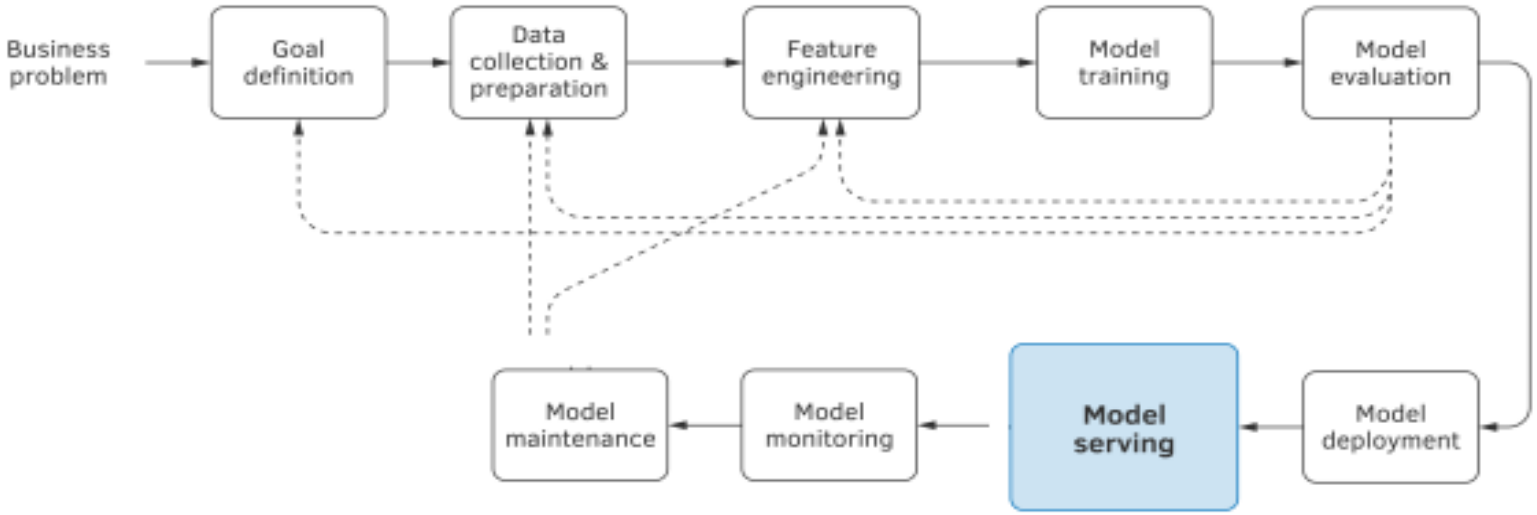
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Essentials of Model Serving

Pravin Y Pawar

Adapted from " Machine Learning Engineering" and "Reliable Machine Learning"

Model Serving in ML Lifecycle



Machine learning project life cycle.

Model Serving

- Model built in training phase, needs to be taken into the world so that it can start predicting!
- Aka Model Serving

- Process of creating a structure to ensure system can ask the model to make predictions on new examples, and return those predictions to the people or systems that need them

Properties of the Model Serving Runtime

- The model serving runtime is the environment in which the model is applied to the input data
  - The runtime properties are dictated by the model deployment pattern

- However, an effective runtime will have several additional properties such as
  - Security and Correctness
  - Ease of Deployment
  - Guarantees of Model Validity
  - Ease of Recovery
  - Avoidance of Training/Serving Skew
  - Avoidance of Hidden Feedback Loops

Properties of the Model Serving Runtime(2)

Security and Correctness

- The runtime is responsible for authenticating the user's identity, and authorizing their requests.

- Things to check are:
  - whether a specific user has authorized access to the models they want to run,
  - whether the names and the values of parameters passed correspond to the model's specification,
  - whether those parameters and their values are currently available to the user.

Properties of the Model Serving Runtime(3)

Ease of Deployment and Recovery

- Ease of Deployment
  - The runtime must allow the model to be updated with minimal effort
    - ideally, without affecting the entire application
  - If the model was deployed as a web service on a physical server,
    - then a model update must be as simple as replacing one model file with another, and restarting the web service
  - If the model was deployed as a virtual machine instance or container,
    - then the instances or containers running the old version of the model should be replaceable by gradually stopping the running instances and starting new instances from a new image
    - The same principle applies to the orchestrated containers

- Ease of Recovery
  - An effective runtime allows easy recovery from errors by rolling back to previous versions
    - The recovery from an unsuccessful deployment should be produced in the same way, and with the same ease, as the deployment of an updated model
    - The only difference is that, instead of the new model, the previous working version will be deployed.

