Domain 2: Data Transformation

2.1 Choose a modelling approach

2.1.1 AWS Model Approaches

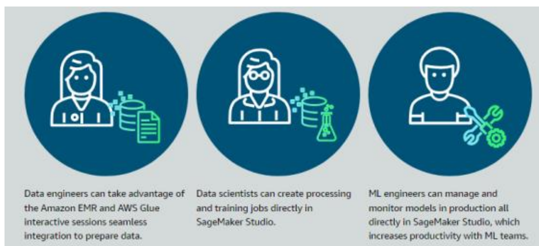
AWS AI/ML stack:

- AWS AI services: NO fine-tune option
- AWS ML services: fine-tune option
- Customized ML model solutions (using AWS infrastructure and frameworks)

2.1.1 SageMaker Offerings

Studio

1. **Roles and Persona**



Choice of IDEs

Jupyterlab	Code Editor	RStudio

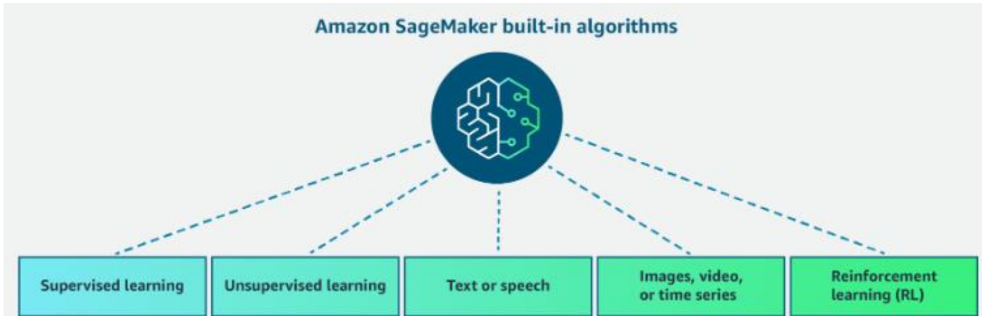
Choice of IDEs SageMaker notebook instances

SageMaker notebook instances initiate Jupyter servers on Amazon Elastic Compute Cloud (Amazon EC2) and provide preconfigured kernels with the following packages:

- Amazon SageMaker Python SDK, AWS SDK for Python (Boto3)
- AWS Command Line Interface (AWS CLI)
- Conda
- Pandas
- Deep learning framework libraries
- Other libraries for data science and ML

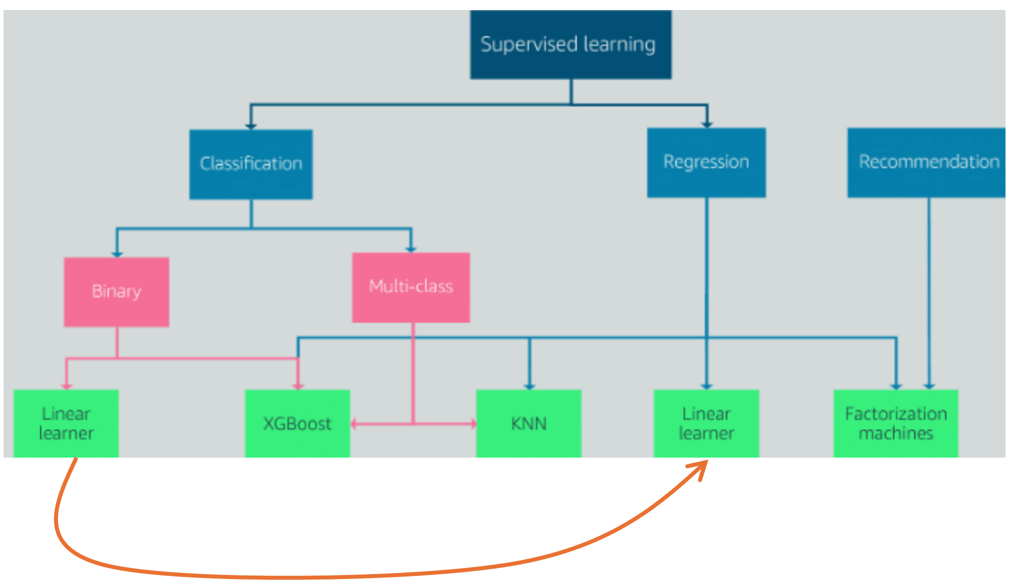
2.1.1 SageMaker Model types

SageMaker notebook instances initiate Jupyter servers on Amazon Elastic Compute Cloud (Amazon EC2) and provide preconfigured

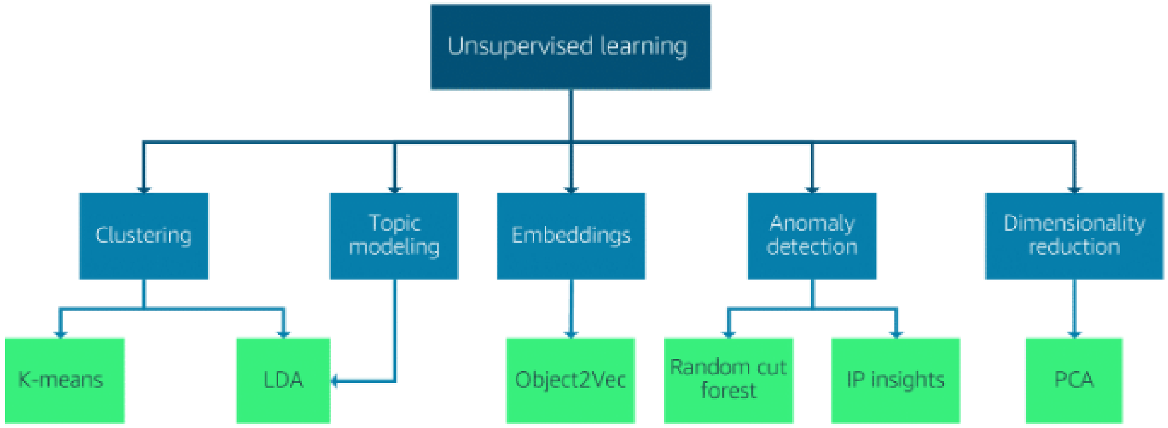


1. ***Supervised***

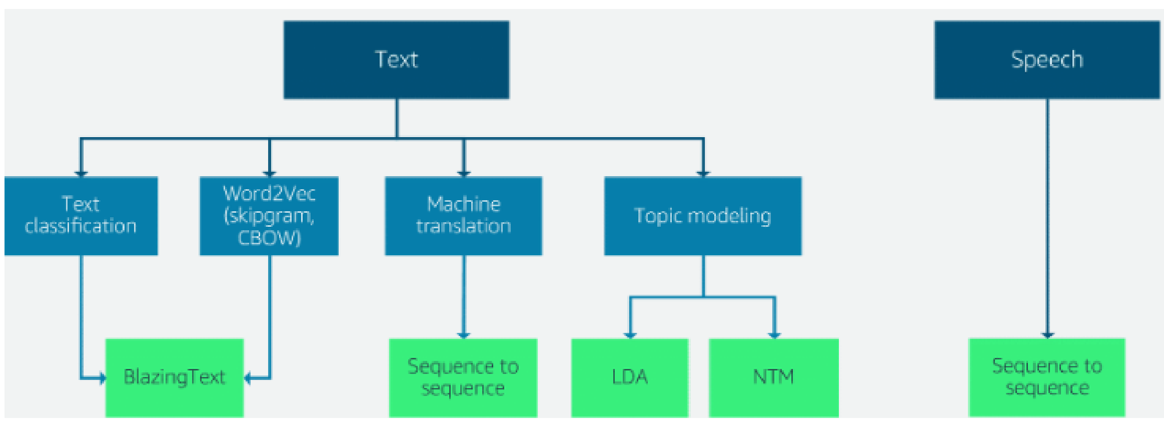
Algorithm	*Classification: Binary*	*Classification: Multi-class*	*Regression*
Linear Learner	*Yes*	*No*	*Yes*
XGBoost	*Yes*	*Yes*	*Yes*
K-Nearest Neighbors	*No*	*Yes*	*Yes*
Factorization Machines	*No*	*No*	*Yes*



