

Chapter 6: Model Development and Offline Evaluation

Group 4 (Team Crickets)

Saturday, May 18, 2024



<https://www.youtube.com/watch?v=JrgqA3xL8AY>

CHAPTER 6: Model Development and Offline Evaluation

Designing ML Systems Book Club

Cohort 1, Group 4, “Team
Crickets”
May 17th, 2024



Team Crickets



Suchi

SWE @ Meta,
Head TA, Applied Crypto @
Georgia Tech,
SW/HW @ AMD, Oracle, Intel,
MS CS+ MS ECE @ GT
www.linkedin.com/in/suchithraravi



Sarma

SDE @ Amazon. Prev Lyft,
Tubemogul

https://www.linkedin.com/in/sarma_tangirala/



Charles

MLE @ Verily (Google Life Sciences)
BSc CS and Math at UW Madison

<https://www.linkedin.com/in/foo/>

Chapter Summary

- **FUN PART OF ML:** ML model selection!!!
- How to **select models** for your task - aspects to consider
- Different aspects of Model development, including **ensembling**
- While experimenting with different models and architectures, need to **track and version experiments**
- **Distributed training** necessary as models get bigger - different techniques of parallelism used
- How to **evaluate models** to pick the best one - establishing baseline, evaluation techniques

Key Takeaways

- Models are like code and need to be evaluated in the context of the theory and the business domain.
- Models at scale, like databases, require a different set of skills to train and manage.
- Engineering leadership might need to understand how scaling works, like they typically do for backend systems.
- Model testing includes an online and offline component. Offline eval is not a substitute for online performance.

Part 1: Model Development and Training

Evaluating ML Models

- Given an ML task, what model should you use for it?
 - With unlimited time and energy, you will *eventually* find the optimal solution.
 - There is always a cost associated with developing software.
 - Models solving a business problem have the same constraint. They were needed yesterday.
- ML/AI split into DNN and Classical ML: very different practical considerations
 - Classical ML (generalized linear models, boosted trees etc) still very relevant for many problems.
 - E.g. XGB can solve a lot of problems out of the box and is *fast*.
- Knowledge of common ML tasks and typical approaches essential.



Evaluating ML Models

- Things to consider while evaluating trade-offs:
 - Explainability
 - Performance
 - Resources
- Depending on which model we use, we are looking at differences in:
 - Data collection process and pipeline
 - Training pipeline and time to train
 - Deployment considerations
 - E.g. Logistic regression has lower accuracy than a DNN but,
 - Needs less data
 - Faster to train
 - Is interpretable
 - Easier to deploy.



Six Tips for Model selection

1. Avoid the **state-of-the-art** trap
 - State-of-the-art means it performs better on some *static* datasets
2. Start with the **simplest models**
 - Simpler models easier to deploy, debug and provide a baseline
3. Avoid **human biases** in selecting models
 - When comparing different architectures, compare them under similar setups
4. Evaluate good performance **now** vs. good performance later
 - Account for potential future improvements, validate that improvements are even possible!
5. Evaluate **trade-offs**
 - FP/FN tradeoff, Compute vs Accuracy tradeoff etc.
6. Understand your model's **assumptions**
 - Understand model's assumptions and whether data satisfies them



Six Tips - A comparison

Principle	<i>In Software, this applies to..</i>	<i>In Machine Learning systems, this applies to..</i>
Don't use the latest <> just because Google did it	Don't use the latest framework just because Google did it	Don't use the latest model just because Google published a paper
Establish a baseline: Simplify the problem, start with a simple solution, understand the domain better and set up an iterative process.	Starting with a POC first	Starting with a simple model first
Don't bikeshed a <> from experience or preference	Choosing a framework or language	Choosing a model or architecture

Six Tips - A comparison

Principle	<i>In Software, this applies to..</i>	<i>In Machine Learning systems, this applies to..</i>
Deploy the best now rather than optimizing for later	Deploy the best code now rather than optimizing for later	Deploy the best possible model now rather than optimizing for later.
Understand the trade-offs <ul style="list-style-type: none">- Design for the use-case- Estimate resource consumption	Understand the trade-offs <ul style="list-style-type: none">- Is this feature something that will change later?- How many boxes?	Understand the trade-offs <ul style="list-style-type: none">- Do I care more about FPs or FNs?- Does this need a GPU?
Understand the assumptions well before committing to them	Understand your tools well before you commit to them.	Understand your model well and the assumptions in the theory.

