesis Markup Language (SSML). Polly reads this syntax and generates a lifelike voice using one of the prebuilt voices in different languages such as English, Arabic, Danish, Mandarin, Spanish, or Russian.

The exam will not test on different languages supported by Polly, but it will often contrast Polly with Transcribe, so it is important to remember the difference.

So how does speech synthesis work? Standard speech synthesis TTS works by stringing together basic speech units called *phonemes* into a natural-sounding synthesized speech.

More recently, artificial intelligence techniques such as deep learning have been applied to this problem to produce neural TTS (NTTS). A neural TTS model consists of what is called a sequence-to-sequence model, which takes an input sequence (in this case a line of text) and generates an output sequence (a spectrogram consisting of frequency bands that mimic the acoustic features used by the brain while processing speech). The output of this model then passes to a neural vocoder. A *vocoder* is the voice equivalent of the phoneme that converts the spectrogram to speech.

So when would you use TTS versus NTTS?

Training NTTS models to be accurate requires a lot of data, so naturally they are not available in all languages yet. So first, you want to check whether NTTS is available for the language you are interested in translating into. If you are looking for Newscaster speaking style, you have to use NTTS. Generally, if you have the option, NTTS generates more superior speech compared to standard TTS.

What if you want to have control of how the speech output is produced, such as slowing down or increasing the speed or controlling pitch or speaking style? This is where SSML tags come in. Think of SSML as a language similar to HTML that allows you to use tags to define how particular objects will be rendered. For more information on SSML tags and supported tags with Amazon Polly, refer to the documentation here: `https://docs.aws.amazon.com/polly/latest/dg/supportedtags.html`.

SSML is analogous to HTML for voice. By specifying tags such as the `<prosody>` tag, you can control the pitch, volume, and rate of speech. Similarly, if you want a word to be spelled out in a different language, use the `<lang>` tag. An example from the AWS documentation (`https://docs.aws.amazon.com/polly/latest/dg/supportedtags.html`) follows:

```
<speak>
   Sometimes it can be useful to <prosody volume="loud"> increase the volume
   for a specific speech.</prosody>
</speak>
```

By specifying **`volume="loud"`**, you can make the volume louder based on the applied tag. To make the volume softer, you would use **`volume="soft"`**.

Optionally, Amazon Polly also returns the metadata associated with the generated speech in the form of *speech marks*. These can tell you at what timestamp a particular audio started, the type of speech mark (sentence, word, SSML, etc.), the start and end offsets, which of the available Polly voice IDs is speaking, and so on. For more information on speech marks, refer to the documentation here: `https://docs.aws.amazon.com/polly/latest/dg/SynthesizeSpeechMarksSample.html`.

- A typical exam question might be as follows: "A mobile app company wants to develop a chatbot with a voice output to respond to the user's query. What service would you use to generate the voice portion of the output?" Be careful of such questions because the "chatbot" may immediately make you think of Amazon Lex (a service we will cover next). But

in reality, the question is asking about how to convert text to speech and that is Amazon Polly.

- Alternatively, the question may ask you about a service that can generate speech from text that can be stored in MP3 or OGG formats that can be played later, such as in an IoT (Internet ofThings) device. Amazon Polly is again the answer.

Amazon Lex

In many applications today, customers interact with the application using chatbots, and the application understands the user's intent and provides responses. Common examples include using e-commerce applications that provide customer support, booking doctor's appointments, and making hotel or airline reservations.

Amazon Lex is an AWS service, powered by natural language understanding (NLU) and automatic speech recognition (ASR), that allows users to build and deploy conversational interfaces for their applications. With Amazon Lex, you can build a tailored and personalized experience for your customers to engage with your platform without any deep learning expertise.

- As with all these AI services, the key is that these services allow you to build applications without expert deep learning knowledge.
- Also, do not confuse Lex with Polly. Although Polly also employs ASR, Polly converts text to speech. Amazon Lex is used to build and deploy the chat interface and responds to user intents. We explain intents in more detail in the following section.

The core steps behind developing an application are as follows:

1. Create a bot in a desired language or languages. You want to configure the bot to understand the user's goal and engage in conversation to fulfill the user's intent.
2. Test the bot.
3. Publish the bot as a version so that you can roll back to a prior version if needed.
4. Deploy the bot to an end application such as Facebook Messenger, Slack, or Twilio.

To execute these steps, you first need to understand some key concepts.

Lex Concepts

Let's start with a bot. A *bot* is the entity that will perform the desired action. For an e-commerce application, this action could be fulfilling a customer order, connecting the customer to a human representative, or providing the customer with information by performing a lookup in a database.

With Amazon Lex, the backend actions can be performed by using an AWS Lambda function. For example, if your bot is designed to make an appointment at your local doctor's office, you could have the Lambda function write to Amazon Relational Database Service (RDS) or Amazon Aurora or even a DynamoDB Appointments table. Likewise, if a customer wanted a reminder of their appointment, the bot could call a Lambda function to read from the table and return the appointment details.

On the front end, the bot needs to understand the user's intent. This requires that the user type in one of the supported languages. A full list of them can be found here: `https://docs.aws.amazon.com/lex/latest/dg/how-it-works-language.html`. You will not be required to memorize this list.

An intent is the action you want the bot to perform. An utterance is what the user actually asks for. For example, if you are ordering a pizza, the utterance could be "I would like a pizza" and the intent is "OrderPizza."

This is where the NLU and deep learning comes in. Amazon Lex needs to take a few sample intents provided by the user to build a model that can generalize to the myriad of ways in which a user can ask for something.

Let's think about how this would work prior to deep learning. A developer would either need to first provide an ontology of all the ways in which a particular request can be made and then build a set of rules to react to the intents in certain ways or prompt the user to ask a question differently if the user's utterance was not part of the bot's vocabulary.

With deep learning, the model can generalize to new utterances without requiring a predefined set of rules. You can provide a few sample utterances such as "I want a pizza," "Can I order a pizza," and "I want to get a pizza," and Lex will build a model to generalize to other intents.

A *slot* is a set of parameters that define the user's ask and a *slot type* is a characterization of that slot. The slot can be used to make the chatbot conversational. For example, if a user wants to order a pizza, the slot type can be size, and the slots can be small, medium, and large. The bot can ask the user to provide a size, or a list of toppings. Once the required slots are provided, the chatbot can connect with the backend Lambda function, which will then call an API to place the order or write the order to an Orders table. To simplify building the chatbot application, Lex provides a set of built-in slots and slot types for common items like Date, Name, Number, Email, Address, and Time.

If you are building a bot using Lex, and if your bot is not performing well, try increasing the number of sample utterances. The more examples you provide, the better the model will be able to generalize to unseen utterances.

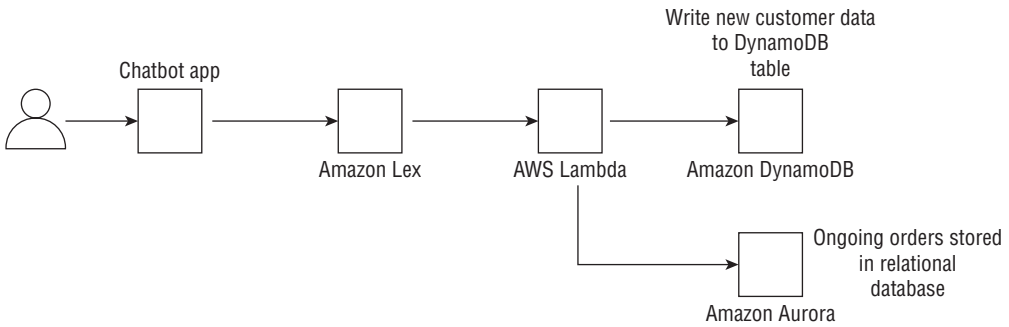
What if the bot does not understand the user? Amazon Lex automatically includes a fallback intent, so you don't have to build one yourself. This intent is invoked when the bot does not recognize the user after a configured number of retries, or an intent does not recognize the user's input as a slot value or a response to a confirmation prompt.

