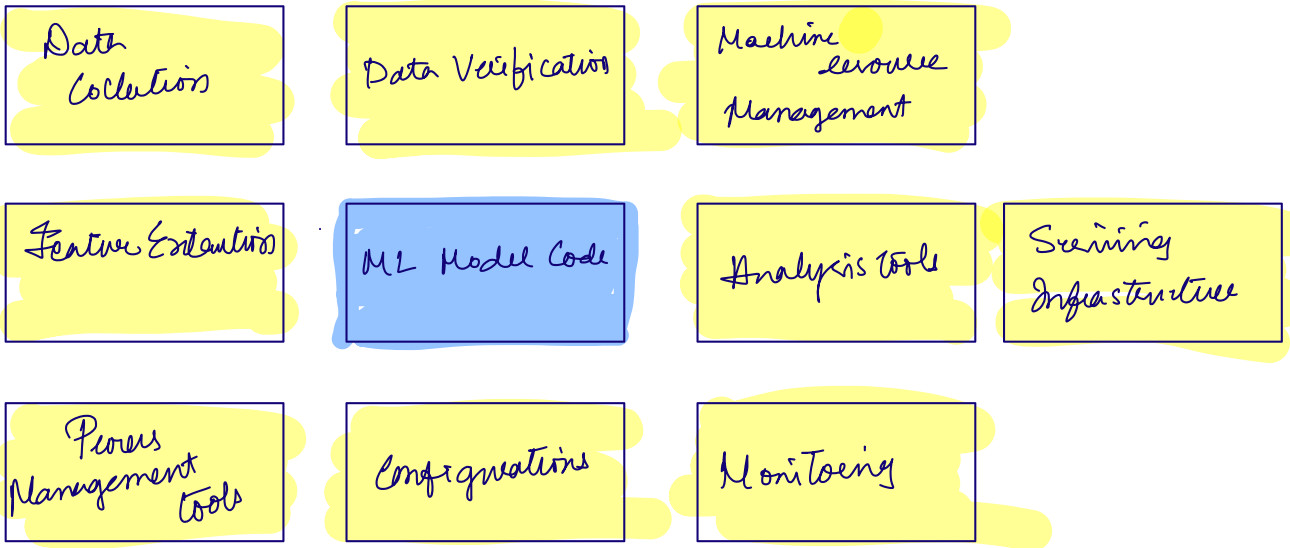
Real World ML

Model is small part of whole large ecosystems.



ML Model is heart of whole Machine learning production system. So overall Model constitutes 5% of whole ecosystem. Major of efforts goes in collecting the data, validating it, and extracting features from it.

Static Vs Dynamic Training

① Static Training → which is also called "offline training" means you train model only once, then serve the model for a while. [pro's : simpler, disadvantage → stale, it degrades and can not upto date]

② Dynamic Training → "Online training" you train the model continuously or atleast frequently, serve the recently trained model. [adv → Mostly upto date, cons → Hard work and resource]

* Data don't change then choose static training
  Cheaper to create & maintain
  data which is tend to constant can be change

hence always be the practice to monitor input values & features.

Static versus Dynamic Inference

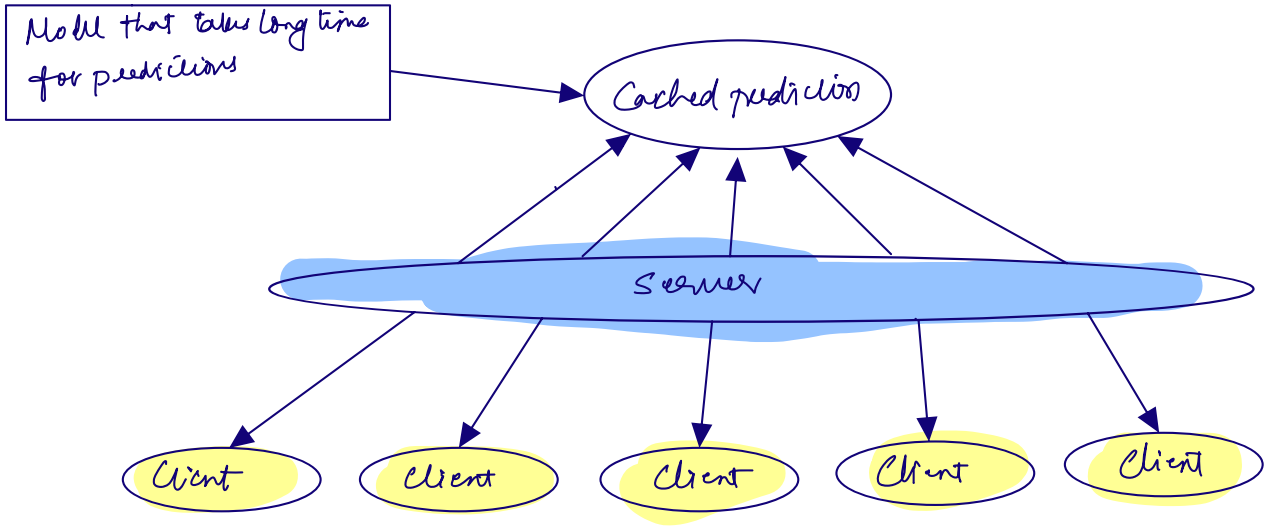
Inference is the process of making predictions by applying trained model on unlabeled examples.