Case Study: GNN vs XGBoost

- Predicting relationships between users in a network : a classic use case for Graph Neural Network Models.
- Original implementation followed the theoretical example- GNN resulting in high precision (which was a priority for this case)
- After about a year, exploratory studies to improve model performance
 - Basic heuristics (or rule-based predictions) are quite useful!
 - XGBoost can cover majority of the remaining cases, but with much fewer resources

Lessons

- Even when the problem is a textbook example use case for model type, *your specific* situation may not need it!
 - Data might be simpler or richer and lend itself to simpler solutions
 - Your metric requirements and resource requirements may be different
- Always revisit your model architecture choices and see if it still makes sense
 - GNN requires retraining as the user graph changes - may not be feasible
- Establishing baselines is invaluable!

Ensembles

- Using multiple models (called *base learners*) to make predictions
 - Pros: Consistently gives performance boost
 - Cons: Less favored in production- more complex to deploy, harder to maintain

Bagging (*Bootstrap Aggregating*)

- E.g. Random Forest
- Sample **with** replacement to create different datasets (independent from each other)
- Pros: Improves training stability and accuracy, reduces variance, helps avoid overfitting.
- Cons: Can degrade the performance of stable methods like kNN

Boosting

Train on same set of samples but weight them differently (weakest samples from previous weighted higher)

E.g. Gradient Boosting Machine like XGBoost

Stacking

Create a meta-learner to combine the outputs of the base learned and make final prediction

Expt Tracking and Versioning

- Important to keep track of all the definitions to re-create experiments and relevant artifacts
 - Allows comparisons and helps understand effect of changes on model performance
 - *Artifact*: a file generated during an experiment
- **Experiment tracking**: Process of tracking progress and results of experiment
 - Big part of training ML model is babysitting the learning process.
 - Tradeoff between tracking everything you can or select things
 - List of things to consider tracking: Loss curve, Model perf metrics, Log of corresponding sample, prediction, ground truth label, Speed of model, System perf metrics, Values over time of any parameter and hyperparameter whose changes can affect model perf
- **Versioning**: Process of logging all details in experiments for the purpose of recreating it later or comparing it with other experiments
 - Need to version code AND data. Code versioning is standard in industry, but data versioning is not.
 - Challenges for data versioning: Data is larger than code, what is a diff for data?, etc.

Debugging ML Models

Main Challenges:

- ML models fail silently
- Can be slow to validate fixes
- High cross-functional complexity


Common reasons for failure:

- Theoretical constraints
- Poor implementation
- Poor hyperparameters
- Bad data
- Poor choice of features


Tried and tested debugging techniques:

- Start simple and gradually add more components
 - Can be a challenge when using off-the-shelf solutions
- Overfit a single batch (should be 100% accurate)
 - If unable to overfit, indicates problems in implementation
- Set a random seed
 - Ensure consistency between different runs, makes it easier to debug

Distributed Training

- As models get bigger, training **at scale** increasingly important
 - Training data doesn't fit in the memory - may need *gradient checkpointing*
 - Data parallelism
 - Split data on multiple machines, train model on all of them and accumulate gradients
 - How to accurately accumulate gradients: synchronous vs asynchronous SGD
 - Asynchronous- gradient staleness may become a problem.
 - Asynchronous requires more steps but acceptable when updates are sparse
 - Batch size can become very big -> account for more learning at each step
 - Main worker may use more resources -> use smaller batch size on main worker
 - Model parallelism
 - Different components of the model trained on different machines e.g. layers in NN
 - Different parts may not be executed in parallel!
 - *Pipeline parallelism* - break the computation on each machine into parts and parallelize micro batches.
 - These two aren't mutually exclusive!
- 

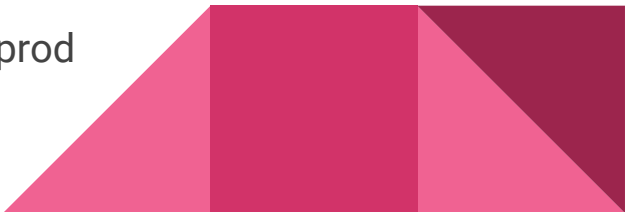
Case Study: Model Baselineing, Memory Tradeoff

- NN to a continuous variable. Fairly large model, required periodic updates, and had good performance. But, EXPENSIVE.
 - XGB model was about as good, baselineing was not done well. Trained with GPU for speed.
 - XGB model works well in prod for a long while. Memory cost then becomes an issue.
 - Plot performance with bin files size to understand the tradeoff.
 - In this case we could pick an order or magnitude smaller model without too much performance loss.
- 

