[DeepLearning.AI] Introduction to Machine Learning in Production

Notes by **José Hilario**

AI Systems	Code	Data
Creation	Software engineers	ML engineers
Quality / Infrastructure	DevOps	MLOps

MLOps is an emerging discipline comprising a set of tools and principles to support progress through the ML project lifecycle. It must ensure consistently high quality data: 1) how to define and collect data? 2) how to modify the data to improve the model performance? 3) What data is it needed to track concept and data drift?

The ML Project Lifecycle

Expect a 50/50 division between the work required to achieve the first deployment, and the work required to stabilize the system in production, after the first deployment.

- **Scoping.** Define the project. Decide metrics (acc., latency, throughput), resources and timeline.
- **Data.** Define data and establish a baseline. Label and organize data. Is data labeled consistently? Do we need to add transformations to the data to make it more realistic to our task (e.g., including silenece in audios or background noise)? How should we normalize the data?
- **Modeling.** Select and train model. Perform error analysis in order to understand *how to improve the data and what additional data needs to be collected*. Remark the approach in academia vs production systems: i) in academia, you hold the data fixed and vary the model and hyperparameters in order to improve the result; while, ii) in production, you hold the model and vary the hyperparameters and the data. Feeds back to **Data** step.
- **Deployment.** How to monitor concept drift and data drift? Challenges regarding statistical issues and software engine issues. Is the software real-time or by batches? Does it run on the cloud or in an edge device? Does it meet the performance requirements? Feeds back to **Data** and **Modeling** steps.

Deployment

- **Shadow-mode deployment.** ML system shadows the human (in assist tasks) and runs in parallel. The system's output is not used for any decisions during this phase; the purpose is to gather performance data.
- **Canary deployment.** Roll out to small fraction of traffic, then, monitor the system and ramp up the traffic gradually.
- **Blue-green deployment.** Have a server running with the old/blue version, and another server running with the new/green version; then, the router will suddenly switch to the green version. Easy rollbacks.

Monitoring

- Use dashboards to monitor sever load, fraction of null outputs, fraction of missing values, etc.
- Brainstorm the things that could go wrong; brainstorm a few statistics/metrics that will detect such problems. Experiment with metrics, and removes the ones found not to be useful.
- Software metrics include memory, compute, latency, throughput, server load.
- Input metrics include avg. input length, avg. input volume, number of missing values, avg. brightness of images, etc.
- Output metrics include fraction of null predictions, fraction of users redoing search, fraction of users which give up trying to use a speech recognition system by switching to typing instead, etc.
- ML model and data $\longrightarrow$ Experiment $\longrightarrow$ Error analysis $\longrightarrow$ REPEAT.
- Deployment and monitoring $\longrightarrow$ Traffic $\longrightarrow$ Performance analysis $\longrightarrow$ REPEAT.
- Set thresholds on monitored metrics for alarms.
- Adapt metrics and thresholds over time.
- Monitor pipeline. If a component within it changes, this *may* affect other components and degrade performance. It is recommended to detect this by means of the metrics being monitored. Monitor and analyze several components of the pipeline at the same time.
- Change rate of data. User data tends to have a slower drift (with exceptions such as when pandemics occur), while enterprise data (B2B applications) can shift fast.

Data

- **Data drift.** A change in the data distribution (e.g., a change in user demographics). The model underperforms on unknown data regions.
- **Concept drift.** A change occurs in the existing relationships. The world has changed and the model needs to be updated. This type of drift can be gradual (this is expected), sudden, or seasonal (recurring).
- **Training-serving skew.** Often mixed with data drift. This problem often happens when training a model on an artificially constructed (e.g. data augmentation) or cleaned datasets; this data does not necessarily represent the real world, or does so incompletely.
- Approaches to deal with drift include: 1) retraining using all available data; 2) use all data but assign higher weights to new data; 3) if enough data is collected, we can simply ignore the old data. Note that naive retraining is not always enough and we may need to modify the model as well.
- Recurring drift should be included in the learning phase so that the model learns to react internally to it, as these are expected occurrences that can be taken into account by using the appropriate data.
- Clean vs noisy data. If you have 500 examples and 12% of them are noisy (incorrectly or inconsistently labeled), then the following are about equally effective: 1) clean up the noise, or 2) collect another 500 new examples (ie. double the training set).
- With a data-centric view, there is significant room for improvement with small datasets.

Model-centric	Data-centric
Collect whatever data is possible, and develop a model good enough to deal with noise in the data.	The consistency of the data is paramount. Use tools to improve the data quality; this will allow multiple models to do well.
Hold the data fixed and iteratively improve the model.	Hold the code fixed and iteratively improve the data.

- Iteratively improving the data in a systematic way.
  1. Train a model.
  2. Error analysis to identify the types of data for which the algorithm underperforms.
  3. Either get more data of that kind via data augmentation, data generation or data collection, or give more consistent definitions for labels if they were found to be ambiguous.
- **From big data to good data.** MLOps most important task if to ensure consistently high-quality data in all phases of the ML project lifecycle. Good data is:
  1. Defined consistently (ie. the definition of the labels is unambiguous).
  2. Having good input coverage, especially of important edge cases.
  3. Having timely feedback from production data (distribution covers data drift and concept drift).
  4. Sized appropriately.
- How to improve labeling consistency?
  1. Have multiple labelers label the same example. Use consensus labeling.
  2. If x does not contain enough information according to the labelers, consider changing it.
  3. Refine instructions iteratively. Use all labelers (or a group) for discussions. Include subject-matter expert (SME) if needed.
  4. Have a class/label to capture inconsistencies (eg. unintelligible audio).

- Do not increase data by more than 10x at a time during one iteration.
- Who is qualified for labeling? Depends on the kind of problem: i) speech recognition $\longrightarrow$ any fluent speaker; ii) medical image diagnosis $\longrightarrow$ SME.
- Get into the iteration for development as quickly as possible. Ask: How much data can we obtain in k days? (Unless it is a requirement to obtain M examples instead).

Model Development

- Milestones.
  1. Doing well on training set.
  2. Doing well on validation and test sets.
  3. Doing well on business metrics and project goals. Note that, even if the test error is low, the model may still be insufficient if it doesn't meet other requirements such as fairness, or working well on slices of data which have been deemed essential from the business perspective.
- **Baseline.** Establish a baseline; only afterwards focus on improving the model. Ways of establishing baselines include: 1) measuring the human-level performance (HLP); 2) using state-of-the-art as reference; 3) implementing state-of-the-art models; 4) referring to the performance of an older system.
- Measuring the HLP is usually a good way of establishing a baseline when dealing with unstructured data.
- Getting started on model development.
  1. Literature search to see what is possible.
  2. Find open-source implementation if available.
  3. A reasonable algorithm with good data will often outperform a great algorithm with lousy data.
  4. Consider deployment constraints when selecting a model after baseline is established.
  5. Perform sanity-checks on the code and the algorithms.

Error Analysis

- Iterative process. Examine/tag examples $\longleftrightarrow$ Propose tags.
- Visual inspection. Specific class labels (scratch, dent, etc.)
- Image properties (blurry, dark background, light background, reflection, etc.)
- Other metadata (phone model, factory, etc.)
- Useful metrics for each tag: 1) What fraction of errors does the tag have? An error of 12% would mean that the tag has a potential 12% of improvement to contribute to the overall task; 2) Of all the data with tags, what fraction is misclassified to each? 3) What room of improvement is there for a particular tag?
- Prioritize which slices of data to work on.

Type	Acc.	HLP	Gap	%Data	Potential
Clean speech	94%	95%	1%	60%	0.60%
Car noise	89%	93%	4%	4%	0.16%
People noise	87%	89%	2%	30%	0.60%
Low band-width	70%	70%	0%	6%	0.00%

- Consider **how easy** and **how important** it is to improve the accuracy on a given category tag.
- For the category tags chosen to be prioritized (according to the analysis):
  1. Collect more data.
  2. Use data augmentation to simulate this kind of data.
  3. Improve label accuracy and data quality.

Performance Auditing

1. Brainstorm the ways in which the system could fail.
   - Performance on subsets of data (eg. ethnicity, gender, etc.)
   - How common are certain errors (eg. FP, FN).
2. Establish metrics to assess performance against these issues on appropriate slices of data.
3. Get the opinion about priority of problems to tackle from the product owner(s).

Data-centric Development

- Can adding data hurt?
  1. Training set may end up coming from a very different dist. than the target.
  2. If model is large enough and has low bias, it will **not** have to compromise on previously OK examples in order to improve on new data (eg. data from data augmentation).
  3. If examples are ambiguous (eg. I vs 1 vs l) then it will hurt performance.
  4. When doing data augmentation consider: i) is it a realistic data point? 2) is the mapping $X \longrightarrow Y$ clear? 3) Is the model currently doing poorly on it?
- Data iteration on structured data results in adding new features to the data in order to improve the performance in the following cycle. The error analysis helps determine which features to add. Designing features for structured data is still necessary, even with deep learning.
- For unstructured problems it is rarely justified to design features.

Experiment Tracking

- What to track? Algorithms used, code verson, dataset used, hyperparameters, results.
- Tracking tools: text files, spreadsheets, experiment tracking systems.
- Desirable features: 1) Contain the necessary information in order to replicate the results; 2) Experiment results, ideally with summary metrics and analysis; 3) Resource monitoring, visualization, model error analysis.

Human-level Performance

- Estimates the Bayes error to help with error analysis and prioritization.
- When the ground truth label is itself estimated by a human, we have to adjust our thinking about HLP.
- HLP is used in academia as a benchmark to beat in support of a publication.
- HLP helps establish a reasonable target to the product owner.
- Caution. Using HLP to prove the ML system is better can be misleading, as label instructions that are inconsistent could be the real problem in HLP, specially when the ground truth is also established by human labelers.
- For applications, it is better to focus on raising HLP (since this will help the learning algorithm) rather than focusing on beating the HLP. Label consistency could help improve the HLP.
- When the ground truth is objectively defined (rather than subjectively), the HLP gives the estimate for the Bayes error.

Data Pipeline

- POC phase.
  1. The goal is to decide if the app. is workable and worth deploying.
  2. Focus on getting the prototype to work.
  3. It is OK if the data preprocessing is manual, but extensive notes and comments must be taken.
- Production phase.
  1. Make sure the data pipeline is replicable.
  2. Keep track of data provenance (where the data comes from) and lineage (sequence of steps needed to get to the end of the pipeline).
  3. Keep meta-data. It can be instrumental in generating the key insights to help move the project forward. Useful for error analysis and data provenance.

Scoping

- What project should we work on?
- What are the metrics for success?
- What are the resources (data, time, prople) needed?
- Process:
  1. Brainstorm business problems.
  2. Brainstorm AI solutions.
  3. Assess the feasibility and value of the potential solutions (diligence).
  4. Determine milestones.
  5. Budget for resources.