OPEN DATA SCIENCE CONFERENCE

Boston | April 30 - May 4, 2019

#ODSC

@ODSC

#ODSC

BOSTON
APR 30 - MAY 3

Deploying Data Science Applications

Garrett Hoffman

Director of Data Science,
StockTwits

Talk Overview

- The Data Product “Needs” Gap
- Machine Learning
- ETL / Batch Processing
- Streaming Analytics

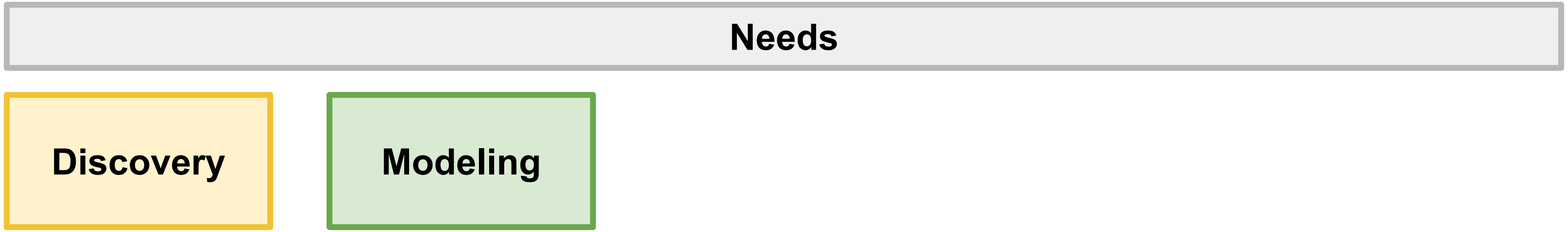
https://github.com/GarrettHoffman/ODSC_East_2019_Deploy_DS_Apps

The Data Product “Needs” Gap

The Challenge

Small Data Science teams are getting really really good at research, prototyping and one off analyses but maturing teams struggle to move their work from prototype to production data products / applications.