Properties of the Model Serving Runtime(4)

Avoidance of Training/Serving Skew

- When it concerns feature extraction, Strongly recommended to avoid using two different codebases,
  - one for training the model, and one for scoring in production
  - even a tiny difference between two versions of feature extractor code may lead to suboptimal or incorrect model performance

- The engineering team may reimplement the feature extractor code for production for many reasons
  - most common being data analyst's code is inefficient or incompatible with the production ecosystem
- Runtime should allow easy access to the feature extraction code for various needs,
  - including model retraining, ad-hoc model calls, and production.

- One way to implement it is by wrapping the feature extraction object into a separate web service

- If cannot avoid using two different codebases to generate features for training and production,
  - then the runtime should allow for the logging of feature values generated in the production environment
  - Those values should then be used as training values

Key questions for model serving

- Lot of ways to create structures around the model that support serving,
- Each with very different sets of trade-offs

- Useful to think through specific questions about needs of system
  - What will be the load to model?
  - What are the prediction latency needs of model?
  - Where does the model need to live?
  - What are the hardware needs for model?
  - How will the serving model be stored, loaded, versioned and updated?
  - What will feature pipeline for serving look like?

What will be the load to model?

- QPS (Queries Per second) – need to know in serving environment what is the level of traffic that model will be asked to handle – when queries are done on demand
  - Model serving predictions to millions of daily users may need handle thousands of QPS
  - Model runs audio recognizer on mobile device may run at a few QPS
  - Model predicting real estate prices might not be served on demand at all!

- Few basic strategies to handle large traffic loads
  - Replicate model across many machines and run these parallel – may be on cloud
  - Use more powerful hardware – accelerators like GPU or specialized chips
  - Tune computation cost of model itself by using fewer features or layers or parameters
  - Model cascades can be effective at cost reduction

What are the prediction latency needs of model?

- Prediction latency is time between the moment request is made and moment response is received
- Acceptable prediction latency can vary dramatically among applications
  - Is major determiner of serving architecture choices
- Taken together, latency and traffic load define overall computational needs of ML system
- If latency is too high, can be mitigated by
  - Using more powerful hardware
  - Making model less expensive to compute
- But creating a larger number of model replicas is not a solution for latency issue!

Where does the model need to live?

Model needs to be stored on physical device in specific location – home of model

- This choice has significant implications on overall serving architecture

- On a local machine
  - Not a practical solution – may be suitable for small batch predictions
  - Not recommended beyond small-scale prototyping or bespoke uses
- On servers owned or managed by our organization
  - Important when specific privacy or security concerns are in place
  - Right option if latency is hypercritical concern or if speciality hardware is needed to run models
  - Can limit flexibility in terms of scaling up/down, require special attention to monitoring
- In the cloud
  - Can allow easy scaling overall computation footprint up or down
  - Can be done by
    - running model servers on own virtual servers and controlling how many of them to be used
    - managed inference service
- On-device
  - Everything from mobile phones to smart watches, digital assistants, automobiles, printers etc.
  - Has strict constraints on model size, because of limited memory and power

What are the hardware needs for model?

- Range of computational hardware and chip options have emerged
  - Enabled dramatic improvements in serving efficiency for various model types
- Multicore CPUs
  - Suitable for non-deep methods, non deep matrix multiplications
- Hardware accelerators – commonly GPU
  - Choice of serving deep learning models as they involve dense matrix multiplications
  - But has drawbacks
    - Specialized hardware – need to invest or use a cloud service – costly options
    - Not suited for operations not involving large amounts of dense matrix calculations

How will the serving model be stored, loaded, versioned and updated?

Model is physical object – has size and needs space

- Model serving in offline environment be stored on disk and loaded by specific binaries in batch jobs
  - Main requirement is disk space to store model and I/O capacity to load model from disk and RAM needed to load model into memory for use

- Model used in online serving needs to be stored in RAM in dedicated machine
  - For high-throughput services in latency critical settings, copies of these models likes to be stored and served in many replica machine in parallel

- Eventually these models needs to be updated by retraining
- means needs to swap version of model currently used in serving on a given machine with a new version

- Deciding exactly how many versions to be supported and at what capacity is important architectural choice
  - Requires balancing resourcing, system complexity and organizational requirements together

What will feature pipeline for serving look like?

Feature needs to be processed at serving time as well as at training time

- Any feature processing or other data manipulation that is done to data at training time will almost certainly needs to repeated for all examples sent to model at serving time
  - Computational requirements for this may be considerable
- Actual code used to turn incoming examples data to features of model may be different at serving time from the code used for similar tasks at training time
  - Main source of classic training-serving skew and bugs notoriously difficult to detect and debug
- Promise is in form of feature stores – handle both training and serving together in a single package
- Creating feature for model to use at serving time is key source of latency
  - Means serving feature pipeline is far from an afterthought
  - But is indeed often most production-critical part of the entire serving stack

Thank You!

In our next session: