

# VERSIONING, PROVENANCE, AND REPRODUCABILITY

Christian Kaestner

Required reading: □ Halevy, Alon, Flip Korn, Natalya F. Noy, Christopher Olston, Neoklis Polyzotis, Sudip Roy, and Steven Euijong Whang. [Goods: Organizing google's datasets](#). In Proceedings of the 2016 International Conference

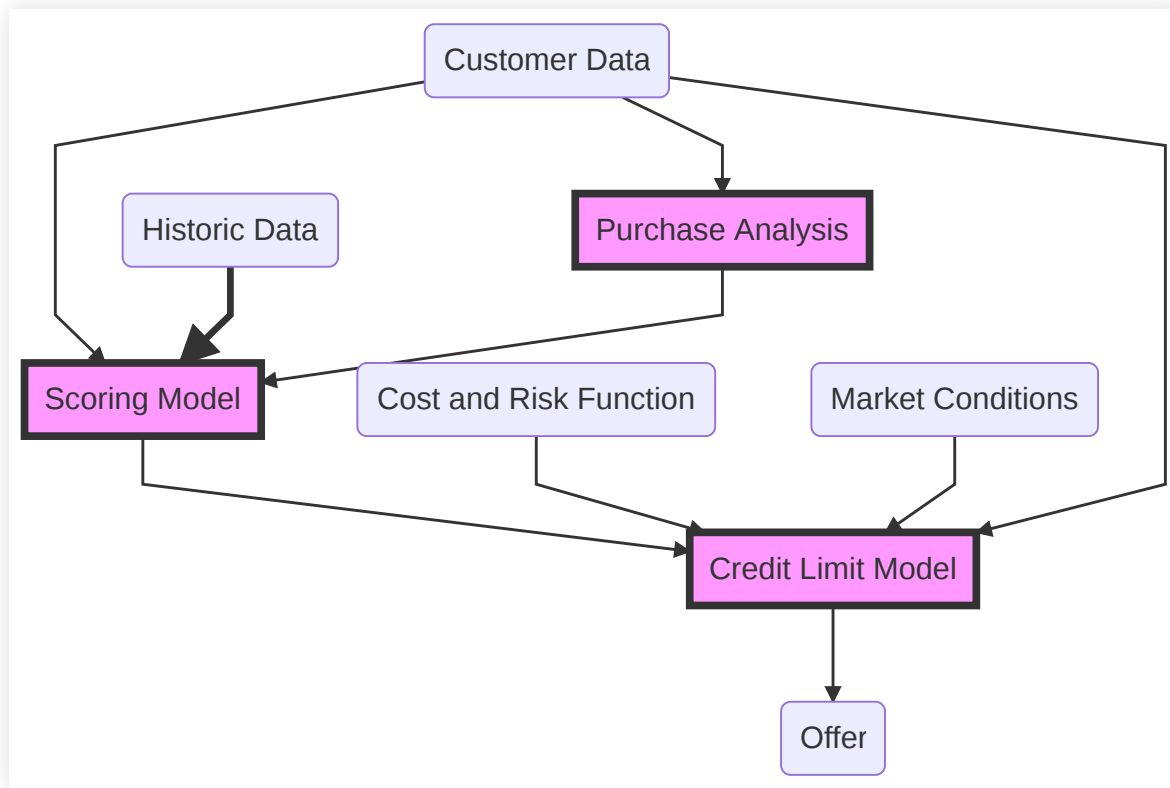
# LEARNING GOALS

- 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**

*Tweet*

*Tweet*



# 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?



# BREAKOUT DISCUSSION: MOVIE PREDICTIONS

*Assume you are receiving complains that a child gets mostly recommendations about R-rated movies*

- Could you identify the problematic recommendation(s)?
- Could you identify the model that caused the prediction?
- Could you identify the training code and data that learned the model?
- Could you identify what training data or infrastructure code "caused" the recommendations?