AutoML

- Automating the process of finding ML algorithms to solve real world problems
- Soft AutoML
 - Most popular form is hyperparameter tuning
 - Goal: Find the optimal set of hyperparameters in the search space
 - Popular methods: Random search, Grid Search, Bayesian optimization
 - *Pro tip*: Sensitive hyperparameters must be carefully tuned!
- Hard AutoML
 - What if treat the model components or entire model as a hyperparameter?
 - Known as architectural search or NAS
 - Search Space:
 - Performance Estimation Strategy:
 - Search Strategy:
 - Challenge: Optimizers are sensitive to hyperparameter settings. Solution: Learned Optimizers
 - Can train on the same dataset as the NN
 - Train once on set of existing tasks then generalize

Phases of Model Development

1. **Before ML:** *Start with non-ML solution first!*
 - a. Try heuristics, simple non-ML solutions (ex - if-else statements).
 2. **Simple ML Models:** *Start with a simple algorithm*
 - a. Logistics regression, gradient boosted trees, knn etc.
 3. **Optimizing simple models:** *Optimize objective functions, hyperparams, features*
 - a. Are you confident you've tuned the hyper params?
 4. **Complex model:** *Try a more complex model, repeat steps 2, 3.*
 - a. Experiment to figure out how much your model decays in prod
- 


Part 2: Offline Evaluation

Baselines

- To translate model metrics to usefulness, need to establish a baseline first
 - Without a baseline you can't compare apples to apples.
- You can test your model against another model that:
 - generates outputs “at random”
 - represents simple heuristics for the for the business problem in mind (ex - decision trees, zero rule)
 - is simply human evaluation
 - that is an oracle (out of the box or API providers).



Evaluation Methods

- Perturbation Tests
 - Add noise to the dataset and test the models performance.
 - Invariance Test
 - Feature importance and bias can be detected offline with data analysis. Sometimes you need to collect more data to figure out if exogenous bias exists.
 - Directional expectation tests
 - Differing to invariance test, certain changes to input should cause predictable changes in outputs.
 - Model Calibration
 - The idea is to compute $P(Y=1|\text{model_probs})$.
 - If a predictive model says “there will be 1000 clicks”, do you trust it in prod when it says 900 clicks?
 - Prediction confidence
 - How much certainty do you want to have in a system exposed to a user? 80%, 90%?
 - Slice-based evaluation
 - What do you need to better understand where the models do well? (Simpson’s Paradox)
 - i. Heuristics - The business domain might tell you something about the slices.
 - ii. Error Analysis - Investigate data points where the model is incorrect.
 - iii. Slice Finder - Beam search, clustering
- 



In the age of Gen AI

Evaluation for LLM

Evaluation of LLMs can be categorized into 3 major groups:

- Knowledge and capability evaluation
- Alignment evaluation
 - Why: The training objective of LLMs which performs next word prediction or determine whether two sentence are contextually related are not necessarily in line with human values. LLM may generate
 - Undesirable content: biased, toxic, privacy sensitive, biases content
 - Unfaithful content: fabricated contents, misinformations, hallucinations
 - How:
 - RLHF
 - Proximal Policy Optimization
 - Direct Preference Optimization: <prompt, worse completion, better completion>
 - Kahneman-Tversky Optimization
 - [More readings](#)



OpenAI leader Jan Leike resigns, says safety has "taken a backseat to shiny products"

CBS NEWS
BAY AREA

Updated on: May 17, 2024 / 9:38 AM PDT / AP



A.I., News, Security

OpenAI disbands safety team 'Superalignment' less than a year after forming it



#ICYMI

Microsoft to release upcoming Call of Duty on Xbox Game Pass: WSJ

May 18, 2024
- Sunnydeep Sarkar -
- 3 minute read



Evaluation for LLM

Evaluation of LLMs can be categorized into 3 major groups:

- Safety evaluation
 - Robustness on adversarial prompts

Google extracted ChatGPT's Training Data using a silly trick.

Top highlight

Scalable Extraction of Training Data from (Production) Language Models



Devansh · Follow

Published in DataDrivenInvestor · 13 min read · Jan 7, 2024



3.2K



23



Evaluation for LLM

Evaluation for LLM vs LLM system

- LLM out of the box
 - Performs various tasks such as chatbot, NER, text generation, summarization, Q&A, sentiment analysis, translation.
 - Undergo evaluation on standardized benchmarks such as
 - Massive Multitask Language Understanding (MMLU)
 - General Language Understanding Evaluation (GLUE)
 - SuperGLUE
 - HellaSwag
 - TruthfulQA

