Managing Data for AI

From Development to Production

Nexla

TABLE OF CONTENTS

Introduction 3
Anatomy of an AI application 4
Development Phase 5
- Model Development 5
- Data Discovery 6
- How to make data discoverable? 7
- Data Analysis 7
- Data Preparation 8
- Feature Engineering 8
- Iterative Feature Addition 9

Production Phase 10
- Data Pipelining 10
- Data Validation 10
- Alerting 10
- Model Feedback 11
- Automating the AI Pipeline 11

Nexla & AI 12
References and Additional Reading 13

Managing Data for AI

From Development to Production

Introduction

AI driven applications have exploded and we are transitioning into a world where every application will apply some degree of AI techniques. This is enabled by vast amount of organized and unorganized data, cheap compute and algorithmic breakthroughs such as deep learning.

The promise of AI, of course, depends on the ability to access, harness and operationalize data at scale. While algorithms and models may get all the hype and glamor, nearly 80% of technical effort actually goes into enabling the underlying data. Easy experimentation with data during development, and robust data pipelines in production are a must have. Today, in most organizations, Data Science teams are having to put together pipelines on their own and are spending too much time manually managing data, because data engineering teams are overwhelmed with requests.

This paper describes the role of data in building AI applications.

What readers can expect from this paper:

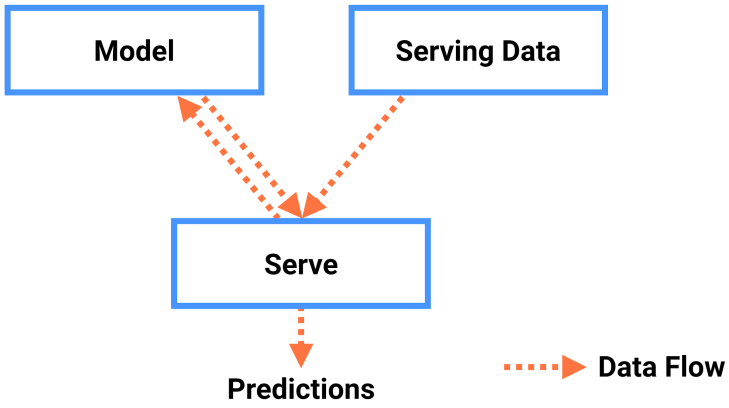
- Technology and business leaders will get a data-centric overview of the AI development process. An understanding of the process and its challenges is essential for making resource allocations
- Data Engineers will get a view into the expectations that AI development puts on data engineering
- Data Scientists and Analysts will see a lot of familiar information, systematically organized and supplemented with information gathered from from across organizations.

Anatomy of an AI application

From an architecture perspective, most AI applications have two distinct sub systems-

1. **Training** - The objective here is to use training data as input to create a model as output

2. **Serving** - The model created in the training phase serves as an input here along with the serving data. The output is a prediction or result that is used in an application.

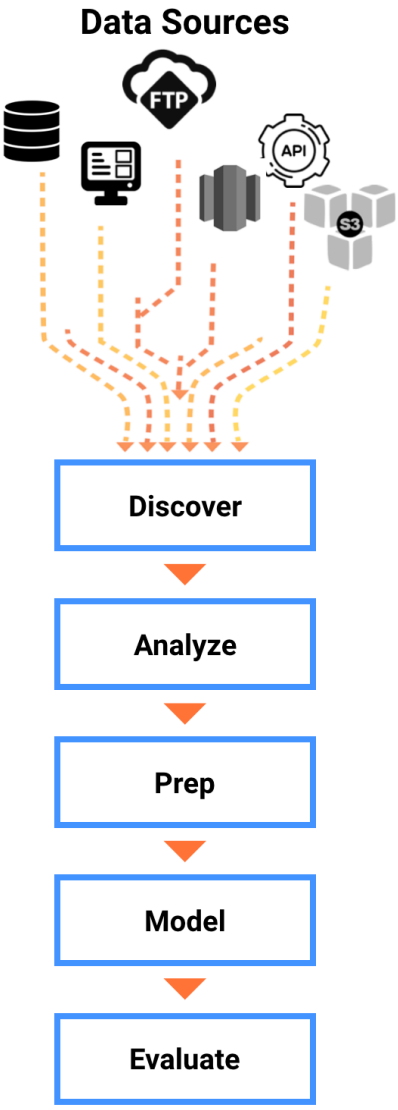


From an implementation perspective, most data scientists think about two phases for an AI project.