Data R&D / Exploration Needs

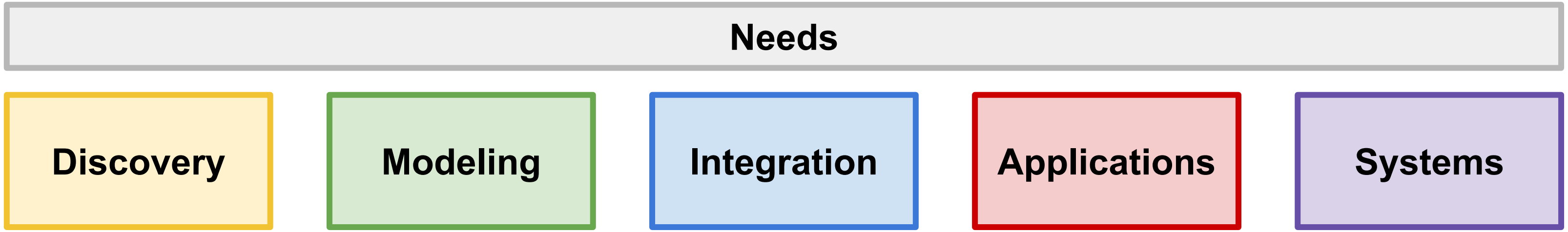


Needs

Discovery

Modeling

Data Product / Application Needs



Needs

Discovery

Modeling

Integration

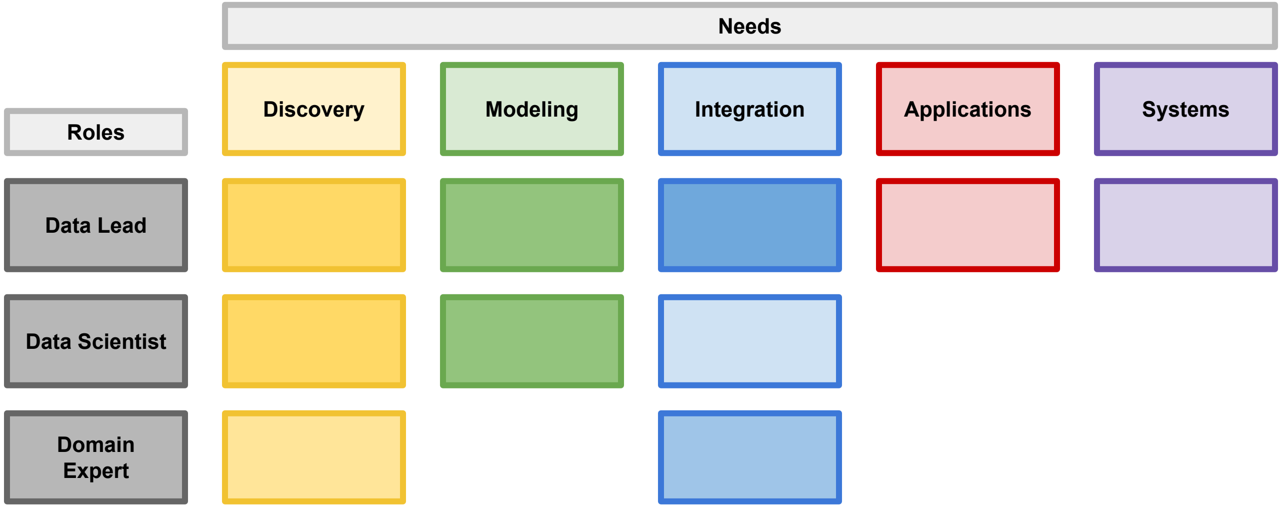
Applications

Systems

Data Product / Application Resources

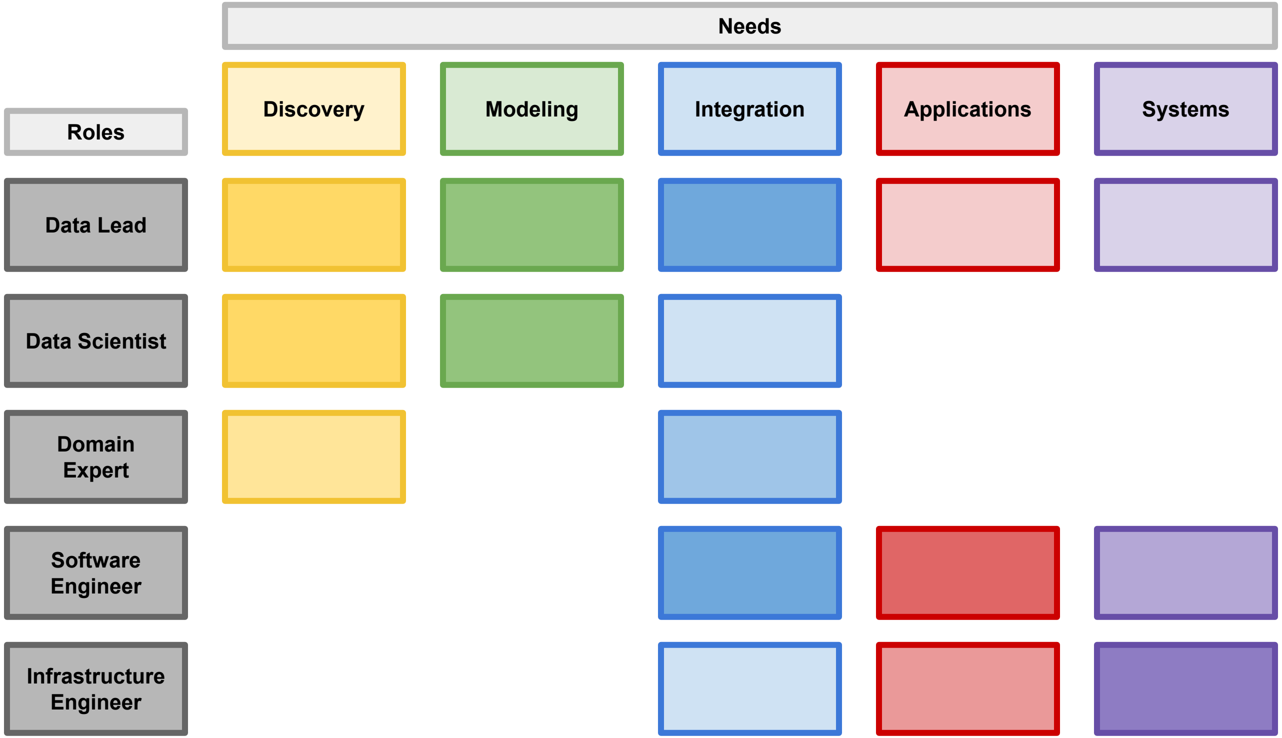
	Needs				
Roles	Discovery	Modeling	Integration	Applications	Systems
Data Scientist	🟨	🟩	🟦		
Domain Expert	🟨		🟦		

Data Product / Application Resources



	Needs				
Roles	Discovery	Modeling	Integration	Applications	Systems
Data Lead	🟨	🟩	🟦	🟥	🟪
Data Scientist	🟨	🟩	🟦		
Domain Expert	🟨		🟦		

Data Product / Application Resources



	Needs				
Roles	Discovery	Modeling	Integration	Applications	Systems
Data Lead	🟨	🟩	🟦	🟥	🟪
Data Scientist	🟨	🟩	🟦		
Domain Expert	🟨		🟦		
Software Engineer			🟦	🟥	🟪
Infrastructure Engineer			🟦	🟥	🟪

Filling in the Gap

- Foster better collaboration with Engineering Team by giving Data Scientists the vocabulary to speak the language of applications and systems
- Implement tooling that abstracts away some of the application and systems requirements
- Teach Data Scientists how to use the tools, creating more generalists

Machine Learning

Machine Learning - Problem Characterization

- We have some data related to some task and want to train some sort of model with which we can make inferences about future data.

- Training this model requires experimentation so we have a compute heavy task (that might require specific hardware) that we repeat many times while tinkering.

- When we are satisfied with our performance we want an API that we can send data to and get back predictions.

Chart vs. Meme Classification

Some StockTwits users are extremely serious traders who like to look at annotated price charts while others just like to have fun and joke about the market. Being able to classify an image that is posted to StockTwits as either a chart or a meme can make it easier for users to find the type of content that they are looking for. We want to train a ML model to perform this classification and deploy it to be able to classify incoming images.

Basic Approach - Flask and Docker

- Train model locally (or on cloud GPU instance) using a Jupyter Notebook to interactively fiddle and experiment
- When we are satisfied with the model, we save the model to disk
- Write a Flask API that loads my model and takes in my image as an input and returns my model classification results
- Build a docker image of the API and push to remote repository
- Deploy the docker image with Kubernetes

Basic Approach - Technologies

- ***Flask*** - Python “micro” framework for writing simple and lightweight web applications and REST APIs very easily - http://flask.pocoo.org/docs/1.0/

- ***Docker*** - Software for building and running containers. Containers are isolated environments that bundle their own application, libraries, configuration and dependencies - https://docs.docker.com/get-started/

- ***AWS ECR*** - Private docker image repository. This is where we store all of our images to eventually deploy. - https://kubernetes.io/docs/home/

Basic Approach - Technologies

- ***Kubernetes*** - Orchestration system for automating deployment, scaling, and operations of containers. We define our deployment with a YAML configuration file and can deploy a job with a single CLI command - https://kubernetes.io/docs/home/

Basic Approach - Docker/ Flask Demo

Basic Approach - Drawbacks

- Hard to version control notebooks, models
- Hard to keep track of parameters
- Decent amount of developer work to get model deployed
- Data is usually stored wherever the model is run so it is hard to keep track of and version control data
- Need to manage hardware training hardware and hardware that model is deployed on (kind of)
- Involves technologies that are not usually associated with the data science “stack” (yet)

Better Approach - AWS Sagemaker

- Write a training script with model training code that takes command line inputs for our data and parameters
- Create a hyperparameter configuration file
- Write and Run a script to train submit our training job to Sagemaker and deploy a Sagemaker endpoint when the model is done training

Better Approach - Technologies

- ***AWS Sagemaker*** - AWS managed Machine Learning service consisting of hosted Jupyter Notebook Instances, Model Training and Hyperparameter Tuning Jobs, and hosted endpoints for Model Inference - https://docs.aws.amazon.com/sagemaker/index.html

Better Approach - Sagemaker Demo

Better Approach - Addressing Drawbacks

- Training files, parameters/experiments can be version controlled via traditional software tools like git
- Model endpoint is deployed with one line of code (almost)
- Data is stored in S3, which is version controlled, and pulled into training job
- “Serverless” GPU and don’t have to manage inference infrastructure
- Python SDK means we can deploy in a familiar language

ETL / Batch Processing

ETL / Batch Processing - Problem Characterization

- We have a compute heavy and possibly memory intensive task that usually requires extracting some raw data, manipulating it with some joins or aggregations and performing some calculations with it and storing this processed result somewhere else for later use.

- These are often long running jobs and typically we need to perform this same processing task on a regular basis, such as a nightly run. Often we have batch jobs that are related in some aspects but a little different in others.

Graph Based User and Room Recs

To facilitate discovery of content on StockTwits we would like to give users personalized recommendations of users to follow and rooms to join. A simple approach to accomplish this is to use the users social graph. We will look at who the user his engaging with the most and recommend users and rooms that those users follow. We want to run this once a night to capture changing behaviors of the user.