K.G Orphanides. [Children's YouTube is still churning out blood, suicide and cannibalism](#). Wired UK, 2018

Kristie Bertucci. [16 NSFW Movies Streaming on Netflix](#). Gadget Reviews, 2020

# PROVENANCE TRACKING

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



# EXCURSION: PROVENANCE TRACKING IN DATABASES

- Whenever value is changed, record:
  - who changed it
  - time of change
  - history of previous values
  - possibly also justification of why
- Embedded as feature in some databases, can also be added in business logic
- Immutable data storage keeps history

# TRACKING DATA LINEAGE

- 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)



(CC BY-SA 4.0, [Skamisetty](#))

# FEATURE PROVENANCE

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

**Example?**

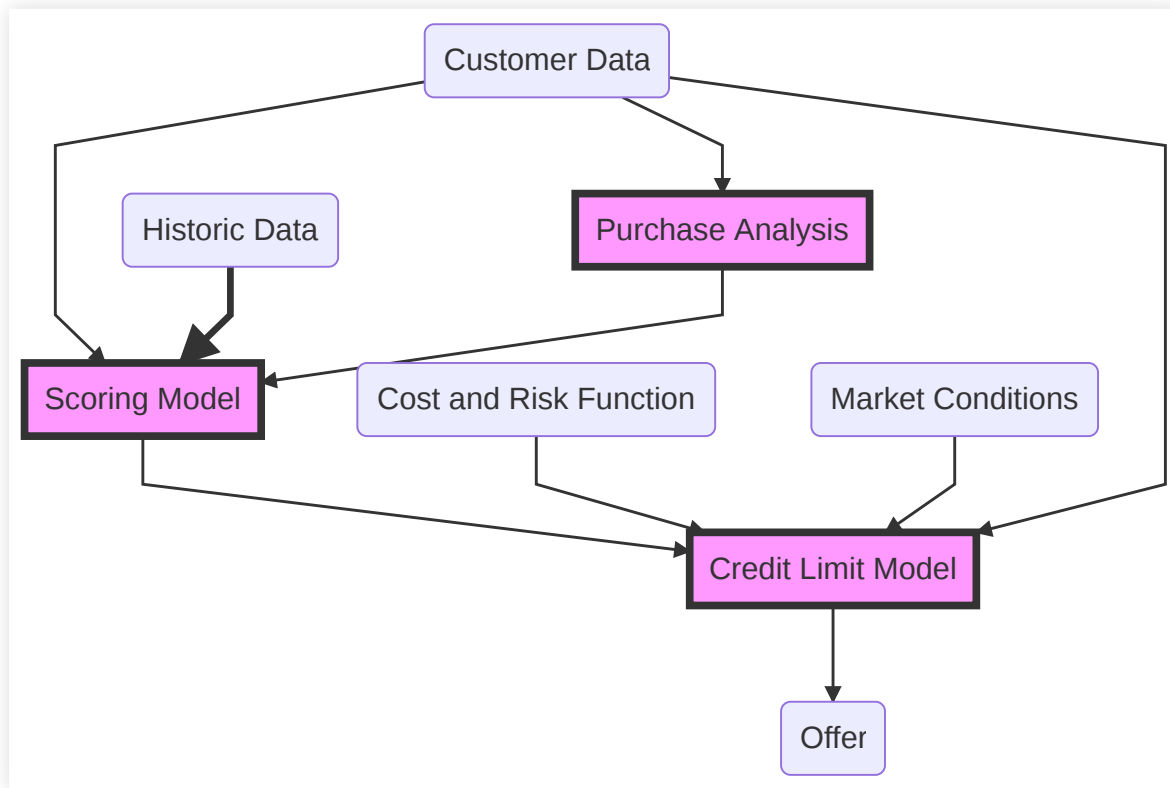


# GOOD PRACTICE: FEATURE STORE

- Encapsulate feature extraction as functions
- Store centrally for reuse
- Use version control
- Use same feature code in training and inference code
- Advanced: Immutable features -- never change existing features, just add new ones (e.g., creditscore, creditscore2, creditscore3)

# MODEL PROVENANCE

- How was the model trained?
- What data? What library? What hyperparameter? What code?
- Ensemble of multiple models?