1. **Development** - In this phase the data scientist analyzes the problem and identifies the datasets which might help in solving the problem. Data scientists typically spend 60-70% of the time in this phase
2. **Production** - This phase involves setting up production infrastructure/processes for training the model and serving the model.

Let's look at the activities performed as part of these phases.

Development Phase

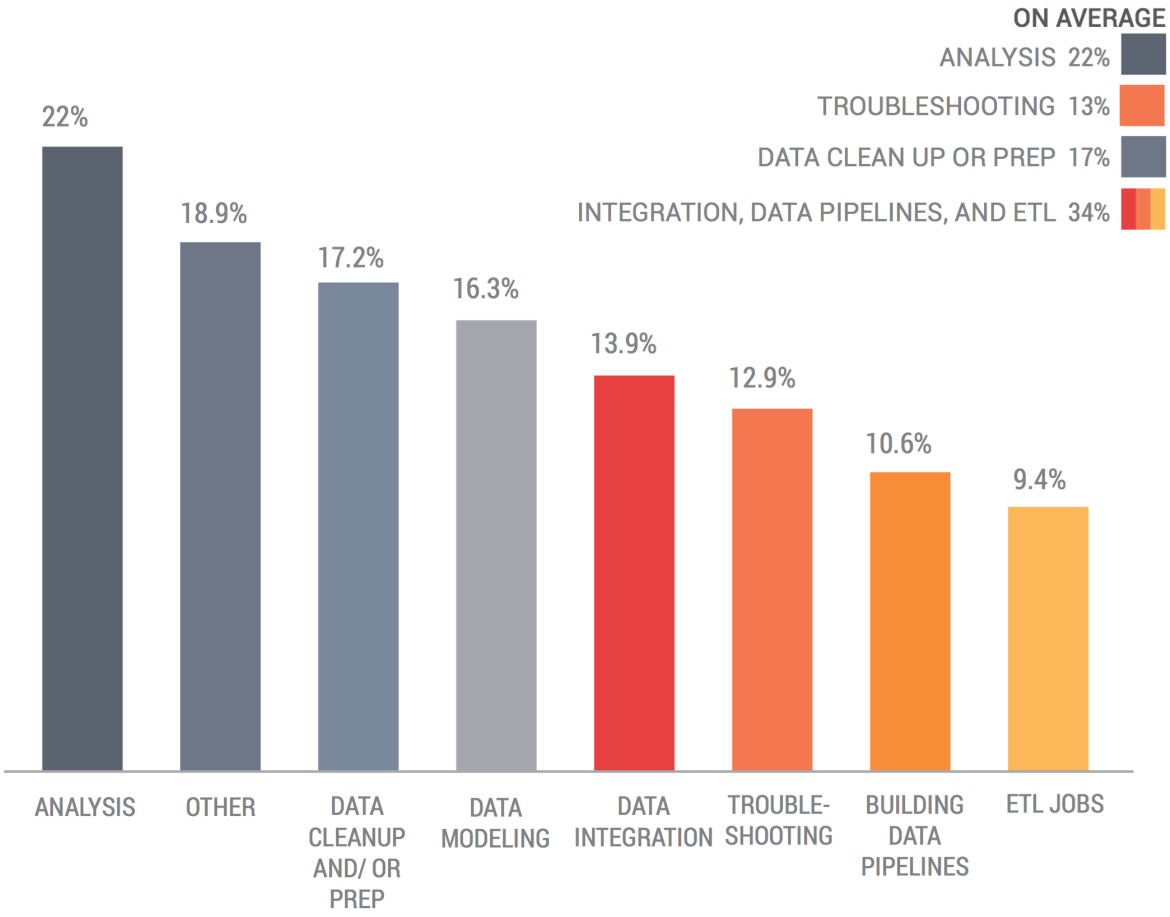


Model Development

Given an AI problem, the data scientist typically starts working backwards. This starts with asking the following questions:

1. What data do I need to solve this problem?
2. How do I find that data?
3. Once I find a dataset, is this the right dataset? How do I validate that?
4. Do I have access to the data? How can I get access to data?
5. Is the data encrypted? Can I read it?
6. If it's the right dataset, is it in the format to be machine learnable? For example if the data is a list of categorical values (such as Low, Medium, and High crime rates in the census data), it is not suitable for machine learning.
7. Does the data contain PII information? Can it be used for modeling? Should it be masked?
8. If not, how do I convert it into the format to be machine learnable
9. Build the model
10. Iterate thru steps 1-9

Surveys have consistently validated that data scientists and ML engineers typically spend 60% to 80% of their time getting the right data to the algorithm to train the model.



Nexla DataOps Report shows distribution of time spent across various data and analytics activities

Data Discovery

In this phase, the data scientist tries to find the datasets they think would help solve the given problem and therefore should be an input into their model. The challenge here is often that of data silos. Enterprises have grown organically and along the way have chosen multiple technologies to solve the problem at hand. This has resulted in fragmented data systems where there is often an absence of formal organization wide data inventory. A CIO at a big bank can track # of employee laptops but he doesn't have an easy way to track how many databases they are running in an organization. Data lakes were touted to be the solution to fragmentation. Get your data to the lake. Enterprises need "Fishing rods" to fish for data in the data lake. In absence of a central data repository or data inventory, the discovery of data ends up being a combination of asking people around and combing through code and log files. There are trends around building a data catalog in an enterprise but catalog is as good as metadata being fed into the catalog.

How to make data discoverable?

Making data discoverable can be a slow journey navigating across many data silos. Large enterprises have millions of datasets and they can't be made discoverable overnight. Based on our experience, here are a few things that work.

1. Identify a data silo
2. Inventory datasets
3. Use a tool to auto discover schemas, metadata
4. Work with business users to review tool output and annotate further
5. Centralize the output into a data catalog
6. Rinse and repeat across the organization.

Data Catalog tools are a great asset here. They capture the point in time data model. Nexla's integration tools supplement that and provide automated detection, evolution, and management of schemas. Users can further add annotations to the schema. Nexla maintains active metadata on datasets and it gets evolved as the underlying data changes. Users also get the ability to search datasets.

Data Analysis

Finding a candidate dataset is a start. The data scientist now needs to ask questions of data to see if this dataset can be used. Typical questions are around statistical properties of the attributes like -

- Is the attribute always present? For example: Do I always have the age of the user"?
- What are the units of the attribute?For example: Is the temperature in Celsius or Fahrenheit
- What is the min, max, median, mean, standard deviation of the attribute?
- Is this a categorical attribute? What is the distribution of categories?
- Are two attributes dependent on each other?

Ideally Data scientists can use SQL as their tool of choice to ask these questions of the data. However SQL may not be an option if data is not available in a database. Depending upon the size of the dataset one of the two routes might be taken. If dataset is small, any spreadsheet tool can be used locally for analysis. If it is a large dataset, some ETL might have to be written to get the data into a warehouse where SQL analysis could be performed. If this dataset ends up being used by multiple data analysts, scientists, it is recommended to productionalize the ETL to avoid code/effort duplication.

Data Preparation

This step is using results from analysis phase and preparing the data to be fed into the model. The step involves feature engineering and adding attributes to the training data.

Feature Engineering

Feature engineering is the process of using domain knowledge of the data to create features that make machine learning algorithms work. As part of feature engineering, a data scientist identifies attributes that could have the most impact on the algorithm. The output from the analysis phase can be very useful to identify features. Feature engineering is fundamental to the application of machine learning and can be both difficult and expensive.

For example, if the problem is to predict the price of house and you are using census dataset.

Important features could be

- Location (lat, lng)
- School District ratings
- Crime Rate
- Age of the house

We probably don't need to consider features like

- # of people in the household
- Age of head of household

Once features have been extracted, features are transformed to be fed into an ML model. Typical transformations include -

- **One hot encoding** - Some ML algorithms don't work well with categorical data. The categorical variable needs to be converted into a binary variable. This is where one hot encoding is helpful. It converts an N categorical variable into N Binary Variables. In the above house example, this might be used to convert a crime rate variable of (Low, Medium, High) into 3 variables as CRIME_LOW, CRIME_MEDIUM and CRIME_HIGH
- **Bucketization/Binning** Bucketization/Binning is a process of converting a continuous value into discrete ranges. This is typically used in following scenarios -
    - A column of continuous numbers having multiple unique values to model effectively. We automatically or manually assign the values to groups, to create a smaller set of discrete ranges.
    - Replace a column of numbers with categorical values that represent specific ranges. For example: Price of a house above can be converted to buckets such as Under 250k, 250k-400k, 400k-500k and so on.
- **Winsorization** Winsorization is the transformation of statistics by limiting extreme values in the statistical data to reduce the effect of possibly spurious outliers.The distribution of many statistics can be heavily influenced by outliers. A typical strategy is to set all outliers to a specified percentile of the data; for example, a 90% winsorization would see all data below the 5th percentile set to the 5th percentile, and data above the 95th percentile set to the 95th percentile.

Extracting features is hard and it is a lot of trial and error. It needs a significant amount of domain knowledge. It is a good idea to invest in tooling which helps to identify features quickly and allow features to be shared in a central repository for other individuals/teams to benefit.

Iterative Feature Addition

You have a model but you realize you need some more features to make this model. This tends to be an iterative process and would need to follow a similar process of discovering a dataset, analyzing it and preparing to extract features. The data engineer will have an additional responsibility to check if this feature is even available at serving time. How can this feature be made available at serving time. How much more latency does it add?

Production Phase

Now our data scientist has a model that has important features, it is working well for training data. Now is the time to take it to production. The first step to taking a model into production involves connecting data pipes.

Data Pipelining

ML projects will typically have two data pipelines. A **training pipeline** which takes training inputs and feeds them into the model. This would involve a ETL/ELT pipeline which would take data from the source, filter/transform it to extract features and feed into the model.

A **serving pipeline** which feeds a real time input to the model and expects a prediction. Serving pipeline is typically more lightweight than the training pipeline. Being, end user facing, the serving pipeline needs much greater reliability.

Enterprise data systems are fragmented and there is a high degree of heterogeneity in the data sources. It is recommended to invest in centralized architecture components to deal with this heterogeneity.

It is also recommended to have a layer of data governance to be able to track the usage of the datasets across the organization.

Data Validation

Garbage In results in Garbage out. Data validation is performed as part of the pipeline to assess if the data is conforming to model's expectations. Typical validations include

- Presence/absence of attributes Example - Is the price of home always present?
- Attributes are within a range - Is age of home in years or suddenly changed to months?
- Attributes have expected values - Did the state field turn into lower case?