Model can infer predictions in two ways:

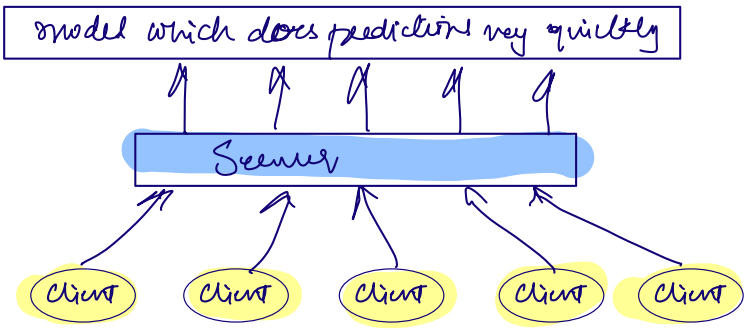
Static Inference is also called as offline inference or batch inference → Means model does the inference on bunch of common unlabeled examples & then caches the predictions somewhere

Dynamic Inference also called as online inference or real-time inference → means Model makes predictions on demand

eg: When client demands the predictions



In static inference, a model generates predictions, which are then cached in on a server



Transforming Data

Raw data must be feature engineered (transformed)

Transforming data can be done at any of the below phases →

When data is feature engineered then this process is called transformed

① Before training the model

② While training the model

Before training the model

⇒ Here you write the code or use specialized tools to transform the raw data

⇒ Store the transformed data on the disk or some where ~~the~~ from which model can pick/ingest

⇒ System transform the data only once.

⇒ System can analyze the entire dataset to determine the best transformation strategy

Disadvantage → Training-Serving Skew.

It is more dangerous when system performs dynamic (online) inference

System use dynamic inference,

→ software transforms the raw dataset

usually differs from software that serves predictions
This causes training-serving skew

In contrast system used for inference can sometimes run the same software.

Transforming data while training

Transformations is part of model code. The model ingests raw data and transforms it

↳ * you can use the same raw data file if you change the transformations
* you are ensured the same transformations at training and at prediction time

↳ Complicated transforms can increase model latency
* transformations occurs for each and every batch

Transforming the data per batch can be tricky, for example
Suppose you want to use Z-score normalization to transform raw numerical data

Z-score normalization requires the mean and standard deviation of the feature.

transformations per batch mean you only have access to one batch only. not full dataset.

So if batches are highly variant, a Z-score of say -2.5 in one batch wont have the same meaning as 2-5 in another batch

Workaround your system can precompute the mean & standard deviation across the entire dataset & then use them as constants in the model.

Deployment Testing

Model which is ready has to deployed

1. Validating input data
2. Validating feature data
3. Validating the quality of new model versions
4. Validating & serving infrastructure
5. Testing integration between pipeline components.

TDD → Test Driven Development

↓

It is difficult in Machine learning

Eg: Before training the model you can write test to validate the loss, Instead you must first discover the achievable loss during the Model development. Then test new model versions against the achievable loss.

Test model updates, with reproducible training

when wanted to improve your model, For example

do some feature engineering and train the model again hoping to get better results (or same results), It is sometimes difficult to reproduce model training

To improve reproducibility below steps to be followed:

=> Deterministically seed the random number generates

=> Initialize model components in a fixed order to ensure the components get the same random from the random number generator on every run. ML libraries typically handle this requirement automatically

=> Take the average of several runs on the model

=> Use version control, even for preliminary iterations so that you can pinpoint code and parameters when investigating your model.

Test calls to machine learning API

write unit test to generate random I/p data & run single step gradient descent.

Write integration tests for pipeline Component.

write integration test, where components work together

Basically when you change in core component cause error in another component.

To run integration test faster, train on a subset of the data or with simpler model.

Validate model quality before serving

Test below two test degradations

① Sudden degradations → A bug in the new version could cause significantly lower quality

② Slow degradations

↓

your test for sudden degradations might not detect slow degradations in model quality

validate new versions by checking their quality against the previous version

Have error Model predictions on a validation dataset meet a fixed threshold. If your validation dataset deviates from quality threshold, update validation dataset.

Validate model–infrastructure compatibility before serving

If your model is updated faster than your server the check software dependencies, majorly causing incompatibilities

Monitoring Pipelines

① Write "data schema" to validate raw data.

↳ to monitor your data, you should continuously check it against expected statistical values by writing rules that the data must satisfy. This collection of rules is called a data schema. Below steps.

① Understand the range and distributions of your feature.

② Encode your understanding into the data schema

→ Ensure that user-submitted ratings are always in range 1 to 5

→ Check that word which occurs more frequently.

→ check each categorical feature is set to a value from fixed set of possible values.

