C2 → Build, Train, and Deploy ML Pipelines using BERT

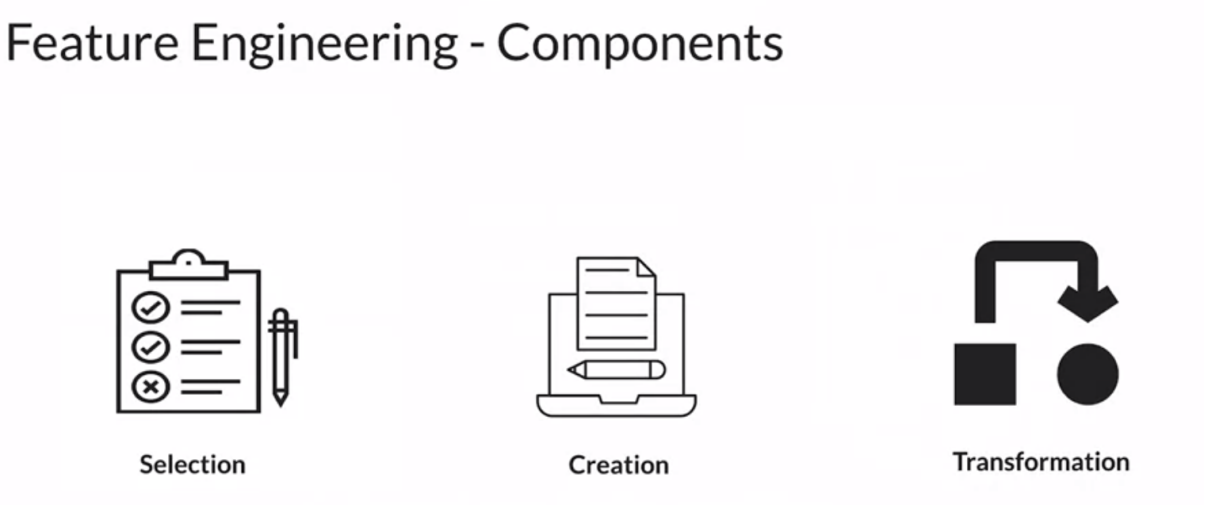
Created	@February 21, 2022 7:53 PM
Tag	Code Trainning

⚠️ Second Course of Practical Data Science Speciallization

Index

W1 - Feature Engineering and Feature Store

Feature Engineering



Feature Selection

Here you identify the appropriate data attributes or features to include in your training dataset, as well as filter out any redundant and unusable features from the training dataset

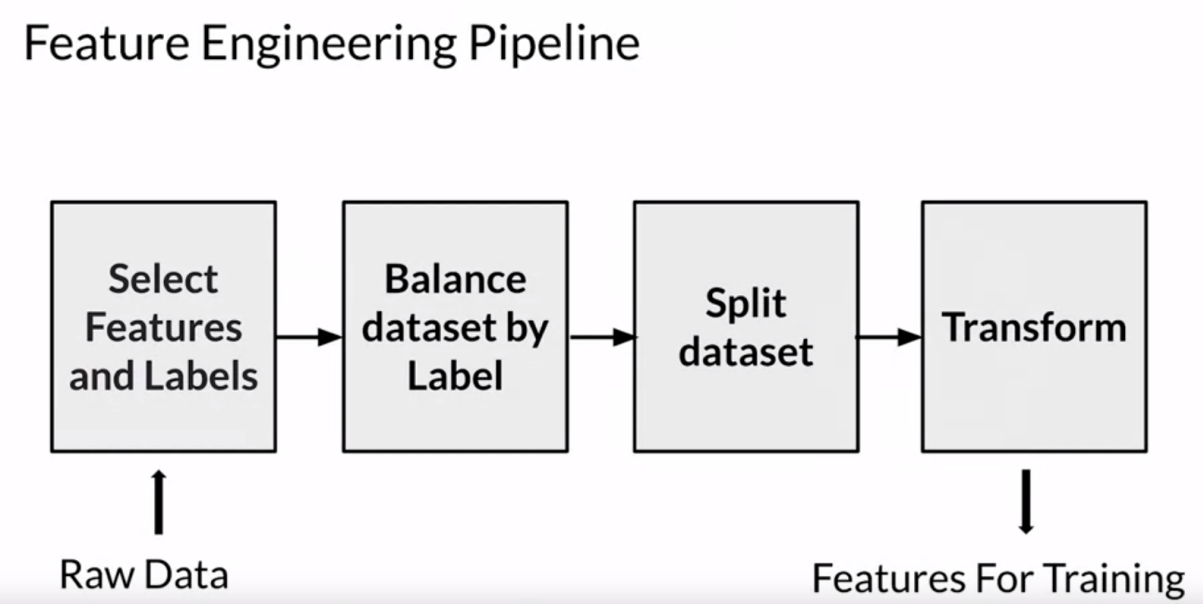
used for dimensionality reductions → saves time

Feature Creation

Combine diferent features or infering new features based on others

Feature Transformation

imputing missing values, encoding, scaling



Select: selecting the appropriate features along with selecting or creating appropriate labels

Balance: The dataset is then balanced so that there is a correct representation from all classes of the labels

Split: train, validation, test datasets

Transform: you perform the transformation techniques: encoding, scaling etc

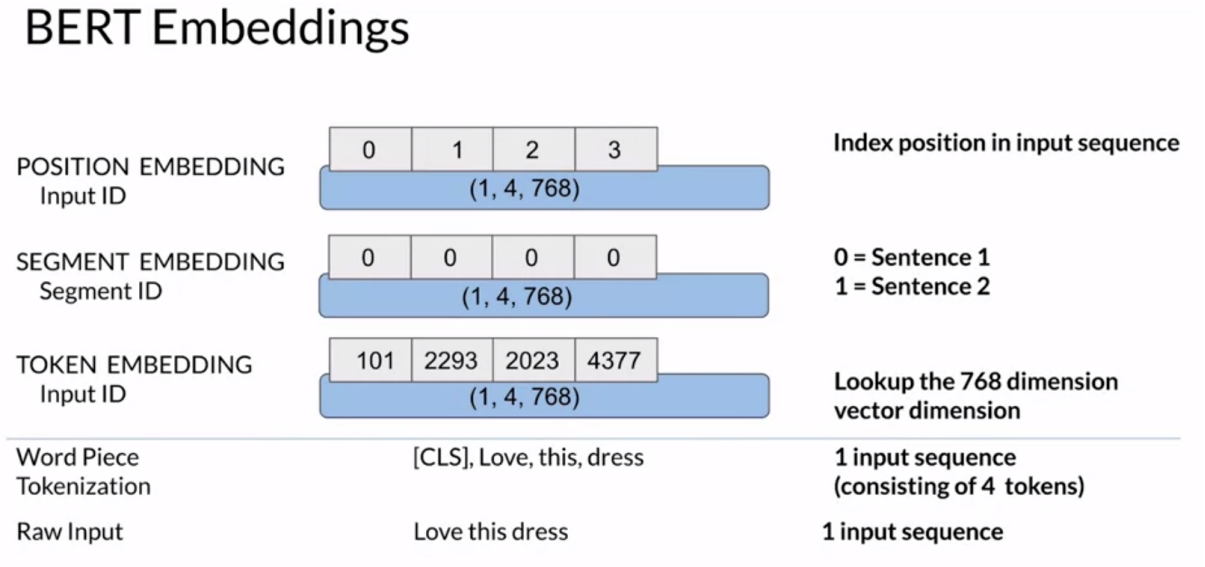
BERT and Feature Engineering at Scale

BERT: Bidirectional Encoder Representations from Transformers which is a neural network-based technique for training NLP-based models

