VERSIONING, PROVENANCE, AND REPRODUCABILITY

Christian Kaestner

LEARNING GOALS

- Judge the importance of data provenance, reproducibility and explainability for a given system
- Create documentation for data dependencies and provenance in a given system
- Propose versioning strategies for data and models
- Design and test systems for reproducibility
CASE STUDY: CREDIT SCORING
The @AppleCard is such a fucking sexist program. My wife and I filed joint tax returns, live in a community-property state, and have been married for a long time. Yet Apple’s black box algorithm thinks I deserve 20x the credit limit she does. No appeals work.
I wasn’t even pessimistic to expect this outcome, but here we are: @AppleCard just gave my wife the VIP bump to match my credit limit, but continued to be an utter fucking failure of a customer service experience. Let me explain...

She spoke to two Apple reps. Both very nice, courteous people representing an utterly broken and reprehensible system. The first person was like “I don’t know why, but I swear we’re not discriminating, IT’S JUST THE ALGORITHM”. I shit you not. “IT’S JUST THE ALGORITHM!”.
DEBUGGING?

What went wrong? Where? How to fix?
DEBUGGING QUESTIONS BEYOND INTERPRETABILITY

- Can we reproduce the problem?
- What were the inputs to the model?
- Which exact model version was used?
- What data was the model trained with?
- What learning code (cleaning, feature extraction, ML algorithm) was the model trained with?
- Where does the data come from? How was it processed and extracted?
- Were other models involved? Which version? Based on which data?
- What parts of the input are responsible for the (wrong) answer? How can we fix the model?
DATA PROVENANCE

Historical record of data and its origin
DATA PROVENANCE

• Track origin of all data
  ▪ Collected where?
  ▪ Modified by whom, when, why?
  ▪ Extracted from what other data or model or algorithm?
• ML models often based on data driven from many sources through many steps, including other models
TRACKING DATA

- Document all data sources
- Model dependencies and flows
- Ideally model all data and processing code
- Avoid "visibility debt"

- Advanced: Use infrastructure to automatically capture/infer dependencies and flows (e.g., Goods paper)
FEATURE PROVENANCE

- How are features extracted from raw data
  - during training
  - during inference
- Has feature extraction changed since the model was trained?

Example?
MODEL PROVENANCE

- How was the model trained?
- Ensemble of multiple models?
Example adapted from Jon Peck. *Chaining machine learning models in production with Algorithmia*. Algorithmia blog, 2019
SUMMARY: PROVENANCE

- Data provenance
- Feature provenance
- Model provenance
PRACTICAL DATA AND MODEL VERSIONING
HOW TO VERSION LARGE DATASETS?
RECALL: EVENT SOURCING

- Append only databases
- Record edit events, never mutate data
- Compute current state from all past events, can reconstruct old state
- For efficiency, take state snapshots
- Similar to traditional database logs

```python
createUser(id=5, name="Christian", dpt="SCS")
updateUser(id=5, dpt="ISR")
deleteUser(id=5)
```
VERSIONING DATASETS

- Store copies of entire datasets (like Git)
- Store deltas between datasets (like Mercurial)
- Offsets in append-only database (like Kafka offset)
- History of individual database records (e.g. S3 bucket versions)
  - some databases specifically track provenance (who has changed what entry when and how)
  - specialized data science tools eg Hangar for tensor data
- Version pipeline to recreate derived datasets ("views", different formats)
  - e.g. version data before or after cleaning?

- Often in cloud storage, distributed
- Checksums often used to uniquely identify versions
- Version also metadata
VERSIONING MODELS

- Usually no meaningful delta, versioning as binary objects
- Any system to track versions of blobs
VERSIONING PIPELINES

data

pipeline

hyperparameters

model
VERSIONING DEPENDENCIES

- Pipelines depend on many frameworks and libraries
- Ensure reproducible builds
  - Declare versioned dependencies from stable repository (e.g. requirements.txt + pip)
    - Optionally: commit all dependencies to repository ("vendoring")
- Optionally: Version entire environment (e.g. Docker container)
- Avoid floating versions
- Test build/pipeline on independent machine (container, CI server, ...)


