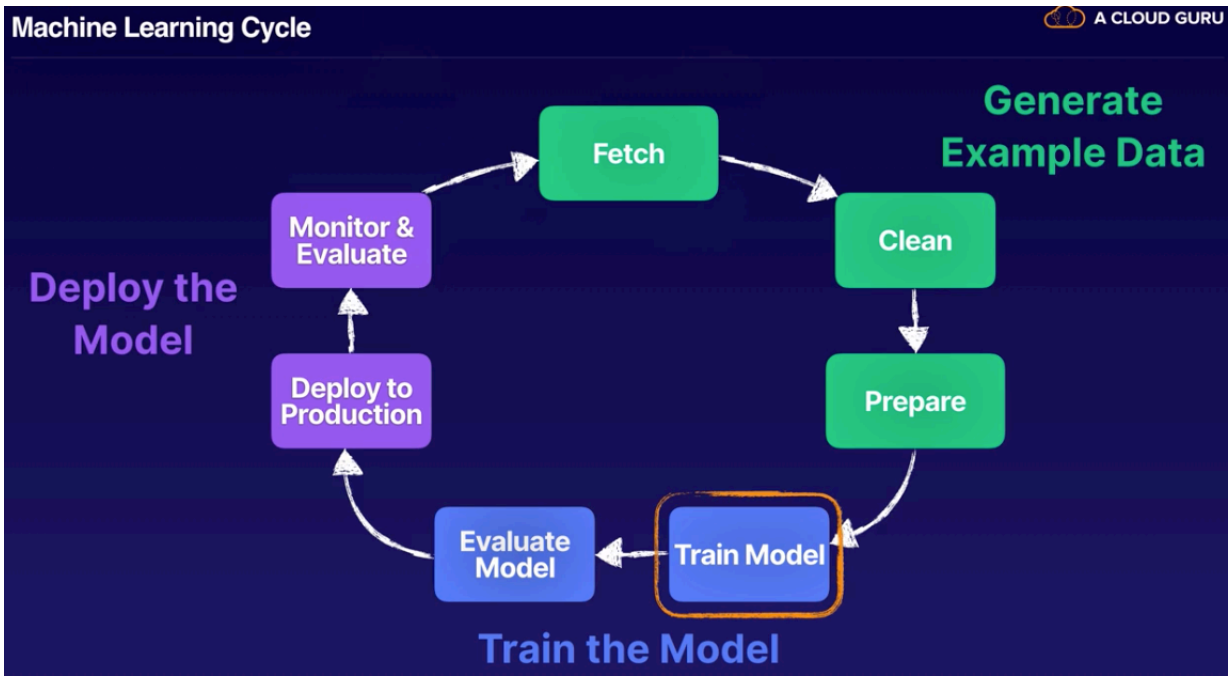
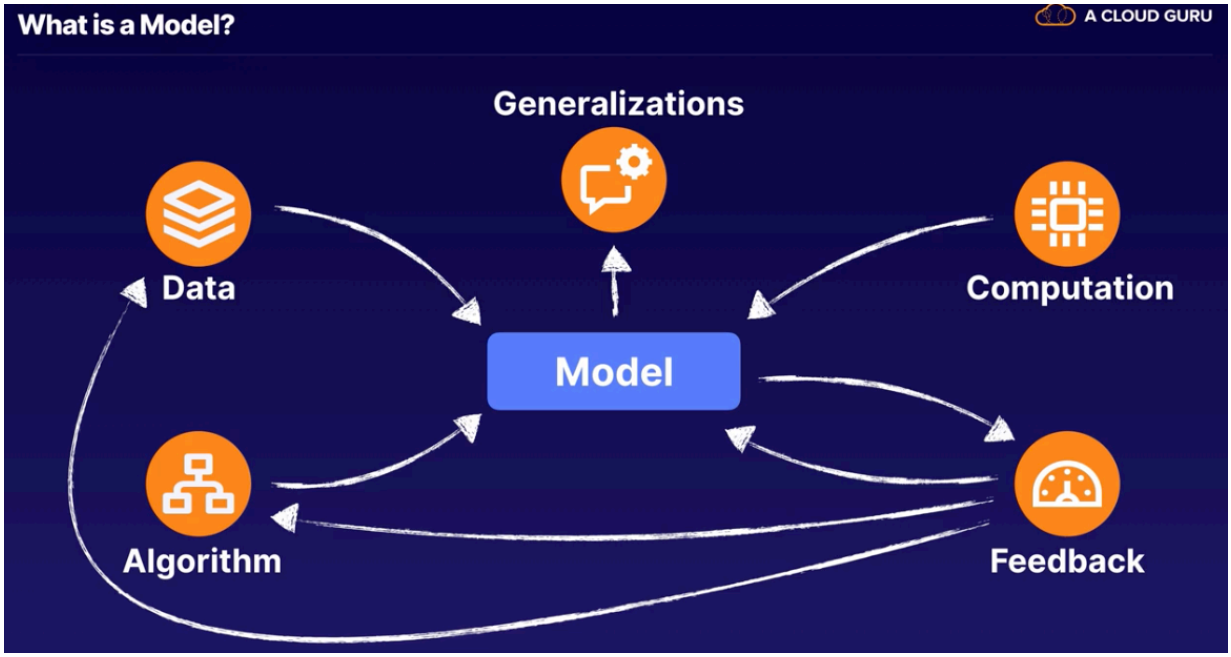
Cloud Guru - 5 - Modeling

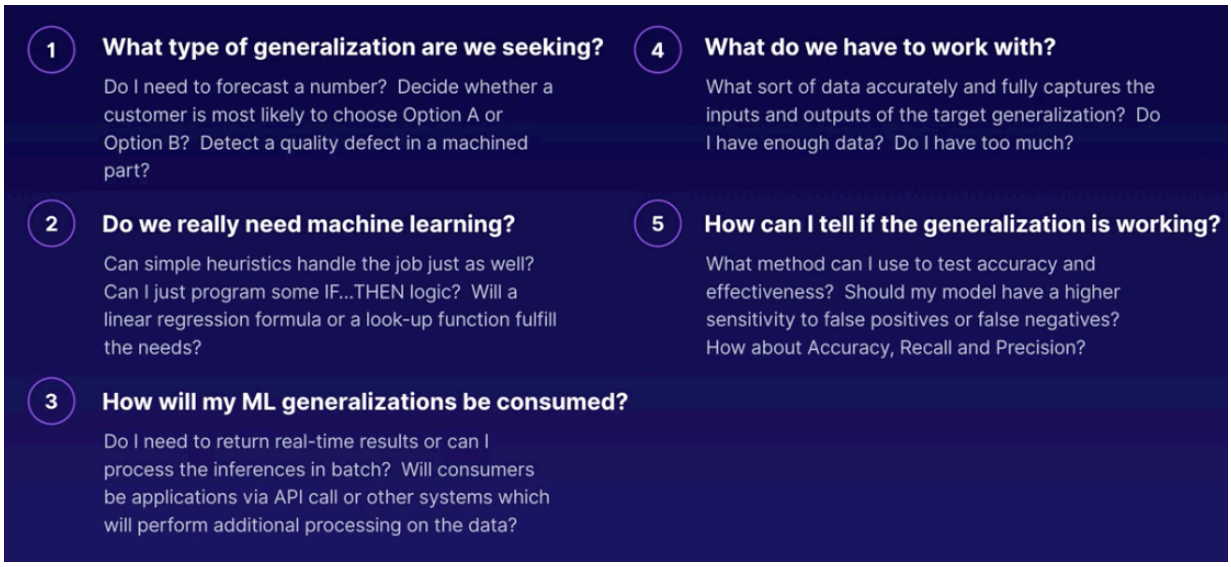
https://acloud.guru/course/aws-certified-machine-learning-specialty/learn/87700af1-74be-5d7b-2143-9e031dd0fb1d/9973b4c0-95fe-4cb0-0c3a-f6adee05d2dd/watch?backUrl=~2Fcourses&backUrl=~2Fcourses&backUrl=~2Fcourses,~2Fcourses



What is a model?



Developing a Good Model



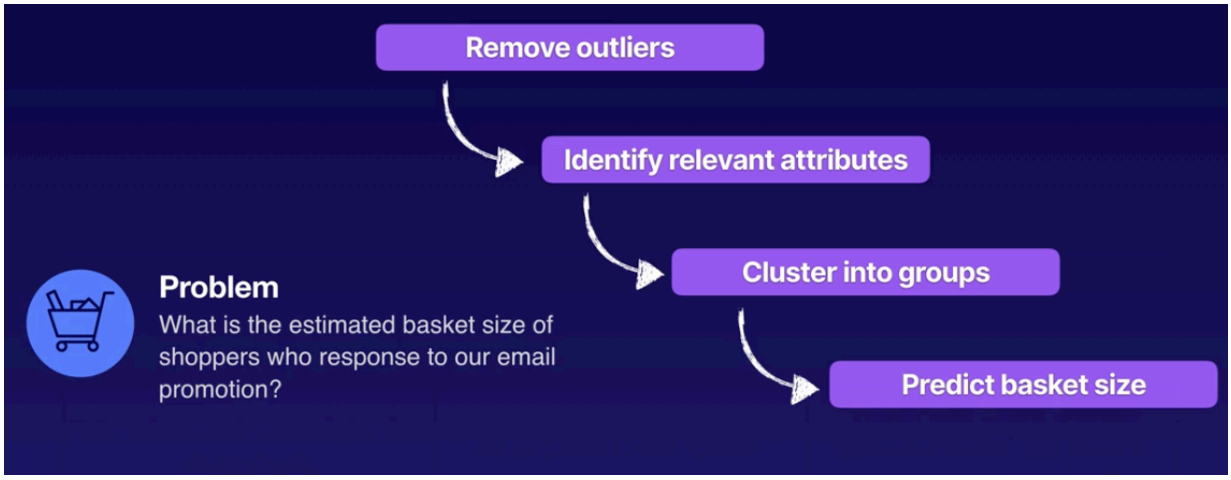
Different types of models:

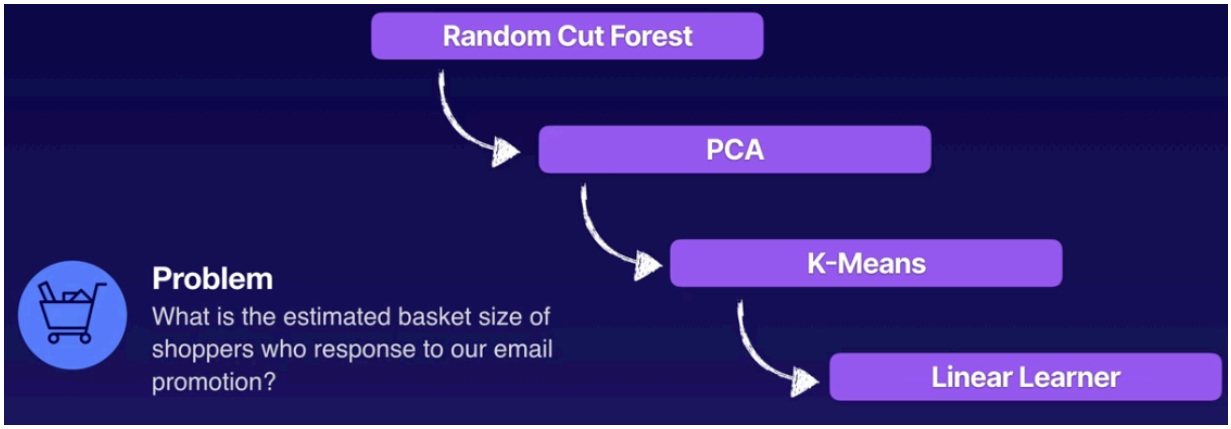
	Supervised Learning	Unsupervised Learning	Reinforcement Learning
Discrete	Classification	Clustering	Simulation-based Optimization
Continuous	Regression	Reduction of Dimentionality	Autonomous Devices

Choosing the right approach

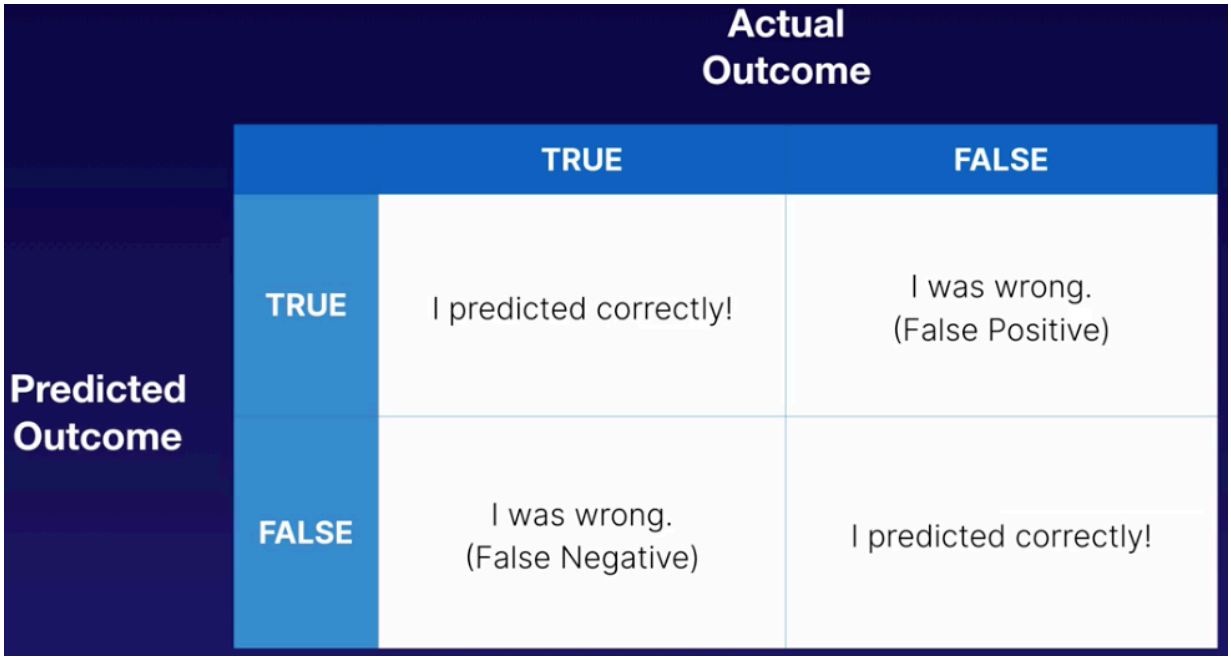
Problem	Approach	Why
Detect whether a financial transaction is fraud.	Binary Classification	Only two possible outcomes: Fraud or Not Fraud
Predict the rate of deceleration of a car when brakes are applied.	Heuristic Approach (No ML Needed!)	Well-known formulas involving speed, inertia and friction to predict this.
Determine the most efficient path of surface travel for a robotic lunar rover.	Simulation-based Reinforcement Learning	Must figure out the optimal path via trial, error and improvement.
Determine the breed of dog in a photograph.	Multi-Class Classification	Which dog breed is most associated with the picture among many breeds?