Generally, if the fallback intent is invoked, you can have your Lambda function perform some predefined action such as connecting to a human representative. In this way, the dialogue flow feels conversational and natural.

Alternatively, if your bot cannot understand a user's request, it can trigger a document search to provide an answer. To do this, you can use the `KendraSearchIntent` API, which leverages Amazon Kendra behind the scenes. We will dive into Kendra next.

Figure 1.3 shows how you can integrate Lex on the back end with different AWS services using Lambda functions. In this case, `AppointmentBot` uses both relational and nonrelational databases to surface relevant information to the end user.

FIGURE 1.3 The `AppointmentBot` can be built using Amazon Lex and backend AWS Services.



- Understand the differences between slots, utterances, and intents. Slots are configuration parameters, utterances are the actual sentences, and intents are the meanings behind them. In the pizza example, slots can be pizza size or individual toppings, the intent is to order a pizza, and utterances may be, "I want olives on my pizza," "I want a small pizza," or "Can I order a pizza."
- Know which external tools Lex can integrate with, namely Facebook Messenger, Slack, and Twilio Short Message Service (SMS).

Amazon Kendra

A global bank contains internal information that needs to be shared among employees. Furthermore, this information is often contained in unstructured textual format across disparate data sources from Microsoft Word documents, Microsoft PowerPoint presentations, PDF documents, and even third-party tools such as Confluence, Salesforce, ServiceNow Microsoft OneDrive, and Microsoft SharePoint.

The CIO of the bank wants to build an application that allows users to query and search this unstructured text and mine it for insights as well as provide users with quick and

relevant responses to their search queries to improve knowledge sharing and user productivity. What AWS service will allow you to build such an application?

At first, you might think this is asking you to extract text from documents and you might think Textract is the correct service, but there are two key points here: (1) The CIO is not asking you to simply extract the text; she is asking you to search text and build an application that provides intelligent answers. (2) The data sources go beyond simple PDFs. Both of these will tell you that Textract is not the solution here but rather Amazon Kendra.

Amazon Kendra allows you to build your own search application using natural language that provides highly relevant responses to user queries as you would get from a human expert within your organization. With Kendra, you can get answers to facts or factoids (such as the height of Mount Everest), descriptive answers to complex questions such as "What is a 10-K form?" or even keyword searches where a user may type "401K match" or "retirement benefits."

Under the hood, Kendra is powered by deep learning algorithms, but the benefit is that this is all abstracted from the end user. The simple reason for this is that building and training highly accurate deep learning algorithms for natural language is expensive and complex, and requires significant corporate investment in scarce machine learning talent. Although not all businesses can afford such an investment, the need to provide a scalable, internal search platform is universal. Recognizing this, Kendra comes in two pricing models: a developer edition and an enterprise edition. Both are pay-as-you-go, but the latter provides higher availability by operating in three availability zones (AZs), allows for more queries, and can ingest more documents and text.

How Kendra Works

So how do you get started with Kendra? The following key concepts are essential:

1. Index: In order to search documents, first you need to index them. An index is an object that is managed by Kendra that carries some metadata about that document, such as when it was created and updated, the version, and custom fields such as date and number that you can modify as a user.
2. Documents: These include the actual documents that Kendra will index. They may include frequently asked questions (FAQs) or purely unstructured documents such as HTML files, PDFs, plain-text or Microsoft Word documents, or Microsoft PowerPoint presentations.
3. Data sources: You may be wondering if you need to manually index documents. The answer is no; you simply provide Kendra with a data source such as a Confluence server, Microsoft SharePoint, Salesforce sites, ServiceNow instances, or an Amazon S3 bucket, and Kendra will index your documents as well as synchronize the data source with the index to keep it relevant and updated. For a full list of the supported data sources for Amazon Kendra, refer to the following document: `https://docs.aws.amazon.com/kendra/latest/dg/hiw-data-source.html`.

Once your index is ready, you can write queries to the index and it will use all the information provided about the documents to return the most relevant items. Additionally, you can filter your search queries by context or categories such as documents authored by a certain person. You can also sort responses based on a custom attribute. The default sort order is specified by the relevance of the response based on your search query according to Kendra.

Often when you search for an item, you may not use the exact name. Examples include if you are searching for Amazon Web Services; you may wish to use "AWS" instead, or Kendra instead of Amazon Kendra. You can provide a list of synonyms to Kendra in a thesaurus file, which may include internal company acronyms or domain-specific common phrases.

Later on, we will cover Amazon Comprehend, which is a general-purpose natural language processing (NLP) tool that allows you to train your own or use pretrained NLP models for sentiment analysis, document classification, and entity recognition. Know the difference between Comprehend and Kendra. Kendra also uses NLP behind the scenes, but it is aimed at document search and question and answering (Q&A) as opposed to a general-purpose tool for NLP.

Recall that in the previous section on Amazon Lex, we covered how you can build a chatbot application on AWS without any ML knowledge. You can now build an end-to-end FAQ chatbot using Lex and Kendra. Lex provides the front end to identify the user intent based on utterances, and it can call Kendra using `KendraSearchIntent` by passing in the intent as the input. Kendra can then search and return the most relevant results that are surfaced by the chatbot.

Amazon Personalize

Personalization has rapidly become a ubiquitous use case across a broad set of verticals such as the following examples:

Financial Services A bank providing its customers with a personalized experience of credit card offers

An E-commerce Platform A company providing its customers with recommendations on purchases or next-best actions and offers based on a customer's search history

A Travel Company A company providing its customers with recommendations on where to travel, activities to do, or places to stay

A Media and Entertainment Business A company recommending what to watch next based on customer preferences and history

Amazon Personalize is a machine learning service that allows businesses to rapidly develop personalized recommendation systems to provide a better customer experience to their end customers.

From a machine learning perspective, all these diverse business problems share some common aspects. They all rely on three forms of data:

User Data This may include data about user's age, location, income, gender, personal preferences, and other demographic information.

Item Data This is data about the actual products a company sells or is trying to recommend.

User-Item Interaction Data This is data about how the set of users have interacted with these items, such as whether they have purchased the items in the past, do they like or dislike these items, or have they provided reviews or ratings for these items.

The goal of any personalization service is to take this data and extract meaningful insights that can lead to a customer purchasing a product or an item or simply spending more time on the digital platform.

Traditionally, this was done in a couple of ways: using only user data, companies would cluster users into similar groups and recommend items others in that cluster had purchased, or they would cluster the items into similar groups and recommend similar items to what a user had purchased or based on user feedback. This is often known as *clustering* and *content-based filtering*, respectively.

More recently, a method called *collaborative filtering* has emerged wherein the user-item interactions data is often used to recommend items. This user-item interaction data is often a very large sparse matrix (proportional to the number of users and number of items). For example, imagine a company like Netflix, with millions of subscribers (users) and hundreds of thousands of movies and shows (items). Collaborative filtering decomposes large sparse matrices into smaller matrices (matrix factorization) to extract hidden or latent vectors for each user and each item. The dot product of these gives the final score, which determines whether or not to recommend an item.

Although this is a highly popular technique, if you don't have a lot of items, matrix factorization can often not work as well. In that case, you can consider other approaches such as supervised learning using models like XGBoost and Factorization machines (which we will cover in depth in more detail in Chapter 8, "Model Training"). Here you predict a probability that a user will purchase an item based on the model and recommend the highest probability items.

One of the drawbacks of collaborative filtering is that it is time invariant; it doesn't take into account a user's purchase or session history. For example, if you were in the market for shoes a month ago, purchased them, and now are interested in a watch, a good recommender should probably not recommend shoes to you.

For this reason, scientists began to use recurrent neural networks (RNNs), which have the ability to retain a user's session history information as part of the model training. If you want to dive deep into the RNN architecture, there are a number of great references and

articles (see, for example, "Session-Based Recommendations with Recurrent Neural Networks," by Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk, available at `https://arxiv.org/pdf/1511.06939.pdf`).