ML VERSIONING TOOLS (SEE MLOPS)

- Tracking data, pipeline, and model versions
- Modeling pipelines: inputs and outputs and their versions
  - explicitly tracks how data is used and transformed
- Often tracking also metadata about versions
  - Accuracy
  - Training time
  - ...

...
EXAMPLE: DVC

dvc add images
dvc run -d images -o model.p cnn.py
dvc remote add myrepo s3://mybucket
dvc push

- Tracks models and datasets, built on Git
- Splits learning into steps, incrementalization
- Orchestrates learning in cloud resources

https://dvc.org/
EXAMPLE: MODELDB

https://github.com/mitdbg/modeldb
EXAMPLE: MLFLOW

- Instrument pipeline with *logging* statements
- Track individual runs, hyperparameters used, evaluation results, and model files
ASIDE: VERSIONING IN NOTEBOOKS WITH VERDANT

- Data scientists usually do not version notebooks frequently
- Exploratory workflow, copy paste, regular cleaning

FROM MODEL VERSIONING TO DEPLOYMENT

- Decide which model version to run where
  - automated deployment and rollback (cf. canary releases)
  - Kubernetes, Cortex, BentoML, ...
- Track which prediction has been performed with which model version (logging)
LOGGING AND AUDIT TRACES

- Version everything
- Record every model evaluation with model version
- Append only, backed up

Key goal: If a customer complains about an interaction, can we reproduce the prediction with the right model? Can we debug the model's pipeline and data? Can we reproduce the model?
Ensure all predictions are logged
DISCUSSION

What to do in movie recommendation and popularity prediction scenarios? And how?
See also Hulten. Building Intelligent Systems. Chapter 21
ORCHESTRATING MULTIPLE MODELS

- Try different modeling approaches in parallel
- Pick one, voting, sequencing, metamodel, or responding with worst-case prediction

![Diagram]

- Try different modeling approaches in parallel
- Pick one, voting, sequencing, metamodel, or responding with worst-case prediction
CHASING BUGS

- Update, clean, add, remove data
- Change modeling parameters
- Add regression tests
- Fixing one problem may lead to others, recognizable only later
PARTITIONING CONTEXTS

- Separate models for different subpopulations
- Potentially used to address fairness issues
- ML approaches typically partition internally already
OVERRIDES

- Hardcoded heuristics (usually created and maintained by humans) for special cases
- Blocklists, guardrails
- Potential neverending attempt to fix special cases
REPRODUCABILITY
DEFINITIONS

- **Reproducibility**: the ability of an experiment to be repeated with minor differences from the original experiment, while achieving the same qualitative result.

- **Replicability**: ability to reproduce results exactly, achieving the same quantitative result; requires determinism.

In science, reproducing results under different conditions are valuable to gain confidence:

- "conceptual replication": evaluate same hypothesis with different experimental procedure or population.
- many different forms distinguished "... replication" (e.g. close, direct, exact, independent, literal, nonexperiential, partial, retest, sequential, statistical, varied, virtual).

PRACTICAL REPRODUCABILITY

- Ability to generate the same research results or predictions
- Recreate model from data
- Requires versioning of data and pipeline (incl. hyperparameters and dependencies)
Some machine learning algorithms are nondeterministic
  - Recall: Neural networks initialized with random weights
  - Recall: Distributed learning
Many notebooks and pipelines contain nondeterminism
  - Depend on snapshot of online data (e.g., stream)
  - Depend on current time
  - Initialize random seed
Different library versions installed on the machine may affect results
(Inference for a given model is usually deterministic)
RECOMMENDATIONS FOR REPRODUCIBILITY

• Version pipeline and data (see above)
• Document each step
  ▪ document intention and assumptions of the process (not just results)
  ▪ e.g., document why data is cleaned a certain way
  ▪ e.g., document why certain parameters chosen
• Ensure determinism of pipeline steps (→ test)
• Modularize and test the pipeline
• Containerize infrastructure -- see MLOps
SUMMARY

- Provenance is important for debugging and accountability
- Data provenance, feature provenance, model provenance
- Reproducability vs replicability
- Version everything
  - Strategies for data versioning at scale
  - Version the entire pipeline and dependencies
  - Adopt a pipeline view, modularize, automate
  - Containers and MLOps, many tools
- Strategies to fix models