BERT vs BlazingText

BlazingText is based on word2vec whereas BERT is based on transformer architecture. Both BlazingText and BERT generate word embeddings. However, BlazingText operates at word-level whereas BERT operates at a sentence level. Additionally, using the bidirectional nature of the transformer architecture, BERT can capture context of a word

BERT Algorithm Embeding



sentence: “Love this dress”

tokens: “ [CLS], Love, this, dress”

CLS → classification problem

SPE → it would appear if there were two sentences separating them

segment embeding → when there are more than when sentences, segment = 0 → 1º sentence, segment 1 = 2º sentence and so on

position embeding → position in the sequence/sentences

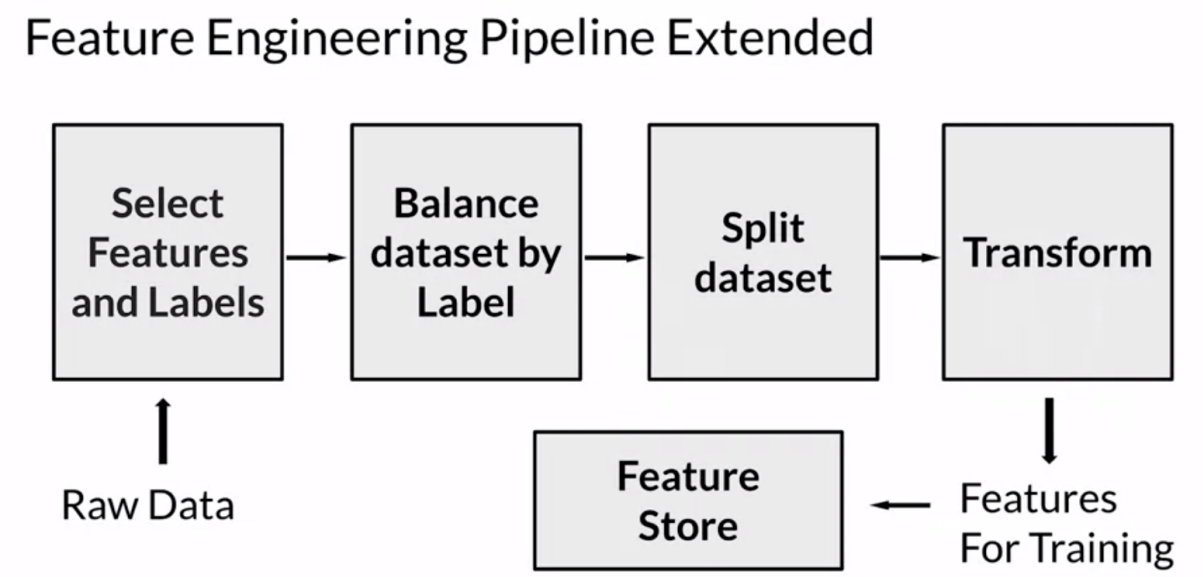
Feature Store

Feature store: is a repository, to store engineered features

Feature Store must be:

- centralized
- reusable
- discoverable

It would be integrated in the pipeline after the transformation of the features. You can do this AWS SageMaker Feature Store



SageMaker Feature Store:

- centralized repository for the whole organization
- it allows to reuse features across diferent projetcs
- allows to create features in batch or real time

feature group allows to group multiple features to treat them as a set and apply the same transformations, it must be stored in a S3 bucket

Reading Materials

RoBERTa: A Robustly Optimized BERT Pretraining Approach

Fundamental Techniques of Feature Engineering for Machine Learning

W2 → Train, Debug and Profile a ML Model

Pre-Trained models

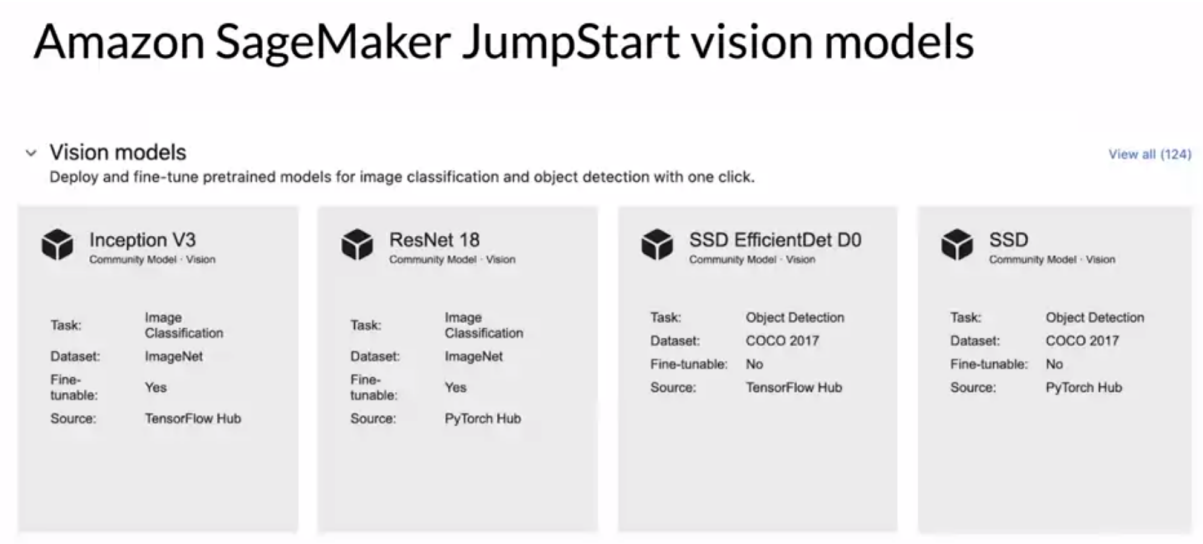
Model Pre-trained models vs built-in algorithms

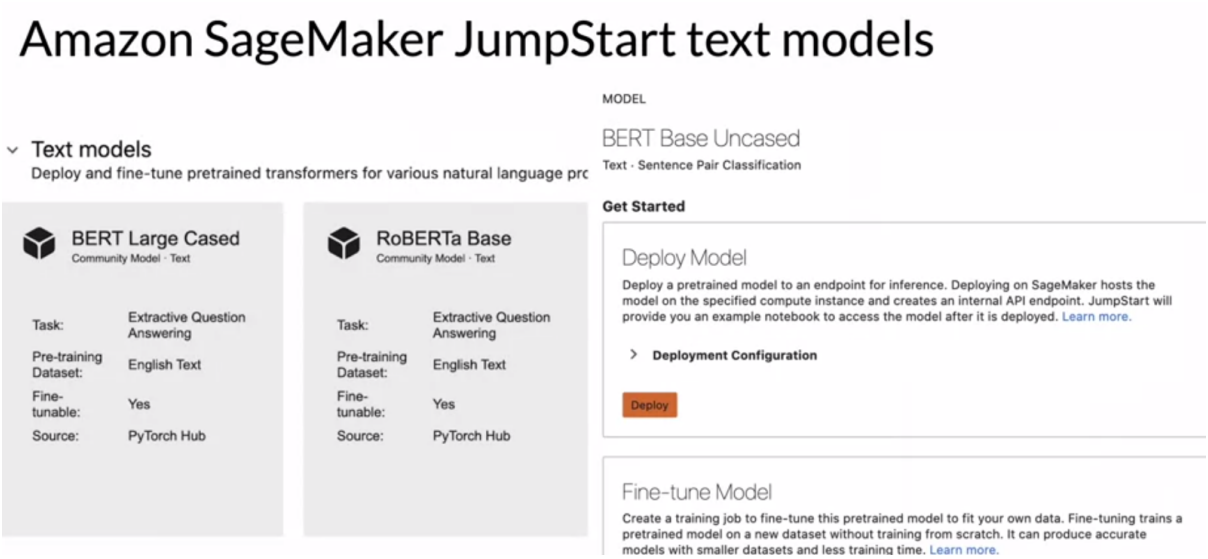
Before I discuss the concept of model pre training and fine tuning, I want to highlight the difference between buit in algorithms and pre trained models. In course one, you learned how to use built in algorithms, for example the blazing text algorithm to quickly train a model. The built in algorithm provided all required code to train the text classifier. You just pointed the algorithm to the prepared training data, this week you will work with pre trained models. The main difference here is that the model has already been trained on large collections of text data. You will provide the product previous data to adapt the model to your text domain and also provide your cast and model training code. Telling the pretrained model to perform a text classification task with a three sentiment classes.

Pre-Trained Models are trained using in unsupervised learning using large amounts of data such as wikipedia files. Nowadays there are already specific pre-trained models for medical data, or other fields. Also BERT has pre-trained models in other languages such as german, dutch, french.

Amazon SageMaker JumpStart

contains a variety of pre-trained models



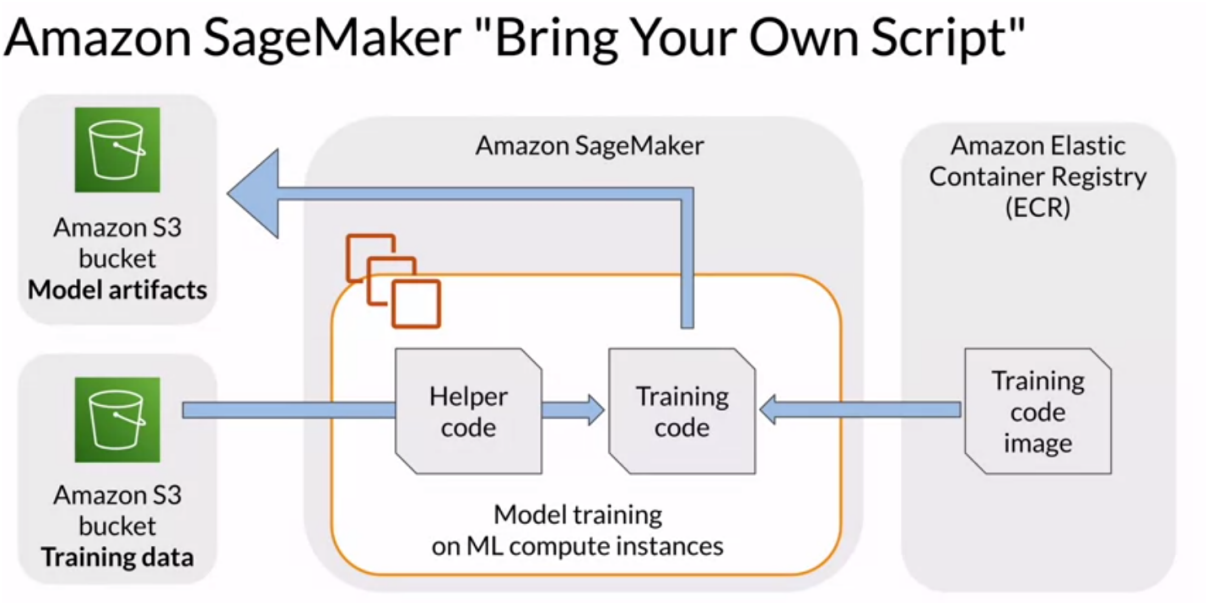


It also provides solutions for popular machine learning use cases, such as for example fraud detection in financial transactions, predictive maintenance, demand, forecasting and prediction and more. When you choose a solution, JumpStart provides a description of the solution and the launch button, there is no extra configuration needed. Solutions launch all of the resources necessary to run the solution including training and model hosting instances

Train a costum model with SageMaker

this option is called: “*bring your own script” or Scrip Mode*

For doing this you need to specify a training job containing the following info:



- S3 bucket to save artifacts
- S3 bucket of training data
- Image from ECR (pre built images)
- instance type and instance count

Steps necessary for doing this:

- **configure dataset and evaluation metrcis, code examples bellow**

```
from sagemaker.inputs import TrainingInput

s3_input_train_data = TrainingInput(s3_data="s3://...")
s3_input_validation_data = TrainingInput(s3_data="s3://...")
s3_input_test_data = TrainingInput(s3_data="s3://...")
```

Amazon S3

```
metric_definitions = [
     {'Name': 'validation:loss', 'Regex': 'val_loss: ([0-9\\.]+)'},
     {'Name': 'validation:accuracy', 'Regex': 'val_acc: ([0-9\\.]+)'}
]
```

Amazon CloudWatch Logs

```
metric_definitions = [
     {'Name': 'validation:loss', 'Regex': 'val_loss: ([0-9.]+)'},
     {'Name': 'validation:accuracy', 'Regex': 'val_acc: ([0-9.]+)'},
]
```

For example, these sample log lines...

```
[step: 100] val_loss: 0.76 - val_acc: 70.92%
```

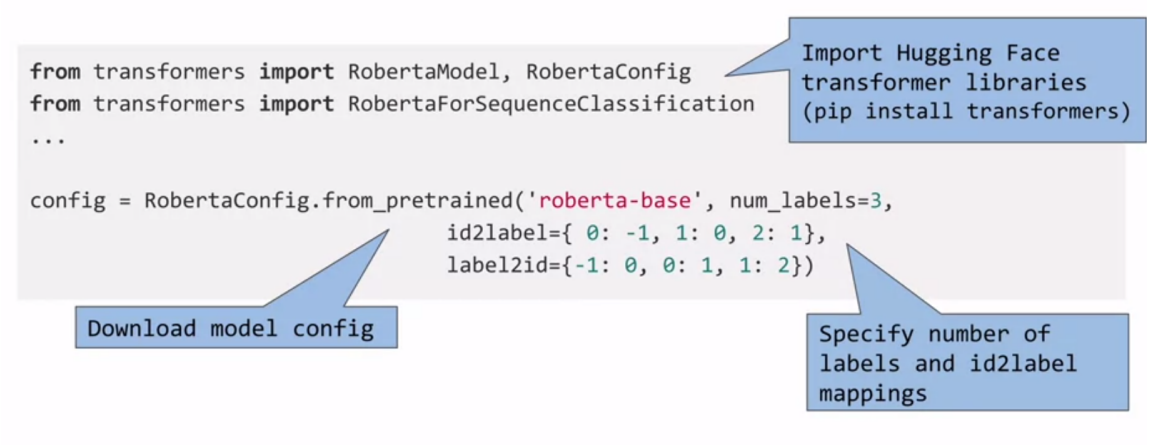
...will produce the following metrics in CloudWatch:

`validation:loss` = 0.76

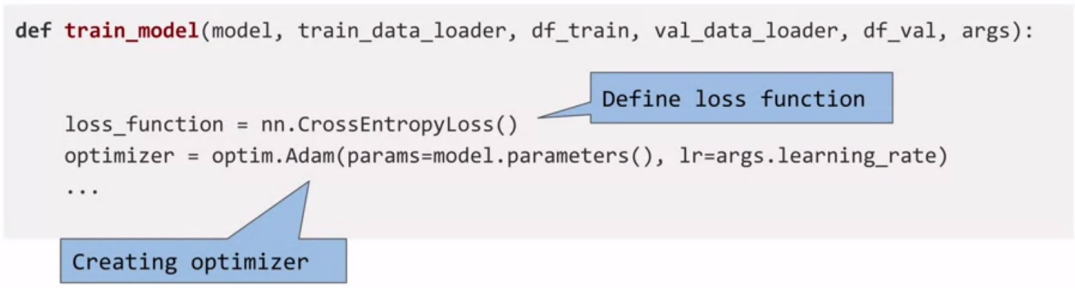
`validation:accuracy` = 70.92

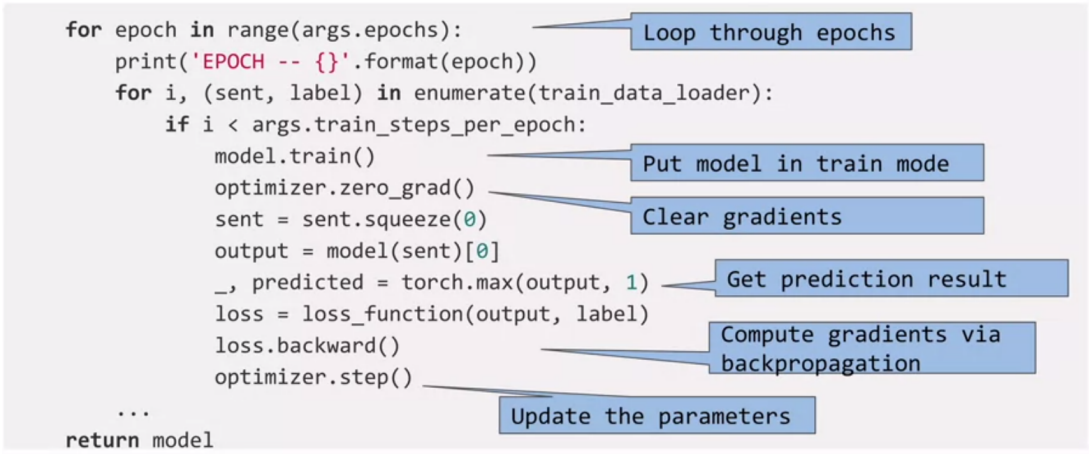
- **configure model hyperparameters:** epochs, learning_rate and so on.
    - For **NLP, max_seq_length:** maximum sequence length refers to the maximum number of input tokens you can pass to the bert model per sample. I choose the value of 128 because the word distribution of the reviews showed that one 100% of the reviews in the training data said have 115 words or less. It must be a power of 2.
- **provide costum training script (src/train.py):** here you're importing a pre-traind model and then adding providing the number of classes in your model (3 in this case) and also the dict of how each label transforms into the predicted numbers: *id2label and label2id,* neste caso as labels eram:

text	nº	y
negative	-1	0
neutral	0	1
positive	1	2



- With a pre-trained model at hand, you need to write the code to fine-tune the model. A function of optimization such as the following:





- **fit the model:** you need to create the estimator first and then you can call estimator.fit()

```
from sagemaker.pytorch import PyTorch as PyTorchEstimator

estimator = PyTorchEstimator(
    entry_point='train.py',
    source_dir='src',
    role=role,
    instance_count=1,
    instance_type='ml.c5.9xlarge',
    ...
```

Model training script

Directory containing additional assets such as requirements.txt

Compute instance types and count

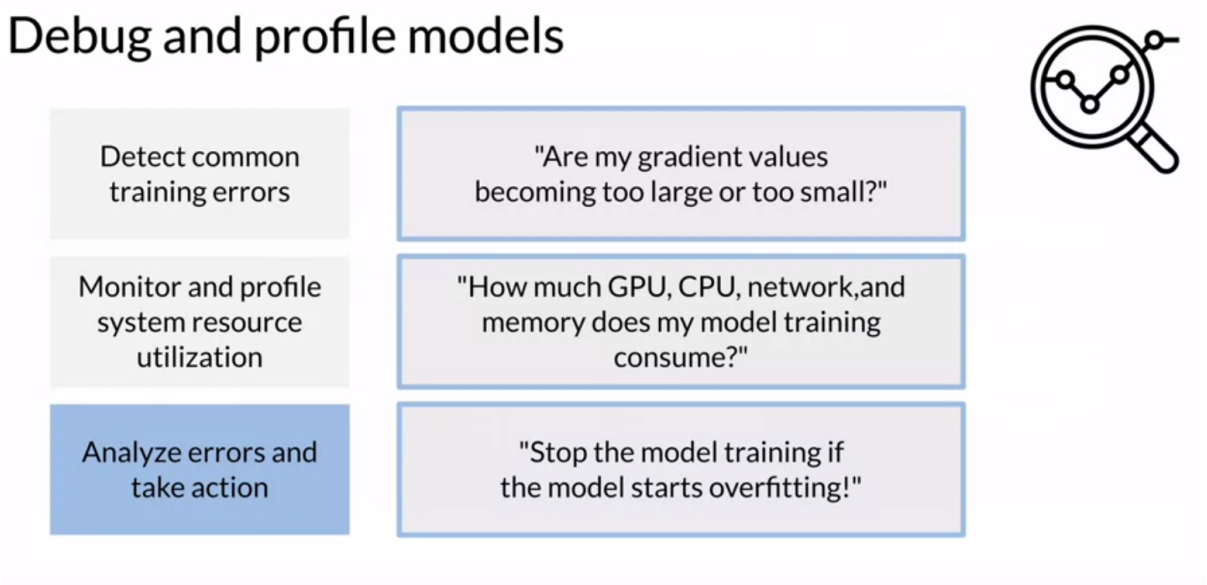
```
estimator = PyTorchEstimator(
    ...
    framework_version=<PYTORCH_VERSION>,
    hyperparameters=hyperparameters,
    metric_definitions=metric_definitions)

estimator.fit(...)
```