③ Test your data against the data schema. Your schema should catch data errors such as

→ Anomalies

→ Unexpected values of categorical variables

→ Unexpected data distributions

⇒ Write unit Tests to validate feature engineering

Data schema is nothing but certain rules to pass the data - in this scenario it might pass the data schema model cannot used to train the model.

Data which is feature engineered is the one model is trained on.

Eg: Model will be trained on normalized feature values, featured engineered data can be very different from raw input data

Write unit tests based on understand. Eg the conditions such as

① Numeric features are scaled, for example 0 → 1

② One-hot-encoded vectors only, contains single 1 and N-1 zeros.

③ data distributions after Transformation contains a single 1 & N-1 zeros

④ Outliers are handled, such as by scaling or clipping

Check Metrics for important data Slices

A successful model sometimes obscures an unsuccessful subset. In other words, a model with great overall metrics might still make terrible predictions for certain situations. for example

AUC → Area Under the Curve

If engineer satisfied with great AUC then you might not notice the model is more in the sahara desert. If making good predictions for every region is important then track performance for every region.

Subset of data like sahara desert is called data slice

Identify data slices of interest. Then compare model Metrics for these data slices against the metrics of your entire dataset.

Use real World Metrics

Model Metrics don't necessarily measure the real-world impact of your model. For example changing a hyperparameter might increase a model AUC, but how did change affect user experience. To measure real world impact, you need to define separate metrics. For example you could survey

Training Serving - Skew

input data during training differs from your input data in serving.

① Schema Skew

Definition: Training & serving input data do not conform to the same schema.

Eg: The format or distributions of the serving data changes while your model continues to train on old data.

Solution: Use same schema to validate training and serving data

→ Separately check the statistics not checked by schema such as the fraction of missing val

② Feature Skew: Engineered data differs from training & serving

Eg: Feature engineering code differs between training and serving, producing different engineered data.

Solution: Similar to schema skew, apply the same statistical rules across training and serving engineered data.

⇒ Track the number of detected skewed features and ratio of skewed examples per feature.

Causes: Training & Serving Skew can be subtle. Always consider what data is available to your model at prediction time.

Underutilized → when you have model to predict daily revenue based on no of customers as feature

⟹ Problem → you dont have customers at prediction time before the daily sales are complete → Hence This feature is not useful

Check for Label Leaks

Label leaks mean ground truth labels that you are trying to predict have inadvertently entered your training features.

Sometimes very different to detect.

Monitor Model Age

If the serving data evolves with time but your model is retrained regularly. Then you will see decline in quality of model.

Track the time since the model was retrained on new data and set threshold age for alerts

You should monitor the age throughout the pipeline.

⟹ Test the model weights & outputs are numerically stable.
write test to check NaN, inf values.

⟹ Monitor Model Performance

⟹ Track Model performance by versions of code

model & data.

=> Catch Memory leaks.

=> Monitor API response time

=> Number of queries answered per second.

Test the quality of live model on served data

① generate labels using human raters.

② Investigate models that show significant statistical bias in predictions.

③ Track real world metrics for model.

if Comedy Spam, compare with user reported spam.

④ Mitigate potential divergence between training & serving data using a new model.

Randomization

data generation pipeline reproducible

Eg: Suppose you want to add feature to check model quality. your dataset should be identical except for this new feature. In that spirit make sure any randomizations in data generation can be made deterministic.

① Seed your Random Number Generator (RNG's)

Seeding ensure same set of values in same order whenever you run it.

② Use Invariant hash keys

Hashing is common way to split data

Eg:
hash each example & use the resulting integer to decide in which split to place the example
Input to your hash function shouldn't change every time

Considerations for Hashing

imagine you are using search queries & using hashing to include or exclude queries.

⇒ your training set will see a less diverse set of queries :-

⇒ Evaluation time will be artificially hard, because they won't overlap with your training data, so your evaluation should reflect that

Considerations before your data & model in production systems

① Is each feature helpful

⇒ Continue you should monitor your features if they are contributing little or nothing to the model's prediction

ability

⇒ If input data for that feature abruptly changes models behaviour will also change undesirably.

② Does the usefulness of the feature justify the cost of including it. ?

⇒ It is always tempting to add more features to the model,

Eg: You find a feature which result you better predictions, which is better than slightly worst prediction but that add maintainance burden.

③ Is your data source reliable ?.

⇒ Is the signal always going to be available or is it coming from an unreliable source ?

⇒ Is the signal coming from a server that crashes under heavy load.

⇒ If signal coming from human who go to vacation every August.

⇒ Does the system that computes your models input data ever change ? Then how often

⇒ How to know when system changes. !

⇒ Please consider creating your own copy of the data you receive from the upstream process. Then only advance to the next version of the upstream data when you are certain that it is safe.

✱ Is your Model part of a feedback loop?

Model can affect its own training data.

For example

Result of some model can become directly or indirectly feature to that same model

→ Model can affect another model for example, consider two models for predicting stock prices.

Eg:

→ Model A, (bad predicting model)

→ Model B

Here Model A is buggy, it mistakenly decides to buy stock in stock X. Then purchases drive up the price of stock X. Model B uses the price of stock X as an input feature, thus model B can come to false conclusions