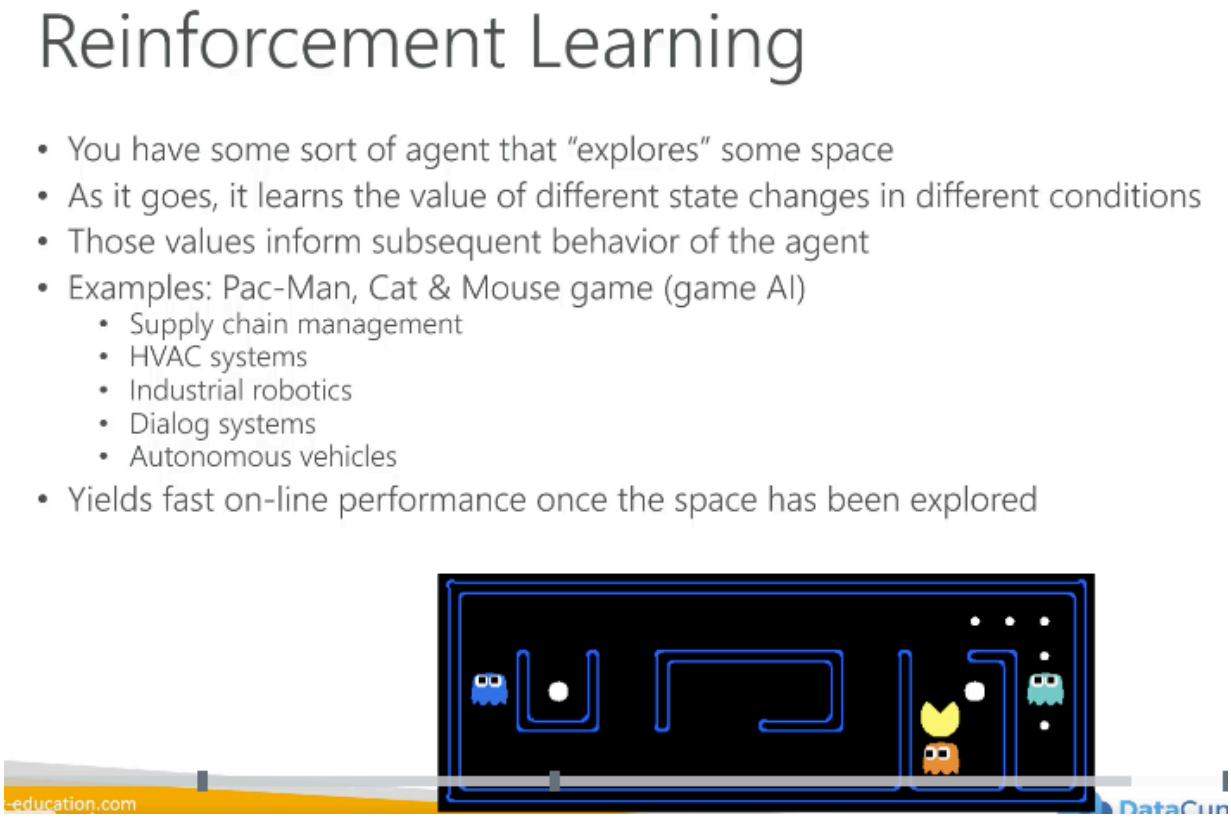
Udemy - 3 - Modeling - ML services - part 2

Reinforcement Learning in SageMaker



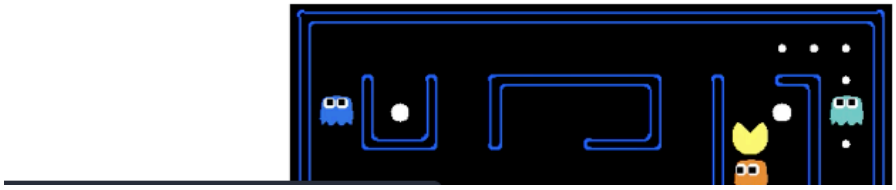
Agent = PAC-MAN
As the agent goes randomly through the space, it learns based on positive/ negative rewards
Once it has explored the entire space, the model is quick to deploy.
Online performance is then very fast based on where the agent is, he knows what to do.

Q-Learning

- A specific implementation of reinforcement learning
- You have:
  - A set of environmental states *s*
  - A set of possible actions in those states *a*
  - A value of each state/action Q
- Start off with Q values of 0
- Explore the space
- As bad things happen after a given state/action, reduce its Q
- As rewards happen after a given state/action, increase its Q

Q-Learning

- What are some state/actions here?
  - Pac-man has a wall to the West
  - Pac-man dies if he moves one step South
  - Pac-man just continues to live if going North or East
- You can "look ahead" more than one step by using a discount factor when computing Q (here s is previous state, s' is current state)
  - Q(s,a) += discount * (reward(s,a) + max(Q(s')) – Q(s,a))



The exploration problem

- How do we efficiently explore all of the possible states?
  - Simple approach: always choose the action for a given state with the highest Q. If there's a tie, choose at random
    - But that's really inefficient, and you might miss a lot of paths that way
  - Better way: introduce an epsilon term
    - If a random number is less than epsilon, don't follow the highest Q, but choose at random
    - That way, exploration never totally stops
    - Choosing epsilon can be tricky

Fancy Words

- Markov Decision Process
  - From Wikipedia: **Markov decision processes (MDPs)** provide a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker.
  - Sound familiar? MDP's are just a way to describe what we just did using mathematical notation.
  - States are still described as s and s'
  - State transition functions are described as $P_a(s, s')$
  - Our "Q" values are described as a reward function $R_a(s, s')$
- Even fancier words! An MDP is a *discrete time stochastic control process.*

So to recap

- You can make an intelligent Pac-Man in a few steps:
  - Have it semi-randomly explore different choices of movement (actions) given different conditions (states)
  - Keep track of the reward or penalty associated with each choice for a given state/action (Q)
  - Use those stored Q values to inform its future choices
- Pretty simple concept. But hey, now you can say you understand reinforcement learning, Q-learning, Markov Decision Processes, and Dynamic Programming!

Reinforcement Learning in SageMaker

- Uses a deep learning framework with Tensorflow and MXNet
- Supports Intel Coach and Ray Rllib toolkits.
- Custom, open-source, or commercial environments supported.
  - MATLAB, Simulink
  - EnergyPlus, RoboSchool, PyBullet
  - Amazon Sumerian, AWS RoboMaker

Distributed Training with SageMaker RL

- Can distribute training and/or environment rollout
- Multi-core and multi-instance

Reinforcement Learning: Key Terms

- Environment
  - The layout of the board / maze / etc
- State
  - Where the player / pieces are
- Action
  - Move in a given direction, etc
- Reward
  - Value associated with the action from that state
- Observation
  - i.e., surroundings in a maze, state of chess board

Reinforcement Learning: Hyperparameter Tuning

- Parameters of your choosing may be abstracted
- Hyperparameter tuning in SageMaker can then optimize them

Reinforcement Learning: Instance Types

- No specific guidance given in developer guide
- But, it's deep learning – so GPU's are helpful
- And we know it supports multiple instances and cores

Automatic Model Tuning

Hyperparameter tuning

- How do you know the best values of learning rate, batch size, depth, etc?
- Often you have to experiment
- Problem blows up quickly when you have many different hyperparameters; need to try every combination of every possible value somehow, train a model, and evaluate it every time

Automatic Model Tuning

- Define the hyperparameters you care about and the ranges you want to try, and the metrics you are optimizing for
- SageMaker spins up a "HyperParameter Tuning Job" that trains as many combinations as you'll allow
  - Training instances are spun up as needed, potentially a lot of them
- The set of hyperparameters producing the best results can then be deployed as a model
- **It learns as it goes**, so it doesn't have to try every possible combination