Scientists at Amazon extended this even further to a model called *HRNN-Metadata*, which uses an RNN to store user histories but also has the ability to incorporate user and item metadata (remember the user data we mentioned earlier?) as part of the training. This allows them to solve not only the temporal history problem, but simultaneously the cold start problem, which is the inability of a recommender to recommend products to completely new users (see `https://openreview.net/pdf?id=ByzxsrrkJ4`).

Finally, most recently the use of so-called multiarmed bandits (MABs) has become popular in personalization. The topic of MABs goes beyond the scope of this book, but at a high level, a MAB uses the concept of exploration-exploitation trade-off. The goal is to maximize the total reward after a fixed number of steps. In the exploration phase, the algorithms explore different possible combinations that could maximize the gain, recording the rewards at each step to build up a reward distribution. In the exploitation phase, it selects a known option that is known to increase the overall gain. The trade-off arises when exploration may reduce the gain as compared to the current choice. However, unless you explore, you will not know if there are other options that beat your current choice. For more details on MABs, see the following site: `https://en.wikipedia.org/wiki/Multi-armed_bandit`.

So as you can see, personalization has a long history in machine learning, in particular, at Amazon itself. Although we can by no means do the topic justice, setting the context is important in order to understand the benefits provided by Amazon Personalize.

Amazon Personalize is an AI service that takes this domain knowledge of personalization into a web service that allows customers to build recommendation systems into their applications.

Amazon Personalize employs the concept of recipes, grouped into three types for a given use case. Recipes allow you to build recommender systems without any prior ML knowledge:

User Personalization Recipes These recipes come in three flavors. First, user-personalization uses the user-item interaction data and tests different recommendation scenarios. It is the recommended personalization recipe and built using the exploration-exploitation trade-off we discussed earlier. Second, popularity count recommends the most popular item among all your users and is a good baseline to compare other recipes against. Finally, there are legacy recipes that involve the HRNN and HRNN-meta models we discussed earlier.

Ranking-Based Recipes This recipe also uses an HRNN but it also ranks the recommendations.

Related Item Recipe This is the collaborative filtering algorithm we described earlier.

In addition to recipes, Amazon Personalize recognizes three kinds of datasets: user data, item data, and interaction data, which we described earlier. The user and item datasets are metadata types and only used by certain recipes. For more information on this, refer to the

following document: https://docs.aws.amazon.com/personalize/latest/dg/how-it-works-dataset-schema.html.

When Amazon Personalize uses the HRNN recipe, the interaction data needs to include a timestamp to pass in the history of the interaction. The exam will generally not test you on the specifics of the data, but this is good to know for your own use. The user-personalization recipe requires interaction data.

Creating a personalization model using Amazon Personalize is called a *solution*. Once you upload the data, the first step to creating a solution is to pick a recipe. Amazon Personalize will create a solution version based on the provided data and will automatically split your data into data for training versus testing. Amazon Personalize will use 90 percent of your data for training and the remaining 10 percent for testing. You can also tune the algorithm's hyperparameters or have Personalize do that on your behalf. It will then evaluate the solution version on the evaluation or test datasets but feeding it the oldest 90 percent of the test data as input. It will then evaluate the recommendations on the latest 10 percent of the test data.

The performance of the model is based on evaluation metrics such as Precision at K and Mean Reciprocal Rank at K. It is important to know what these metrics mean to be able to judge the quality of your model:

- Precision at K: Of the K items recommended, how many were actually relevant, divided by K.
- Mean Reciprocal Rank at K: The mean of the reciprocal rank of the first recommendation out of K, where the mean is taken over all queries.

For more details on other metrics used by Amazon Personalize, refer to the following document: https://docs.aws.amazon.com/personalize/latest/dg/working-with-training-metrics.html.

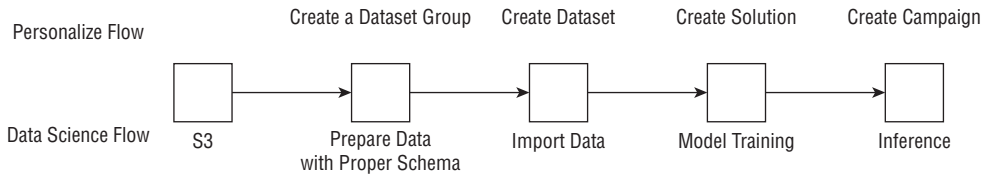
Once your solution version is created, you can create a campaign to score your items in real time or in batch. A *campaign* is used to make recommendations for your users. Personalize provides a SQL-like interface to filter the recommendations based on queries both in real time and in batch use cases. For more information on campaigns, visit https://docs.aws.amazon.com/personalize/latest/dg/campaigns.html.

A common question that you may ask is how often you need to retrain your models as new user-item data comes in. For certain recipes, Personalize allows you to include real-time events data in your recommendations without having to retrain a model each time, by adding the new data to your user history and automatically updating the model with the new data. For more information on making recommendations based on real-time events data, see the following document: https://docs.aws.amazon.com/personalize/latest/dg/recording-events.html.

That said, many customers will retrain their models at a certain fixed cadence (nightly, weekly) depending on the freshness (or relevance) of their recommendations. This is often a business-driven question—in some industries, a customer's preferences may change much more quickly than others. Generally, when building any ML use case, you will want to work with the relevant business stakeholders to answer these questions.

Amazon Personalize uses some unique terminology, and it is useful to map this to the typical data scientist workflow (Figure 1.4).

FIGURE 1.4 The end-to-end flow with Amazon Personalize (text on top) and how it maps to the common machine learning terminology (text on bottom)



- Generally, on the test, if you see a question about recommender systems or real-time personalization, think Amazon Personalize.
- The test may also give you a personalization question and ask you to recommend real-time or batch depending on the use case. Remember that Personalize supports both modalities.
- Later in this chapter, we will cover Amazon SageMaker, which has a built-in algorithm called Factorization Machine. Understand that this is a supervised learning algorithm that works well when you have a small number of items compared to algorithms like HRNN that are ideally suited for large numbers of items (>100). In the test, if the question asks about Personalization on SageMaker, think Factorization Machine.

For the test, it may be important to understand the distinction between Personalize and what is provided with SageMaker. Remember that as an AI service, the key benefit of Personalize is that it handles the heavy lifting of the underlying algorithm code, training the models, deploying the models for both batch and real time with little to no code. Although you can do all of this with SageMaker, you will have to train the models yourself and deploy the model endpoints using code on your own. However, SageMaker gives you the flexibility of trying out any algorithm, whereas Personalize offers a limited set of algorithms.

Amazon Forecast

Like Personalization, forecasting is another problem that affects nearly all industries:

- A shipping company needs to forecast the demand for its ships at various ports to schedule shipping routes accordingly.
- A health care company needs to forecast the demand of medical supplies at different hospitals.

- A major retailer needs to forecast product demand to determine the inventory to keep in their warehouse.
- A cloud computing provider needs to forecast infrastructure and compute demand from their customers to plan capacity for their data centers.
- A customer service agency needs to forecast the volume and duration of calls during various periods in their call center to plan for hiring staff.

Amazon Forecast is an AI service that uses both statistical and deep learning–based algorithms to provide highly accurate forecasts. Similar to personalization, as a major retailer and cloud services provider, forecasting has a long history at Amazon as well, and Amazon Forecast provides that experience to customers.

From a data science perspective, traditional approaches to forecasting were autoregressive in nature, one of the most popular algorithms being the autoregressive integrated moving average (ARIMA). ARIMA is a statistical algorithm where past values are used to make predictions about future values. The ARIMA model predicts the value of a time series at time t, based on the historical value at times $t - 1 \ldots t - p$ plus some error terms. The moving average is the regression error and the autoregressive part is the lagged temporal features. The integrated part is to add a differencing component (taking the difference of the time series value at adjacent points) in order to make the time series stationary.

The key benefit of this model is its simplicity, and there are many packages in different languages (Python, R, etc.) to use ARIMA to forecast a time series. Over the years, the ARIMA approach has been improved upon in many ways—for example, S-ARIMA or seasonal-ARIMA, which includes seasonality, and more recently Prophet, which was introduced by a team at Facebook (`https://research.fb.com/prophet-forecasting-at-scale`).