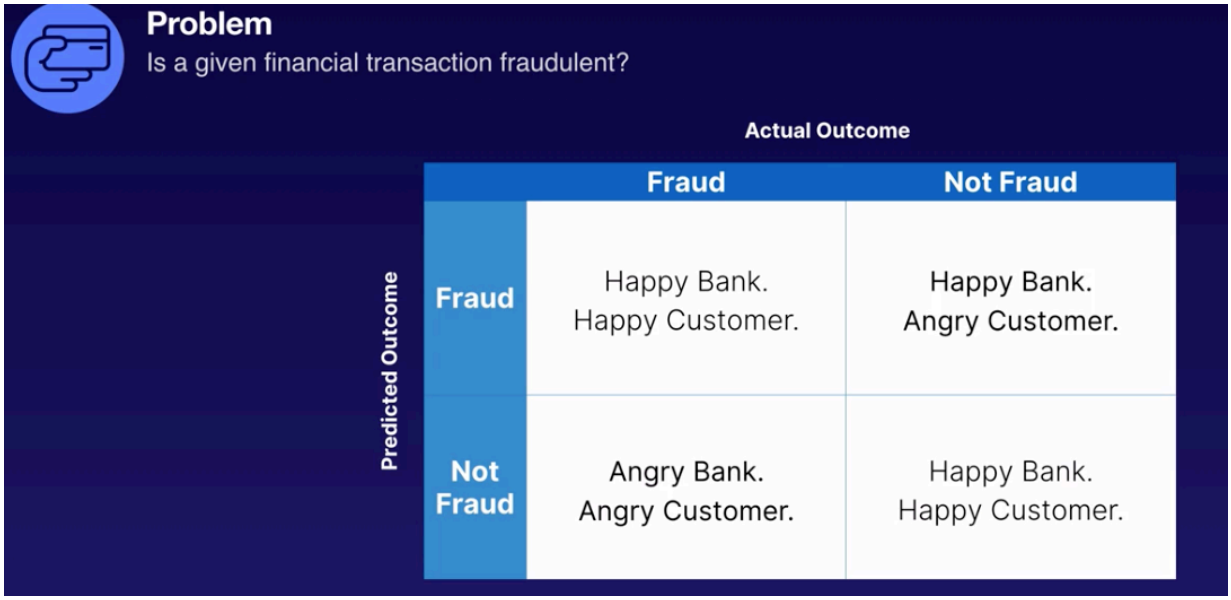
Cascading algorithms



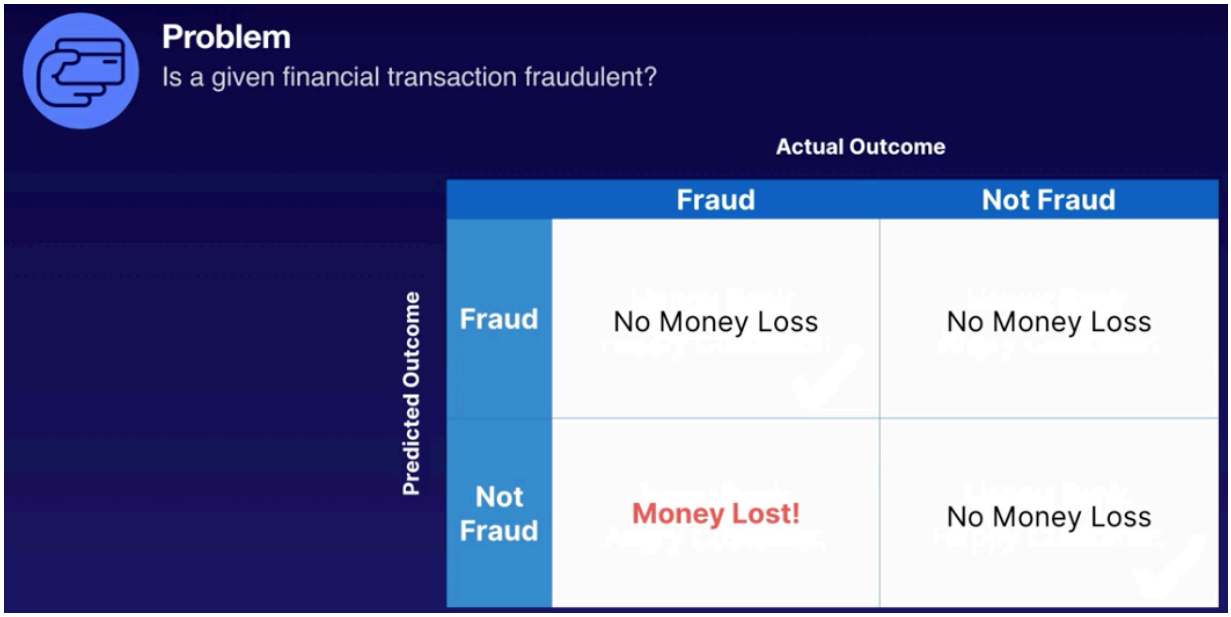


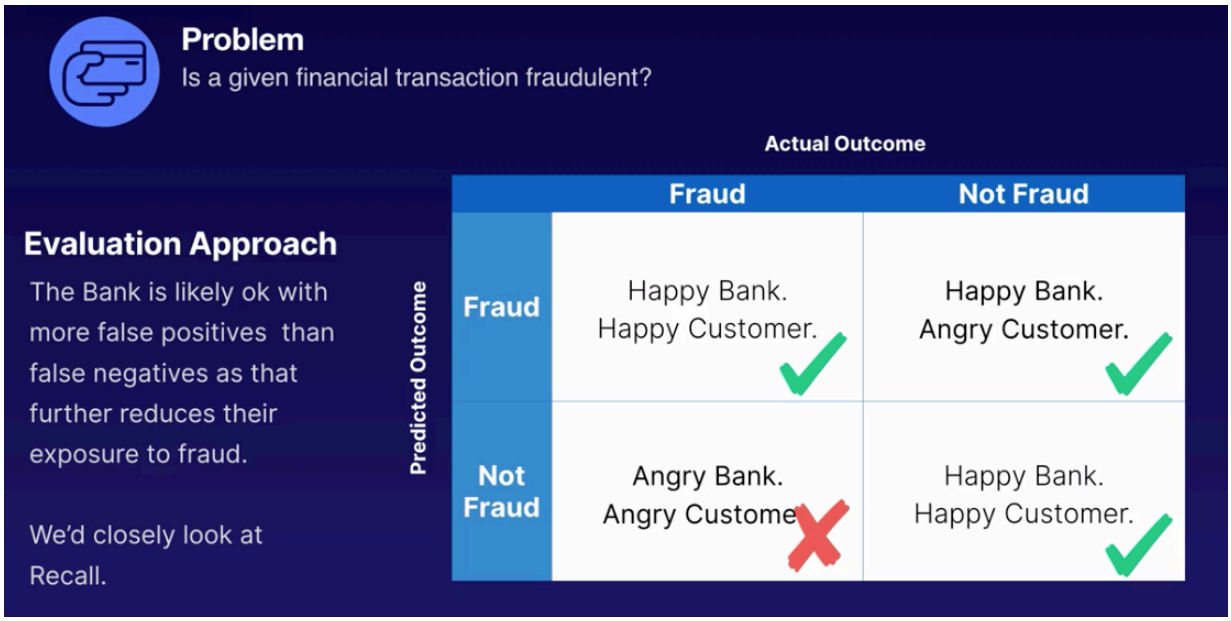
Confusion Matrix





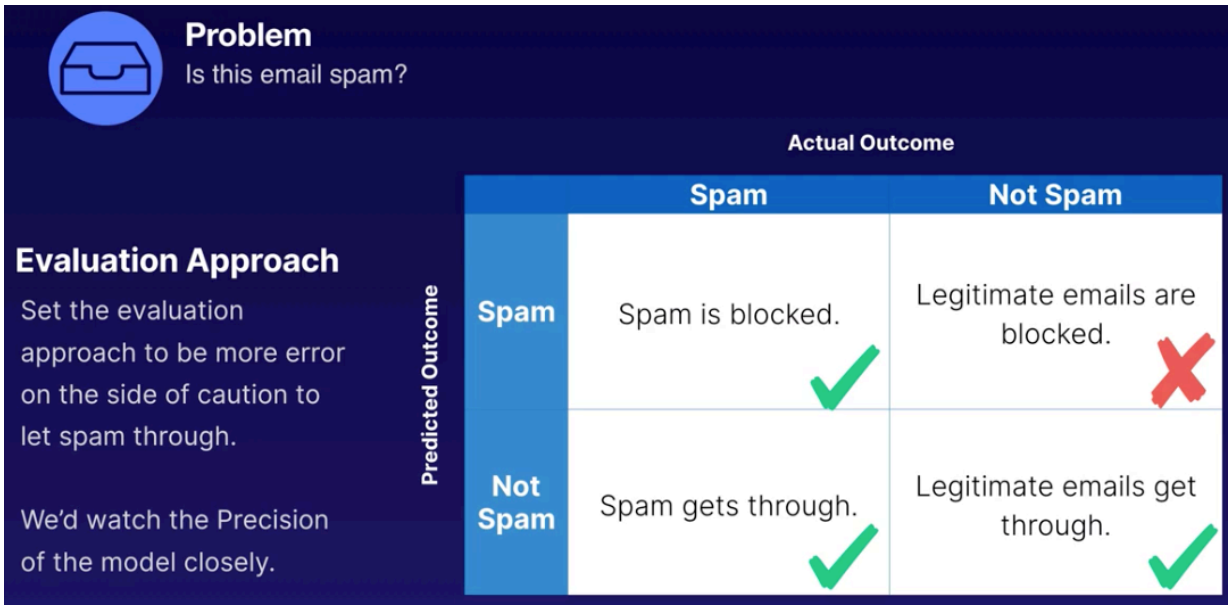
From the bank perspective, there is only one case where we lose money.
So this is where we will focus there => specify the evaluation approach





Bank would probably prefer to err on the side of more false positives than false negatives

Here is another example of Confusion matrix - is an email spam?

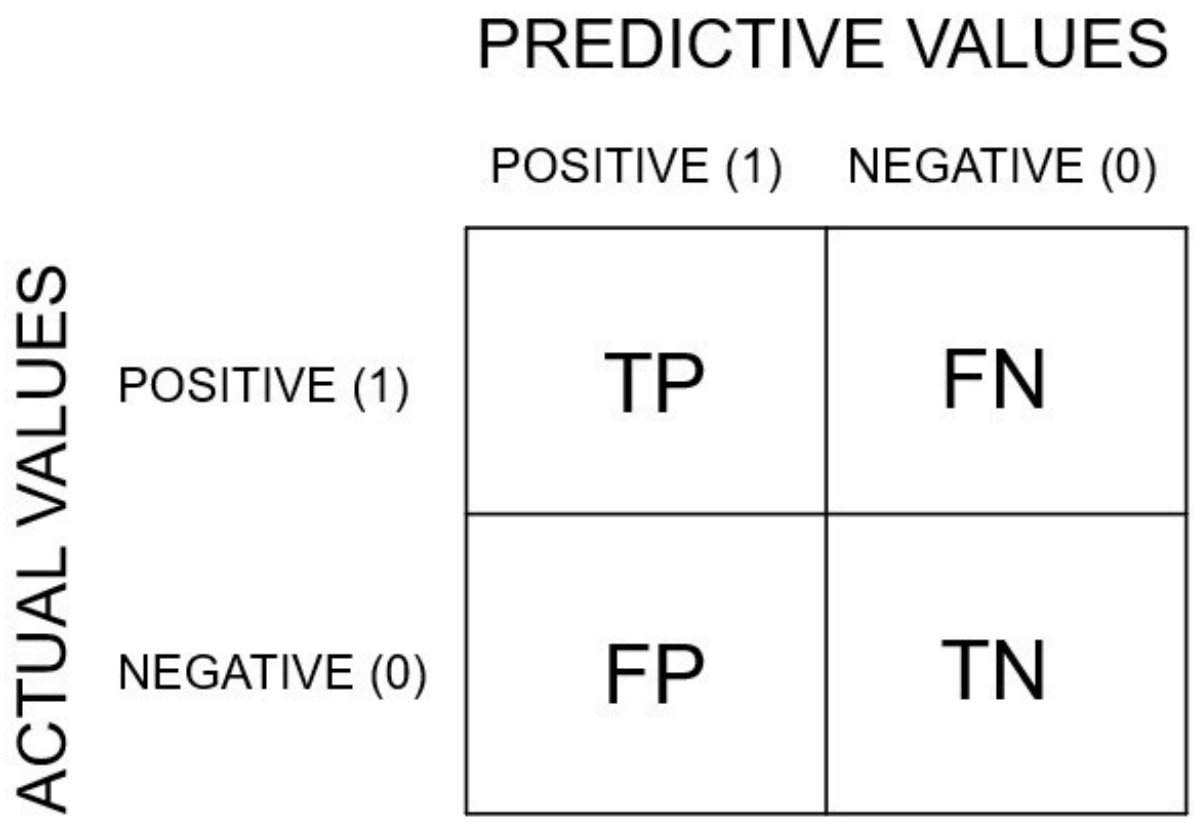


https://towardsdatascience.com/decoding-the-confusion-matrix-bb4801decbb

1. Confusion Matrix:

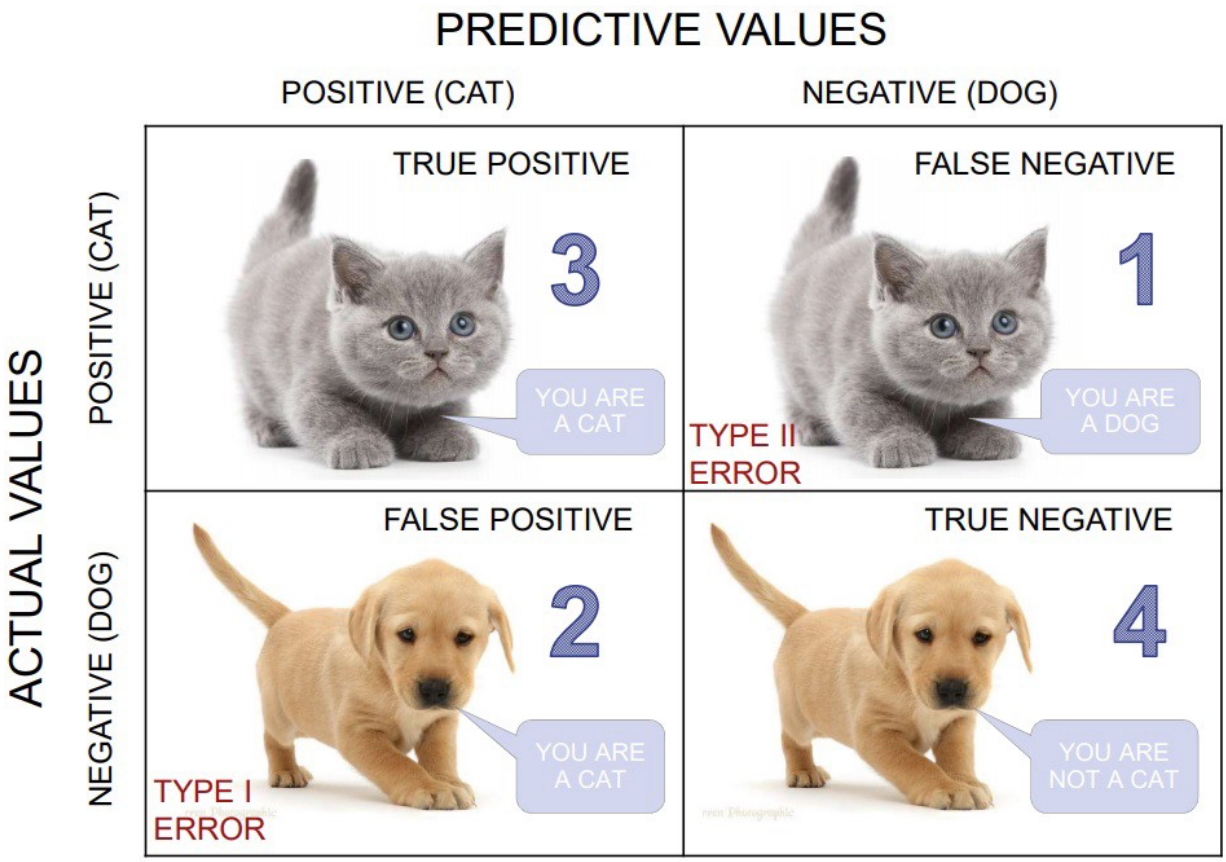
A confusion matrix provides a summary of the predictive results in a classification problem. Correct and incorrect predictions are summarized in a table with their

values and broken down by each class.