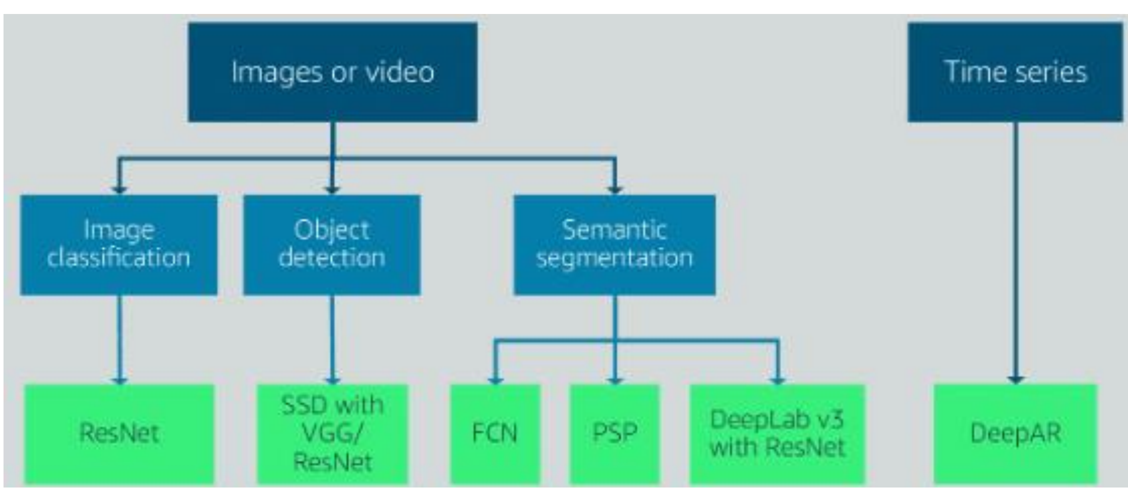
2. *Unsupervised*



3. *Text or speech data*



4. *Images and video (or time series data)*



5. *Reinforcement learning (RL)*

To train RL models in SageMaker RL, use the following components:

- **A deep learning (DL) framework.** Currently, SageMaker supports RL in TensorFlow and Apache MXNet.
- **An RL toolkit.** An RL toolkit manages the interaction between the agent and the environment and provides a wide selection of state of the art RL algorithms. SageMaker supports the Intel Coach and Ray RLlib toolkits. For information about Intel Coach, see https://nervanasystems.github.io/coach/(opens in a new tab). For information about Ray RLlib, see https://ray.readthedocs.io/en/latest/rllib.html(opens in a new tab).
- **An RL environment.** You can use custom environments, open-source environments, or commercial environments. For information, see RL Environments in Amazon SageMaker(opens in a new tab).

2.1.3 SageMake AutoML

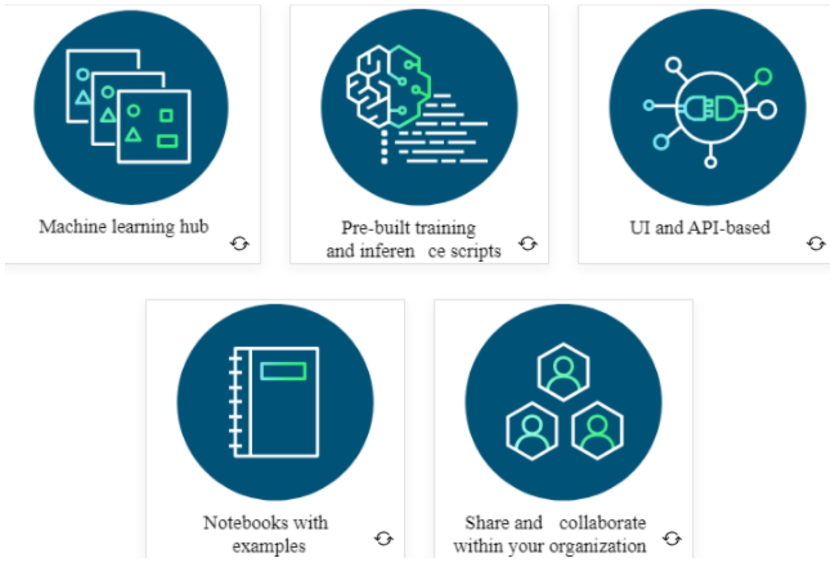
SageMaker

- **Data analysis and processing**: SageMaker Autopilot identifies your specific problem type, handles missing values, normalizes your data, selects features, and prepares the data for model training.
- **Model selection**: SageMaker Autopilot explores a variety of algorithms. SageMaker Autopilot uses a cross-validation resampling technique to generate metrics that evaluate the predictive quality of the algorithms based on predefined objective metrics.
- **Hyperparameter optimization**: SageMaker Autopilot automates the search for optimal hyperparameter configurations.
- **Model training and evaluation**: SageMaker Autopilot automates the process of training and evaluating various model candidates.
    - It splits the data into training and validation sets, and then it trains the selected model candidates using the training data.
    - Then it evaluates their performance on the unseen data of the validation set.
    - Lastly, it ranks the optimized model candidates based on their performance and identifies the best performing model.
- **Model deployment:** After SageMaker Autopilot has identified the best performing model, it provides the option to deploy the model. It accomplishes this by automatically generating the model artifacts and the endpoint that expose an API. External applications can send data to the endpoint and receive the corresponding predictions or inferences.

2.1.3 SageMake JumpStart

SageMaker JS is a ML hub with foundation models, built-in algorithms, and prebuilt ML solutions that you can deploy with a few clicks.

Features



Foundation Models

With JumpStart foundation models, many models are available such as:

- *Jurassic models from AI21*
- *Stable Diffusion from Stability.ai*
- *Falcon from HuggingFace*
- *Llama from Meta*
- *AlexaTM from Amazon*

JumpStart industry-specific solutions

- ***Demand forecasting***

Amazon SageMaker JumpStart provides developers and data science teams ready-to-start AI/ML models and pipelines. SageMaker JumpStart is ready to be deployed and can be used as-is. For demand forecasting, SageMaker JumpStart comes with a pre-trained, deep learning-based forecasting model, using Long- and Short-Term Temporal Patterns with Deep Neural Networks (LSTNet).

- ***Credit rating prediction***

Amazon SageMaker JumpStart solution uses Graph-Based Credit Scoring to construct a corporate network from SEC filings (long-form text data).

- ***Fraud detection***

Detect fraud in financial transactions by training a graph convolutional network with the deep graph library and a SageMaker XGBoost model.

- ***Computer vision***

Amazon SageMaker JumpStart supports over 20 state-of-the-art, fine-tunable object detection models from PyTorch hub and MxNet GluonCV. The models include YOLO-v3, FasterRCNN, and SSD, pre-trained on MS-COCO and PASCAL VOC datasets.

Amazon SageMaker JumpStart also supports image feature vector extraction for over 52 state-of-the-art image classification models including ResNet, MobileNet, EfficientNet from TensorFlow hub. Use these new models to generate image feature vectors for their images. The generated feature vectors are representations of the images in a high-dimensional Euclidean space. They can be used to compare images and identify similarities for image search applications.

- ***Extract and analyze data from documents***

JumpStart provides solutions for you to uncover valuable insights and connections in business-critical documents. Use cases include text classification, document summarization, handwriting recognition, relationship extraction, question and answering, and filling in missing values in tabular records.

- ***Predictive maintenance***

The AWS predictive maintenance solution for automotive fleets applies deep learning techniques to common areas that drive vehicle failures, unplanned downtime, and repair costs.

- ***Churn prediction***

After training this model using customer profile information, you can take that same profile information for any arbitrary customer and pass it to the model. You can then have it predict whether that customer is going to churn or not. Amazon SageMaker JumpStart uses a few algorithms to help with this. LightGBM, CatBoost, TabTransformer, and AutoGluon-Tabular used on a churn prediction dataset are a few examples.

- ***Personalized recommendations***

Amazon SageMaker JumpStart can perform cross-device entity linking for online advertising by training a graph convolutional network with a deep graph library.

- ***Healthcare and life sciences***

You could use the model to summarize long documents with LangChain and Python. The Falcon LLM is a large language model, trained by researchers at the Technology Innovation Institute (TII) on over 1 trillion tokens using AWS. Falcon has many different variations, with its two main constituents Falcon 40B and Falcon 7B, comprised of 40 billion and 7 billion parameters, respectively. Falcon has fine-tuned versions trained for specific tasks, such as following instructions. Falcon performs well on a variety of tasks, including text summarization, sentiment analysis, question answering, and conversing.

- ***Financial pricing***

Many businesses dynamically adjust pricing on a regular basis to maximize their returns. Amazon SageMaker JumpStart has solutions for price optimization, dynamic pricing, option pricing, or portfolio optimization use cases. Estimate price elasticity using Double Machine Learning (ML) for causal inference and the Prophet forecasting procedure. Use these estimates to optimize daily prices.

- ***Causal inference***

Researchers can use machine learning models such as Bayesian networks to represent causal dependencies and draw causal conclusions based on data.

2.1.5 Bedrock

Use cases

Text generation

Create new pieces of original content, such as short stories, essays, social media posts, and webpage copy.

Chatbots

Build conversational interfaces, such as chatbots and virtual assistants, to enhance the user experience for your customers. Build assistants that understand user requests, automatically break down tasks, engage in dialogue to collect information, and take actions to fulfill the request.

Search

Search for and synthesize relevant information to answer questions and provide recommendations from a large corpus of text and image data.

Text summarization

Get concise summaries of long documents such as articles, reports, research papers, technical documentation, and even books to quickly and effectively extract important information.

Image generation

Quickly create realistic and visually appealing images for ad campaigns, websites, presentations, and more from language prompts.