Unlike ARIMA, which assumes a mathematical form for the time series, Prophet attempts to fit a time series to the data by detecting trends at different intervals such as seasonality; daily, weekly, and yearly trends; and even holiday effects. This has led to Prophet becoming one of the most popular models in use for forecasting today. But remember that this also adds complexity; the ARIMA model is easy to explain as it has a simple, linear mathematical form. More recently Facebook released a neural network version of Prophet, called Neural Prophet (`https://m.facebook.com/FacebookAI/photos/a.360372474139712/1710575115786101/?type=3`).

With the advent of deep learning, in particular the ability of deep learning models to learn sequences, it is natural that these methods also be applied to time series forecasting, which can be viewed as a sequence. Amazon scientists developed an algorithm called Deep-AR (`https://arxiv.org/abs/1704.04110`) that uses a long short-term memory (LSTM)–based model and a probabilistic sampling technique to generate a probabilistic forecast. We will not cover LSTM architectures in detail in this book, but there are many excellent books and courses on machine learning for time series (see `https://www.oreilly.com/library/view/hands-on-machine-learning/9781492032632` and `https://www.manning.com/books/deep-learning-with-python`).

Like Amazon Personalize, Amazon Forecast aims to combine the externally used popular forecasting algorithms and algorithms researched within Amazon into a service that makes it easy for customers to build accurate forecasts without prior machine learning knowledge.

You provide your data in a specific schema, and either let Amazon Forecast choose an algorithm for you or pick one of the many algorithms available within Forecast such as ARIMA, DeepAR+, and Prophet.

Let's cover some of the algorithms available in Forecast (also known as predictors):

- DeepAR+: This is an extension of DeepAR we discussed earlier and trains a single model on many similar time series (>100s). It works by splitting your time series randomly into fixed-length "windows" called context length and aims to predict the future up to a length called the *forecast horizon*. By doing this over many epochs and different time series, DeepAR can learn common patterns across different time series to generate an accurate global model. DeepAR+ treats the context length as a hyperparameter in your model, allowing you to tune them to get better performance. Additionally, DeepAR accepts metadata about the item in the form of related time series or simply item metadata. So if you are forecasting sales, a related time series could be a time series of weather data or foot traffic data to your store. Although DeepAR+ can handle missing values in your data, if your data has many missing values, then your forecasts may suffer because the model is not able to learn useful patterns.
- ETS: ETS, or exponential smoothing, is a statistical algorithm that is useful for datasets with seasonality. It works by computing a weighted average of prior features, but instead of a constant weight, it applies an exponentially decaying function as the weighting parameter. ETS does not accept any metadata or related time series.
- Prophet: Prophet is useful when your time series has strong seasonal variations over many months/years and if you have detailed time series information. It is also useful when your data contains irregularities (such as spikes during holidays) or has missing values.
- CNN-QR: The convolutional neural network quantile regression algorithm also uses deep learning, but this time uses convolutional neural networks (CNN) over recurrent neural nets like DeepAR. It uses a concept called sequence-to-sequence learning, where a model is fed an input sequence and it generates a hidden representation of that sequence. This is called an encoder. That representation is then used to predict the output sequence using another network called the decoder. We don't cover sequence-to-sequence models in great depth here, but it is useful to understand the distinction between this and DeepAR. DeepAR uses LSTMs whereas CNN-QR uses causal convolutional networks. To learn more about CNNs, we refer you to the many courses and textbooks on this subject (see `https://www.oreilly.com/library/view/hands-on-machine-learning/9781492032632/` and `https://www.manning.com/books/deep-learning-with-python`).
- Nonparametric time series (NPTS): Unlike ARIMA or ETS, which use a formal mathematical form to fit the time series, NPTS is nonparametric. It is useful when you have seasonal data or bursty data, or data with a lot of intermittent values. In practice, however, NPTS is often outperformed by Prophet.

If you only have a handful of time series, consider algorithms like ARIMA, ETS, or Prophet. Once you have hundreds of time series, only then consider DeepAR+ or CNN-QR.

When should you use CNN-QR versus DeepAR+?

Both these models accept metadata and related time series inputs. However, CNN-QR does not require the related time series to extend to the forecast horizon, but DeepAR does. Imagine you have a time series of item sales up to time t and you are trying to predict sales from time $t + 1$ to $t + n$ into the future. If you are using weather data as your related time series, DeepAR requires you to have a weather forecast handy from time $t + 1$ to $t + n$ in order to predict your future sales. CNN-QR does not have that requirement.

One of the other benefits of Amazon Forecast is that in addition to algorithms, it allows you to include weather information as well as holiday information in your time series.

Forecasting Metrics

One of the complications of time series algorithms is how to measure model performance. Unlike a typical classification problem where you can randomly split data into train/test, that is no longer possible with time series as different rows of data in a time series are related to one another temporally.

Instead, time series use a concept called *backtesting*, where a model is tested against historical data where you have ground truth. For example, say you are an intrepid and avid fan of the markets and are trying to predict a stock price. You have trained a model to take historical stock data and predict future values. How will you test your model quality?

In this case, you can backtest your model on historical stock prices where you know exactly how the market behaved. Generally it is a good practice to conduct multiple backtests each with a little more training data (expanding window), but the fixed-length test horizon or fixed but sliding training data (sliding window) and fixed test horizon.

In each case, a good metric to evaluate your model is the mean-squared error (MSE), often reported as a root mean-squared error (RMSE). This is encapsulated in the following formula:

$$RMSE = \sqrt{\frac{1}{NT}\sum_{i,t}\left(y'_{i,t} - y_{i,t}\right)^2}$$

where $y'_{i,t}$ is the predicted value for time series i and time t; $y_{i,t}$ denotes the actual/observed value of the time series; and the sum is over all time series and times, normalized by NT: the number of points in the backtest window.

This metric amplifies outlier values whereas another metric called the weighted absolute percentage error (WAPE) is more robust against outliers since it does not take the square. WAPE is also sometimes known as mean absolute percentage error (MAPE).

$$WAPE = \sqrt{\frac{\sum_{i,t} | y'_{i,t} - y_{i,t} |}{\sum_{i,t} | y_{i,t} |}}$$

Which metric to rely on for your forecast depends on the use case: if an outlier might have an outsized business impact and a few large mispredictions can be expensive, you may consider RMSE over WAPE.

In both these forecasts, the model does not care if you underpredict versus overpredict. However, in many business problems, the cost of underpredicting can be very different from that of overpredicting. For example, in retail, you may want to be overstocked versus understocked. Similarly, if you are a cloud computing provider like AWS, you want your data centers to be overstocked versus understocked.

Amazon Forecast also provides a probabilistic forecast by providing you with quantiles such as p10, p50, or p90. It is useful to understand what these quantiles mean:

- p10 means that your model predicts that the true value will be less than this value only 10 percent of the time.
- p90 means that your model predicts that the true value will be less than this value 90 percent of the time.

As a retail firm, if the value of being understocked exceeds the cost of being overstocked, a p75 or p90 forecast may be more useful to you as a business, as you prefer to be overstocked rather than understocked.

In this case, you can choose a weighted quantile loss (wQL), which allows you to set a quantile, which can take values from 0.01 to 0.99. You may want to set this to 0.75, and your model will incorporate automatically different penalties for underfitting versus overfitting. We will not show the weighted quantile loss formula here, but for the test, it suffices to understand that this loss distinguishes between under- and overpredicting, unlike RMSE or WAPE.

When should you use WAPE versus RMSE versus wQL loss? If your business will have an outsized impact for a few large mispredictions, then consider RMSE. If your business costs change based on whether your forecast under- or overpredicts, consider wQL loss. Otherwise, consider WAPE. In general, it is a good practice to look at your model performance against multiple metrics and visualize your predictions with different quantiles, such as p10, p50, and p90.

Note that the p50 quantile is identical to the WAPE forecast. The WAPE forecast is often known as mean absolute percentage error (MAPE) or median forecasting.