2. Calculate a confusion matrix:

Let’s take an example:
We have a total of 10 cats and dogs and our model predicts whether it is a cat or not.
Actual values = [‘dog’, ‘cat’, ‘dog’, ‘cat’, ‘dog’, ‘dog’, ‘cat’, ‘dog’, ‘cat’, ‘dog’]
Predicted values = [‘dog’, ‘dog’, ‘dog’, ‘cat’, ‘dog’, ‘dog’, ‘cat’, ‘cat’, ‘cat’, ‘cat’]

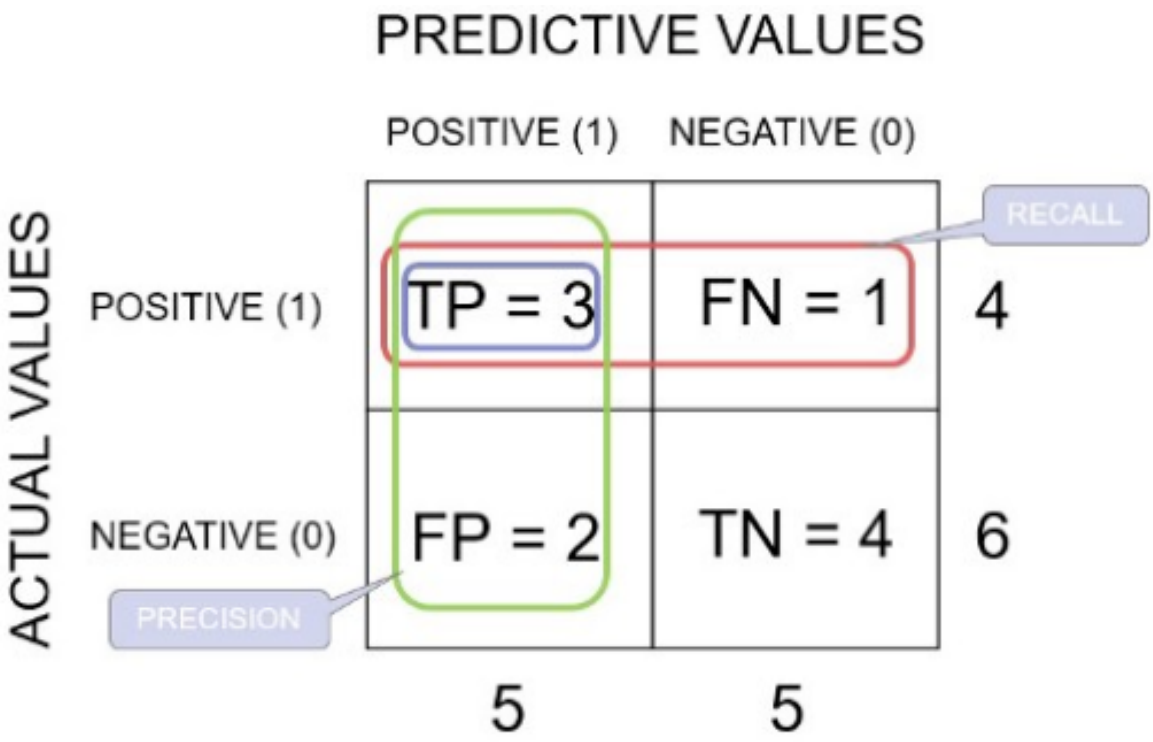


Remember, we describe predicted values as Positive/Negative and actual values as True/False.

$$\textbf{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\textbf{Recall} = \frac{TP}{TP + FN} \quad \text{or} \quad \frac{\text{True Positive}}{\text{Actual Results}}$$

$$\textbf{Precision} = \frac{TP}{TP + FP} \quad \text{or} \quad \frac{\text{True Positive}}{\text{Predictive Results}}$$



Recall: Recall gives us an idea about when it's actually yes, how often does it predict yes.
Recall = TP / (TP + FN) = 3/(3+1) = 0.75

Precision: Precision tells us about when it predicts yes, how often is it correct.
Precision = TP / (TP + FP) = 3/(3+2) = 0.60

F-score:
F-score = (2*Recall*Precision)/(Recall+Precision) = (2*0.75*0.60)/(0.75+0.60) = 0.67

Specificity:
Specificity = TN / (TN + FP) = 4/(4+2) = 0.67

4. Summary:

- Precision is how certain you are of your true positives. Recall is how certain you are that you are not missing any positives.

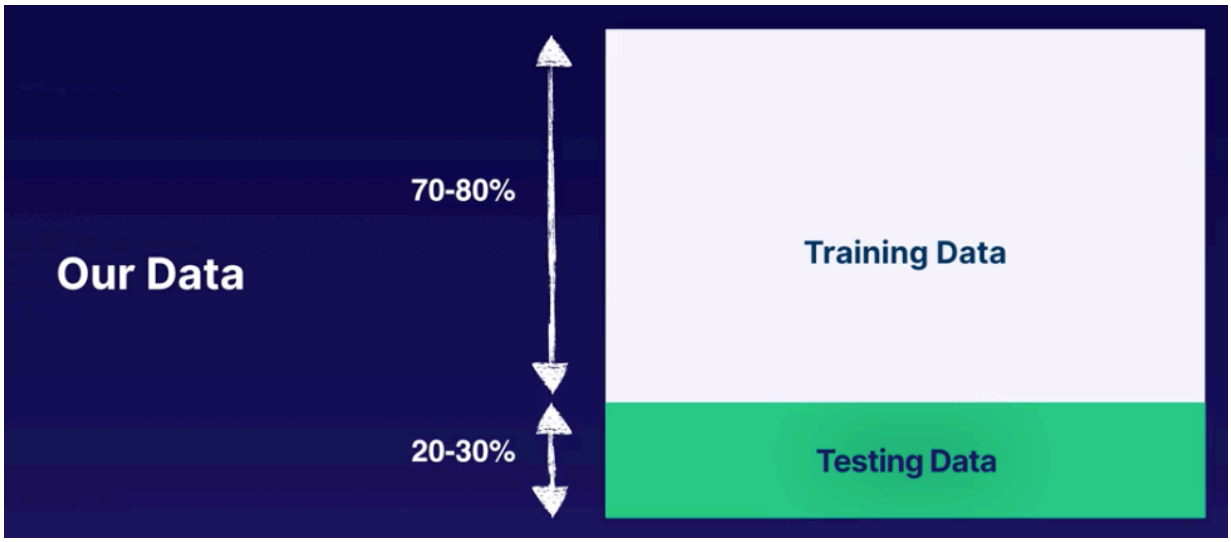
- Choose **Recall** if the occurrence of **false negatives is unaccepted/ intolerable.** For example, in the case of diabetes that you would rather have some extra false positives (false alarms) over saving some false negatives.
- Choose **Precision** if you want to be more **confident of your true positives**. For example, in case of spam emails, you would rather have some spam emails in your inbox rather than some regular emails in your spam box. You would like to be extra sure that email X is spam before we put it in the spam box.
- Choose **Specificity** if you want to **cover all true negatives,** i.e. meaning we do not want any false alarms or false positives. For example, in case of a drug test in which all people who test positive will immediately go to jail, you would not want anyone drug-free going to jail.

We can conclude that:

- Accuracy value of 70% means that identification of 3 of every 10 cats is incorrect, and 7 is correct.
- Precision value of 60% means that label of 4 of every 10 cats is a not a cat (i.e. a dog), and 6 are cats.
- Recall value is 70% means that 3 of every 10 cats, in reality, are missed by our model and 7 are correctly identified as cats.
- Specificity value is 60% means that 4 of every 10 dogs (i.e. not cat) in reality are miss-labeled as cats and 6 are correctly labeled as dogs.

Data Preparation

Make sure to carve out 20-30% of your dataset for testing

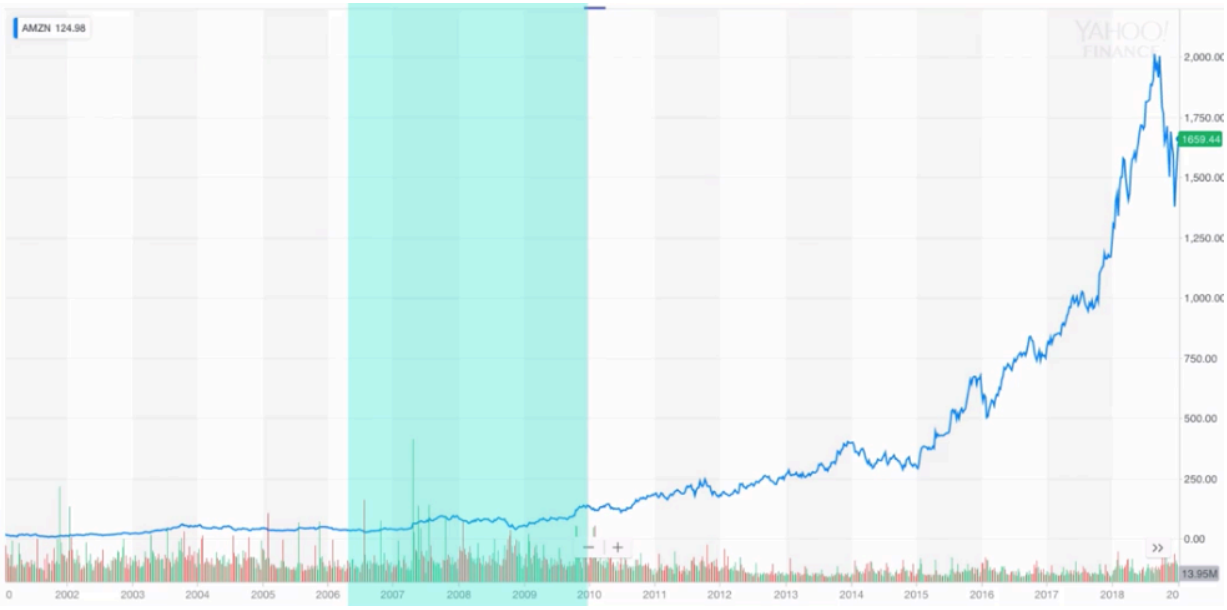


Example with numpy
Going with a 90/10 split

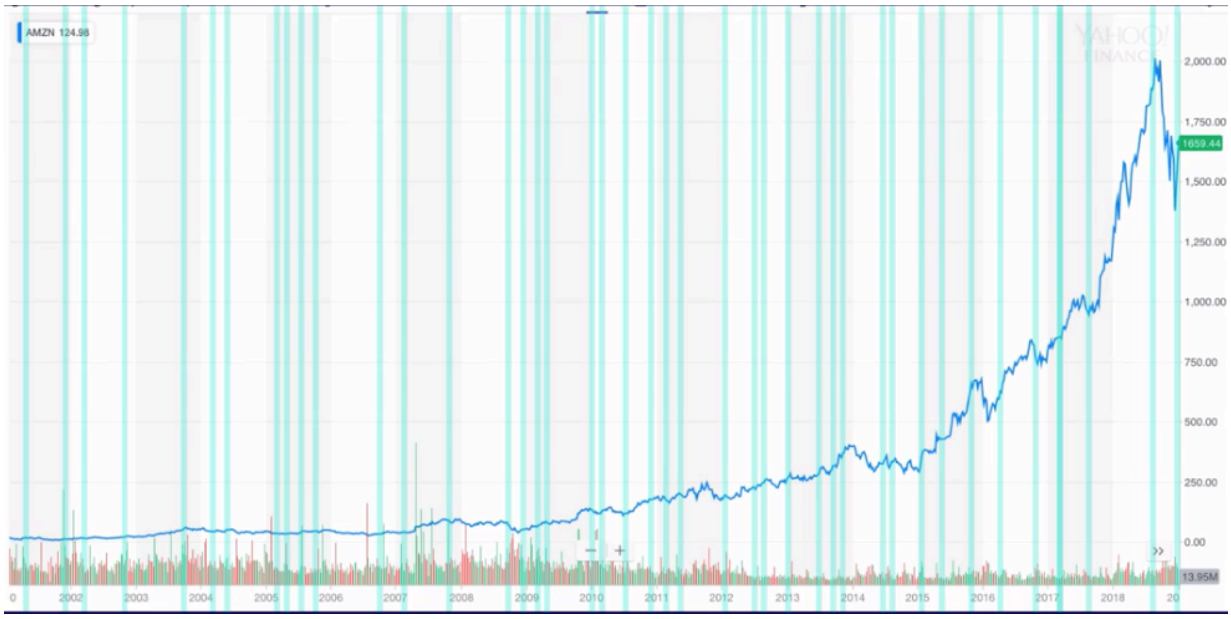
```python
import numpy as np
import os


# read raw data
print("Reading raw data from {}".format(raw_data_file))
raw = np.loadtxt(raw_data_file, delimiter=',')

# split into train/test with a 90/10 split
np.random.seed(0)
np.random.shuffle(raw)
train_size = int(0.9 * raw.shape[0])
train_features = raw[:train_size, :-1]
train_labels = raw[:train_size, -1]
test_features = raw[train_size:, :-1]
test_labels = raw[train_size:, -1]
```