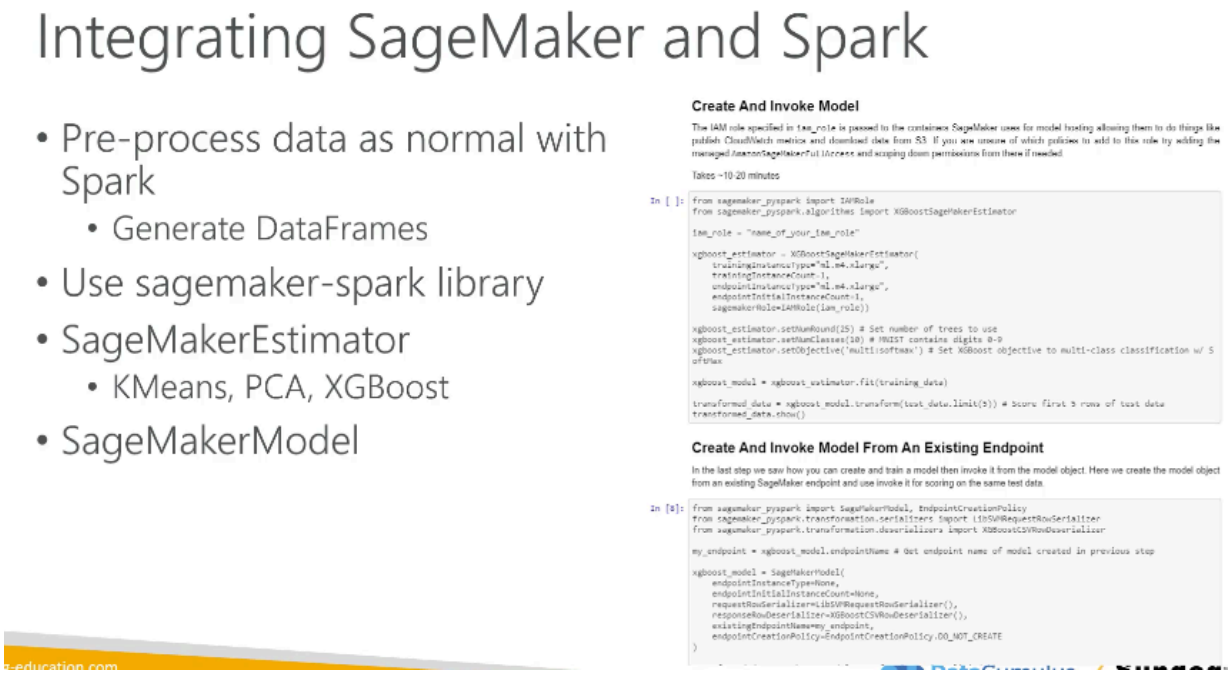
Automatic Model Tuning: Best Practices

- Don't optimize too many hyperparameters at once
- Limit your ranges to as small a range as possible
- Use logarithmic scales when appropriate
- Don't run too many training jobs concurrently
  - This limits how well the process can learn as it goes
- Make sure training jobs running on multiple instances report the correct objective metric in the end

Apache Spark with SageMaker
=> can use SageMaker within a Spark driver script
=> perfect to **train with Spark and infer with SageMaker**.

Ex: process data with Apache spark as usual.
In the end we get a DataFrame with all the pre-process data.

Then instead of using SparkMLLib, we can use **SageMaker Estimator** to run ML: Means, PCA, XGboost
In the end it will create a SageMaker model that we will sue to get inferences.



Integrating SageMaker and Spark

- Connect notebook to a remote EMR cluster running Spark (or use Zeppelin)
- Training dataframe should have:
  - A features column that is a vector of Doubles
  - An optional labels column of Doubles
- Call fit on your SageMakerEstimator to get a SageMakerModel
- Call transform on the SageMakerModel to make inferences
- Works with Spark Pipelines as well.

```
val estimator = new KMeansSageMakerEstimator(
 sagemakerRole = IAMRole(roleArn),
 trainingInstanceType = "ml.p2.xlarge",
 trainingInstanceCount = 1,
 endpointInstanceType = "ml.c4.xlarge",
 endpointInitialInstanceCount = 1)
 .setK(10).setFeatureDim(784)
 // train
 val model = estimator.fit(trainingData)
 val transformedData = model.transform(testData)
 transformedData.show
```

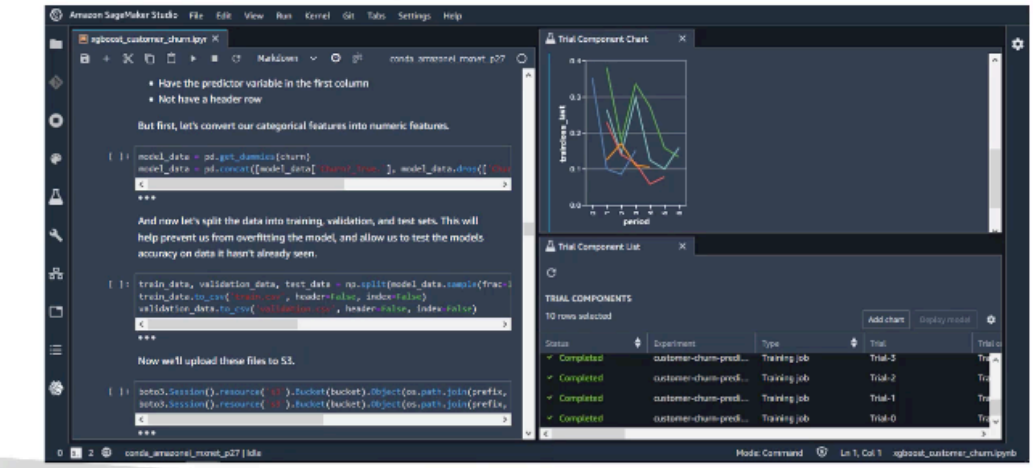
Why bother?

- Allows you to combine pre-processing big data in Spark with training and inference in SageMaker.