PyTorch version

Pass in hyperparameters and metrics

Start model training

Debug and Profile Models

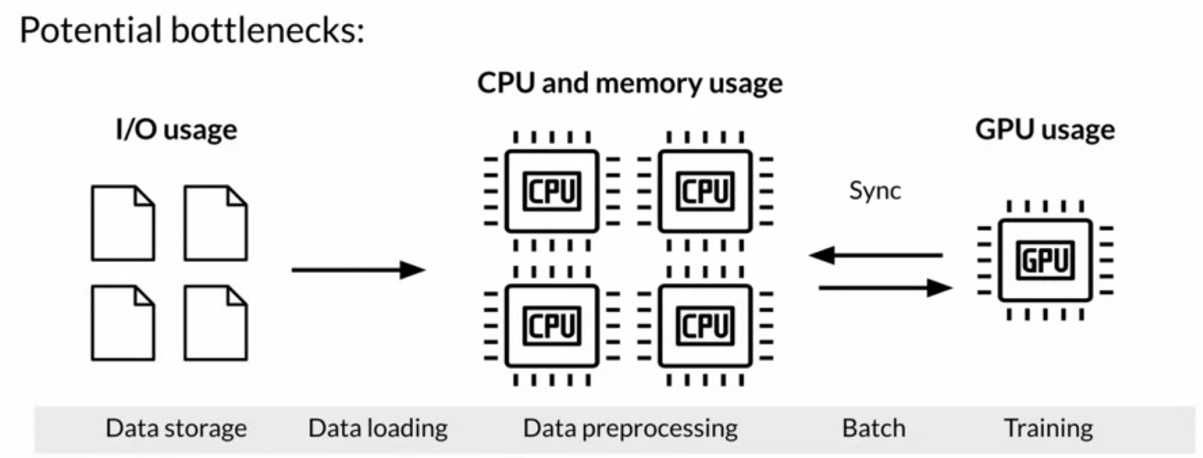


Detect common training errors:

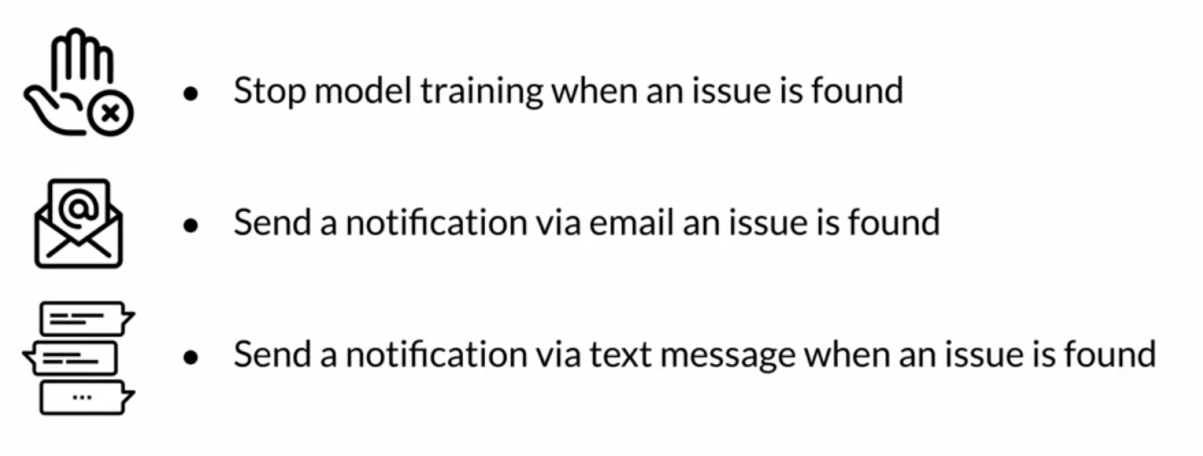
- **vanishing gradient:** Deep neural networks typically learn through back propagation, in which the models losses trace back through the network. The neurons weights are modified in order to minimize the loss. If the network is too deep, however, the learning algorithm can spend its whole lost touch it on the top layers and waits in the lower layers, never get updated

- **exploding gradient**: the learning algorithm might trace a series of errors to the same neuron resulting in a large modification to that neurons wade that it imbalances the network
- **bad initialization**: Initialization assigns random values to the model parameters. If all parameters have the same initial value, they received the same gradient and the model is unable to learn. Initializing parameters with values that are too small or too large may lead to vanishing or exploding gradients again
- **overfitting**: the training loop consists of training and validation. If the model's performance improves on a training set but not on a validation data set, it's a clear indication that the model is overfitting. the model's performance initially improves on the validation set but then begins to fall off, training needs to stop to prevent the overfitting

Monitor and profile system resource utilization



Analyse errors and take action



Amazon Debugger

Capture real-time debugging data during the model training metrics such as:

- **system metrics** (CPU, GPU, and memory utilization)
- f**ramework metrics** (convolutional operations in the forward pass, batch normalization operations in backward pass. And a lot of operations between steps and gradient descent algorithm operations to calculate and update the loss function)
- **output tensors** (scaler metrics: accuracy and loss, matrices: weights and layers))

In case Debugger detects an issue, for example, the model starts to over fit. You can use Amazon CloudWatch events to create a trigger to send you a text message, email you the status or even stop the training job. You can also analyze the data in your notebook environment using the Debugger SDK. Or you can visualize the training metrics and system resources using the SageMaker Studio IDE

the outputs allow you to also save costs by analysing if the CPU/GPU are being underused. if so you can sclae down to save costs, still maintaining the same performance

Reading Material

- PyTorch Hub
- TensorFlow Hub
- Hugging Face open-source NLP transformers library
- RoBERTa model
- Amazon SageMaker Model Training (Developer Guide)
- Amazon SageMaker Debugger: A system for real-time insights into machine learning model training
- The science behind SageMaker’s cost-saving Debugger
- Amazon SageMaker Debugger (Developer Guide)

- Amazon SageMaker Debugger (GitHub)

W3 → Deploy End-to-End ML Pipelines

Machine Learning Operations (MLOps) Overview

MLOps practices aim to be able to deliver machine learning workloads quickly to production while still maintaining high quality consistency and ensuring end-to-end traceability.

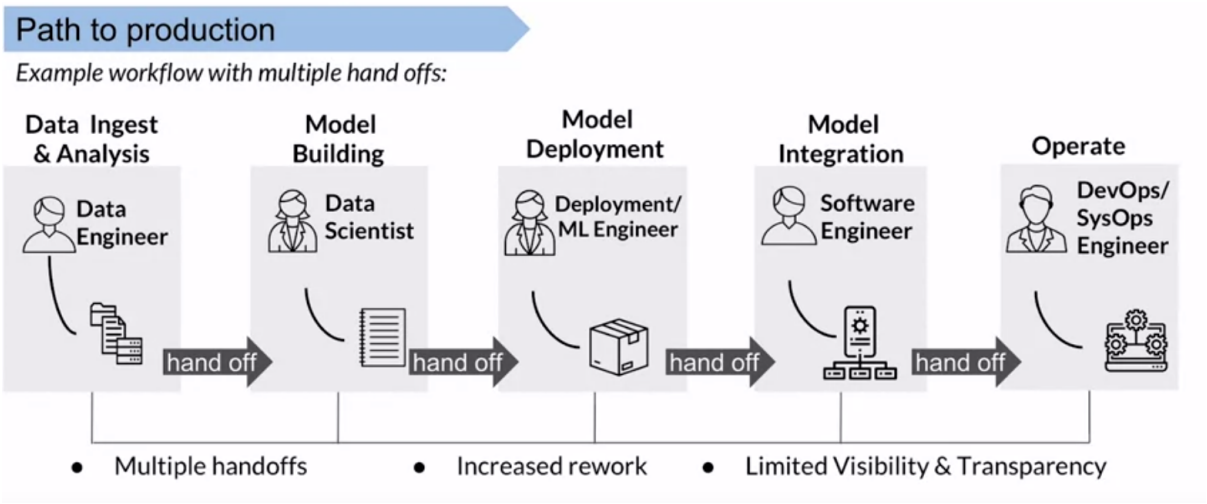
A few considerations in ensuring your models have a **path to production:**