Alerting

Data pipelines often fail. Validations help in catching data issues. It is recommended to build a layer to alert on validation failures/ model errors. The alerts should be directed to the right teams. Infrastructure alerts should go to an infra team. Data validation errors should go to an engineer. Following are major categories of alerts -

- Alert on a required field missing
- RPC timeout from a service

- Incorrectly formatted data e.g. for phone number, email etc.
- Data security alerts
- Unexpected or erroneous data values

Model Feedback

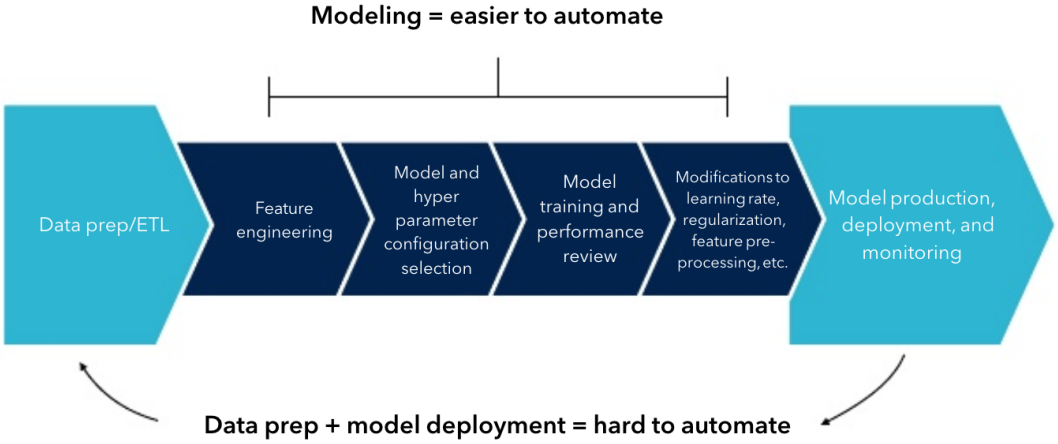
It is essential to build a feedback layer around model predictions and actual results and iterate on model as needed.

In the earlier example of predicting home prices, a good feedback layer might be to compare the actual sale prices of home with the model prediction. If there is a variance higher than the threshold, it should be investigated. The variance can stem from multiple factors -

1. New Data, New Features - May be there was a change in taxation policy which caused price fluctuations in the market. This might need to take new data into account and add new features.
2. Same Data, Same Features - There is probably a micro change in the market causing prices to move in a certain direction. This can be handled by retraining the model more frequently.
3. Same Data, New Features - We already have the data, but we are realizing a new feature from the data might make the model more accurate.

Automating the AI Pipeline

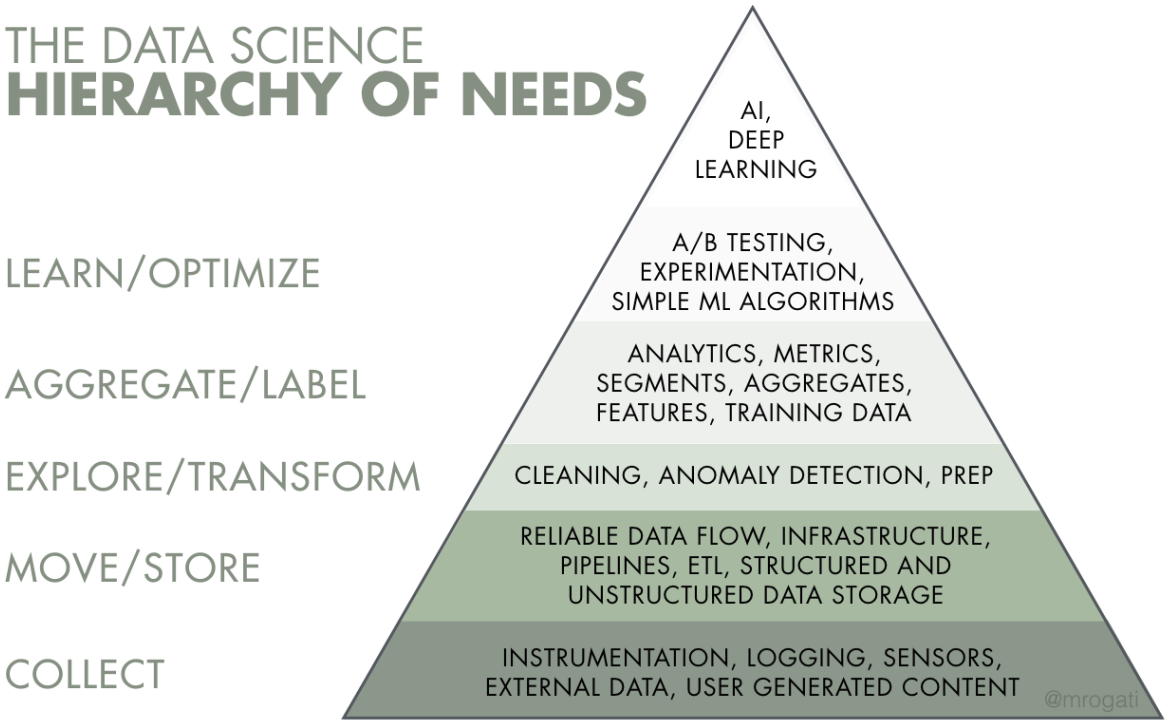
The desire to accelerate AI development has led to many products the domain of AutoML. Many vendors from major cloud providers all the way to niche AI enterprises have AutoML offerings. While feature extraction and model testing lends itself well to automation, the same doesn't hold true for automating DataPrep, ETL, and pipelines for model serving. As further reading, check the References at the end of this document.