Amazon stock as a time series

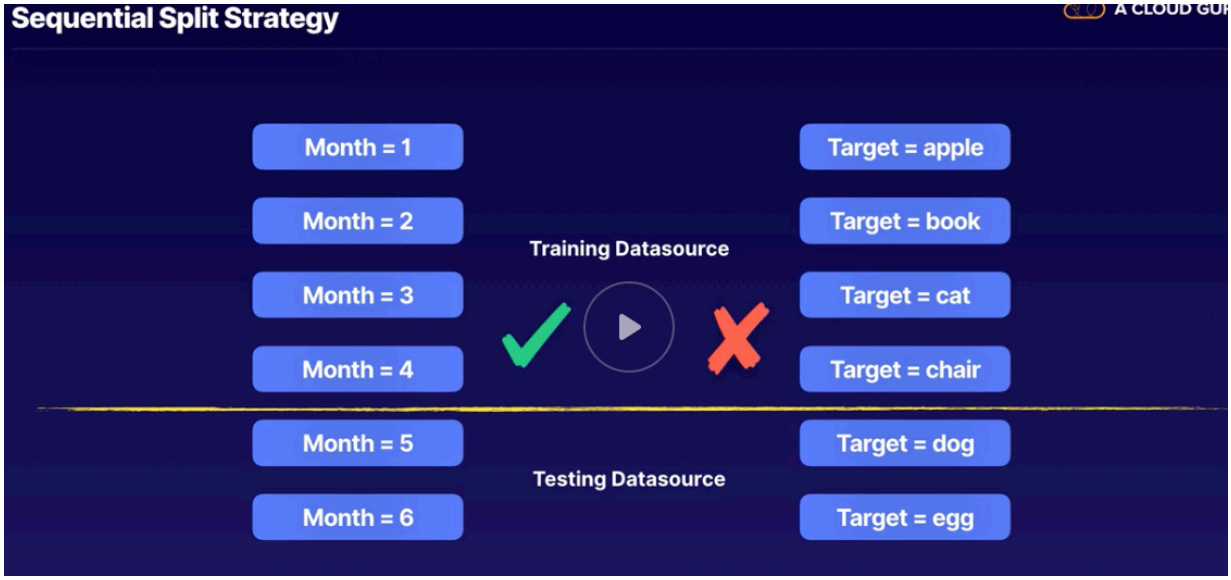
For time series, the approach is different.
We can’t just pick 20% in the middle



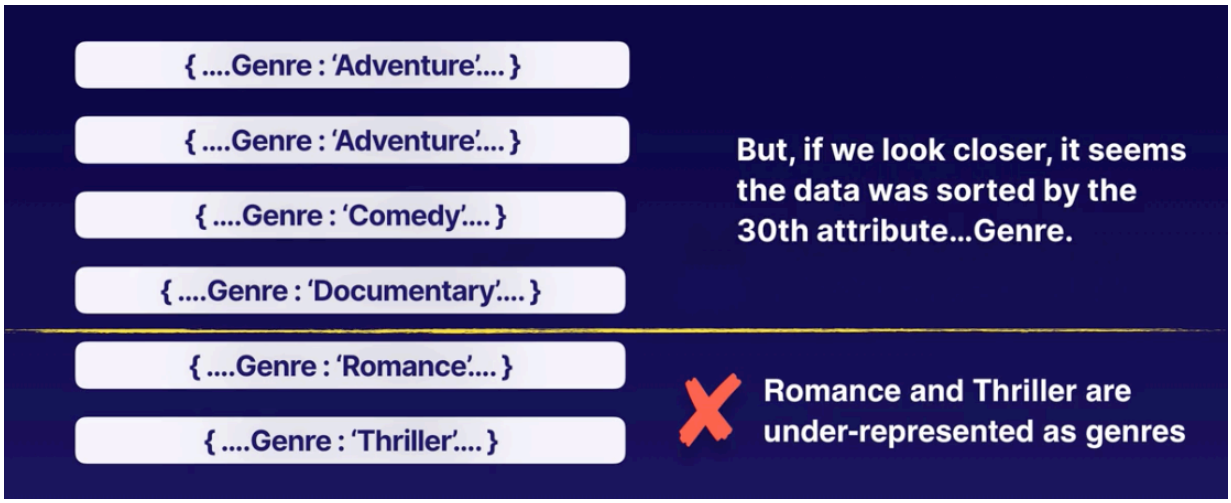
Picking points rankly may not be effective either

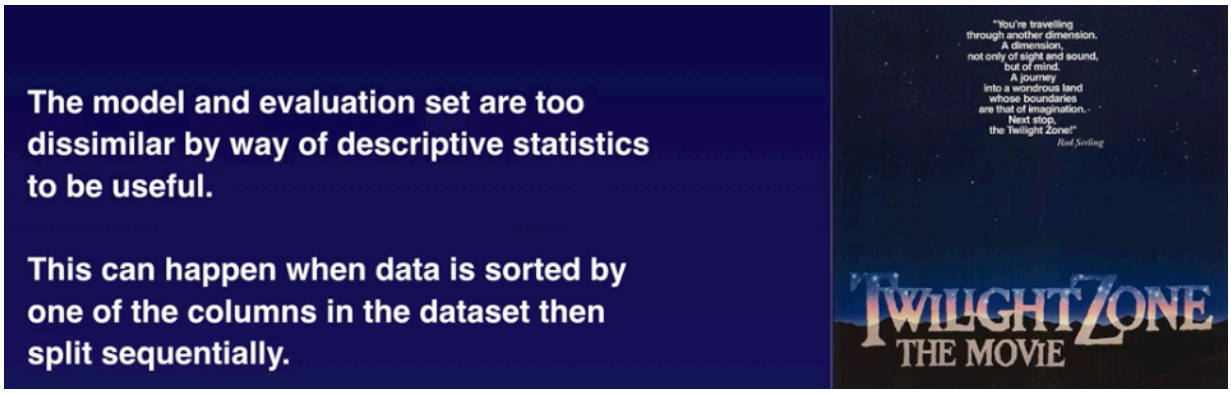


We need instead to focus on a training set over consecutive months, then use test set as next months (in sequence)
That is a sequential split strategy.
But this is ONLY for time series.
That would not work for other datasets like multi class



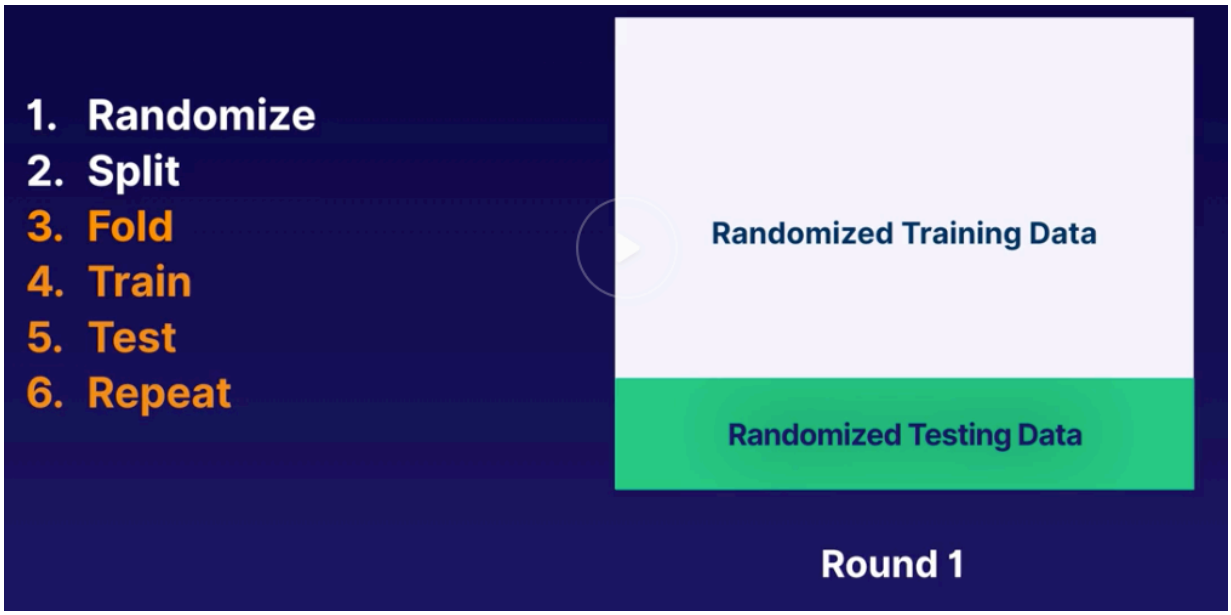
Example of wrong sequential split :





K-fold cross validation => K means how many times we will fold the data

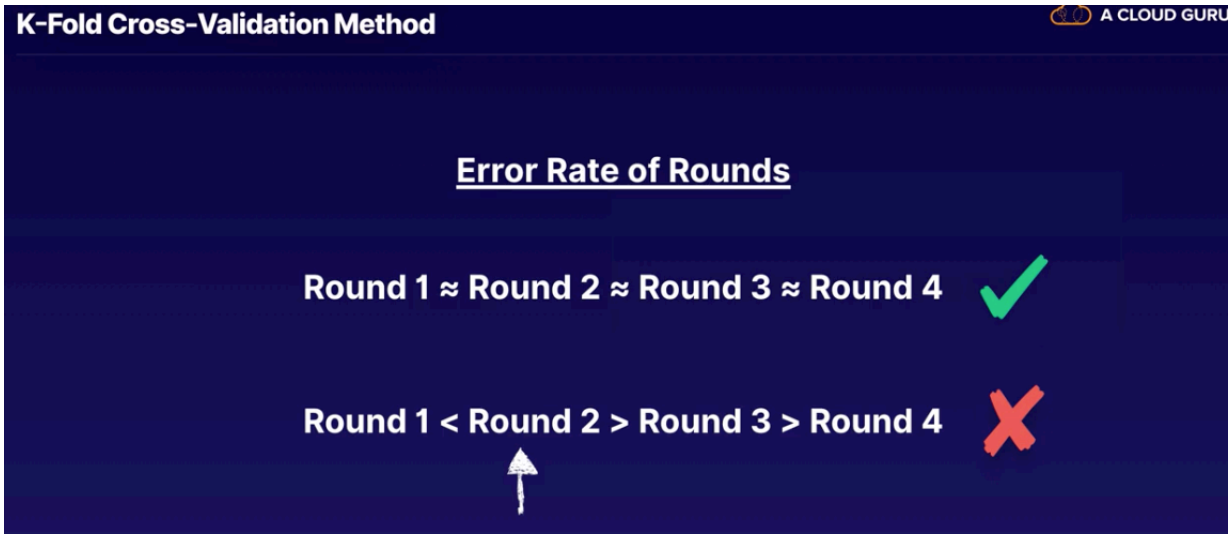
Round 1:



Round 2: take a different section and hold that out for testing

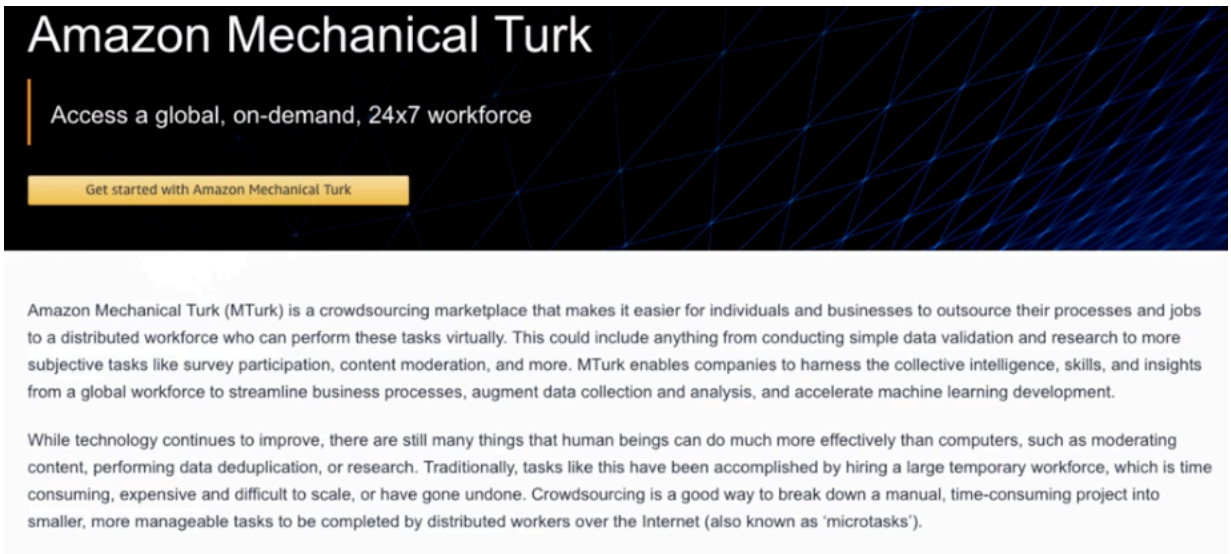
Round 4:

Then we look at error rate of each round.