- ML Development Life cycle (MLDC) ≠ Software Dev Life cycle (SDLC):
  - more testing
  - more artifacts and models for versioning
  - different tasks in the pipeline
  - harder to plan from a PM prespective
  - longer experimentation cycles
- a model might not be a stand alone, sometimes the model require specific tasks of pre-processing and so on. It is also important to integrate the estimations in the future application in which they’re gonna be used such as a web app...
- multiple personas are included from different areas and with different priorities and needs
- integrate your pipeline with company and IT practices

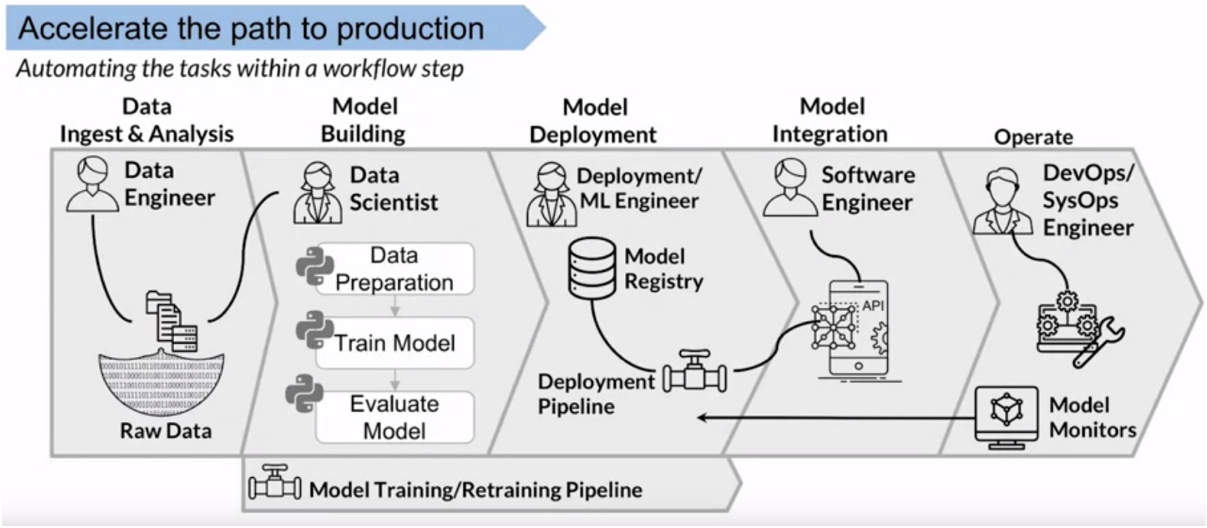
These considerations include/span **People , Processes and Technology** which is the scope of MLOps

Operationalizing ML

Without any MLOps practices applied:



With MLOps tecniques



Model Registry

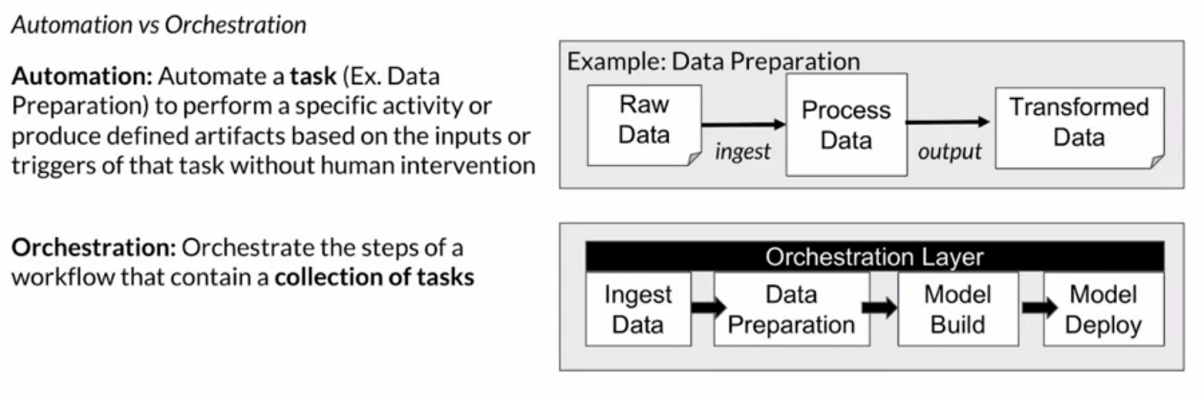
Model Registry holds key metadata and information about how your model was built, how the model performed with evaluation metrics. It's no longer a manual hand-off of a black box model. A Model Registry can also be used to trigger downstream deployment workflows as well

Model Monitors

These monitors include traditional systems or performance monitors, but they also include model specific monitors like checking for things like model quality drift or

data drift. As you can see here, visibility into those monitors is provided back to the personas that are involved in creating that end-to-end workflow

Automation VS Orchestration



Creating ML Pipelines

The pipeline will be started through pipeline triggers.

Traceability is a benefit of a ML pipeline, it allow you to keep track of versions both for code and all artifacts such as datasets

You can create triggers for data quality , bias and schema inside the ML pipeline in an automated form.

Model Lineage & Artifact Tracking

Model lineage

Essentially refers to understanding and tracking all of the inputs that were used to create a specific version of a model.

What is Model Lineage?

For **EACH** version of a trained model:

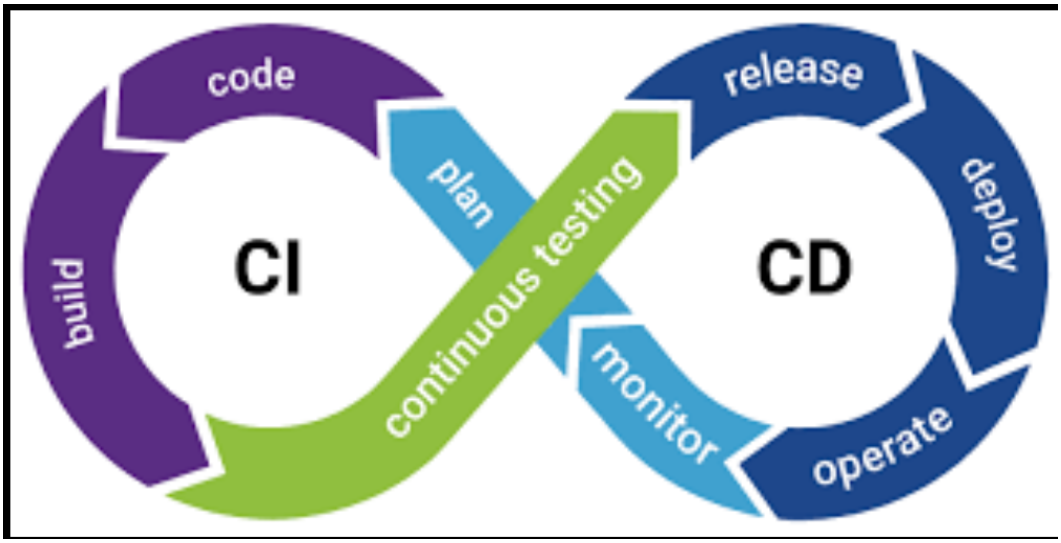
- ❑ Version(s) of data used
- ❑ Version(s) of code/hyperparameters used
- ❑ Version(s) of algorithm/framework
- ❑ Version(s) of training docker image
- ❑ Version(s) of packages/libraries