Amazon Comprehend

Amazon Comprehend provides a set of natural language processing–based APIs to pretrained and custom models that can extract insights from text. Amazon Comprehend can analyze a document for the following characteristics:

1. *Entities*—Date, location, organization, persons, quantity, title, event, commercial item, and other entities.
2. *Key phrases*—A noun phrase that describes a particular thing; for example, the sentence "Your latest statement was mailed to 100 Main Street, Anytown, WA 98121." has a key phrase: "Your latest statement."
3. *Personally identifiable information (PII)*—Data that could be used to identify an individual such as a name, address, or bank account number. In the previous example, "100 Main Street, Anytown, WA 98121" is PII data.
4. *Language*—Amazon Comprehend can be used to identify what the dominant language is in the text. This can be one of 100 recognized languages.
5. *Sentiment*—Amazon Comprehend can determine the sentiment of the text provided; this can be positive, negative, mixed, or neutral.
6. *Syntax*—This is used to extract the part of speech for each word in the document.

Apart from these API calls to pretrained models, you can also train custom models on Amazon Comprehend using your own data. Training these custom models is a synchronous process—the customer prepares a set of documents, trains a model, and then uses these trained models to predict with a new set of documents. Three types of custom models you can train with Amazon Comprehend are as follows:

1. *Custom document classification*—For this, you provide a set of documents that are each associated with a label. Once your custom model is trained, you can pass in a new document to get a predicted label with a confidence value.
2. *Custom entity detection*—This can be used to extract custom entity types. As you can imagine, custom terms like policy numbers or part numbers are not included in the default entity detection on Comprehend. Custom entity detection can be trained with a list of entities and a set of documents that contain them. Once the model is trained, you can use this custom model to extract entities custom to your use case.
3. *Document topic modeling*—Topic modeling on Comprehend uses an unsupervised learning technique called Latent Dirichlet Allocation. A set of words that frequently show up in the same context across many documents form a topic. The same word can be associated with different topics.

Some overall guidelines that apply to Amazon Comprehend that are worth remembering are that the character encoding used in all text documents is UTF-8, and the size of each document must be less than 5,000 bytes. If you plan to send more than 25 documents per second, you should use batch detect operations. For example, you can use the

DetectDominantLanguage API if you are sending 20 documents per second, but using the BatchRequestDominantLanguage API, you can send up to 250 documents per second, but since it's a batch job, it may take more time for you to receive final results for all the documents in Amazon S3. The maximum quotas for asynchronous operations such as StartEntitiesDetectionJob or StartSentimentDetectionJob include a maximum document size of 1 MB, a maximum size of a batch of documents of 5 GB, and less than a million documents in a batch operation. For other guidelines and limits, visit this documentation page on Amazon Comprehend: https://docs.aws.amazon.com/comprehend/latest/dg/guidelines-and-limits.html.

Real World Scenario

Email Parsing Model

A customer collects incoming email in a JSON document and wants to use the subject line to redirect emails to the right department. The customer first organizes their data into a set of two-column CSV files; the first column is the department label, and the second column is the subject line. They can then use the console or the CreateDocumentClassifier API to start a custom training job. Amazon Comprehend uses between 10 and 20 percent of the training data for testing the final trained model and provides a classifier performance metric to help you determine if the model is trained well enough for your purposes. The customer can then analyze incoming emails by passing in the subject line to the hosted model endpoint and routing the email using the results obtained from this API call.

Amazon CodeGuru

Amazon CodeGuru uses program analysis and machine learning built from millions of lines of Java and Python code from the Amazon codebase to provide intelligent recommendations for improving code performance and quality. CodeGuru consists of two main services: Reviewer and Profiler.

Amazon CodeGuru Reviewer proactively detects potential code defects and offers suggestions for improving your Java or Python code. CodeGuru Reviewer does not identify syntax errors (an IDE is a better way to do this), but it does suggest improvements related to AWS best practices, resource leak prevention, concurrency, sensitive information leak prevention, refactoring, input validation, and security analysis. CodeGuru Reviewer can analyze code in AWS CodeCommit, Bitbucket, GitHub, or Amazon S3.

Amazon CodeGuru Profiler collects runtime performance data from your live applications and provides recommendations on how to fine-tune performance. Using machine learning,

CodeGuru Profiler helps find the most expensive lines of code, provides visualizations of profiling data, and suggests ways to reduce CPU bottlenecks. CodeGuru Profiler provides a single dashboard to understand application performance issues and can also be used to profile code running in AWS Lambda and other AWS compute services. On AWS Lambda, the easiest way to add `codeguru_profiler_agent` if you are using Python is by first adding a Lambda layer containing the package, and then using a function decorator as follows:

```
from codeguru_profiler_agent import with_lambda_profiler

@with_lambda_profiler(profiling_group_name="MyGroupName")
def handler_name(event, context):
    return "Profiler is active"
```

Amazon Augmented AI

Amazon Augmented AI (or A2I) is used to get a secondary human review of a low-confidence prediction from machine learning models. A2I works out of the box with Amazon Rekognition and Textract, but you can also use A2I with your own custom ML models. A2I is usually used when you want to review low-confidence predictions or to audit a random sample of predictions regardless of confidence levels. The first thing you need to do is define a human review workflow.

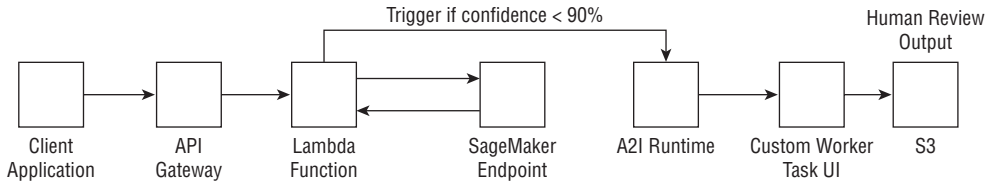
A human review workflow involves (1) defining a *work team* that will review predictions, and (2) using a UI template for providing instructions and the interface for humans to provide feedback (called the *worker task template*). A work team can be made up of a public workforce (powered by Amazon Mechanical Turk), a vendor-managed workforce, or your own private workforce. The worker task template displays your input data such as images or documents, and provides interactive tools to allow human reviewers to complete their task of reviewing the machine learning model's prediction.

For the two built-in task types with Amazon Rekognition and Textract, the corresponding AWS service will automatically trigger a human review loop. Conditions for when to trigger a human review loop are defined in a JSON document. For example, if you are using Amazon Textract to extract a specific key from submitted forms, you can use the *ImportantFormKey* parameter along with the *KeyValueBlockConfidenceLessThan* parameter to trigger a human review job when the confidence associated with a specific key is less than a particular threshold. When using a custom machine learning model, you can use the Amazon A2I Runtime API to start a human loop by calling the *StartHumanLoop* API.

Real World Scenario

Detecting Loan Application Fraud

A financial services company has a machine learning model that predicts whether a loan application is fraudulent or not. A recent mandate states that this company must review predictions of fraudulent loan applications by humans before making a decision on the loan. The company uses A2I to support automated machine learning by first calling the machine learning model endpoint and analyzing the confidence score. If the confidence score is less than 90 percent, the client triggers a human review loop in A2I and later analyze these results from humans from output files stored in Amazon S3 (see the following graphic).



Amazon SageMaker

Amazon SageMaker is an end-to-end machine learning platform that lets you build, train, tune, and deploy models at scale. SageMaker provides features through every step in the typical machine learning lifecycle. Here, we discuss each major step in this lifecycle and provide details about features that SageMaker provides. Note that we will not be discussing every single feature of SageMaker here but will focus on the features that support the most important phases of a typical machine learning lifecycle. For your reference, please see Table 1.1 for features of SageMaker that are relevant to the different phases of machine learning (for the latest set of features, please refer to the developer guide: `https://docs.aws.amazon.com/sagemaker/latest/dg/whatis.html`).

