Published in Towards Data Science

Jojo John Moolayil Follow

Jun 8, 2019 · 8 min read ★ · Listen

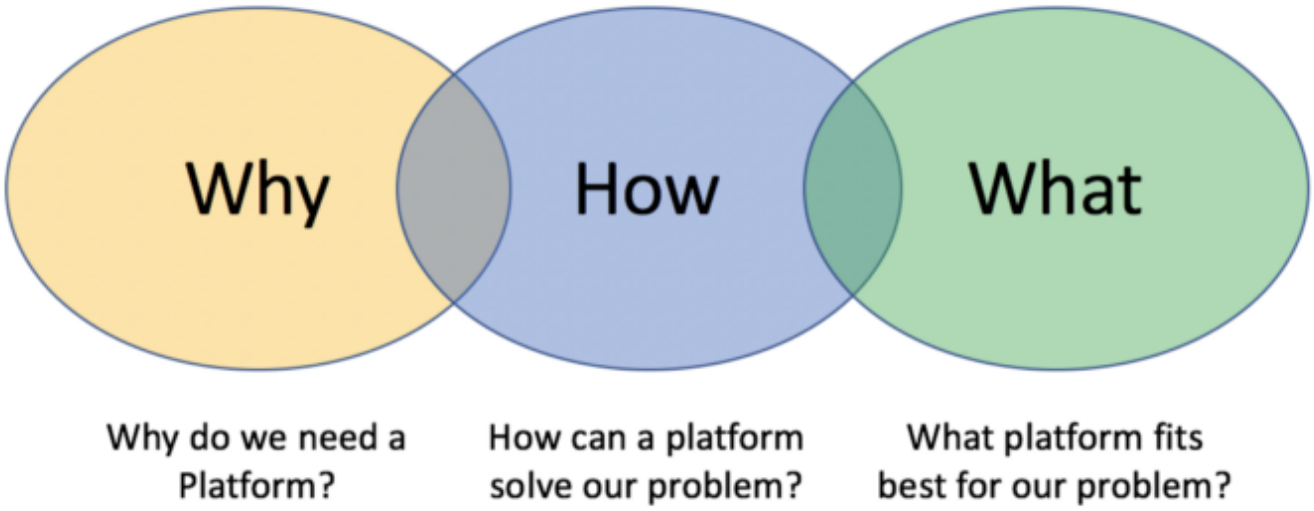
Why do we need AWS SageMaker?

A more serious thought for large Enterprises developing Machine Learning products and solutions.

Today, there are several platforms available in the industry that aid software developers, data scientists as well as a layman in developing and deploying machine learning models within no time. Some platforms have made big claims to democratize AI and data science to an extent where you can solve your data science problem with almost no coding skills, while some others claim the fastest and easiest way to deploy a solution in the cloud. All these claims, some very profound, would make you wonder one simple question — **What platform works best for me?**

start by understanding WHY do we need a platform in the first place. Once we understand the '**WHY**', we then understand '**HOW**' a given platform will solve the problem for us and then we can finally answer the ultimatum '**WHAT** platform works best for me?'

Starting with the WHY always heads us in the right direction.



Inspired by Simon Sinek's 'Start with WHY'

In this blog, my objective is to enlighten you by answering the 'WHY', 'HOW' and 'WHAT' sequences of questions for selecting the best platform for a significant majority of the machine learning solutions in enterprises that we solve in the present day. I would then like to connect '**AWS Sagemaker**' as a valid candidate for the problem and emphasize the reasons outlined with 'Why do we need AWS Sagemaker?' There are obviously other groups of problems and experiments in enterprises, for which this wouldn't be the best choice; I will try to also highlight a few areas where this would not be the best choice.

Let's start with the 'WHY'.

TAKE 1 — WHY do we need a platform for developing Machine Learning solutions?

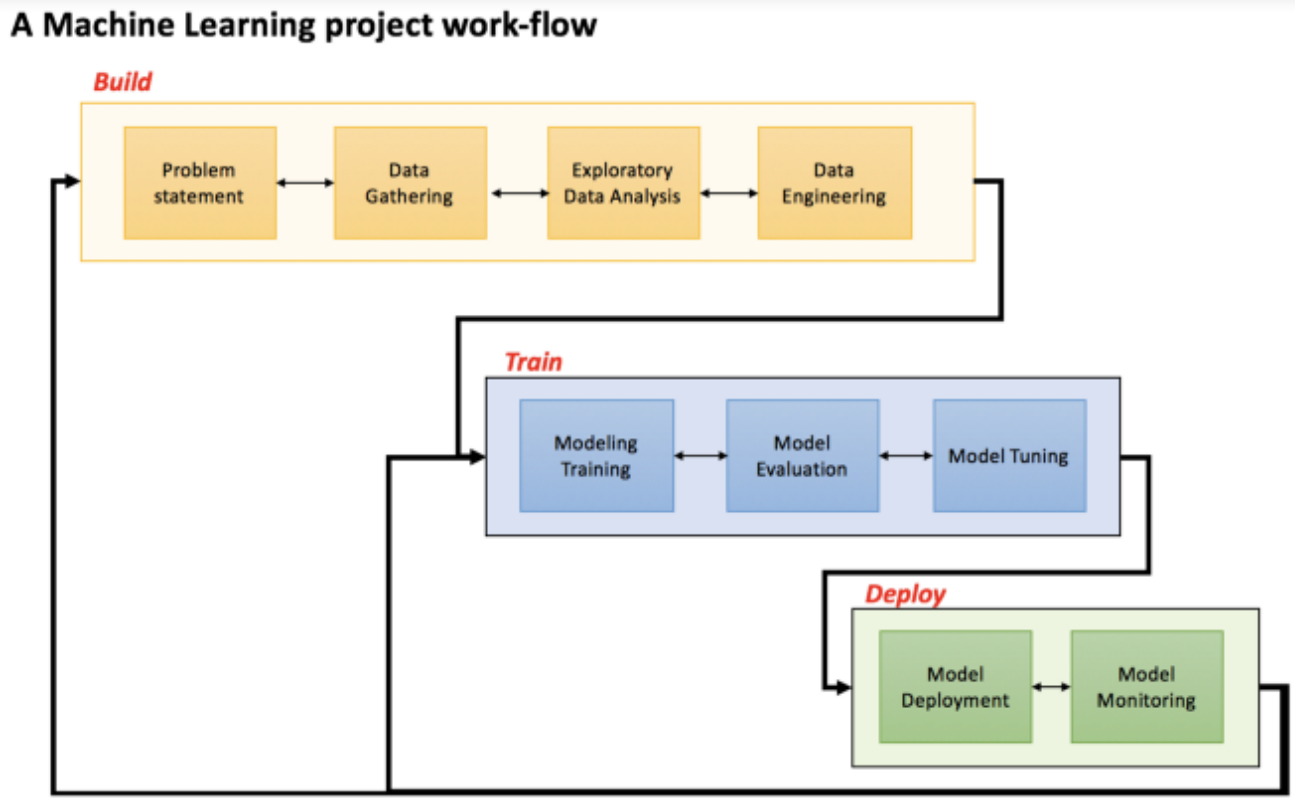


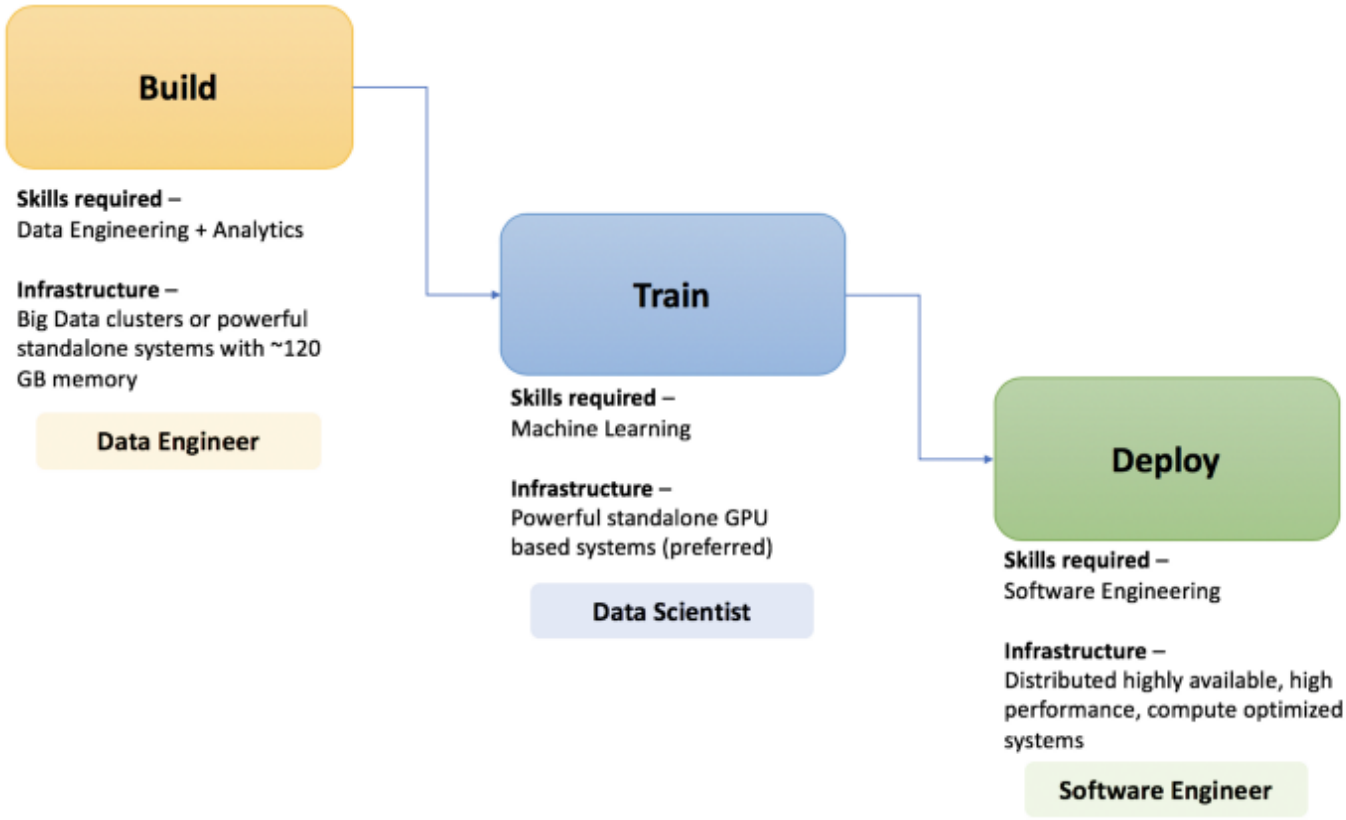
Fig 1- An illustration for a typical Machine Learning project workflow

We can typically categorize the list of associated tasks into 3 broad areas -

1. **Build**: Here we define the problem, gather data, analyze it and then clean, transform and engineer the data into the desired form.
2. **Train**: We leverage one or more machine learning algorithms and train (fit) the model to learn the patterns from the engineered data. Based on the model's performance, we tune hyperparameters and retrain and repeat the process until we have fairly acceptable results.
3. **Deploy**: We deploy the model into a production system where it caters its services to the larger ecosystem.

This entire process is highly iterative and changes can be expected to loop back the progress to any state in the entire process (as shown in the diagram). In most of the Proof of Concepts (PoC) we experiment and the available study resources over the internet

Consider a fairly complex ML problem we would deal in an enterprise. Say, you have a very large dataset with millions of records (~10GB — 1TB) to process and would need at least a few hundred iterations (say, for a deep neural network) and around 100,000 API calls per minute once the service is deployed. Then, the vanilla PoCs we developed during experimentation is no longer a comparison for the use-case. Each group (Build, Train, Deploy) would need a different kind of compute infrastructure, a different approach to execute the problem and a distinct set of disciplines to accomplish. Fig 2 (below), helps in understanding how each of these groups within the workflow, fork out as an independent branch with diverse requirements of skills and infrastructure to execute. **Build** phase requires a combination of Data Engineering and Data Analytics. **Train** phase requires Machine Learning skills and finally, **Deploy** phase requires primarily Software Engineering skills and a bit of ML skills. Each of these phases would need a person with a completely different skillset.



Usually, executing a large-scale ML project demands much more from a team than what

Data Engineer — who engineers the entire data pipeline to gathers the required data from several heterogeneous sources under one roof in the desired form. A **Data Scientist**, who analyzes the data and develops machine learning models for the use-case. Finally, a **Software Engineer** who would take the model developed by the scientist into a production system.

The major problem in this equation lies in the transition from **BUILD to TRAIN** and then to **DEPLOY**. Data scientists are not software engineers and similarly, software engineers are not data scientists. It is indeed a mammoth task for a data scientist to deploy a machine learning solution (that he researched and prototyped) into a full-scale web service (an API that can be integrated into a software ecosystem). Data scientists lack the required software skills to take a research prototype and offline machine learning model into a large complex model (service) that can inference thousands of API calls in real-time.

On the other hand, software engineers are proficient in this task and can easily have this done for a large software system. However, the lack of understanding of machine learning and math skills brings in several impediments for the implementation. Developing a full-scale ML service does require the developer to have a deeper understanding of the process which in most cases is quite overwhelming.

Most organization follow the same practice of hiring data scientists to research, experiment, prototype and develop a PoC that can validate an ML business use-case. Later, hire a software development team to take the PoC into a full-blown and scalable web-service or software product as per the required business standards. However, the collaboration between software engineers and data scientists are again a not so easy one to accomplish. It is indeed far too overwhelming for each group to understand the other group's requirements, tech-language, and ideas.

Now that we fairly understand the pain points, we can take a stab at formulating a more succinct question with **WHY**.

TAKE 2 — Why do we really need a platform?

into the required software systems and abstract the ML service as just another service wrapped around an API. (Software engineers love APIs). Therefore, we need a platform that can enable a data scientist with the necessary tools to independently execute a machine learning project in a truly end-to-end way.

HOW can a platform solve this problem?

Given, the 3 phases in the project are extremely diverse, we need to devise a solution with the data scientist in the driver's seat. We can solve the problems discussed above if we have a platform that supplements the required remainder skills in each of these phases with neat abstractions while still being highly effective and flexible for a data scientist to deliver results.

Thus, we need a platform where the data scientist will be able to leverage his existing skills to engineer and study data, train and tune ML models and finally deploy the model as a web-service by dynamically provisioning the required hardware, orchestrating the entire flow and transition for execution with simple abstraction and provide a robust solution that can scale and meet demands elastically.

With the succinct question framed and answered, we can now come to the final question.

What platform works best for me?

Here, I can say, **AWS Sagemaker** fits best for us. It provides Jupyter NoteBooks running R/Python kernels with a compute instance that we can choose as per our data engineering requirements on demand. We can visualize, process, clean and transform the data into our required forms using the traditional methods we use (say Pandas + Matplotlib or R +ggplot2 or other popular combinations). Post data engineering, we can train the models using a different compute instance based on the model's compute demand, say memory optimized or GPU enabled. Leverage a smart default high-performance hyperparameter tuning settings for a variety of models. Leverage performance-optimized algorithms from the rich AWS library or bring our own algorithms through industry standard containers. Also, deploy the trained model as an API, again using a different compute instance appropriate to meet business requirements and scale elastically. And the entire process of provisioning hardware instances, running high capacity data jobs, orchestrating the entire flow with simple commands while abstracting the mammoth complexities and finally enabling serverless elastic deployment works with a few lines of code and yet is cost effective. Sagemaker is a game-changing solution for the enterprise.

Some scenarios where Sagemaker might not be suitable.

The problems we discussed above though represent a vast majority of ML projects, there are certainly few scenarios where this might not be a valid option. Sagemaker, expects you to code your data engineering and analysis needs(in your choice of language). The entire orchestration of infrastructure and transition from one phase to another though happens with few lines code; definitely needs a small amount of code. For professionals, who prefer a drag and drop (absolutely no code) solutions for the data engineering and modeling phase, Azure ML studio would be a more ideal choice. Both Azure ML Studio and AWS Sagemaker are great platforms for developing ML solutions but are targetted for a completely different set of users. This article by Steve Dille does a great job of enlightening us about how both are different.

Concluding thoughts

AWS Sagemaker has been a great deal for most data scientists who would want to

experiments and supplements the remainder necessary skills with easy abstracted tools similar to our existing workflow.