SageMaker Studio and new SageMaker for 2020

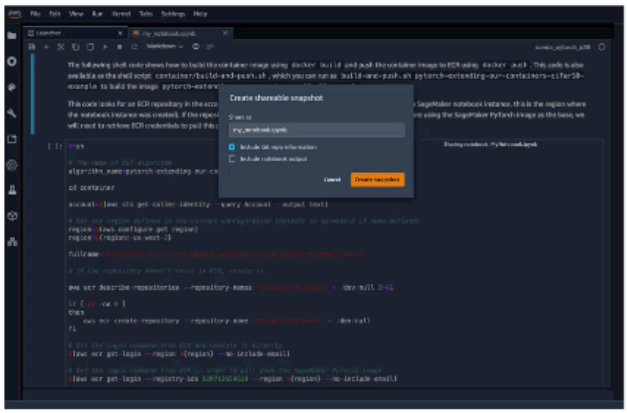
SageMaker Studio

- Visual IDE for machine learning!
- Integrates many of the features we're about to cover.



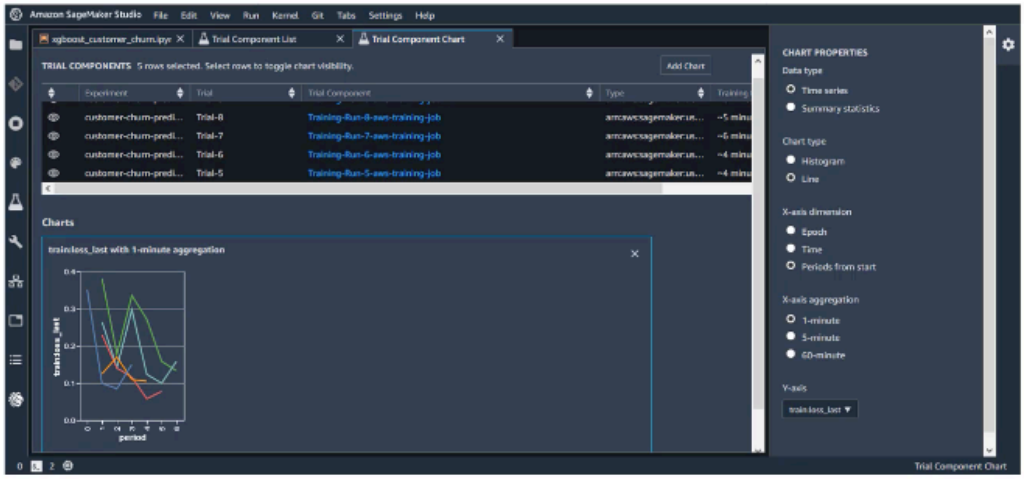
SageMaker Notebooks

- Create and share Jupyter notebooks with SageMaker Studio
- Switch between hardware configurations (no infrastructure to manage)



SageMaker Experiments

- Organize, capture, compare, and search your ML jobs

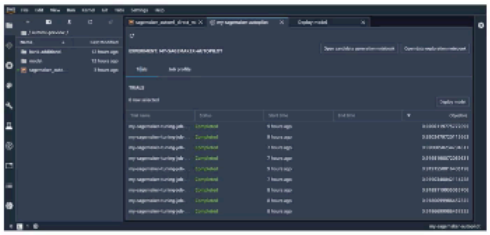


SageMaker Debugger

- Saves internal model state at periodical intervals
  - Gradients / tensors over time as a model is trained
  - Define rules for detecting unwanted conditions while training
  - A debug job is run for each rule you configure
  - Logs & fires a CloudWatch event when the rule is hit

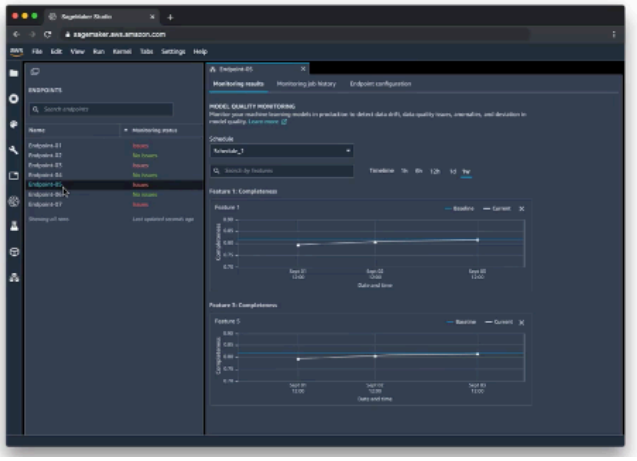
SageMaker Autopilot

- Automates:
  - Algorithm selection
  - Data preprocessing
  - Model tuning
  - All infrastructure
- It does all the trial & error for you
- More broadly this is called AutoML



SageMaker Model Monitor

- Get alerts on quality deviations on your deployed models
- Visualize data drift
  - Example: loan model starts giving people more credit due to drifting or missing input features
- No code needed



Putting them together