Source: 2018 Enterprise Almanac by Workbench

Nexla & AI

Monica Rogati (Former VP of Data @Jawbone & @LinkedIn data scientist) describes AI as the top of a pyramid of needs. Yes, self-actualization (AI) is great, but you first need food, water and shelter (data literacy, collection, and infrastructure).



As shown in this diagram, a solid data infrastructure is needed for success on the AI journey. Consumer AI companies have built a very robust data infrastructure. This infrastructure typically consists of

- Data Acquisition - Tools to collect, acquire data from multiple sources. This could include data from IoT devices, page views, user actions.
- Data Movement - Move data from source to a storage engine (Database, files, etc)
- Data Cleaning - Clean/Prep data
- Data Labeling - Label data for training
- Learning - Apply ML algorithms

- External Signals: The right external signals can make the model even better. Think about - Weather impacting sales or a big event like Superbowl impacting footfall.

About Nexla:
Nexla delivers a data integration platform that enables **executives** to provide a collaborative and secure method of accessing and using internal and external data throughout the enterprise, **engineers** to reduce overhead for creating and operating data pipelines, and **data users** to access, refine, and deliver any internal or external data to fuel data-driven applications and processes.

Nexla takes a unique approach to the need for a modern system that can seamlessly acquire, move, clean and label data. It is built on the principles of

- **Collaborative** workflows that replace a dependency based process. Sharing and reuse of datasets and functions brings scale to large organizations
- **Self Service tools** that empower non-engineers while allowing engineers to focus on complex problems and reusable extensions
- **Automatic and continuous data organization** through observation of both data and metadata.
- **Modern architecture** built with a scalable realtime and streaming data engine.

To learn more of how Nexla helps with building data pipelines and accelerating AI initiatives visit www.nexla.com

References and Additional Reading

1. The Workbench Enterprise Almanac 2018 by Michael Yamnitsky : https://medium.com/work-bench/the-work-bench-enterprise-almanac-2018-edition-6796f3941337
2. The AI hierarchy of needs by Monica Rogati: https://hackernoon.com/the-ai-hierarchy-of-needs-18f111fcc007
3. Hidden technical debt in Machine Learning, by Sculley, Holt, Golovin, Davydov, Phillips https://papers.nips.cc/paper/5656-hidden-technical-debt-in-machine-learning-systems.pdf
4. Machine Learning: High Interest Credit Card of Technical Debt: https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/43146.pdf