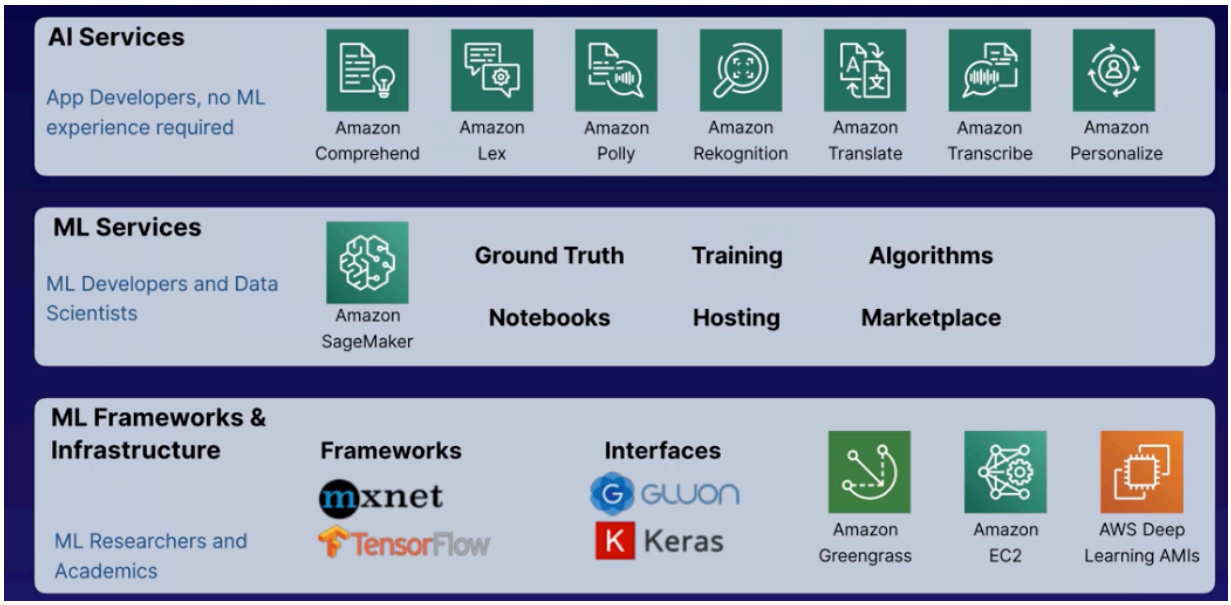


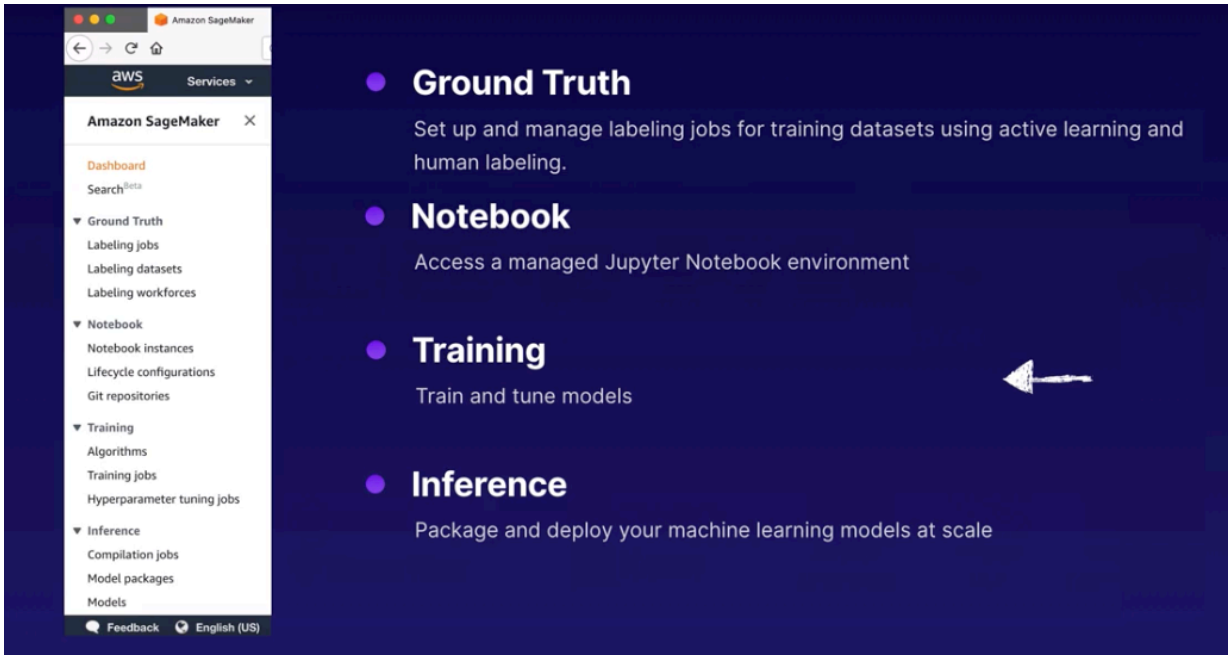
SageMaker Modeling

Mechanical Turk - provide an interface to farm out tasks for human beings to complete

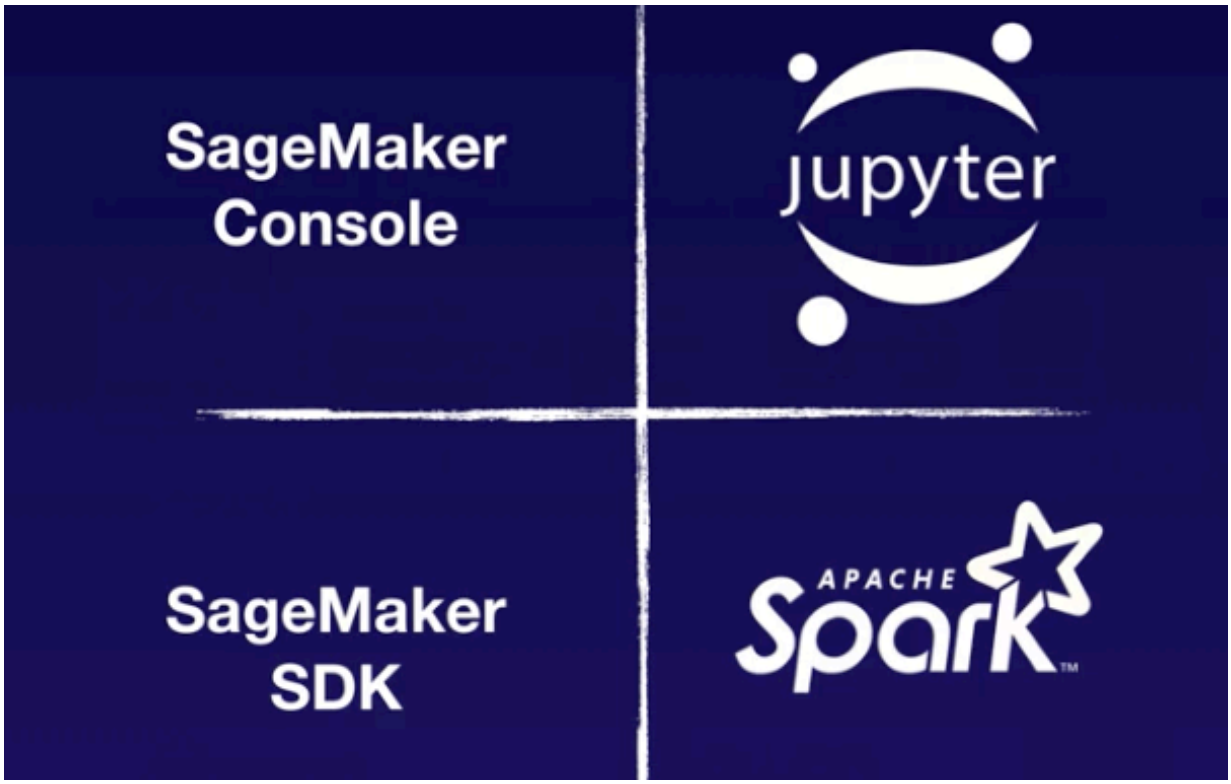


Ironically, some of the things we use are noe wrapped as services

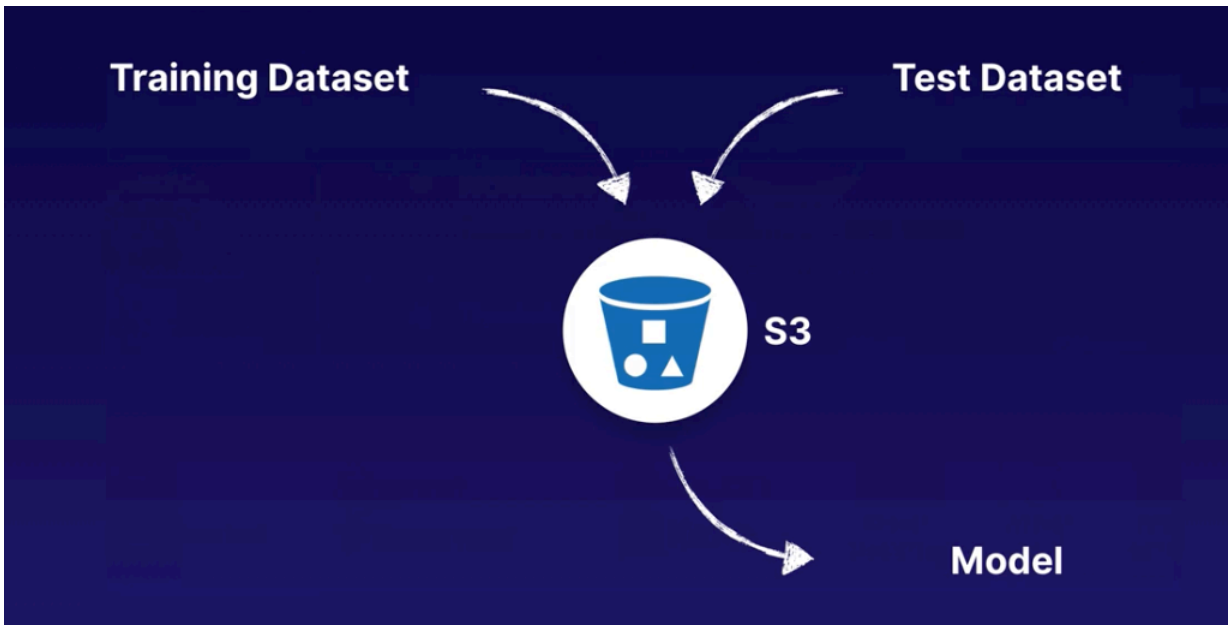




4 options for model creation:



All start of S3



Data is split in a randomized training and test set:

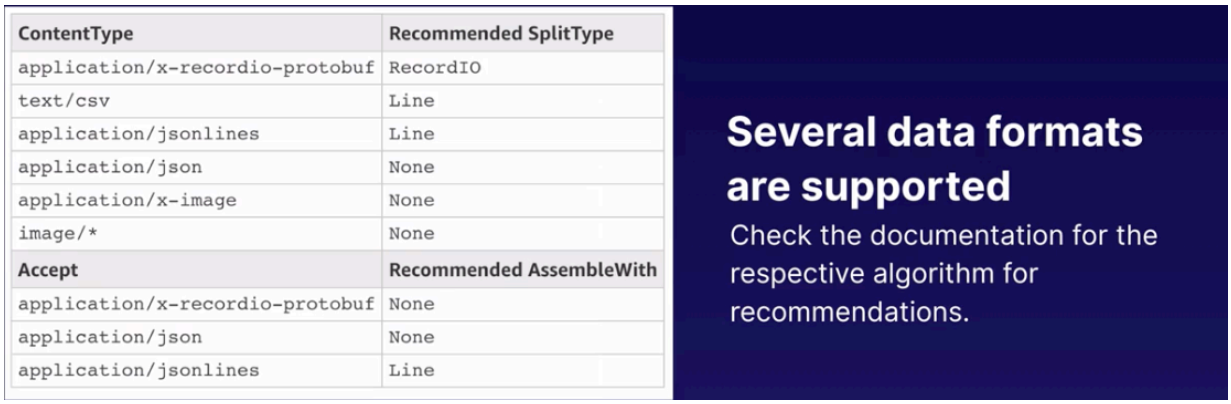
Evil	Name	Affiliation
0	Luke	Rebel
1	JarJar	Empire
0	Han	Rebel
1	Vadar	Empire
0	Yoda	Rebel
1	Jabba	Empire
1	Maul	Empire
0	Leia	Rebel

0	Luke	Rebel
1	JarJar	Empire
0	Han	Rebel
1	Vadar	Empire
0	Yoda	Rebel

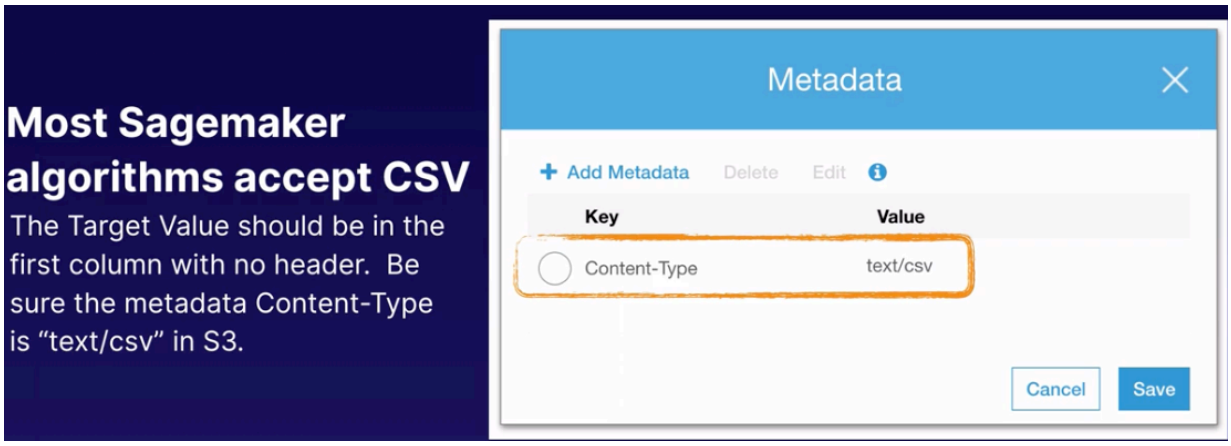
1	Jabba	Empire
1	Maul	Empire
0	Leia	Rebel

We then upload it to S#

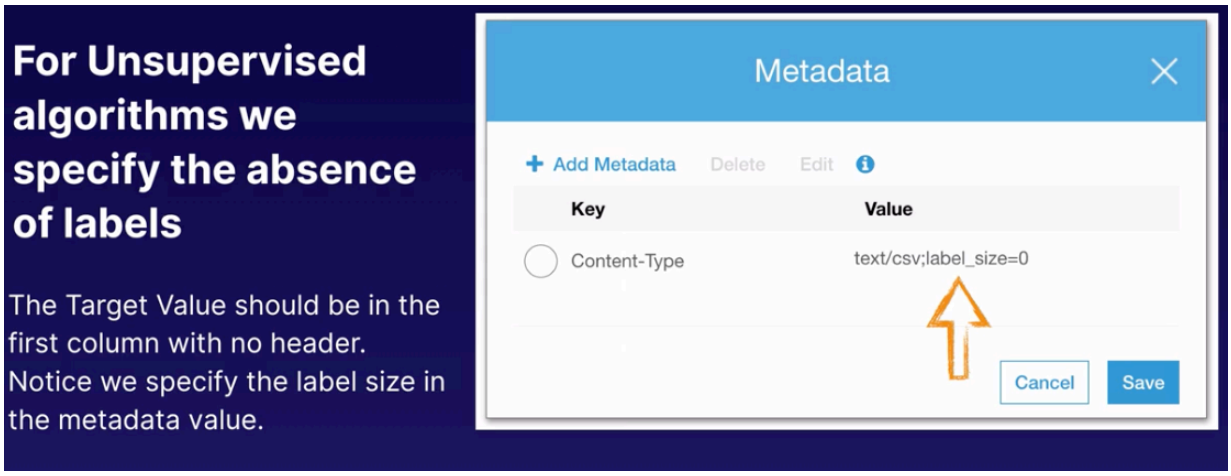
Be careful about the format though

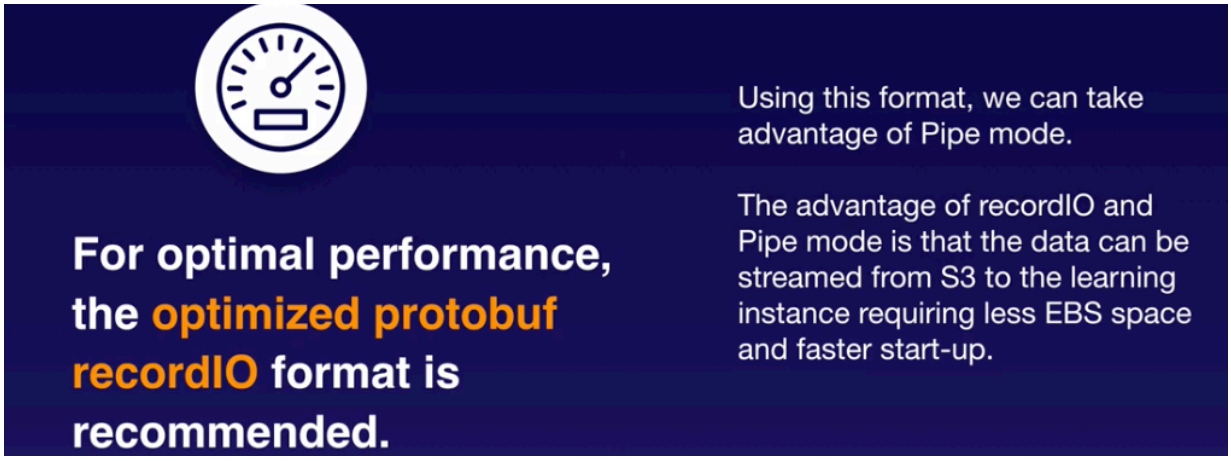


ContentType	Recommended SplitType
application/x-recordio-protobuf	RecordIO
text/csv	Line
application/jsonlines	Line
application/json	None
application/x-image	None
image/*	None
Accept	**Recommended AssembleWith**
application/x-recordio-protobuf	None
application/json	None
application/jsonlines	Line