ML pipelines with SageMaker

SageMaker Pipelines allows:

- create & visualize automated workflows
- chose the best model to deploy
- automatic tracking models
- bring CI/CD techniques to ML

CI/CD → Continuous Integration / Continuous Delivery



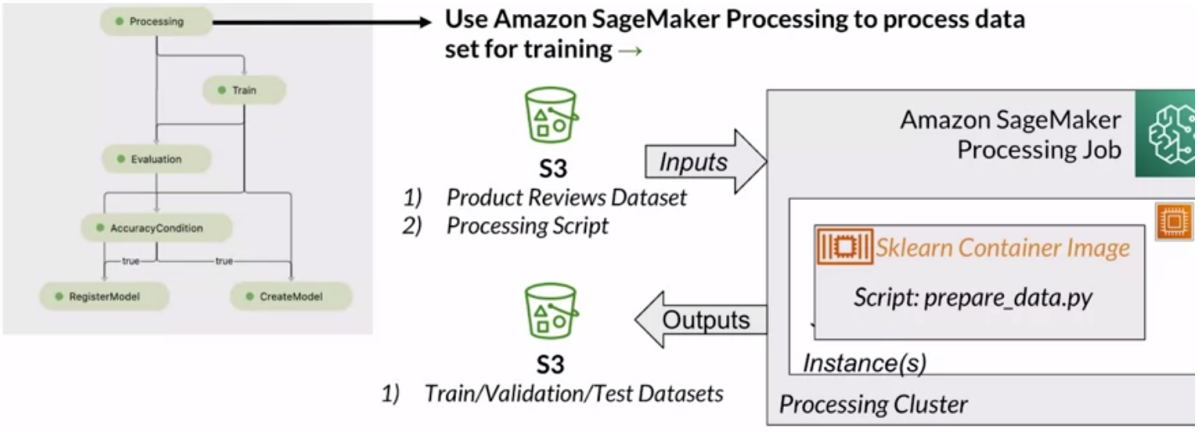
SageMaker Pipeline components:

- **pipelines**
  - model building
  - batch predictions
- **model registery**
  - model metadata
  - deployment approval
- **projects:**
  - built-in project templates
  - incorporating CI/CD practices

Pipelines

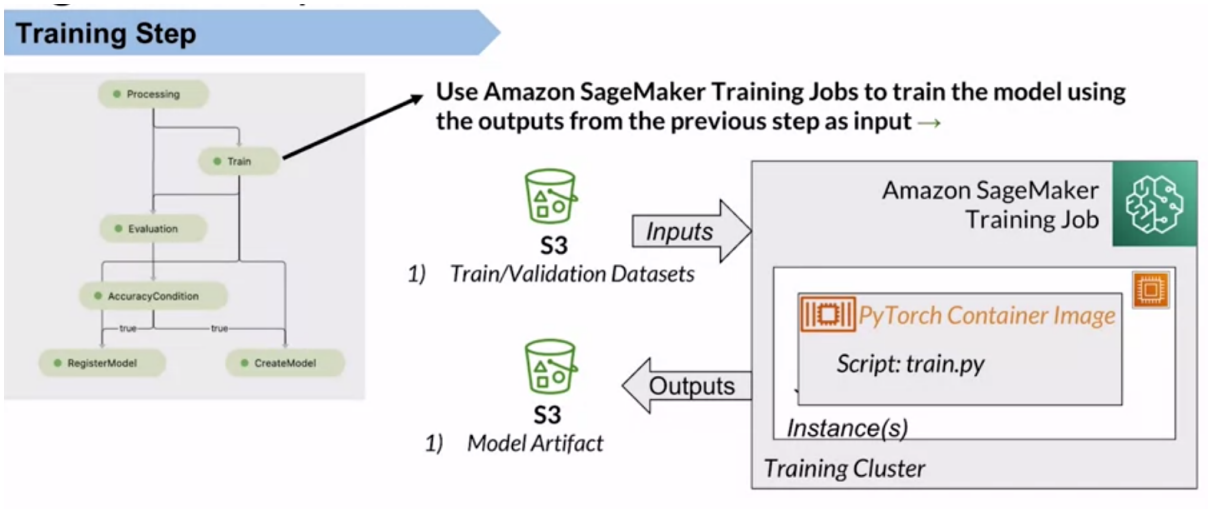
> ⚠️ details on the code, in the course slides as well as in the course labs

Processing step: inputs the raw data and pre-processes it and divide it into train/test/val datasets, the output will be the input of the training job

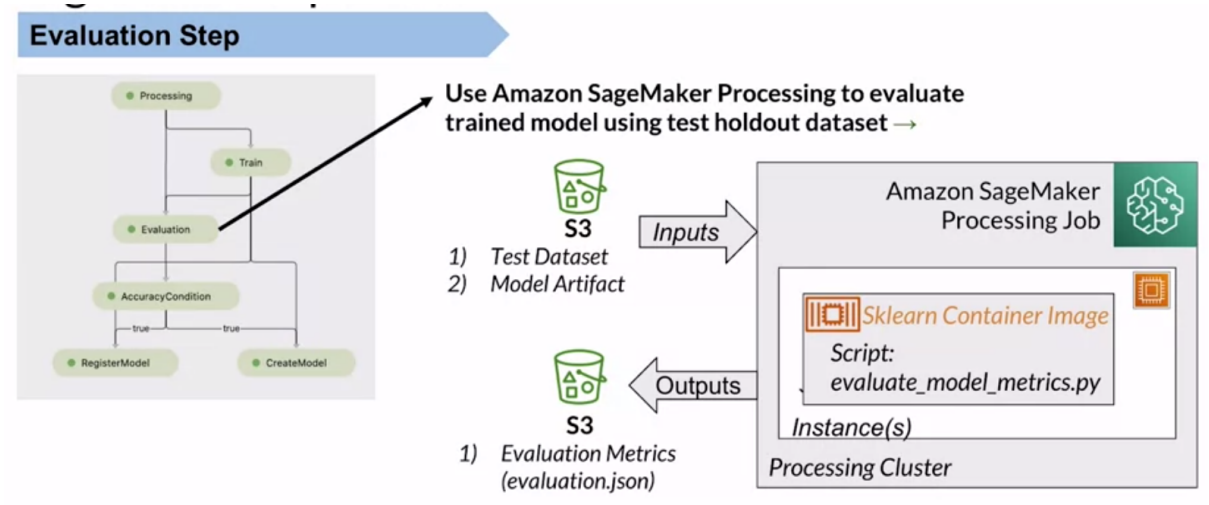


training step: the output is the model artifact. The inputs also contains the hyperparameters of the model, the inputs, the instances to use and the training