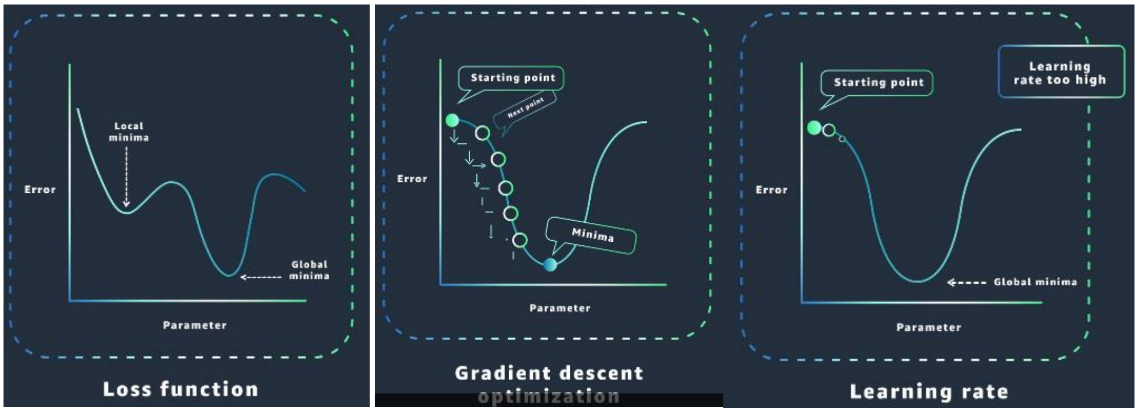
Personalization

Help customers find what they're looking for with more relevant and contextual product recommendations than word matching.

2.2 Train Models

2.2.1 Model Training Concepts

Minimizing loss:



- Loss function needs to be minimum (Global minimum)
- Gradient descent optimization used to reach “global minima” loss
- Hyperparameter tuning uses Gradient descent to reach “global minima” loss
  - Too much tuning can result in “Overshoot”, learning rate too high
  - Too less tuning can result in “Undershooting”, learning rate too small

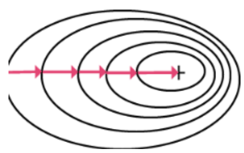
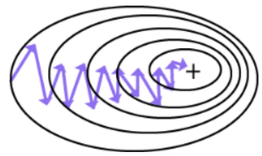
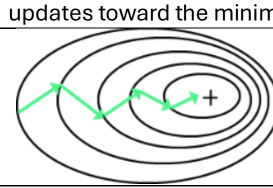
(Measuring) Loss function:

Root mean square error	Log-likelihood loss
• The most basic form of a loss function, commonly used in regression tasks such as predicting continuous values is known as Root Mean Square Error (RMSE). • A regression task can be used to predict home prices.	• A variation of a loss function is log-likelihood loss, also known as cross-entropy loss, is used in logistic regression. • With log-likelihood loss, instead of the raw probabilities of predictions of each class, the logarithm of probabilities is considered.
$\sqrt{\frac{\sum_{i=1}^{n}\left(Y_{target\ ,i} - Y_{pred\ ,i}\right)^2}{n}}$	$-(y \log p + (1-y)\log(1-p))$

When to use which

Log-likelihood loss is an algorithm used for classification tasks, where the goal is to predict whether an input belongs to one of two or more classes. For example, you might use logistic regression to predict whether an email is spam.

Optimizing - Reducing Loss function:

Optimization technique	***Gradient descent***	***Stochastic gradient descent***	***Mini-batch gradient descent***
Weights updated	Every epoch	Every datapoint	Every batch
Speed of each epoch calculation	Slowest	Fast	Slower
Gradient steps	Smooth updates toward the minima	Noisy or erratic updates toward the minima	Less noisy or erratic updates toward the minima
			

Gradient descent

As mentioned, gradient descent only updates weights after it's gone through all of the data, also known as an epoch. Of the three variations covered here

- gradient descent has the slowest speed to finding the minima as a result, but
- also has the fewest number of steps to reach the minima.

In stochastic gradient descent or SGD, you update your weights for each record you have in your dataset.

Stochastic Gradient Descent (SGD)

For example, if you have 1000 data points in your dataset, SGD will update the parameters 1000 times. With gradient descent, the parameters would be updated only once in every epoch.

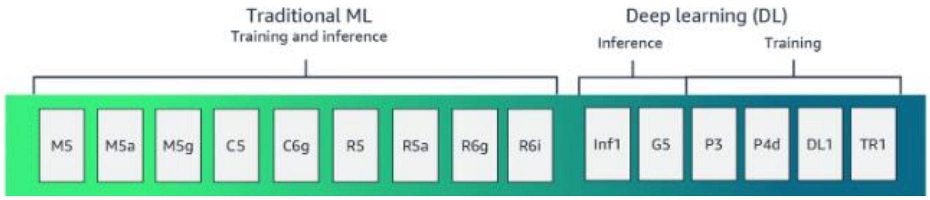
- SGD leads to more parameter updates and, therefore, the model will **get closer to the minima more quickly.**
- One drawback of SGD, however, is that it will oscillate in different directions, unlike gradient descent, hence lot more steps.

Mini-batch gradient descent

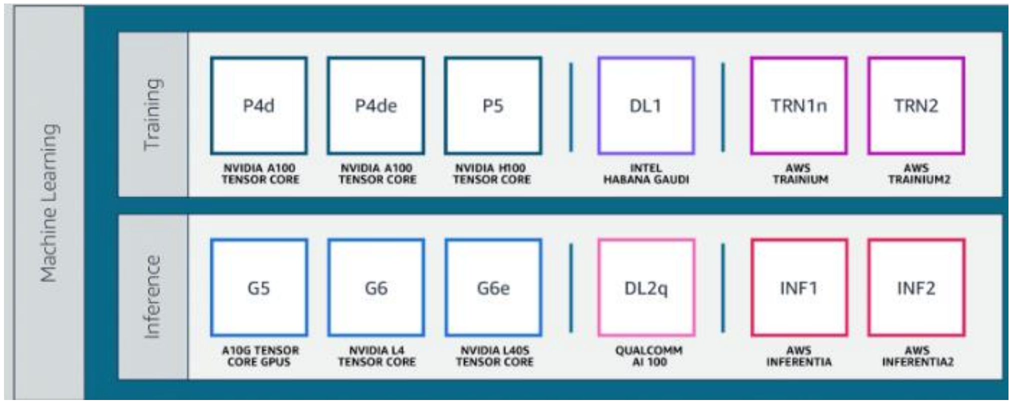
Hybrid of gradient descent and SGD, this approach uses a smaller dataset or a batch of records, also called batch size, to update your parameters.

- Mini-batch gradient descent updates more than gradient descent while having less erratic or noisy updates as compared to SGD. The user-defined batch size helps you fit the smaller dataset into memory. Having a smaller dataset helps the algorithms run on almost any average computer that you might be using.

2.2.2 Compute Environment



AWS Instances for ML:



AWS offers solutions for a variety of specific ML tasks, and this permits you to optimize on your particular use case scenarios.

AWS Container Services:

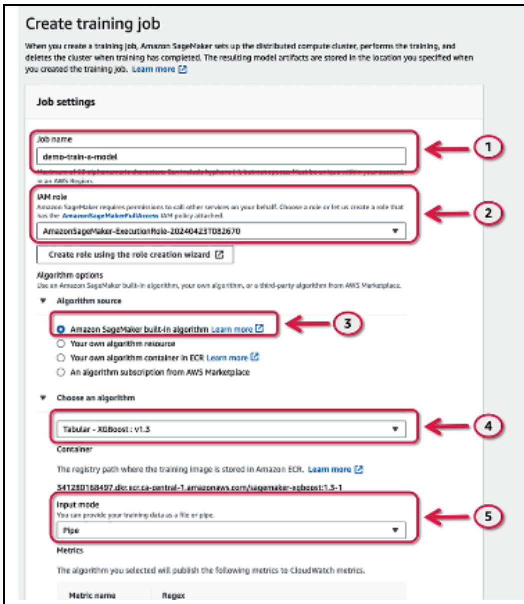
		Keyword
Amazon ECS	*ECS simplifies the process of running and managing containerized applications on AWS, offering various deployment options, and seamlessly integrating with other AWS services.*	*General/custom Container*
Amazon EKS	*Amazon EKS provides a fully managed Kubernetes control plane and seamless integration with other AWS services*	*Kubernetes*
AWS Fargate		*Fully Managed Container*
Amazon ECR	*ECR makes it easy to store, manage, and deploy container images.*	*Container Registry*

Containers in SageMaker for ML model generation

1. ***SageMaker managed container images***
   - You can use built-in training algorithms included in these containers, ML Framework, settings, libraries, and dependencies included in these container images, but provide your own custom training algorithms. This approach is referred to as script mode.
2. ***Customer-managed container images (BYOC)***
   - You can build your own container using the Bring Your Own Container (BYOC) approach if you need more control over *the algorithm, framework, dependencies, or settings.*
   - *Some industries might require BYOC or BYOM to meet regulatory and auditing requirements.*