TABLE 1.1 Various features of SageMaker corresponding to the different phases in typical machine learning workflows

Prepare	Build	Train and tune	Deploy
GroundTruth	Notebooks	Managed Training	Managed Endpoints
Data Wrangler	Studio	Experiments	Model Monitor
Processing	Autopilot	Tuning	Pipelines
Feature Store	Jumpstart	Debugger	
Clarify		Spot Training	

Analyzing and Preprocessing Data

Typically, the first step in the machine learning lifecycle is to prepare data for training. Generally, the tool of choice for developing code that can help prepare data is an Integrated Development Environment (IDE), and more commonly a Jupyter Notebook. A notebook contains a mix of Markdown and runnable code that records outputs of each runnable cell. Once you are done experimenting with code on a notebook, it is also typical to perform the same preprocessing in stand-alone Python code. SageMaker provides the following components that helps with this phase in the ML lifecycle:

1. SageMaker notebook instance
2. SageMaker Studio
3. SageMaker Data Wrangler
4. SageMaker Processing
5. SageMaker GroundTruth

We will discuss all these components in detail next.

SageMaker Notebook Instance

A SageMaker notebook instance is a managed ML compute instance running the Jupyter server. Users can create a notebook instance from the SageMaker console or using the *CreateNotebookInstance* API. When creating the notebook instance, SageMaker first creates a network interface in the chosen VPC and associates the security group that you provide in your request with the subnet in a particular availability zone. SageMaker then launches an instance in the service VPC and enables traffic between your VPC and the notebook instance. SageMaker then installs common packages and ML frameworks and additionally runs any lifecycle configuration scripts that you define; these scripts can be used to pull the latest updates from a Git repository, mount a shared drive, or download data and packages. SageMaker then attaches an EBS storage volume (you can choose a size between 5 GB and

16 TB). Files stored inside the `/home/ec2-user/SageMaker` directory persist between notebook sessions (that is, when you turn the notebook instance off and on again). Note that scheduling a notebook to be turned off during idle times is important to reduce costs; this can be done using lifecycle configuration scripts or via Lambda functions. For more details on this, please read the following document: `https://aws.amazon.com/blogs/machine-learning/right-sizing-resources-and-avoiding-unnecessary-costs-in-amazon-sagemaker`.

From the AWS Console, SageMaker displays a list of notebook instances that are in your account as well as ones that are stopped or running. When you access your notebook instance, the console uses the credentials you used to sign in to get a presigned URL by calling the *CreatePresignedNotebookInstanceUrl* API call. If you are signing in through your company's single sign-on, Active Directory, or another identity provider like Google or Facebook, identity federation using identity and access management (IAM) roles are already set up, and this lets you assume a role indirectly that allows access to SageMaker resources, such as a notebook instance. Lastly, SageMaker uses the `nbexamples` Jupyter extension to provide over 200 examples showcasing various use cases and SageMaker features. SageMaker also lets you associate your notebook instance with a project Git repository that automatically gets cloned to your instance on startup.

When using a SageMaker notebook instance, you can edit the notebook execution role to access other AWS services. For example, you can use the notebook instance to manage large-scale data preprocessing by making API calls to AWS Glue or connect your notebook to Amazon EMR to run a PySpark kernel. You can also query an Amazon Redshift data warehouse for data that you need to prepare for training.

SageMaker Studio

SageMaker Studio is a web-based IDE for machine learning and is based on a highly customized JupyterLab environment. In a single visual interface, you can write and maintain notebooks, track experiments, deploy and monitor models, and more. Compared to notebook instances, SageMaker Studio launches containerized images that are used to run kernels for your notebooks. This lets you have multiple back-end compute instances run your notebooks. For example, one notebook tab on Studio could be running a general-purpose m4 instance, while another notebook may run a GPU instance for local training. SageMaker Studio lets you share a snapshot of your notebook with your teammates, which other users who share the same domain can copy and use in their own personal workspace. The workspace setup is a folder in an Amazon EFS drive that can elastically scale in size as your local data grows.

SageMaker Studio also provides a visual interface to many other SageMaker features, such as the following:

- Visual Git workflow
- Experiment tracking
- SageMaker Autopilot for AutoML on tabular datasets
- Curated one-click solutions for applications on SageMaker Jumpstart

- Pretrained and fine-tunable models for typical vision and NLP jobs through SageMaker Jumpstart
- Model-building pipelines using SageMaker pipelines
- SageMaker Clarify for detecting pretraining bias
- SageMaker Feature store for creating, sharing, and managing curated data for ML development
- SageMaker Data Wrangler for preparing data (we will discuss this next)

SageMaker Data Wrangler

SageMaker Data Wrangler lets you import, transform and analyze data through a visual workflow, and then export that workflow. Data Wrangler allows you to import data from Amazon S3, Athena, and Redshift. A data preparation pipeline on SageMaker Data Wrangler is called a *data flow*.

SageMaker Data Wrangler automatically creates a new intermediate data frame when you add a new step to the data flow. You can add four different types of steps:

Data Transform Step Data Wrangler provides over 300 built-in transforms to normalize, transform, and combine columns without writing any code. You can also create your own custom steps using Python or PySpark code.

Data Analysis Step This step uses 100,000 rows of your dataset to provide built-in or custom visualizations, a quick machine learning model to assess feature importance scores, statistical summaries of your columns, and a correlation or target leakage report.

Join This step joins two datasets and produces one data frame, including left joins, right joins, and inner joins.

Concatenate This step concatenates one dataset to another and adds the result to your data flow.

SageMaker Data Wrangler steps can be exported to a Data Wrangler job, a notebook with all the steps, a feature store, or stand-alone Python code. This lets you modularize your preprocessing step and run it on demand, usually with SageMaker Processing, which we will discuss next.

SageMaker Processing

SageMaker Processing is a simple, managed feature on SageMaker that allows you to run common data processing workloads such as preprocessing, feature engineering, and model evaluation. SageMaker takes your Python or PySpark script, copies data from an Amazon S3 location, processes your data, and writes back output data to another Amazon S3 output location in your account. You can also provide a custom container image rather than using one of the built-in images provided by SageMaker Processing. When passing in a Python script, you use `SKLearnProcessor`, and when passing in a PySpark script, you use the

`PySparkProcessor` classes in the SageMaker Python SDK. It is common to use multiple instances to process data, in which case you can shard the input objects by S3 key so that each instance receives the same number of input files to process.

SageMaker GroundTruth

SageMaker GroundTruth provides an important functionality in this preprocessing phase of the ML lifecycle. To train an ML model using a supervised training algorithm, you need high-quality, labeled data. GroundTruth provides built-in labeling functionality for common task types (like image classification or document classification), and also allows completely customized workflows. With GroundTruth, you can use a public workforce (Amazon Mechanical Turk), a private workforce, or a vendor company. You can optionally use automated data labeling for some task types, which uses active learning to train a model in parallel and decide which samples of data to send to human labelers. Built-in task types involving images, video frames, text data, and LiDAR data have managed UIs that are surfaced to workers in a secure way. You can also provide custom UIs for your data labeling jobs. For more information about SageMaker GroundTruth, visit `https://docs.aws.amazon.com/sagemaker/latest/dg/data-label.html`.

Training

Once you have prepared your training data, you can train your models using one of the following training modes on Amazon SageMaker:

- Amazon SageMaker provides 17 built-in algorithms for typical use cases. These include binary or multiclass classification, regression, time series forecasting, anomaly detection, IP address anomalies, embedding generation, clustering, topic modeling, text classification and summarization, image classification, object detection, and semantic segmentation. We cover these in more detail in Chapter 8, "Model Training."

Read up on the typical use cases and algorithm mappings at the following documentation site: `https://docs.aws.amazon.com/sagemaker/latest/dg/algos.html`.

- When using a popular ML framework like TensorFlow, PyTorch, or MXNet, you can submit a script to SageMaker. SageMaker will provide a managed container that can run several versions of these popular frameworks. This is useful when you want to bring your own algorithm without the additional heavy lifting of managing a custom container.
- Lastly, you can create a completely custom container for your training job. SageMaker will simply run your container in a managed training instance of your choice with the entry point that you provide. For example, your entry point in the Dockerfile could look like: `ENTRYPOINT ["python", "train.py"]`.