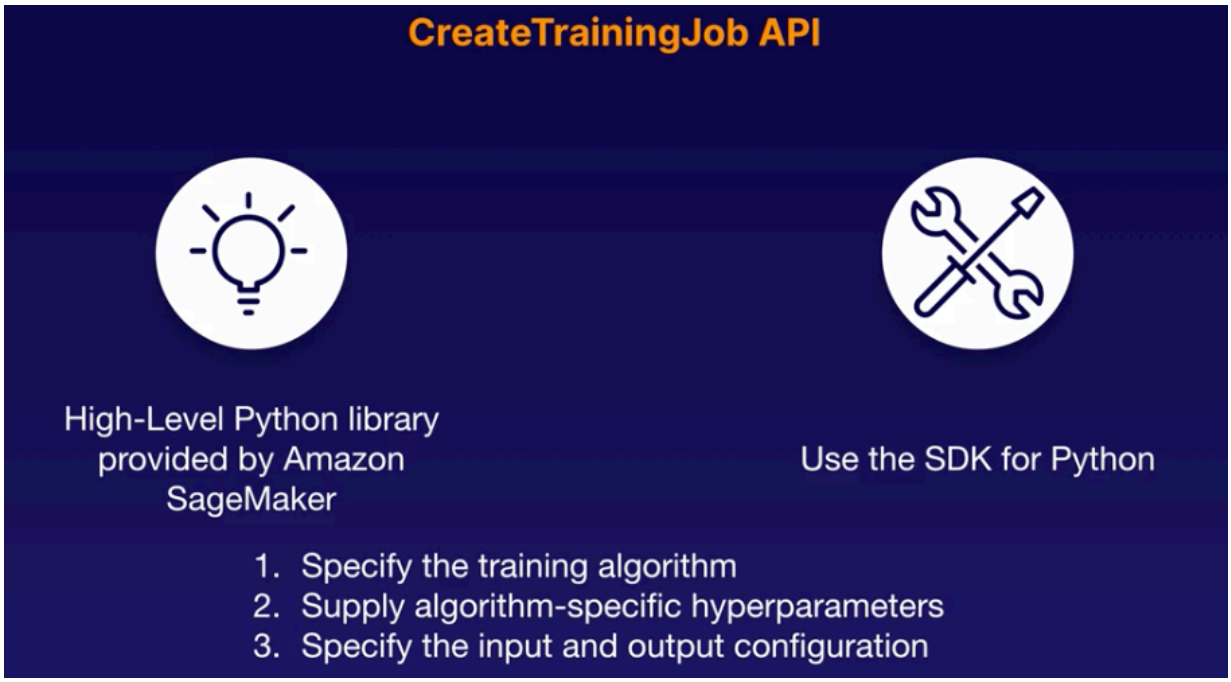


For unsupervised ML, we need to specify the absence of a label (the absence of a target value we are trying to predict)





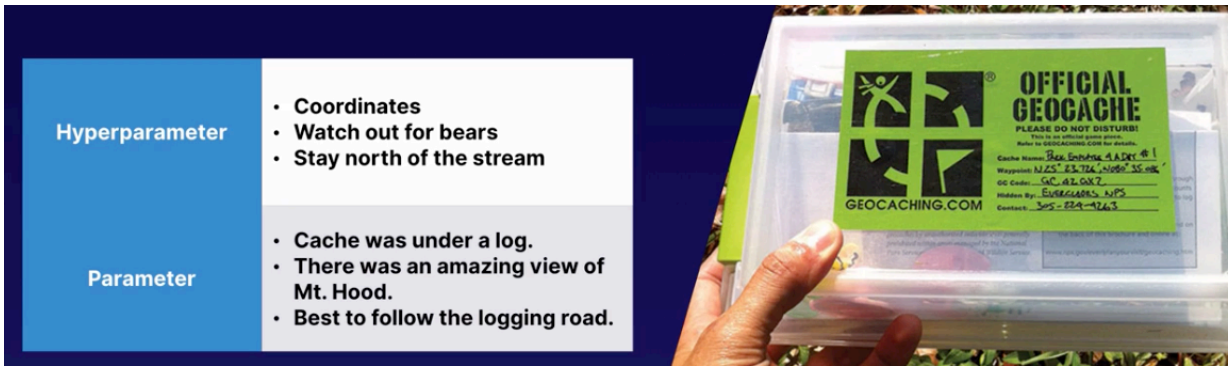
For training:



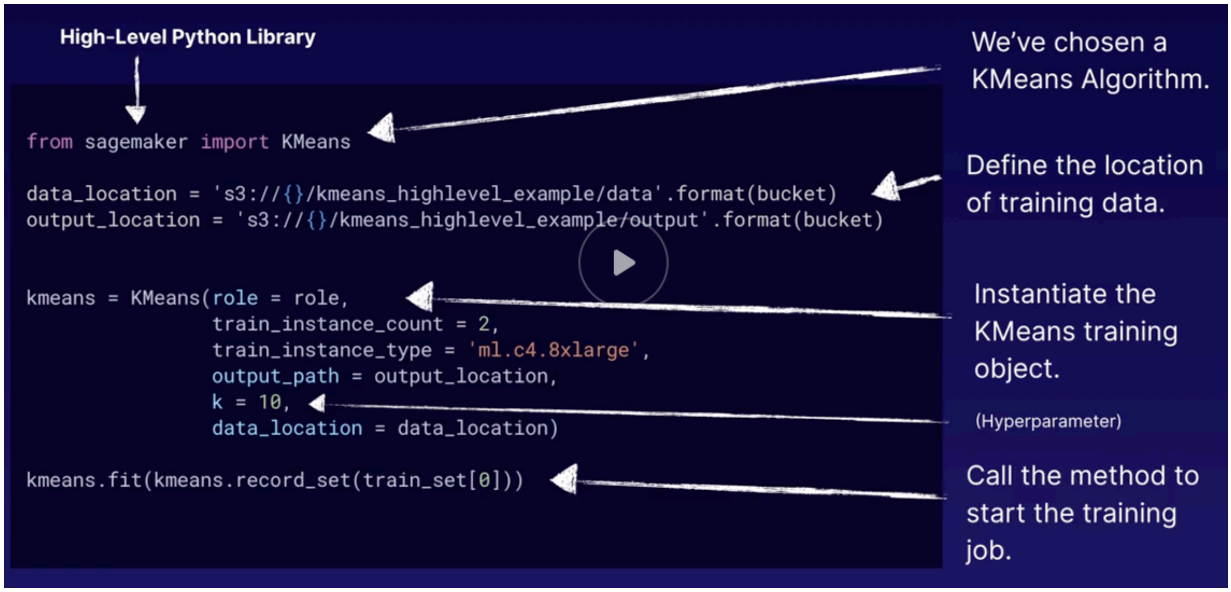
Hyperparameters

Hyperparameter	Values set **before** the learning process.
Parameter	Values **derived via** the learning process.

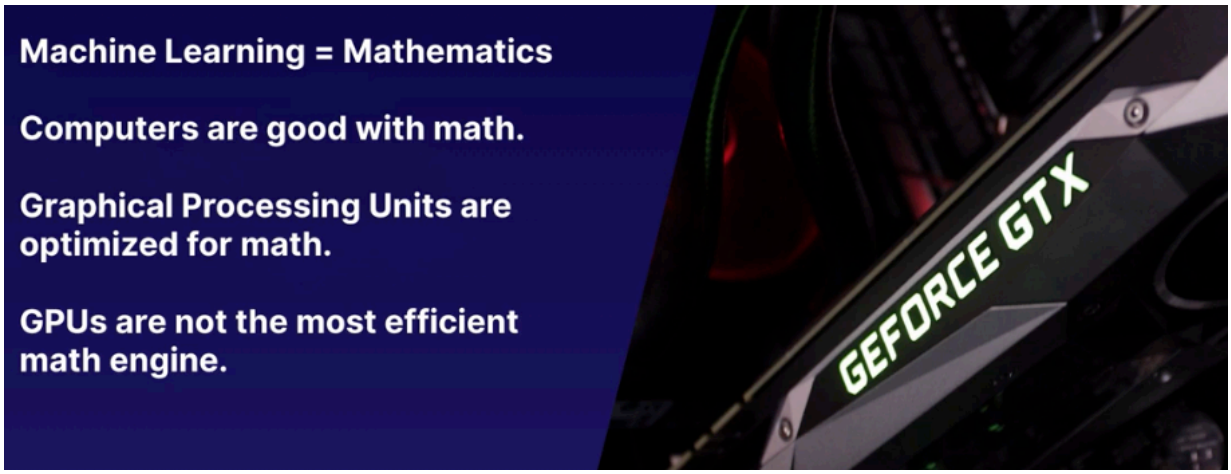
Example with Geocache

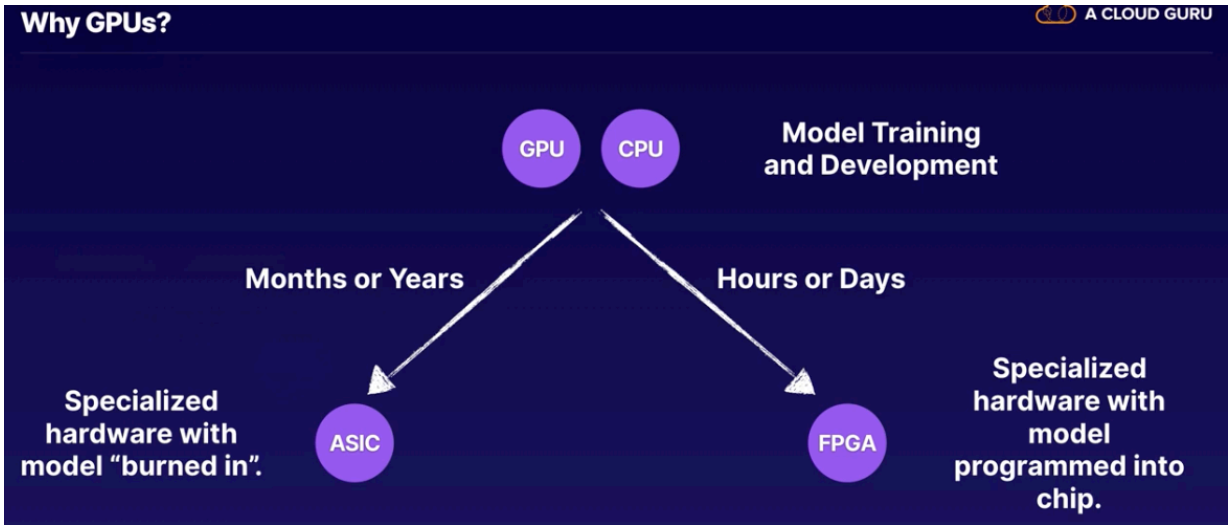


Example of Training Job
K = # of clusters to create



Sagemaker training





To change an FPGA, it could take hours or days.
But with ASIC, it's much easier

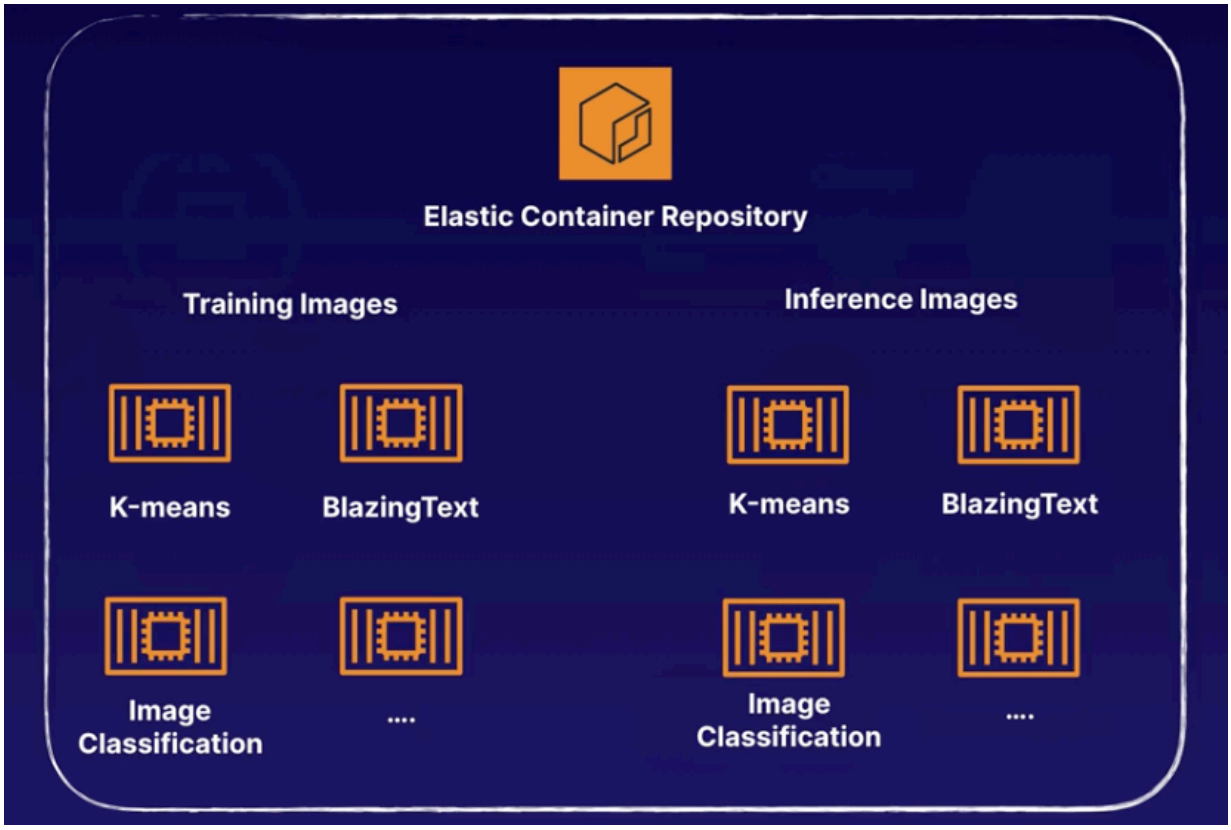
What happens when we submit a job to SageMaker