Gemini: A Family of Highly Capable Multimodal Models

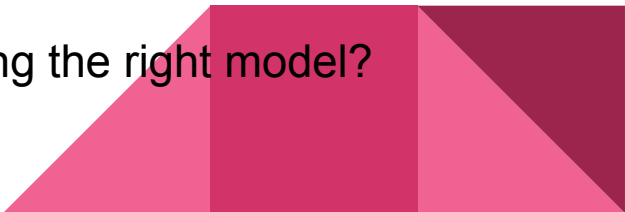
	Gemini Ultra	Gemini Pro	GPT-4	GPT-3.5	PaLM 2-L	Claude 2	Inflection-2	Grok 1	LLAMA-2
MMLU Multiple-choice questions in 57 subjects (professional & academic) (Hendrycks et al., 2021a)	90.04% CoT@32	79.13% CoT@8	87.29% CoT@32 (via API**)	70% 5-shot	78.4% 5-shot	78.5% 5-shot CoT	79.6% 5-shot	73.0% 5-shot	68.0%***
GSM8K Grade-school math (Cobbe et al., 2021)	94.4% Maj1@32	86.5% Maj1@32	92.0% SFT & 5-shot CoT	57.1% 5-shot	80.0% 5-shot	88.0% 0-shot	81.4% 8-shot	62.9% 8-shot	56.8% 5-shot
MATH Math problems across 5 difficulty levels & 7 subdisciplines (Hendrycks et al., 2021b)	53.2% 4-shot	32.6% 4-shot	52.9% 4-shot (via API**)	34.1% 4-shot (via API**)	34.4% 4-shot	—	34.8% 4-shot	23.9% 4-shot	13.5% 4-shot
BIG-Bench-Hard Subset of hard BIG-bench tasks written as CoT problems (Srivastava et al., 2022)	83.6% 3-shot	75.0% 3-shot	83.1% 3-shot (via API**)	66.6% 3-shot (via API**)	77.7% 3-shot	—	—	—	51.2% 3-shot
HumanEval Python coding tasks (Chen et al., 2021)	74.4% 0-shot (IT)	67.7% 0-shot (IT)	67.0% 0-shot (reported)	48.1% 0-shot	—	70.0% 0-shot	44.5% 0-shot	63.2% 0-shot	29.9% 0-shot
Natural2Code Python code generation. (New held-out set with no leakage on web)	74.9% 0-shot	69.6% 0-shot	73.9% 0-shot (via API**)	62.3% 0-shot (via API**)	—	—	—	—	—
DROP Reading comprehension & arithmetic. (metric: F1 score) (Dua et al., 2019)	82.4 Variable shots	74.1 Variable shots	80.9 3-shot (reported)	64.1 3-shot	82.0 Variable shots	—	—	—	—
HellaSwag (validation set) Common-sense multiple choice questions (Zellers et al., 2019)	87.8% 10-shot	84.7% 10-shot	95.3% 10-shot (reported)	85.5% 10-shot	86.8% 10-shot	—	89.0% 10-shot	—	80.0%***
WMT23 Machine translation (metric: BLEURT) (Tom et al., 2023)	74.4 1-shot (IT)	71.7 1-shot	73.8 1-shot (via API**)	—	72.7 1-shot	—	—	—	—

Evaluation for LLM

Evaluation for LLM vs LLM system

- LLM systems
 - LLMs immediate applicability off the shelf are often times constrained for specific requirements.
 - This oftentimes requires the need to fine tune a LLM using domain specific dataset, RAG etc.
 - Research scientists need to ensure that they are utilizing appropriate prompt templates, implement effective data retrieval pipelines, and when fine-tuning properly consider the model architecture.
 - To evaluate the system, we need to compare the performance of the system against ground truth dataset. This quickly becomes tricky
 - Offline evaluation
 - Eyeball
 - Utilize a powerful LLM to generate meaningful synthetic evaluation dataset
 - LLM guided evaluation – using LLM to evaluate LLM
 - Existing frameworks: [OAI Evals](#), [Vertex AI studio](#), [LM evaluation harness](#), [UpTrain](#), [PromptBench](#)

Discussion Questions

1. If you use LLMs, what do you think of HumanEval?
 - a. Do you care about leaderboards?
 - b. If a model is higher up on a leaderboard do you just use it?
 - c. What happens if in your work the model isn't doing too well? What if you're using a hosted API model?
 2. Do we care about AutoML with DNN?
 - a. Throw all the data at the NN? Will it always “just work”.
 3. In your experience experimenting with model architectures, did you ever have to switch from a more complex to a simpler model over the course of a training and deploying a model?
 - a. Are there any lessons/pro tips you have for choosing the right model?
- 



Thank You!