2.2.3 Train a model

Create training job

<table>
<tr>
<td>


</td>
<td>

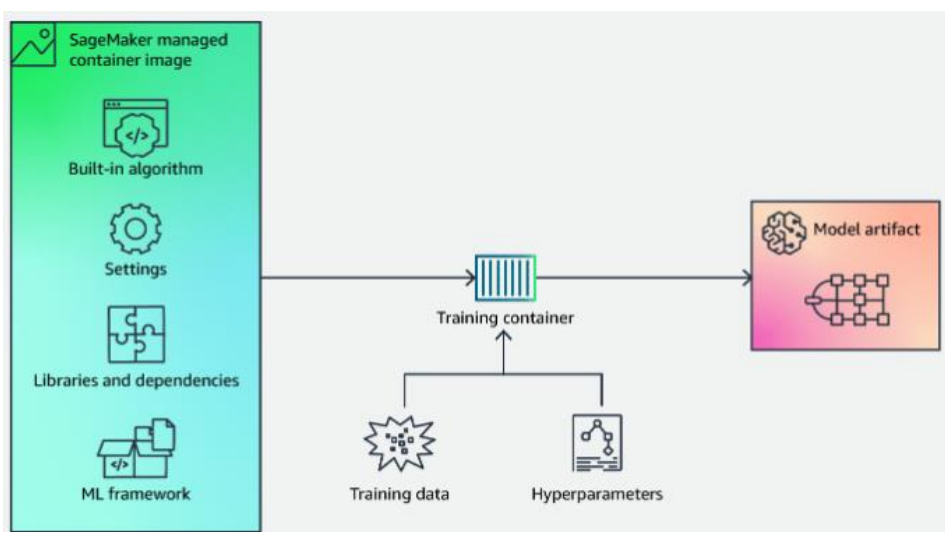
- Create IAM role
- Chose algorithm source (built-in, etc.)
- Choose algorithm
- Configure compute resource
- Set hyperparameters (+ default)
- Specify data type
- Choose Data source (default S3)

Model created

- Store in S3
- Package and distribute
- Register model (in registry)

</td>
</tr>
</table>

Train a model



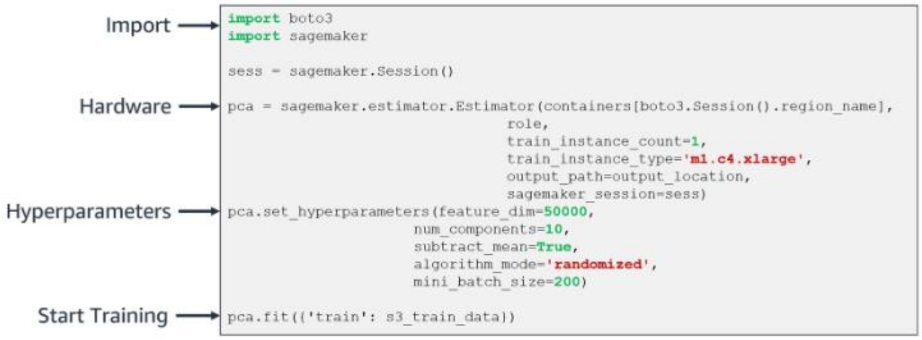
For built-in algorithms, the only inputs you need to provide are the

- *training data*
- *hyperparameters*
- *compute resources.*

Amazon SageMaker training options

When it comes to training environments, you have several to choose from:

- *Create a training job using SageMaker console (see the Creating a Training Job Using the Amazon SageMaker Console lesson for an example using this method).*
- *Use AWS SDKs for the following:*
    - *The high-level SageMaker Python SDK*
    - *The low-level SageMaker APIs for the SDK for Python (Boto3) or the AWS CLI*



Training data sources

- S3
- Amazon EFS
- Amazon FSx for Lustre

Training ingestion modes

	Pipe mode	File mode	Fast File mode
What	SageMaker streams data directly from Amazon S3 to the container, without downloading the data to the ML storage volume	SageMaker will download the training data from S3 to the provisioned ML storage volume. Then it will mount the directory to the Docker volume for the training container.	SageMaker can stream data directly from S3 to the container with no code changes. Users can author their training script to interact with these files as though they were stored on disk.
Pros	Improve training performance by reducing the time spent on data download	In a distributed training setup ,the training data is distributed uniformly across the cluster.	Fast File mode works best when the data is read sequentially.
Cons		Manually ensure ML storage volume has sufficient capacity to accommodate data from Amazon S3.	Augmented manifest files are not supported. The startup time is lower when there are fewer files in the S3 bucket provided.

When to use which

	Pipe mode	File mode	Fast File mode
large training datasets	X		X
training algorithm reads data sequentially		X	X

Amazon SageMaker training – Script mode

Amazon SageMaker script mode provides the flexibility to develop custom training and inference code while using industry-leading machine learning frameworks.

Steps to bring your own script using SageMaker script mode

1. Use your local laptop or desktop with the SageMaker Python SDK. You can get different instance types, such as CPUs and GPUs, but are not required to use the managed notebook instances.
2. Write your training script.
3. Create a SageMaker estimator object, specifying the
   a) training script
   b) instance type
   c) other configurations.
4. Call the *fit* method on the estimator to start the training job, passing in the training and validation data channels.
5. SageMaker takes care of the rest. It pulls the image from Amazon Elastic Container Registry (Amazon ECR) and loads it on the managed infrastructure.
6. Monitor the training job and retrieve the trained model artifacts once the job is complete.

Example

```
import sagemaker
from sagemaker.pytorch import PyTorch

# Define the PyTorch estimator
estimator = PyTorch(
    entry_point="train.py",
    role=sagemaker.get_execution_role(),
    instance_count=1,
    instance_type="ml.p3.2xlarge",
    framework_version="1.12.0",
    py_version="py38",
)
# Launch the training job
estimator.fit({"train": "s3://my-bucket/train_data"})
```

In this example, the *PyTorch* estimator is configured with the training script using the entry_point: *train.py*, instance type *ml.p3.2xlarge*, and other settings. The *fit* method is called to launch the training job, passing in the location of the training data.

Reducing training time

Amazon SageMaker script mode provides the flexibility to develop custom training and inference code while using industry-leading machine learning frameworks.

a) ***Early stopping:***
Early stopping is a regularization technique that shuts down the training process for a ML model when the model's performance on a validation set stops improving.

How early stopping works in Amazon SageMaker

Amazon SageMaker provides a seamless integration of early stopping into its hyperparameter tuning functionality, so users can use this technique with minimal effort. Here is how early stopping works in SageMaker:

- *a)* ***Evaluating objective metric after each epoch****: During the training process, SageMaker evaluates the specified objective metric (for example, accuracy, loss, F1-score) for each epoch or iteration of the training job.*
- *b)* ***Comparing to running median of previous training jobs***
- *c)* ***Stopping current job if performing worse than median****:*

b) ***Distributed training***

- ***A.*** ***Data parallelism*** *is the process of splitting the training set in mini-batches evenly distributed across nodes. Thus, each node only trains the model on a fraction of the total dataset.*

  Done in SageMaker using SageMaker distributed data parallelism (SMDDP) library

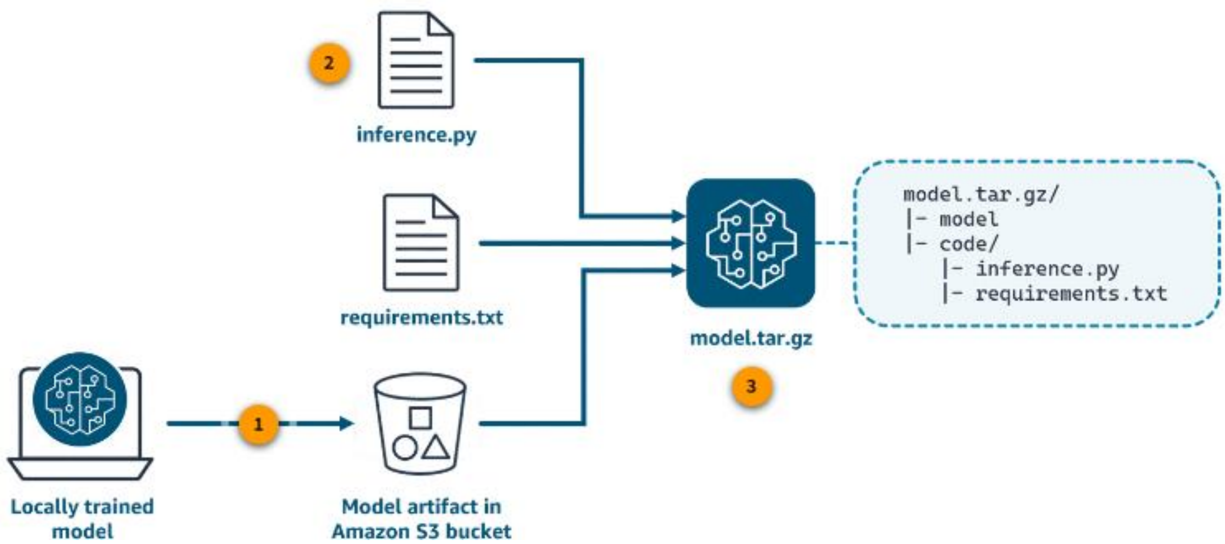
- *B.* ***Model parallelism*** *is the process of splitting a model up between multiple instances or nodes.*

  SageMaker model parallelism library v2 (SMP v2)

Guidance on choosing data parallelism compared to model parallelism

- *If model can fit on a single GPU's memory but your dataset is large,* ***data parallelism*** *is the recommended approach. It splits the training data across multiple GPUs or instances for faster processing and larger effective batch sizes.*
- *If model is too large to fit on a single GPU's memory,* ***model parallelism*** *becomes necessary. It splits the model itself across multiple devices, enabling the training of models that would otherwise be intractable on a single GPU.*

Building a deployable model package



Step 1: *Upload your model artifact to Amazon S3.*

Step 2: *Write a script that will run in the container to load the model artifact. In this example, the script is named inference.py. This script can include custom code for generating predictions, as well as input and output processing. It can also override the default implementations provided by the pre-built containers.*

To install additional libraries at container startup, add a requirements.txt file that specifies the libraries to be installed by using pip.

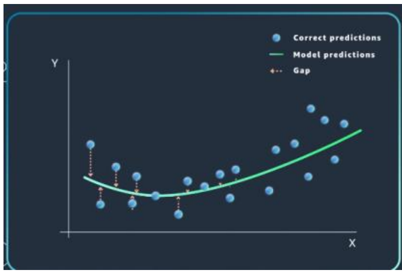
Step 3: *Create a model package that bundles the model artifact and the code. This package should adhere to a specific folder structure and be packaged as a tar archive, named model.tar.gz, with gzip compression.*

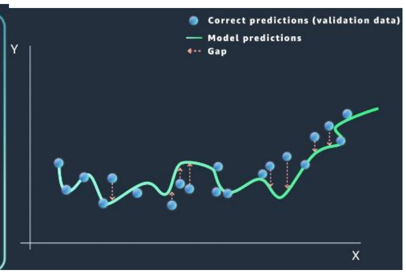
2.3 Refine Models

2.3.1 Evaluating Model Performance

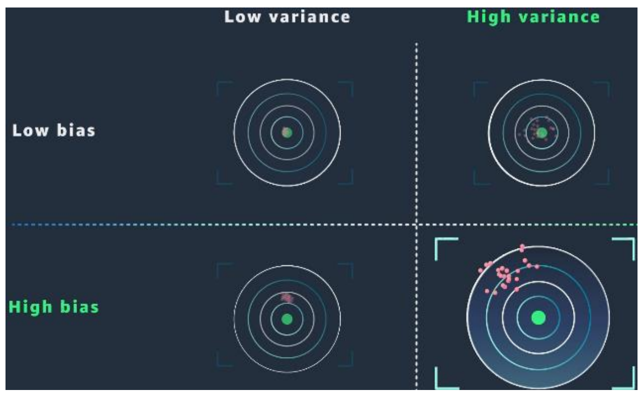
Bias and Variance

a) ***What are these***





Bias ***Variance***



Common cause of high model bias vs Variance

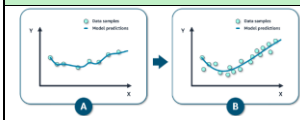
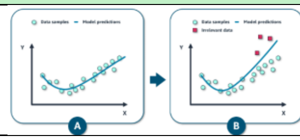
Bias	*Variance*
The model is too simple	The model is too complex
Incorrect modeling or feature engineering	Too much irrelevant data in training dataset
Inherited bias from the training dataset	Model trained for too long on training dataset

2.3.2 Model Fit (Overfitting and Underfitting)

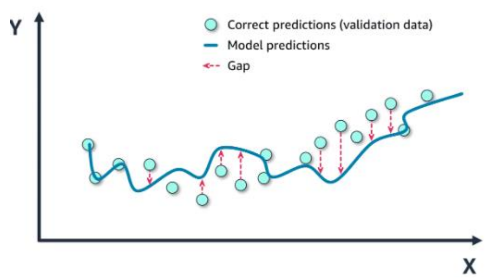
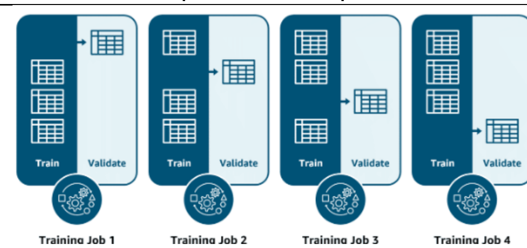
1. Overfit/Underfit

- **Overfit**

1. Reasons

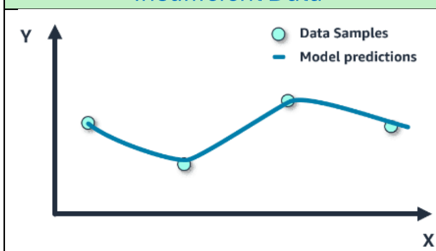
Training data too small	*Too much irrelevant data*	*Excessive training time*	*Overly complex architecture*
			
		Prolonged training on the same data can cause model to memorize training examples instead of learning underlying patterns.	*A model with too many parameters (weights and biases) can start memorizing the training data and noise.*

2. Detecting model overfitting

Check for model variance	**Use K-fold cross-validation**
If your model performed well with the training set but poorly with the validation set, it indicates high variance and overfitting.	You split the input data into k subsets of data, also called folds. You train multiple models on this dataset. For each model, you change which fold is set aside to be the evaluation dataset. This process is repeated k times
	

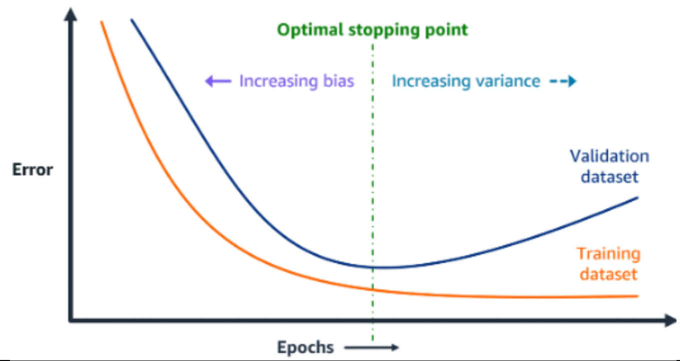
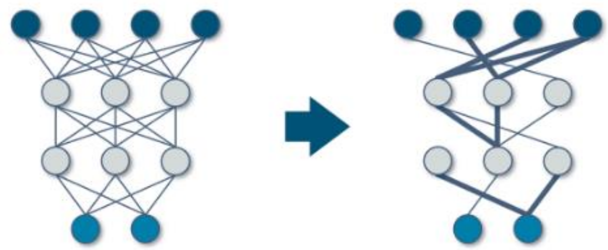
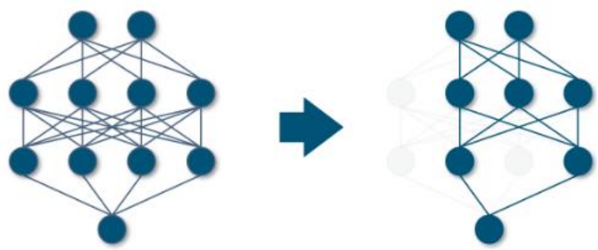
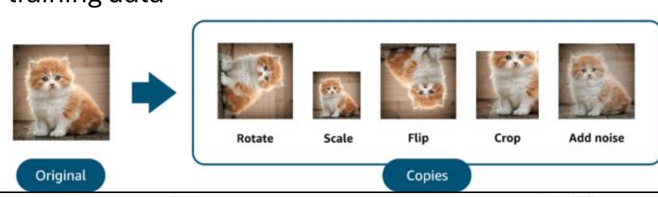
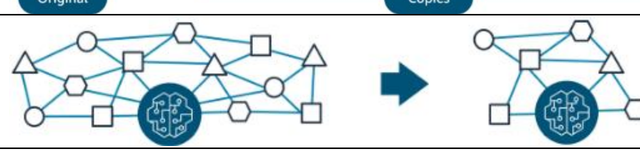
- **Underfit**

Reasons

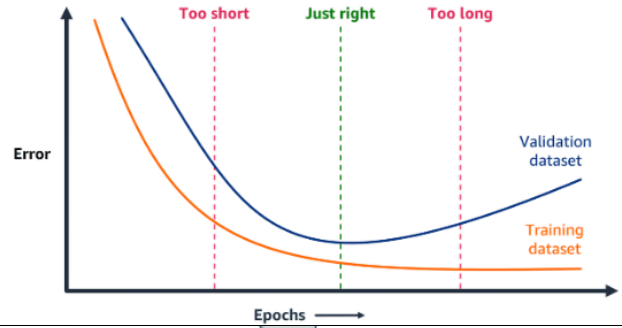
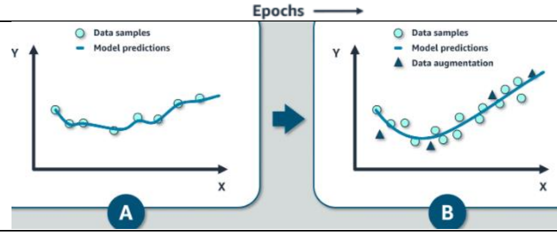
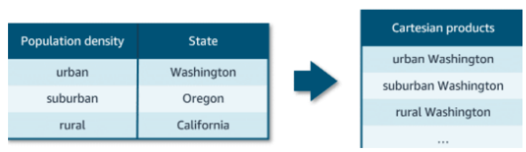
Insufficient Data	*Insufficient Training Time*	*Excessive training time*
	*Model might not have had the opportunity to learn the necessary patterns and relationships in the data*	*A model with too few parameters (weights and biases) will likely not be able to accurately capture the nonlinear relationships or intricate patterns within the data instead of learning underlying patterns.*

b) Preventing Overfitting and Underfitting

a) Remediating Overfitting

Early stopping Pauses the training process before the model learns the noise in the data.	
Pruning Aims to remove weights that don't contribute much to the training process	
Regularization	**a) Dropout** Randomly drops out (or sets to 0), a number of neurons in each layer of the neural network during each epoch.  **b) L1 regularization** Push the weights of less important features to zero. **c) L2 regularization** Results in smaller overall weight values (and stabilizes the weights) when there is high correlation between the input features.
Data augmentation	perform data augmentation to increase the diversity of the training data 
Model architecture simplification	

b) Remediating Underfitting

<table>
<tr>
<td>Train for an appropriate length of time</td>
<td>
</td>
</tr>
<tr>
<td>Use a larger number of data points</td>
<td>
</td>
</tr>
<tr>
<td>Increase model flexibility</td>
<td>a) Add New Domain -specific features
For example, if you have length, width, and height as separate variables, you can create a new volume feature to be a product of these three variables.
b) Add Cartesian Products
Consider generating new features through Cartesian products.


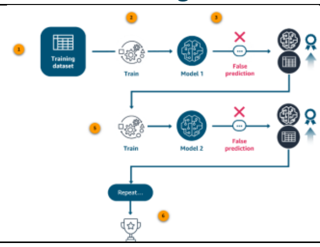
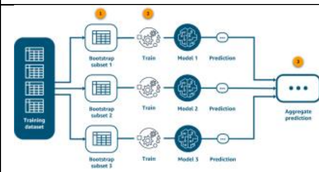
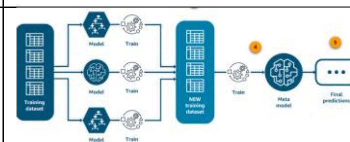
c) Change Feature Engineering
Adjusting types of feature processing techniques can increase model flexibility. E.g., in NLP tasks, you increase the size of n-grams, etc.
d) Decrease Regularization
Such as reducing the regularization strength or using a different regularization technique,</td>
</tr>
</table>

c) *Combining models for improved performance*

Ensembling: *Process of combining the predictions or outputs of multiple machine learning models to create a more accurate and robust final prediction.*

The idea behind ensembling is that by combining the strengths of different models, the weaknesses of individual models can be mitigated. This leads to improved overall performance.

The following are commonly used ensembling methods:

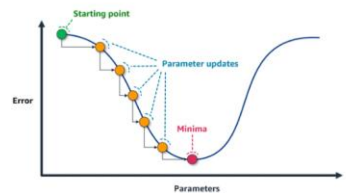
	Boosting	Bagging	Stacking
	trains different machine learning models sequentially	combines multiple models trained on different datasets	combines both
When	**Accuracy**	**Interpretability**	
Prevents	• **Overfitting** • underfitting	• **overfitting**	
			
	a) Adaptive Boosting (AdaBoost) b) Gradient Boosting (GB) c) Extreme Gradient Boosting (XGBoost)		

Boosting algorithm		
Adaptive Boosting (AdaBoost)	**Gradient Boosting (GB)**	**Extreme Gradient Boosting (XGBoost)**
classification	• classification • regression	• classification • regression • large datasets and big data applications.
Bagging (bootstrap aggregation)		
Random forests		
Stacking		
??		

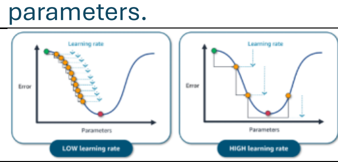
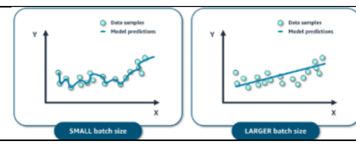
2.3.3 Hyperparameter Tuning

Benefits of Hyperparameter tuning

a) Impact of Hyperparameter tuning on model performance



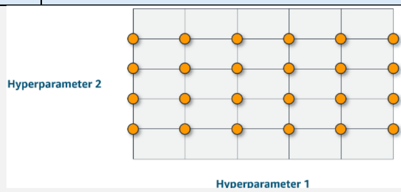
b) Types of hyperparameters for tuning

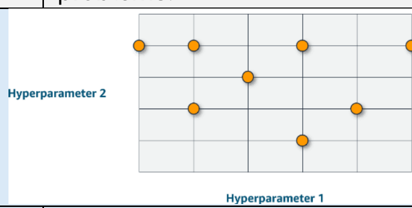
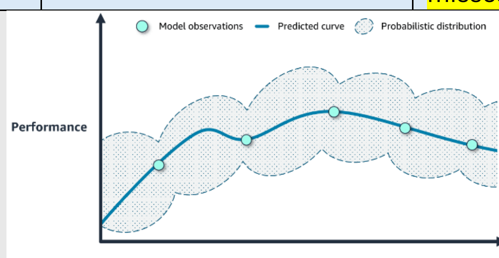
Gradient Descent algorithm			
	Learning Rate	Batch Size	Epochs
	Determines the step size taken by the algorithm during each iteration. This controls the rate at which the training job updates model parameters.	# of examples used in each iteration	# of passes through the entire training dataset
			
Careful	If the learning rate is too high, the algorithm might overshoot the optimal solution and fail to converge.	A larger batch size can lead to faster convergence but might require more computational resources.	However, too many can result in overfitting.

Neural networks			
# of layers	# of neurons in each layer	Choice of activation functions	Regularization Techniques
more layers -> more complex	more neurons -> more processing power	introduce non-linearity into the neural network Common activation functions include: • Sigmoid function • Rectified Linear Unit (ReLU) • Hyperbolic Tangent (Tanh) • Softmax function	helps prevent overfitting . Common regularization techniques • L1 /L2 regularization • Dropout • Early stopping
increasing the depth of a network risks overfitting.	Increasing number of neurons risks overfitting		

Decision Tree		
Maximum Depth of tree	# of neurons in each layer	Choice of activation functions
helps manage complexity of the model and prevent overfitting	Sets a threshold that the data must meet before splitting a node. prevents the tree from creating too many branches. This also helps to prevent overfitting	Options to select how algorithm evaluates node splits: • **Gini impurity:** measures purity of data and the likelihood that data could be misclassified. • **Entropy:** Measures randomness of data. The child node that reduces entropy the most is the split that should be used.

Hyperparameter tuning techniques

<table>
<tr><th></th><th></th><th>Pros</th><th>Cons</th><th>When</th></tr>
<tr><td>Manual</td><td></td><td>When you have a good understanding of the problem at hand</td><td>time-consuming</td><td>Domain knowledge, and prior experience with similar problems</td></tr>
<tr><td rowspan="2">Grid search</td><td colspan="4">

Systematic and exhaustive approach to hyperparameter tuning. It involves defining all possible hyperparameter values and training and evaluating the model for every combination of these values.</td></tr>
<tr><td></td><td>Reliable technique, especially for smaller-scale problems.</td><td>Computationally expensive.</td><td>Small scale and accuracy</td></tr>
<tr><td rowspan="2">Random search</td><td colspan="4">
</td></tr>
<tr><td></td><td>More efficient than Grid Search</td><td>Optimum hyperparameter combination could be missed.</td><td></td></tr>
<tr><td rowspan="2">Bayesian optimization</td><td colspan="4">

Uses the performance of previous hyperparameter selections to predict which of the subsequent values are likely to yield the best results.</td></tr>
<tr><td></td><td>can handle composite objectives
can also converge faster than random search.</td><td>More complex to implement.
Works sequentially, so difficult to scale.</td><td>multiple objectives and/or speed.</td></tr>
<tr><td rowspan="2">Hyperband</td><td colspan="4">

Dynamically allocates resources to well-performing configurations and stops underperforming ones early.</td></tr>
<tr><td></td><td>can train multiple models in parallel
can be a more efficient allocation of compute resources than grid search or random search</td><td>Not for regular alogos</td><td>Only be used for iterative algorithms like neural networks
For limited resources</td></tr>
</table>

Hyperparameter tuning using SageMaker AMT

STEPS

1. Define your environment and resources, such as output buckets, training set, and validation set.
2. Specify the hyperparameters to tune and the range of values to use for each of the following: **alpha**, **eta**, **max_depth**, **min_child_weight**, and **num_round**.
3. Identify the objective metric that SageMaker AMT will use to gauge model performance.
4. Configure and launch the SageMaker AMT tuning job, including completion criteria to stop tuning after the criteria have been met.
5. Identify the best-performing model and the hyperparameters used in its creation.
6. Analyze the correlation between the hyperparameters and objective metrics.