High-Level Python Library

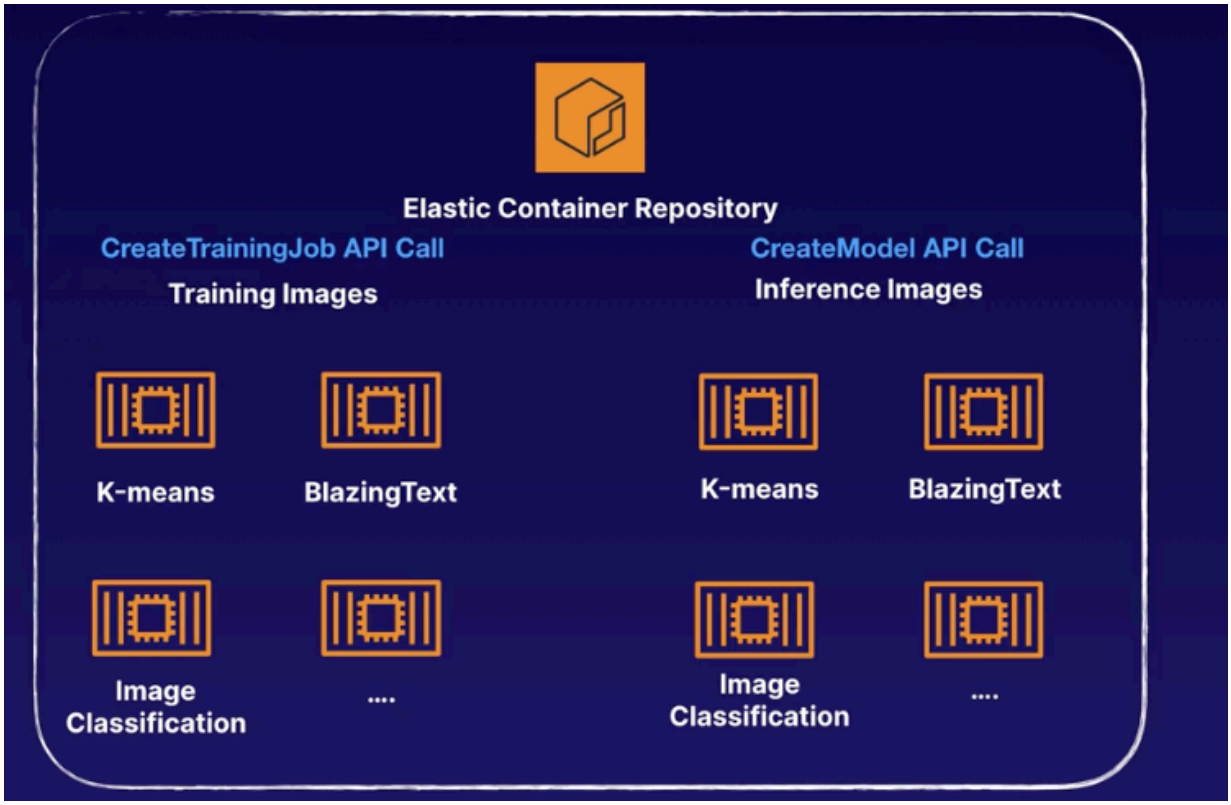
```
from sagemaker import KMeans

data_location = 's3://{}/kmeans_highlevel_example/data'.format(bucket)
output_location = 's3://{}/kmeans_highlevel_example/output'.format(bucket)


kmeans = KMeans(role = role,
                train_instance_count = 2,
                train_instance_type = 'ml.c4.8xlarge',
                output_path = output_location,
                k = 10,
                data_location = data_location)

kmeans.fit(kmeans.record_set(train_set[0]))
```

From a Jupyter Notebook, you can issue commands to launch a training job.

Amazon has an Elastic Container Repository that has containers for each algorithms.

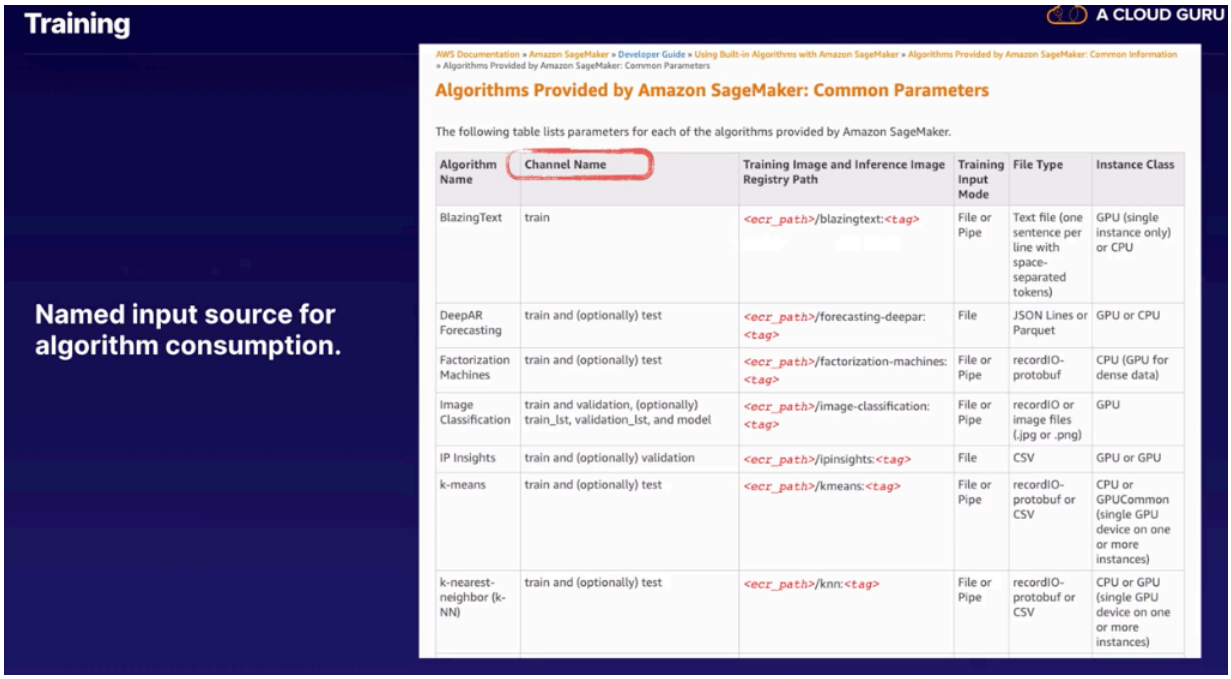


2 jobs:
- training (potentially on large GPU)
- model API call (deploy to prod on a less CPOU expansive)

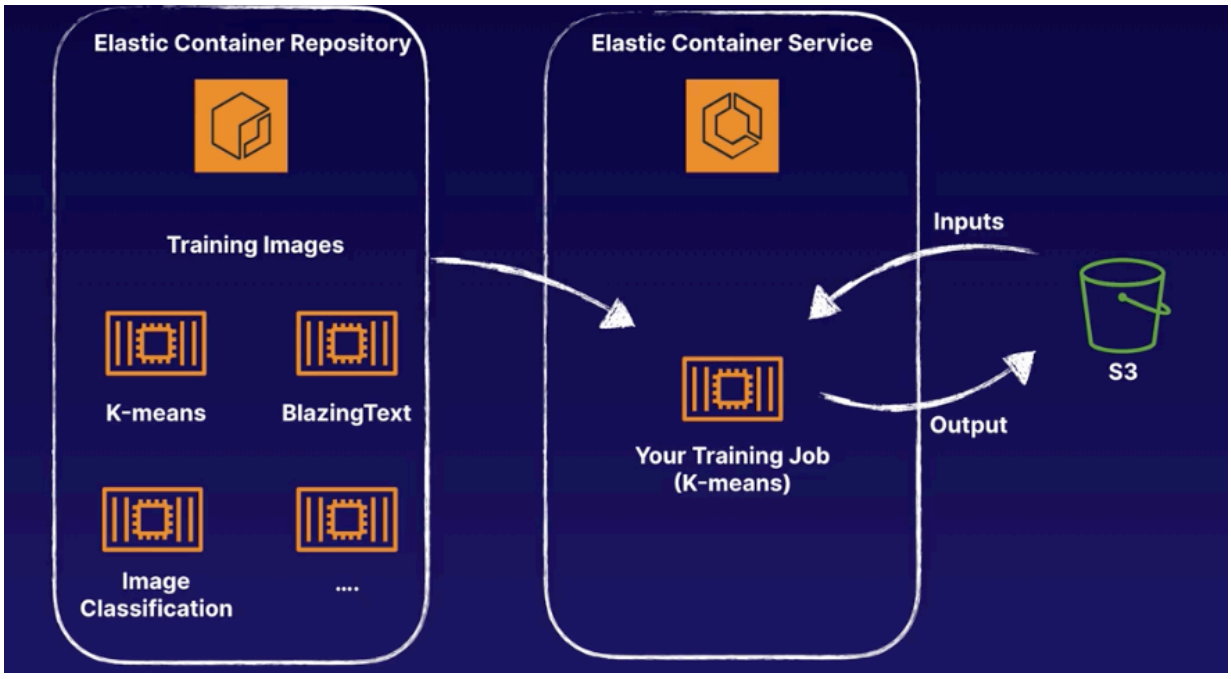


AWS has created image repositories in different regions:

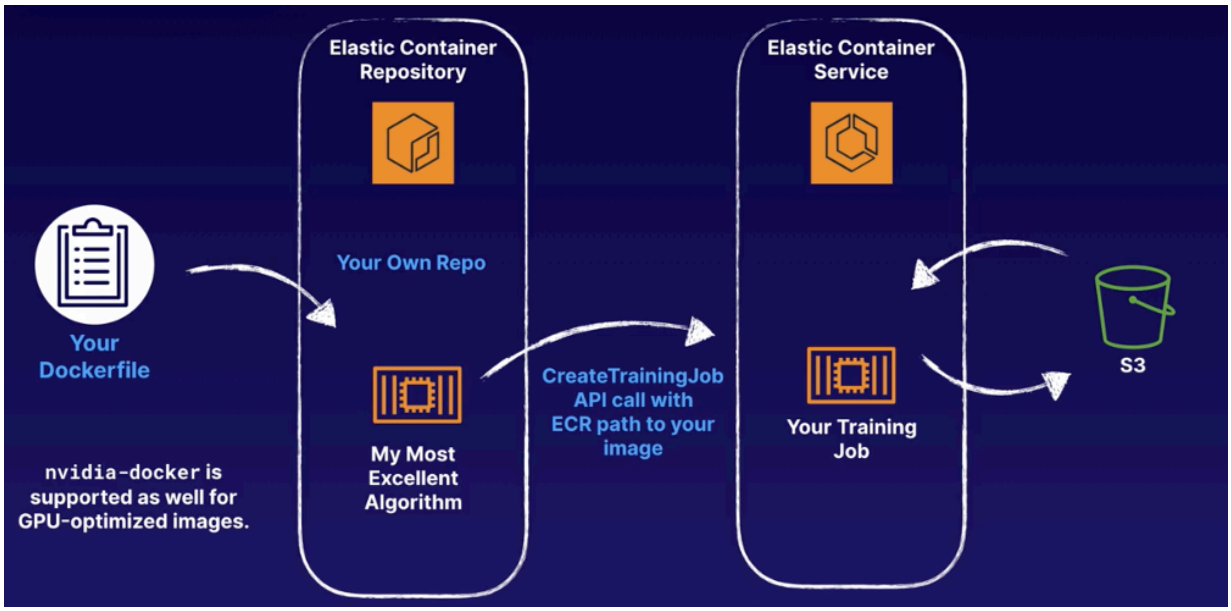
They are named differently depending on the algorithm we want to use

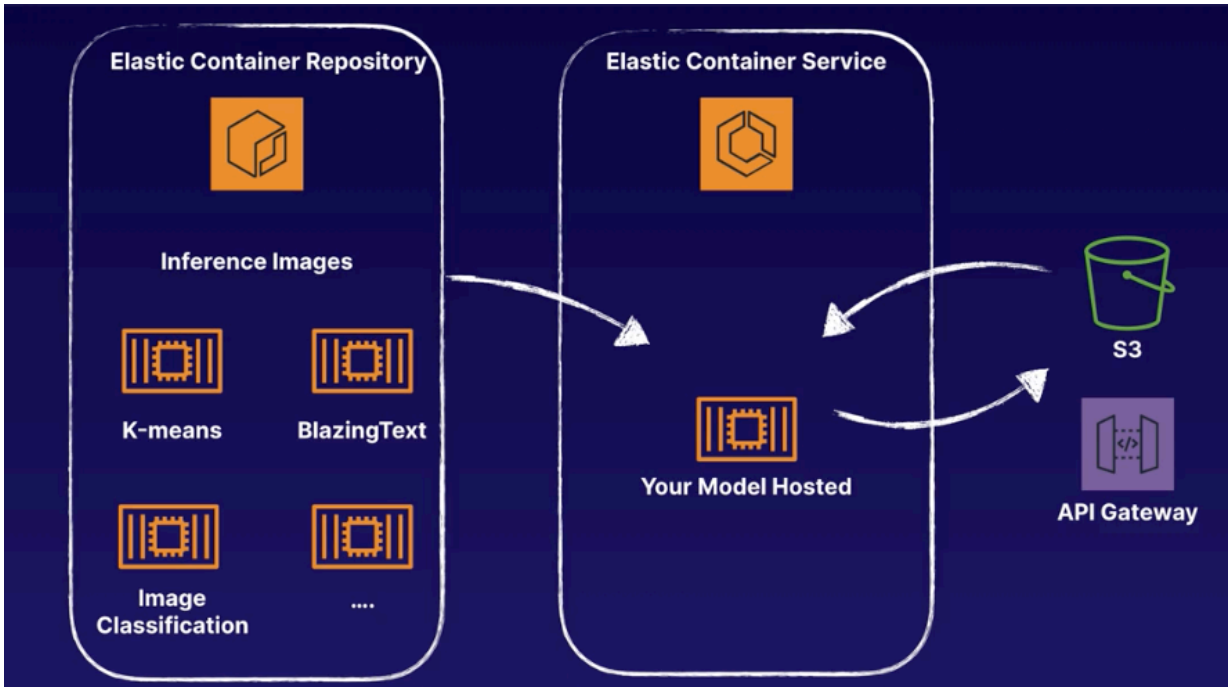


Amazon fetch the proper image and spin up inside Pagemaker space
Then the job ill access the data on S3



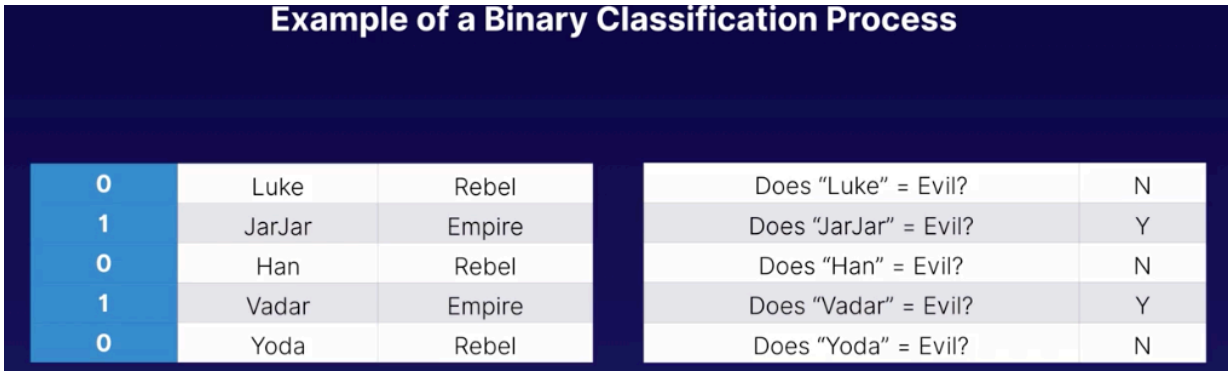
To note we can also specify our own docker file



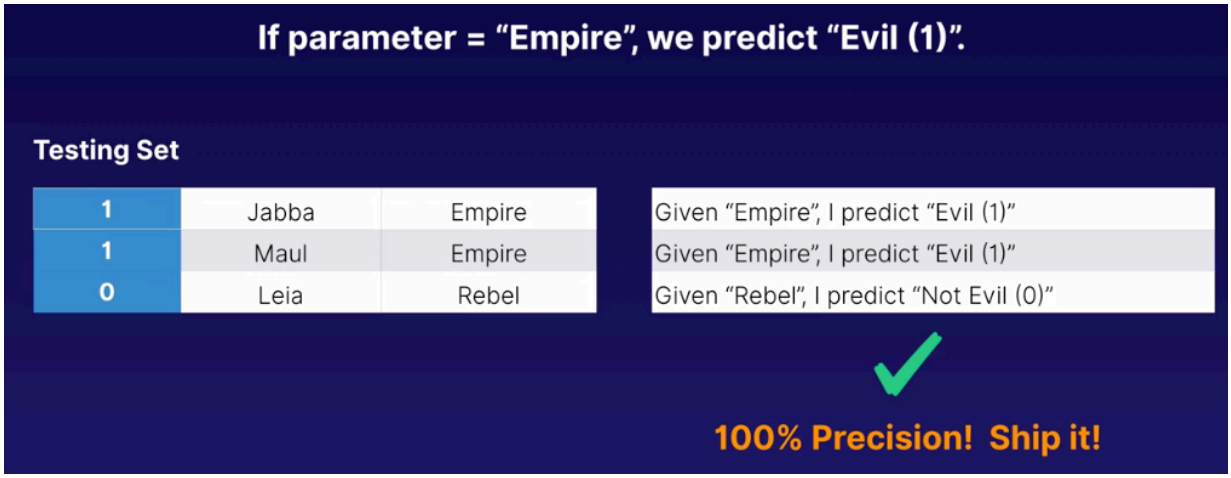


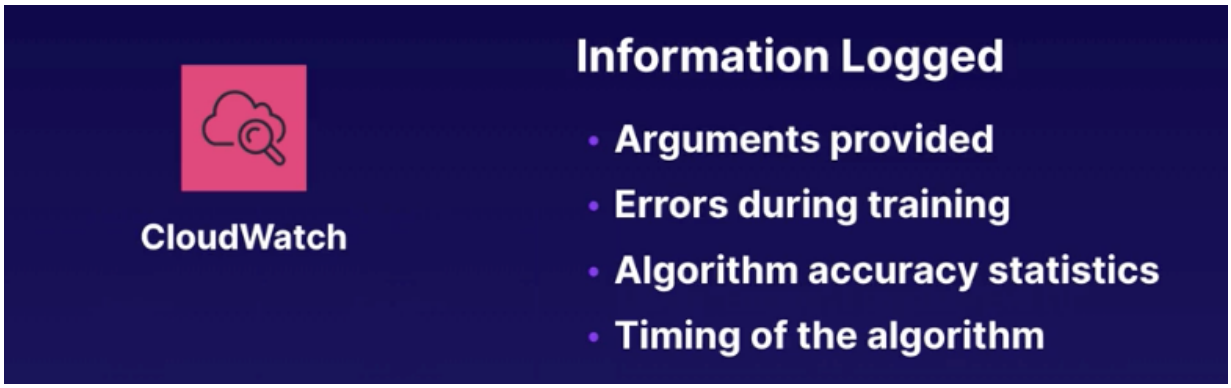
API GW allow us to use REST calls on our own program.

Example of Binary Classification:

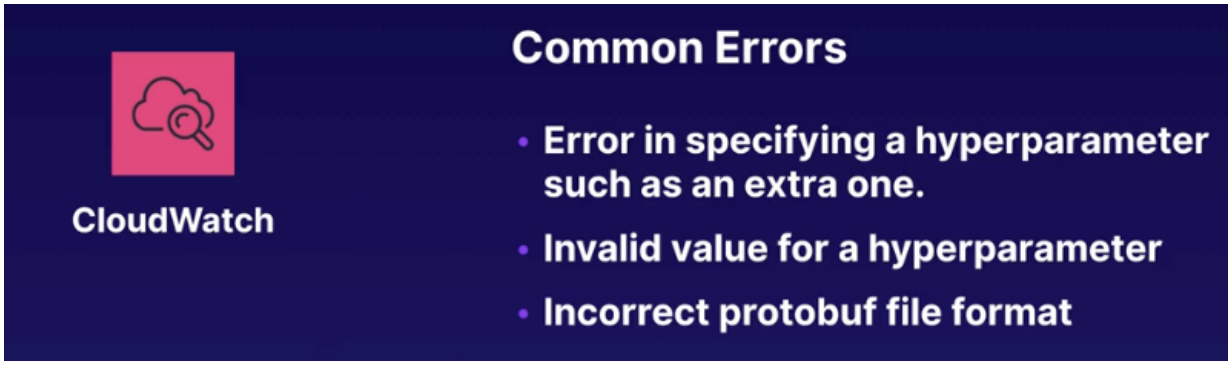


Sometimes, super strong relationship

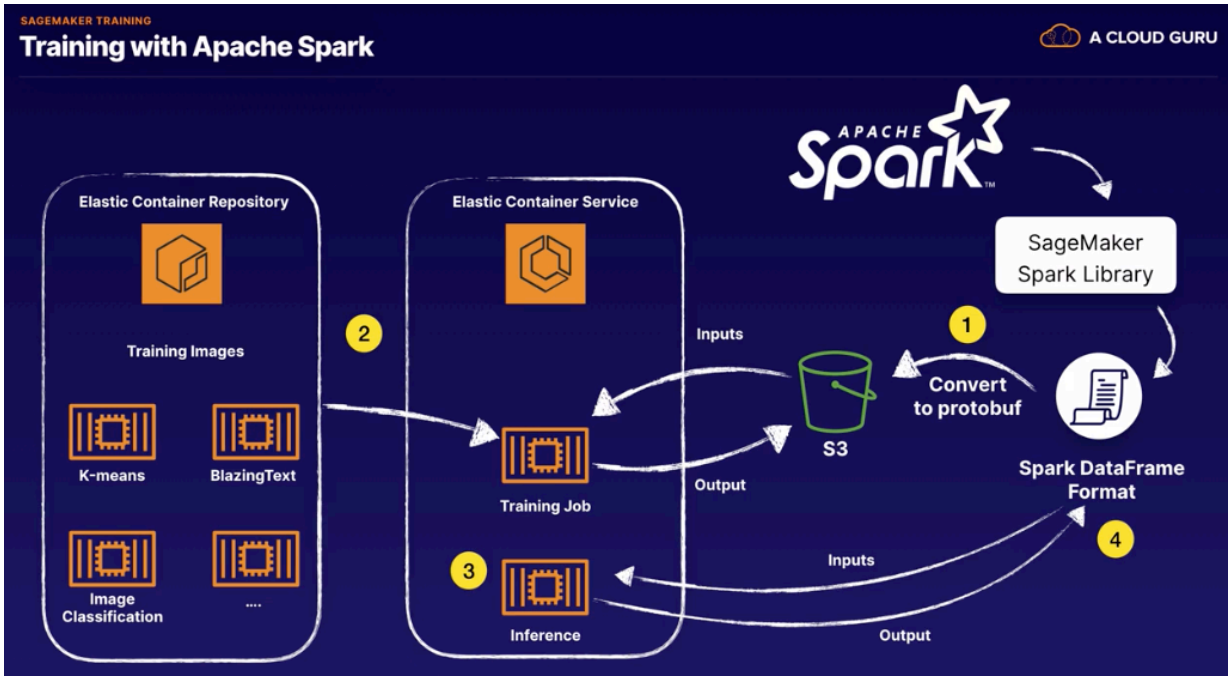




Common errors



Training with Apache Spark
Use Spark library: converts Spark Dataframe format into protobuff.
We can send it to S3, then similar training job as the rest



Modeling Exam Tips

Model Design

- Selecting a model that is a good fit for the objective.
- Choosing the proper ML approach for your objective (regression, binary classification, etc.).
- Choose proper evaluation strategies for your model.
- Steps for training a model.

Data Preparation

- Understand concepts of Training Data and Testing Data.
- Identify potential biases introduced in an insufficient split strategy.
- Know when to use sequential splits versus randomized splits and what additional measures could be used to increase training data value.

Sequential split for Timer Series data makes sense
Randomized split would be for other cases to make sure we have good values in training and test data

Model Training

- Multiple Options for training: SageMaker Console, Apache Spark, Custom Code via SDK, Jupyter Notebook.
- Be familiar with default data types SageMaker algorithms support and the recommended format for best performance.
- Know the difference between a Hyperparameter and Parameter.
- Understand the repository and container image concept for SageMaker training.
- Understand the process if you wish to provide your own algorithm.
- Understand the process for using Apache Spark to interact with SageMaker.

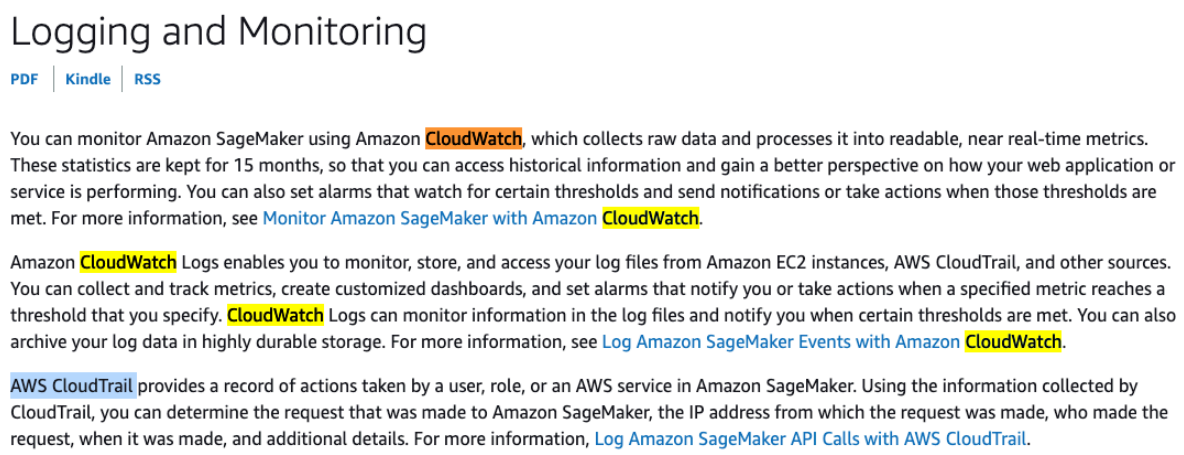
model **parameters** are estimated from data automatically and model

hyperparameters are set manually and are used in processes to help estimate model parameters.
Model **hyperparameters** are often referred to as parameters because they are the parts of the machine learning that must be set manually and tuned.

A **parameter** is something which the deep learning model learns. For example, the weights in **between** layers, **parameters** of batchnorm layer, etc.
On the other hand, a **hyperparameter** is something which you don't learn. For example, the number of layers, number of nodes **in a** layer, size of a convnet, etc.

Additional Resources:
- AWS White Paper: managing ML projects: https://d1.awsstatic.com/whitepapers/aws-managing-ml-projects.pdf
- Using own algorithms: https://docs.aws.amazon.com/sagemaker/latest/dg/your-algorithms.html
- Amazon ML Key concepts: https://docs.aws.amazon.com/machine-learning/latest/dg/amazon-machine-learning-key-concepts.html

Logging and Monitoring
https://docs.aws.amazon.com/sagemaker/latest/dg/sagemaker-incident-response.html



Logging and Monitoring

PDF | Kindle | RSS

You can monitor Amazon SageMaker using Amazon CloudWatch, which collects raw data and processes it into readable, near real-time metrics. These statistics are kept for 15 months, so that you can access historical information and gain a better perspective on how your web application or service is performing. You can also set alarms that watch for certain thresholds and send notifications or take actions when those thresholds are met. For more information, see Monitor Amazon SageMaker with Amazon CloudWatch.

Amazon CloudWatch Logs enables you to monitor, store, and access your log files from Amazon EC2 instances, AWS CloudTrail, and other sources. You can collect and track metrics, create customized dashboards, and set alarms that notify you or take actions when a specified metric reaches a threshold that you specify. CloudWatch Logs can monitor information in the log files and notify you when certain thresholds are met. You can also archive your log data in highly durable storage. For more information, see Log Amazon SageMaker Events with Amazon CloudWatch.

AWS CloudTrail provides a record of actions taken by a user, role, or an AWS service in Amazon SageMaker. Using the information collected by CloudTrail, you can determine the request that was made to Amazon SageMaker, the IP address from which the request was made, who made the request, when it was made, and additional details. For more information, Log Amazon SageMaker API Calls with AWS CloudTrail.

:1 vs :latest container tags

Algorithms that are *parallelizable* can be deployed on multiple compute instances for distributed training. For the **Training Image and Inference Image Registry Path** column, use the `:1` version tag to ensure that you are using a stable version of the algorithm. You can reliably host a model trained using an image with the `:1` tag on an inference image that has the `:1` tag. Using the `:latest` tag in the registry path provides you with the most up-to-date version of the algorithm, but might cause problems with backward compatibility. Avoid using the `:latest` tag for production purposes.

CreateModel API call

https://docs.aws.amazon.com/sagemaker/latest/APIReference/API_CreateModel.html

Creates a model in Amazon SageMaker. In the request, you name the model and describe a primary container. For the primary container, you specify the Docker image that contains inference code, artifacts (from prior training), and a custom environment map that the inference code uses when you deploy the model for predictions.

Use this API to create a model if you want to use Amazon SageMaker hosting services or run a batch transform job.

To host your model, you create an endpoint configuration with the CreateEndpointConfig API, and then create an endpoint with the CreateEndpoint API. Amazon SageMaker then deploys all of the containers that you defined for the model in the hosting environment.

To run a batch transform using your model, you start a job with the CreateTransformJob API. Amazon SageMaker uses your model and your dataset to get inferences which are then saved to a specified S3 location.

In the CreateModel request, you must define a container with the PrimaryContainer parameter.

In the request, you also provide an IAM role that Amazon SageMaker can assume to access model artifacts and docker image for deployment on ML compute hosting instances or for batch transform jobs. In addition, you also use the IAM role to manage permissions the inference code needs. For example, if the inference code access any other AWS resources, you grant necessary permissions via this role.