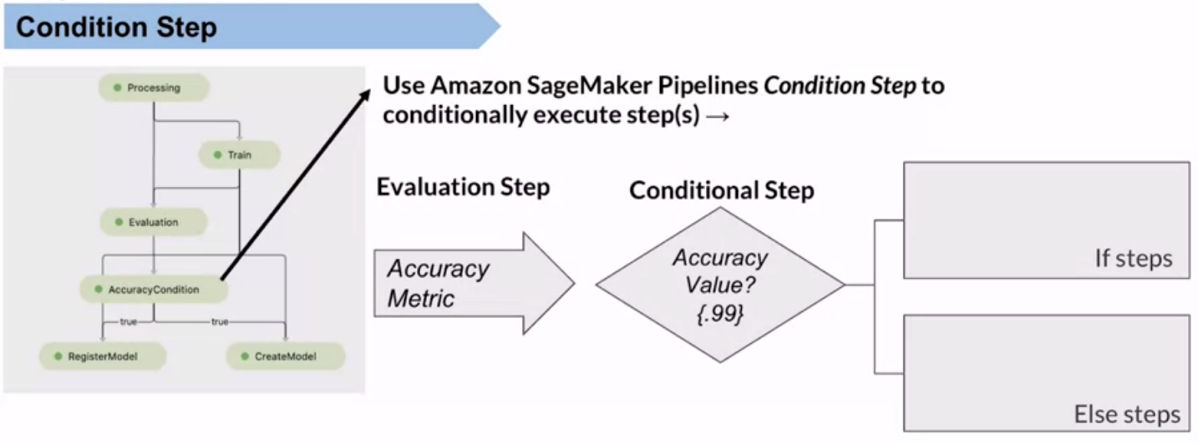
script, using the built in training steo (T*rainingStep())*



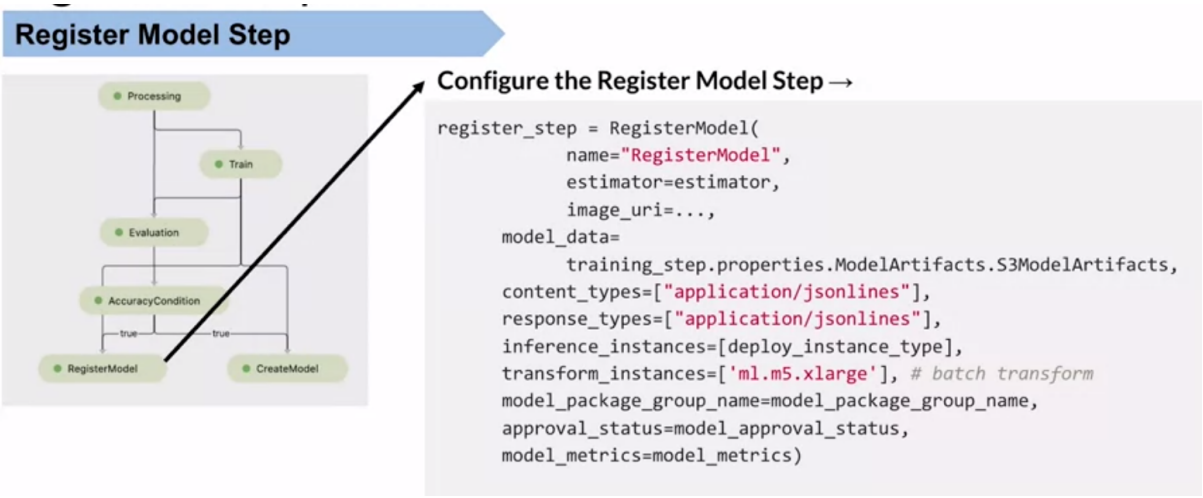
evaluating step: inputs include the test dataset , the trained model artifact (that's stored in S3 and was produced as a result of your previous training step) and Python script to use for model evaluation. the output will be saved into S3 bucket. the ouput contains a property file that will have the metric value that can later be used for a condtional step



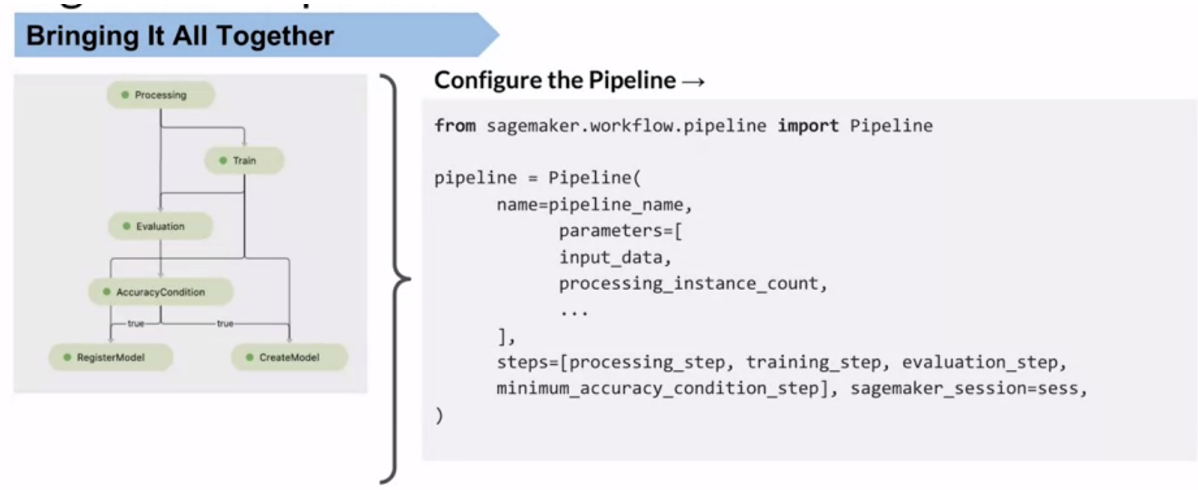
conditional step: evaluates a condition to determine wich is the next step in the pipeline



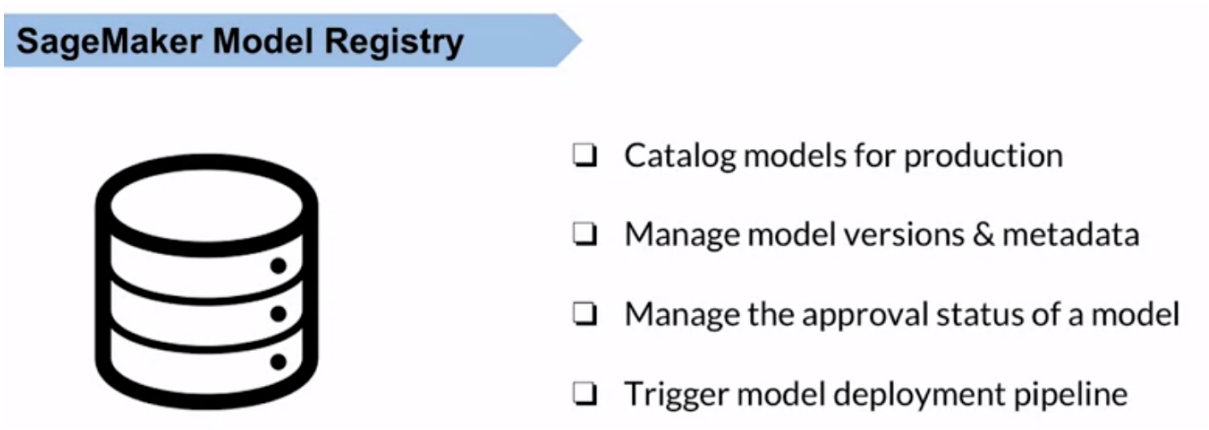
register model step: it saves the model



create pipeline: bring it all together for running the pipeline



Model Registry



SageMaker Projects

SageMaker projects integrates directly with SageMaker pipelines and SageMaker model registry. And it's used to create MLOps solutions to orchestrate and manage your end to end machine learning pipelines, while also incorporating CI/CD practices.

Reading Material

- A Chat with Andrew on MLOps: From Model-centric to Data-centric AI