2.3.4 Managing Model Size

Model Size Overview

a) *Model Size considerations*

Bigger and more complex models can achieve higher accuracy on training data. However, there are several tradeoffs involved with large model sizes that must be considered.

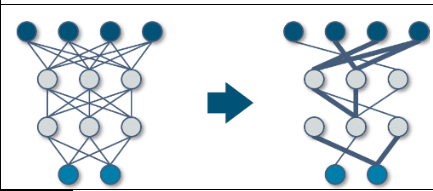
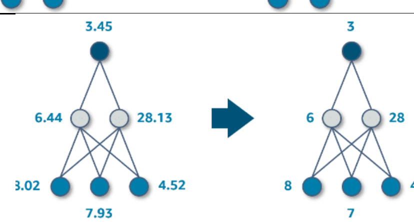
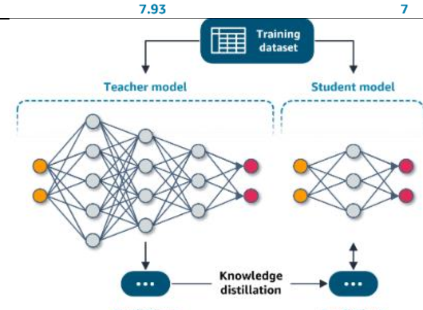
Pros

Smaller models	Larger models
Smaller models have several advantages, including faster training times, reduced memory usage, and lower computational costs. They can be particularly useful in real-time or resource-constrained scenarios where prediction speed and low latency are desired.	Larger models might perform better because they are more likely to have captured more relationships in the data.

Cons

Smaller models	Larger models
Small models might not perform as well because they are less likely to have captured complex patterns in the training data.	More relationships often come at the expense of faster deployment times, prediction latency, and greater compute resource requirements.

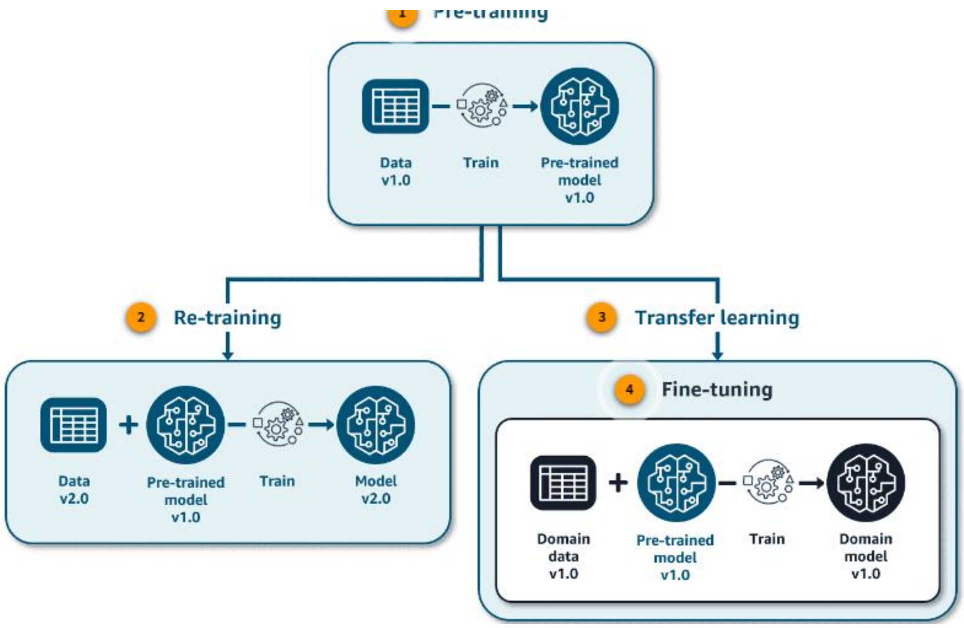
b) *Model size reduction technique: Compression*

Pruning Pruning is a technique that removes the least important parameters or weights from a model.	
Quantization Quantization changes the representation of weights to its most space-efficient representation. E.g., instead of a 32-bit floating-point representation of weight, quantization has the model use an 8-bit integer representation.	
Knowledge distillation With distillation, a larger teacher model transfers knowledge to a smaller student model. The student model is trained on the same dataset as the teacher. However, the student model is also trained on the teacher model's knowledge of the data.	

2.3.5 Refining Pre-trained models

Benefits of Fine tuning

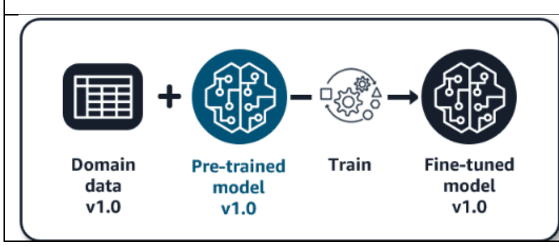
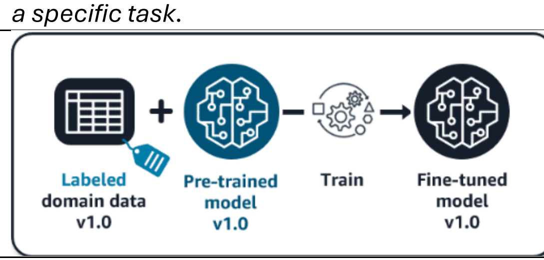
a) Where fine-tuning fits in the training process



Reasons for fine-tuning

- To customize your model to your specific business needs
- To work with domain-specific language, such as industry jargon, technical terms, or other specialized vocabulary
- To have enhanced performance for specific tasks
- To have accurate, relative, and context-aware responses in applications
- To have responses that are more factual, less toxic, and better aligned to specific requirements

b) Fine-tuning approaches

Domain adaption	***Instruction adaption***
Adapting foundation models to specific tasks by using limited domain-specific data	*Uses labeled examples to improve the performance of a pre-trained foundation model on a specific task.*
	

Fine tuning Models with Custom Datasets on AWS

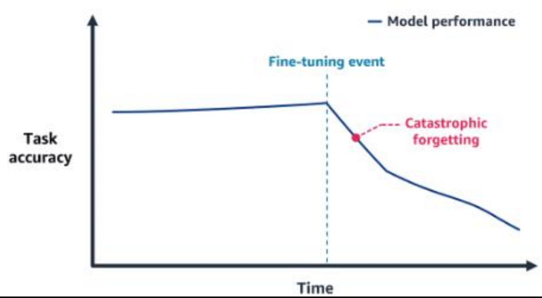
With a Custom Dataset Using Amazon SageMaker JumpStart	*With a Custom Dataset Using Amazon Bedrock*
1. Navigate to the model detail card of your choice in SageMaker JumpStart. 2. Edit your model artifact location. 3. Enter your custom dataset location. 4. Adjust the hyperparameters of the training job. 5. Specify the training instance type. 6. Start the fine-tuning job.	1. Choose a custom model in Amazon Bedrock. 2. Create a fine-tuning job. 3. Configure the model details. 4. Configure the job. 5. Select your custom dataset. 6. Adjust the hyperparameters.

Catatrosphic Forgetting Prevention

Catastrophic forgetting occurs when a model is trained on a new task or data, and it forgets previously learned knowledge.

a) Detecting

Plot your model's performance over time. If the model's performance on specific tasks decreases significantly after training on new data, it might be a sign of catastrophic forgetting.	Make sure your validation sets are representative of historic patterns in the data that are still relevant to the problem.



b) Preventing

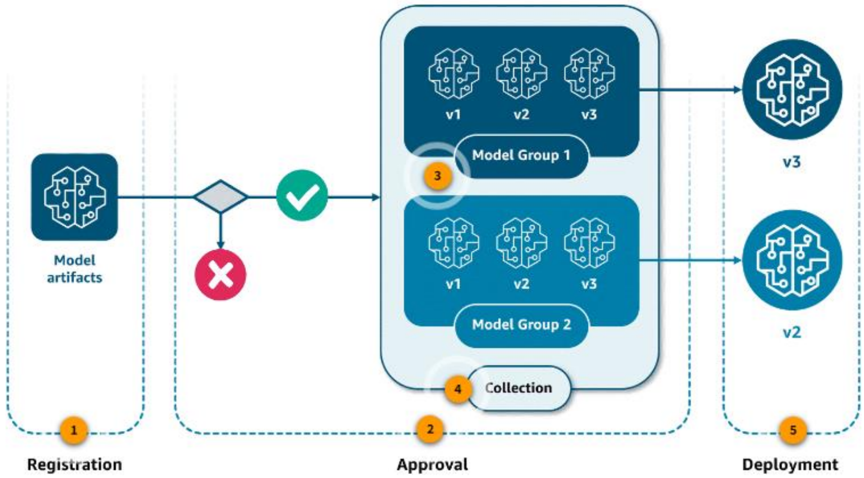
To prevent catastrophic forgetting, consider the following techniques:

1. **Elastic weight consolidation** (EWC): regularization technique that predicts which weights are important to performing previously learned tasks. It adds a penalty term to the loss function that protects these weights when the model is fine-tuned or re-trained on new task-specific data. Monitoring the EWC can indicate how much the model is forgetting older knowledge.
2. **Rehearsal**: This approach includes samples from the original training set during the fine-tuning or re-training process. During this process, the model rehearses the previous task to help it retain the learned knowledge.
3. **Model design**: You can also design your model with the appropriate amount of complexity to learn and retain patterns in the data. You can also use enough features to make sure your model captures diverse patterns in the data that differentiate between tasks.
4. **Renate**: This is an open source Python library for automatic model re-training of neural networks. Instead of working with a fixed training set, the library provides continual learning algorithms that can incrementally train a neural network as more data becomes available.

2.3.6 Model Versioning

Benefits of SageMaker Model Registry

a) SageMaker Model Registry



b) Benefits

- **Catalog models for production**
- **Manage model versions**
- **Control the approval status of models within your ML pipeline**

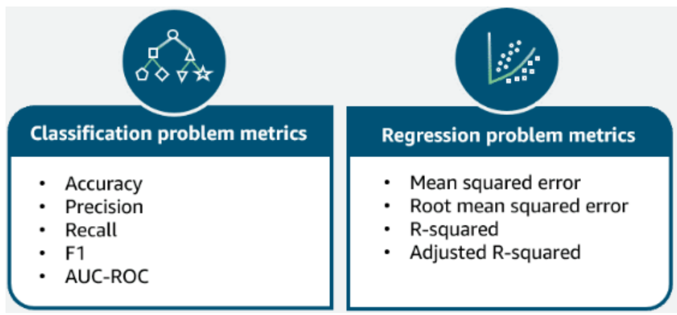
Registering and Deploying models with SageMaker Model Registry

a) SageMaker Model Registry

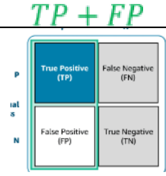
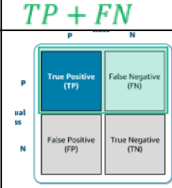
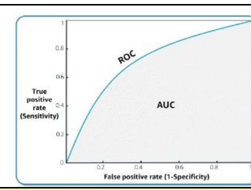
2.4 Analyze Model Performance

2.4.1 Model Evaluation

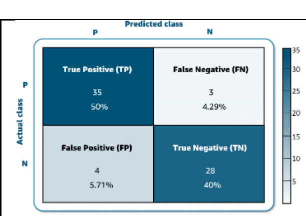
Model Metrics and Evaluation Techniques



a) *Classical Algorithm problems*

Accuracy	***Precision***	***Recall***	***F1 score***	*AUC Curve*
# of matching predictions to the total number of instances .	Proportion of positive that are correct.	proportion of correct sets that are identified as positive.	precision + recall	D
$\frac{TP+TN}{(TP+TN+FP+FN)}$	$\frac{TP}{TP+FP}$	$\frac{TP}{TP+FN}$	$\frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$	
				
	Cost of false positives is high	Cost of false negatives is high (Better to have false +**ves**)		
	classification Emails as spam or not	(e.g. diagnose cancer)		

New one: Heat Maps

graphical representations that use color coding to visualize the performance of a model		identify the areas of interest that have the most impact when your model is making predictions.

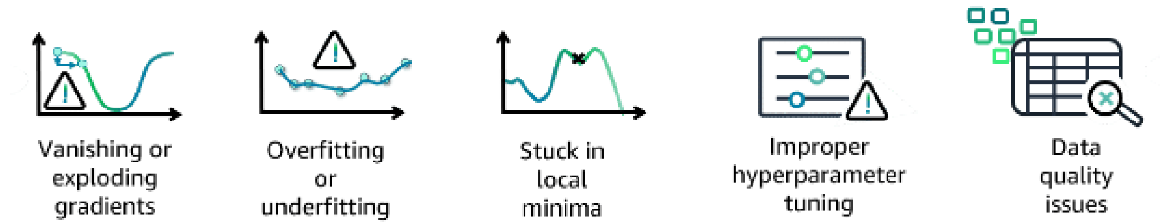
b) Regression Algorithm problems

Metric	Description	When to Use
Mean Squared Error (MSE)	Average of squared differences between predicted and actual values	• When larger errors should be penalized more • For comparing models (lower is better) • When the scale of errors is important
Root Mean Square Error (RMSE)	Square root of MSE, in the same units as the target variable	• When you want the error in the same units as the target variable • For easier interpretation of the error magnitude • When comparing models with different scales
R-Squared (R^2)	Proportion of variance in the dependent variable explained by the independent variables	• To understand how well the model fits the data • When you want a metric bounded between 0 and 1 • For comparing models across different datasets
Adjusted R-Squared	Modified version of R-Squared that adjusts for the number of predictors in the model	• When comparing models with different numbers of predictors • To penalize overly complex models • In feature selection processes

Model Convergence

Convergence refers to the ability of a model to reach an optimal solution during the training process. Failure to converge can lead to suboptimal performance, overfitting, or even divergence, where the model's performance deteriorates over time.

a) Impact of convergence

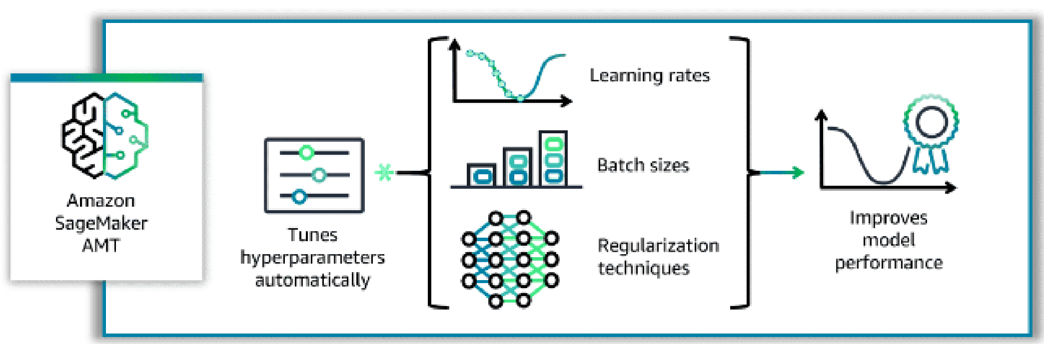


b) How SageMaker AMT (Compiler) helps in convergence issues of convergence

This is where SageMaker AMT can help. It can automatically tune models by finding the optimal combination of hyperparameters, such as

i. learning rate schedules
ii. initialization techniques
iii. regularization methods.

Improve CNN



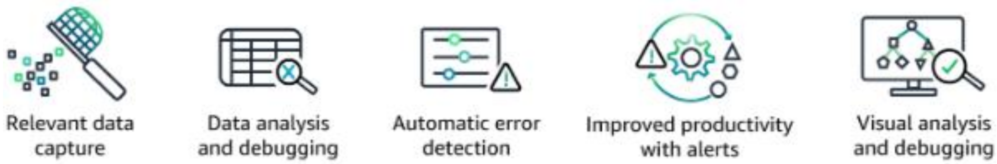
How SageMaker AMT improves issues with local maxima and local minima

When training a deep CNN for image classification tasks can encounter saddle points or local minima. This is because the loss function landscape in high-dimensional spaces can be complex. Having multiple local minima and saddle points can trap the optimization algorithm, leading to suboptimal convergence.

This is where SageMaker Training Compiler can help. It can automatically apply optimization techniques like

- tensor remapping
- operator fusion
- kernel optimization.

Debug Model Convergence with SageMaker Debugger



SageMaker Clarify and Metrics Overview

Bias metrics give visibility into model evaluation process

Data bias metrics	• **Class Imbalance**: Measures the imbalance in the distribution of classes/labels in your training data. • **Facet Imbalance:** Evaluates the imbalance in the distribution of facets or sensitive attributes, such as age, gender, or race across different classes or labels. • **Facet Correlation:** Measures the correlation between facets or sensitive attributes and the target variable.
Model bias metrics	• **Differential validity:** Evaluates the difference in model performance such as accuracy, precision, and recall across different facet groups. • **Differential prediction bias:** Measures the difference in predicted outcomes or probabilities for different facet groups, given the same input features. • **Differential feature importance:** Analyzes the difference in feature importance across different facet groups, helping to identify potential biases in how the model uses features for different groups.
Model explainability metrics	• **SHAP (SHapley Additive exPlanations):** Provides explanations for individual predictions by quantifying the contribution of each feature to the model's output. • **Feature Attribution**: Identifies the most important features contributing to a specific prediction, helping to understand the model's decision-making process. • **Partial Dependence Plots (PDPs)**: Visualizes the marginal effect of one or more features on the model's predictions, helping to understand the relationship between features and the target variable.
Data quality metrics	• **Missing Data:** Identifies the presence and distribution of missing values in your training data. • **Duplicate Data:** Detects duplicate instances or rows in your training data. • **Data Drift**: Measures the statistical difference between the training data and the data used for inference or production, helping to identify potential distribution shifts.