Some additional training features provided by Amazon SageMaker are as follows:

Distributed Training SageMaker provides both model parallel and data parallel distributed training strategies. Let's briefly discuss what these strategies are. Data parallel strategy in distributed training is where the dataset is split up across multiple processing nodes. Each node runs an epoch of training and shares results with other nodes before moving on to the next epoch. In model parallel training, the model is split up across multiple processing nodes. Each node carries a subset of the models and is responsible to run a subset of the transformations as decided by a pipeline execution schedule so that performance losses due to sequential computations are minimized.

Managed Spot Training You can use managed spot instances instead of on-demand instances to reduce the cost of training by up to 90 percent. Spot instance interruptions are handled by SageMaker, but you are responsible for creating your own checkpoints to allow the training to continue post any disruptions.

Automatic Model Tuning SageMaker's Automatic model tuning, also known as hyperparameter optimization, can be used to search for the optimal set of hyperparameters using either a random search or Bayesian optimization. You can use the hyperparameters that result in the best version of your model, measured by a metric that you choose (for example, validation accuracy).

Monitoring Training Jobs SageMaker training job logs can be viewed in Amazon CloudWatch. Additionally, training job metrics such as training error that are emitted can be viewed as graphs in the console. After the training job has ended, you can also view final metrics using the *DescribeTrainingJob* API call.

SageMaker Debugger SageMaker Debugger can be used to profile and debug your training jobs to improve the performance of ML model training by eliminating bottlenecks and detecting nonconverging conditions. Debugger stores instance-level metrics, framework-level metrics, and custom tensors that you define from within your training code. You can use either various built-in rules (like loss not decreasing or overtraining), or your own custom rules to analyze tensors emitted by your training code. Debugger also provides profiler rules such as CPU bottleneck threshold and I/O bottleneck threshold. When a rule is activated, you can trigger an Amazon SNS notification or Lambda function to take further action, such as stopping a training job.

Model Inference

Training a model on SageMaker results in a trained model artifact on Amazon S3 (usually in the format of a `model.tar.gz` file). To get predictions from this model, you can either host a persistent endpoint for real-time predictions or use the SageMaker batch transform APIs to apply model predictions to an entire test dataset.

For real-time predictions, SageMaker provides fully managed model hosting services and generates a private HTTPS endpoint where your model can return prediction outputs. You can deploy multiple production variants of the model to divert different percentages of traffic to different versions of your model. You can host multiple models and target these models from your client application calling the endpoint. And finally, after you deploy your model into production, you can use SageMaker's Model Monitor to continuously monitor model quality metrics in real time and provide you with a notification when deviations such as data drift are detected.

For batch predictions, SageMaker initializes the requested number of compute instances and distributes inference workload involving getting predictions for a large test dataset between these instances. Batch transform jobs create the same number of output files as input files, with an additional `.out` extension. You can reduce the time it takes to do large-scale batch transform jobs by tuning parameters that control the maximum payload accepted, the maximum number of concurrent transforms, or the batch strategy used.

Additional features of model deployment that become important in production are as follows:

Endpoint Autoscaling Dynamically adjust the number of instances used to host your model based on changes in your workload.

Model Compilation SageMaker Neo automatically optimizes your models to run more effectively on cloud or edge hardware.

Elastic Inference (EI) EI lets you add GPU-based accelerators to your hosting instances at a fraction of the cost of using a full GPU instance and supports any TensorFlow, MXNet, PyTorch, or ONNX model.

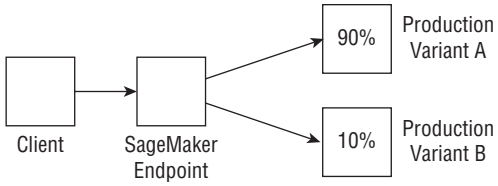
Inference Pipelines This lets you deploy a linear sequence of up to five containers that perform steps on your incoming data. Typically this sequence may involve a preprocessing step, a model prediction step, and a post-processing step done in real time.

SageMaker Model Registry SageMaker Model Registry lets you catalog models for production, manage versions, add and manage manual approval steps for the model, and automate model deployment using continuous integration/continuous delivery (CI/CD).

Real World Scenario

A/B Testing Deployment

A customer who already has a hosted model would like to test a new version with production traffic. To do this, the customer updates the endpoint configuration and diverts 10 percent of the traffic to a new production variant (see the following graphic).



An exhaustive list of features related to SageMaker are discussed in the AWS documentation section here: `https://docs.aws.amazon.com/sagemaker/index.html`. Please take note of the Exam Essentials section at the end of this chapter.

AWS Machine Learning Devices

In this brief section, we describe devices that belong to the machine learning stack:

AWS DeepLens The DeepLens ecosystem lets you learn about vision systems and deep learning by providing you with a fully programmable video camera and several pre-trained models and examples.

AWS DeepRacer The DeepRacer ecosystem lets you learn about reinforcement learning using a fully managed simulation and training environment, as well as a 1/18 scale RC (race car) car that can run your trained model.

AWS DeepComposer DeepComposer is a fully programmable MIDI keyboard that lets you play, record, train, and generate music using generative adversarial networks (GANs).

AWS Panorama Device and SDK This allows you to add computer vision–based applications to your IP camera setup. You can analyze video feeds from multiple cameras in parallel generating predictions from models that you trained and compiled on the cloud with SageMaker.

For the exam, it is sufficient to understand at a high level what these device offerings may be used for. For more information about the Machine Learning Stack on AWS, visit the documentation page here: `https://aws.amazon.com/machine-learning`. To dive deeper on any of these services, follow the detailed documentation pages—for example, `https://docs.aws.amazon.com/panorama/latest/dev/panorama-welcome.html` and `https://docs.aws.amazon.com/deeplens/latest/dg/what-is-deeplens.html`. Trying out the getting started guides can help you go beyond just passing the exam.

Summary

As you have seen in this chapter, machine learning has a wide range of applications such as analyzing images and videos for insights, mining text data, translating and transcribing speech, enterprise search, personalization, and others. You have learned that the AI services like Rekognition, Personalize, Forecast, Transcribe, and others make it easy to build these ML applications with little prior ML knowledge or via little code.

That said, sometimes ML developers need the flexibility to train their own algorithms, tune hyperparameters, explore and analyze data for machine learning, and then deploy those models to production. This requires some ML knowledge and writing code to develop algorithms. This is where Amazon SageMaker comes in; SageMaker lets data scientists focus on the data and ML model development while abstracting the underlying infrastructure needs from them.

Finally, you have seen how even once models are deployed, often humans need to review the ML model outputs. A service like Augmented AI can put humans in the loop in the ML workflow to review model performance.

Exam Essentials

Understand the various areas of machine learning that are covered by AI/ML services on AWS. AI services cover several subdomains of machine learning such as vision (Rekognition, Textract, etc.), speech (Lex, Polly, Transcribe), recommendation systems (Personalize), and forecasting systems (Forecast). For more customer use cases, or for custom implementations of the aforementioned common AI/ML use cases, you can use Amazon SageMaker's built-in algorithms or bring your own algorithm to build, train, tune, and deploy.

Familiarize yourself with the basics of what each service is meant to do and the main features of each service. Use the descriptions provided here as a starting point, and dive deeper into it using the documentation on AWS (`https://docs.aws.amazon.com`).

Review Questions

1. You have raw text data stored in S3 and would like to use each document to train a custom text classification model. What is the easiest way to achieve this?
   - **A.** Download all your data and work with an open-source framework on your laptop.
   - **B.** Use Comprehend Custom labels to train a custom document classification model.
   - **C.** First use SageMaker Processing to preprocess your data; then use the SageMaker built-in Blazing text algorithm to train and deploy your model.
   - **D.** None of the options is correct.

2. A customer would like to run computer vision models at a manufacturing facility and already uses IP cameras and custom edge devices for other purposes. The customer is a current user of SageMaker and needs suggestions on how to deploy these models. Which of the following options would you as a solutions architect suggest?
   - **A.** Replace all IP cameras with DeepLens cameras, and use SageMaker models at the edge.
   - **B.** Use outposts and attach cameras directly to Outpost.
   - **C.** Purchase "smart cameras" from a vendor and retrain your models on the vendor-provided software.
   - **D.** Download and use SageMaker trained models on the custom edge devices.