Basic Approach - K8s Cron Job

- Write a script to process user recommendations and store results in S3
- Write a script to process room recommendations and store results in S3
- Build a docker image that runs each script and push to remote repository
- Deploy the docker images as a CronJob with Kubernetes

Basic Approach - Technologies

- ***Cron*** - A software utility to schedule jobs on unix-like operating systems. Use config file to schedule the execution of unix commands or shell scripts

- ***K8s CronJob*** - A type of Kubernetes workload that creates jobs on a time-based schedule. We use this to run docker containers at schedule intervals - https://kubernetes.io/docs/concepts/workloads/controllers/cron-jobs/

Basic Approach - CronJob Demo

Basic Approach - Drawbacks

- With old school cron jobs all of our code lives on a single machine so managing dependencies can get tricky (not anymore)
- We duplicate a lot of processing across jobs. If we split this code into separate jobs we will need to worry about managing downstream dependencies (failures and timing)
- Have low visibility inside the job to monitor, optimize and debug

Better Approach - Airflow

- Write a script for each “task” in our recommendation processing pipeline. Tasks can be shared across our user and room recommendation jobs.
- Build a docker image that runs each script and push to remote repository
- Define and Airflow DAG that runs these tasks on our Kubernetes cluster. In our DAG definition we define task dependencies.
- Add our new DAG to our DAG bag on our Airflow server

Better Approach - Technologies

- ***Airflow*** - Airflow is a platform to programmatically author, schedule and monitor task dependent workflows. Define workflows as directed acyclic graphs (DAGs) of tasks. Airflow comes out of box with a web interface that makes it easy to visualize pipelines running in production, monitor progress, and troubleshoot issues - https://airflow.apache.org

Better Approach - Airflow Demo

Better Approach - Addressing Drawbacks

- “Traditional” approach to Airflow also has issues with dependency management but this is addressed by defining our tasks as K8s Pod Operators

- Defining our workflows as DAGs allow us to easily split out our pipeline into modular tasks without worrying about downstream task dependencies and also enables parallel execution of tasks.

- Web UI allows us to easily visualize, profile and debug our pipelines

Streaming Analytics

Streaming Analytics - Problem Characterization

- We have high velocity data, that is it is created and expired very quickly, that we need to process with extremely low latency
- A “single unit” of processing is not really compute or memory intensive in isolation, we just need to do some processing very quickly and typically aggregate over a very short time interval

Real-Time Sentiment Index

Some users come to the StockTwits to get a sense of how our community feels in aggregate about a stock. One heuristic of this is the StockTwits sentiment index for the ticker. We have a social sentiment model that classifies the sentiment of all incoming messages. Sentiment scores are assigned at the message level so we need to aggregate these sentiment scores for every ticker in real time.

Basic Approach - Pseudo Real Time Mini Batch

- Write a script to pull the last minute of raw sentiment data, aggregate it for each ticker and write back into the database
- Build a docker image that runs the script and push to remote repository
- Deploy the docker images as a CronJob with Kubernetes to run every minute

Basic Approach - Technologies

We are actually just doing batch processing on a really small interval so we know everything we need to know to do this already

Basic Approach - Mini Batch Demo

Basic Approach - Drawbacks

- Not actually doing streaming analytics, we are just doing batch processing jobs really really often
- This won’t scale. With high volume and higher ticker coverage we won’t be able to process all of this data in less than a minute.

Better Approach - AWS Kinesis and AWS Lambda

- Push raw sentiment data into an Kinesis Stream as it is generated
- Read this data stream into an Kinesis Analytics Application that uses a Lambda function to pre-process the raw sentiment score
- Write and run a series of SQL-esque statements in the Kinesis Analytics application to aggregate the processed scores
- Forward the aggregate stream output to S3 with a Kinesis Firehose Delivery Stream

Better Approach - Technologies

- ***AWS Kinesis Stream*** - AWS service to collect and process large streams of data records in real time. Used for rapid and continuous data intake - https://docs.aws.amazon.com/kinesis/index.html
- ***AWS Kinesis Analytics Application*** - AWS service to process and analyze streaming data using standard SQL
- ***AWS Kinesis Firehose*** - AWS service for delivering real-time streaming data to storage destinations

Better Approach - Technologies

- ***AWS Lambda*** - AWS serverless compute service that runs your code in response to events and automatically manages the underlying compute resources for you - https://docs.aws.amazon.com/lambda/index.html

Better Approach - Kinesis Demo

Better Approach - Addressing Drawbacks

- This is actually a streaming application that should scale with our business

Thanks!

Any questions?

You can find me at

- @garrettleeh (Twitter and StockTwits)
- garrett@stocktwits.com

and talk slides and demo code

- https://github.com/GarrettHoffman/ODSC_East_2019_Deploy_DS_Apps