3. A marketing data provider has 50 GB of time series data from various customers and would like to train a forecasting model to predict future sales. The customer uses an open-source algorithm on premises and is exploring ways to build multiple forecasting models based on cohorts of customers. Which of these solutions will work for this company?
   - **A.** Use Amazon Forecast. It automatically recognizes cohorts and can easily handle up to 100 GB of files on premises or on S3.
   - **B.** Use the open-source algorithm on SageMaker either by using Script mode or by bringing in a custom container and pointing the training job to data on S3.
   - **C.** Redshift is the best option to both store and query data. It can also be used to forecast data in this case.
   - **D.** None of the options is correct.

4. A customer using SageMaker Studio has been manually running each step in a complex workflow. What is the easiest way to automate and manage these manual steps?
   - **A.** Use SageMaker Pipelines. It is integrated with Studio, and converting the manual steps to a workflow is easy with the Python SDK.
   - **B.** Move all steps to Step functions. Author the individual steps on Studio, but run pipelines in the Step functions.
   - **C.** Move all steps to Managed Workflows for Apache Airflow. Author the individual steps on Studio, but run pipelines in Airflow.
   - **D.** Move all steps to an EC2 instance, and use a Bash script to run each step in succession.

5. A customer currently uses Spark on premises to transform datasets for machine learning purposes. The customer is new to AWS and is aware of training options that are available on SageMaker. The customer would like to reuse Spark code that they have developed as is but make it part of their machine learning lifecycle on AWS. What solution will require the least amount of maintenance and would integrate well with other steps in the machine learning lifecycle?

   A. Use EMR to run on-demand Spark jobs.

   B. Use the Spark processing container provided by SageMaker and prepare data for training steps that will also use SageMaker.

   C. Use Glue DataBrew to import your Spark code and run as part of a data preparation pipeline.

   D. Set up an EC2 instance that replicates the on-premises setup. Since the setup on AWS now matches the on-premises setup, the customer can easily run Spark jobs without any additional effort.

6. A customer running a streaming service has 10,000 audio files in S3. The customer would like to easily label these audio files and use them in a deep learning algorithm for music genre classification. Which solution will allow the customer to achieve this?

   A. Use a built-in UI template for audio classification on SageMaker GroundTruth, followed by a built-in audio classification algorithm to train the model.

   B. Use a built-in UI template for audio classification on SageMaker GroundTruth, followed by a custom audio classification algorithm to train the model.

   C. Use a custom UI template for audio classification on SageMaker GroundTruth, followed by a built-in audio classification algorithm to train the model.

   D. Use a custom UI template for audio classification on SageMaker GroundTruth, followed by a custom audio classification algorithm to train the model.

7. A media company wants to process image data to detect persons, objects, and text from a database of images, but the company is concerned about their lack of machine learning expertise to build and deploy a custom solution. Which AWS service would you advise them to use to solve this problem?

   A. Amazon Comprehend

   B. Amazon Rekognition

   C. Amazon SageMaker

   D. Amazon Textract

8. An asset management firm would like to build a chatbot-based solution to automate advice given to their clients by their financial advisers. They are concerned that due to their diverse global client base, the chatbot will need to translate incoming text into English before the advice can be rendered. What services would you use to build this solution?

   A. Lex, Translate

   B. Lex, Polly

   C. Translate, Polly

   D. SageMaker, Lex

9. A retail company wants to build a forecasting model to forecast demand for their products. They have thousands of products and related product metadata. Although they have tried a few models like ARIMA on premises, they are concerned with the model performance and are also looking for a solution that can be scaled and deployed easily. Another concern they have is that they do not want to understock their warehouses. What solution would you recommend?
   - **A.** Train DeepAR on Amazon SageMaker for scalability and pick MAPE loss to solve the understocking problem.
   - **B.** Train ARIMA on EC2 and use EC2 AutoScaling to solve the scalability issue.
   - **C.** Use Amazon ETS on Amazon Forecast. Include product information as item metadata. Pick a 0.75 weighted quantile loss metric to solve the understocking problem.
   - **D.** Use DeepAR+ on Amazon Forecast. Include product information as item metadata. Pick a 0.75 weighted quantile loss metric to solve the understocking problem.

10. You are trying to get your organization excited about machine learning. You host a tournament where employees can race a car around a race track that is programmed using reinforcement learning to teach them about applications of ML to real-world scenarios. Which AWS service is suited for this activity?
   - **A.** AWS Deep Lens
   - **B.** AWS Deep Composer
   - **C.** Amazon S3
   - **D.** AWS DeepRacer

11. You have trained an ARIMA-based forecasting model to forecast electricity prices in ZIP codes across the country. You want to use a metric that penalizes the model differently for under- versus overpredicting the price. Which metric would you use?
   - **A.** Weighted quantile error
   - **B.** Root mean squared error (RMSE)
   - **C.** Mean squared error (MSE)
   - **D.** Mean absolute percentage error (MAPE)

12. You want to train a single model across a multitude of time series ranging in the thousands. You also have contextual data associated with the time series as a related time series, but the related time series data does not extend in the prediction interval. Finally, you wish to use a fully managed service to produce the ML model instead of developing your own algorithm code from scratch. What service and algorithm would you use?
   - **A.** Amazon Personalize, multi-arm bandits
   - **B.** Amazon SageMaker, XGBoost
   - **C.** Amazon Forecast, CNN-QR
   - **D.** Amazon Forecast, DeepAR

13. A major sports company wants to detect helmets on players to ensure player safety. The company has terabytes of video, but it is largely unlabeled. What AWS service would you use to label the data?
    - **A.** Amazon Comprehend
    - **B.** Amazon SageMaker Ground Truth
    - **C.** Amazon SageMaker Processing
    - **D.** Amazon Forecast

14. Consider the same use case as in the previous two questions. Having trained the object detection algorithm, you want to deploy it in production. However, the incoming raw video first needs to be processed before it can be sent to the model for inference. This processing code is written in Spark. You want to jointly deploy the Spark-based processing code and the inference code. Which AWS tool lets you do this?
    - **A.** Inferentia
    - **B.** Neuron SDK
    - **C.** Inference Pipelines
    - **D.** SageMaker Model Monitor

15. You work for an insurance firm trying to automate insurance claims processing. As a first step, you want to parse PDF documents and extract relevant entities. What AWS service could you use to get started with entity detection without much ML experience?
    - **A.** AWS SageMaker
    - **B.** Amazon Comprehend
    - **C.** Amazon Kendra
    - **D.** Amazon Personalize

16. You are the head of a law firm trying to modernize your internal document search systems. What AWS service would you use where users can type their questions and the service will parse the question and provide the most relevant collection of documents that may match the response?
    - **A.** Amazon Kendra
    - **B.** Amazon Comprehend
    - **C.** Amazon Forecast
    - **D.** Amazon Rekognition

17. You work for an insurance firm trying to automate insurance claims processing. As a first step, you want to perform optical character recognition (OCR) and extract forms and tables from PDF documents. What AWS service could you use to get started with this use case without having to build your own or use an open-source OCR solution?
    - **A.** AWS SageMaker
    - **B.** Amazon Textract
    - **C.** Amazon Kendra
    - **D.** Amazon Comprehend

18. You have some custom PySpark code that you use to process data prior to training an ML model on that processed data. Which of the following AWS tools can be used to process the data? (Choose all that apply.)

A. SageMaker Clarify

B. AWS Glue

C. SageMaker Processing

D. Amazon TimeStream

19. Which AWS services allow you to build ML Ops pipelines by defining a directed acyclic graph (DAG) that can be executed to process data, train a model, and deploy the model? (Select all that apply.)

A. AWS Step Functions

B. AWS SageMaker Pipelines

C. Amazon CodeCommit

D. Amazon CodeBuild

20. Which AWS service proactively detects bottlenecks and defects in your code and offers suggestions to improve based on AWS code best practices for code in AWS CodeCommit or GitHub?

A. AWS Guru Code

B. AWS Code Guru

C. AWS DevOps Guru

D. AWS Lookout for Code