# RECALL: MODEL CHAINING

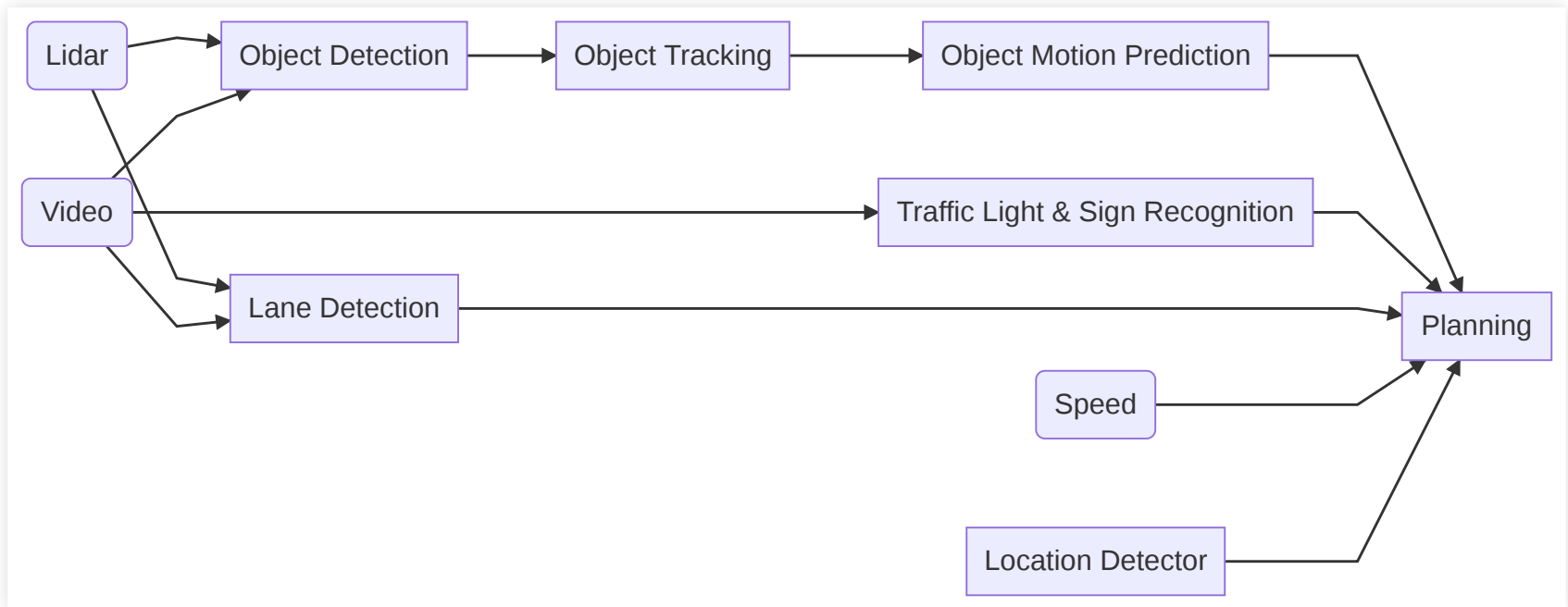
*automatic meme generator*



Example adapted from Jon Peck. [Chaining machine learning models in production with Algorithmia](#). Algorithmia blog, 2019

# RECALL: ML MODELS FOR FEATURE EXTRACTION

*self driving car*



Example: Zong, W., Zhang, C., Wang, Z., Zhu, J., & Chen, Q. (2018). [Architecture design and implementation of an autonomous vehicle](#). IEEE access, 6, 21956-21970.

# SUMMARY: PROVENANCE

- Data provenance
- Feature provenance
- Model provenance

# PRACTICAL DATA AND MODEL VERSIONING

# HOW TO VERSION LARGE DATASETS?



(movie ratings, movie metadata, user data?)



# 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

```
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

# ASIDE: GIT INTERNALS



Scott Chacon and Ben Straub. [Pro Git](#). 2014

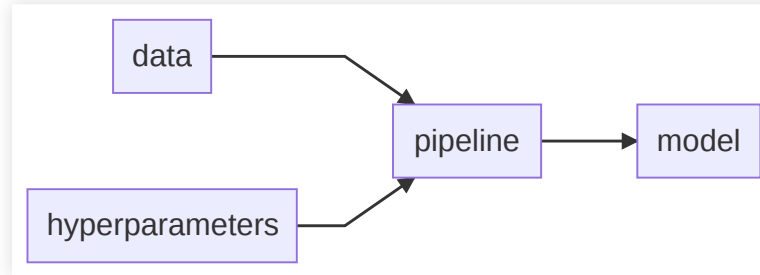
# VERSIONING MODELS



# VERSIONING MODELS

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

# VERSIONING PIPELINES



# 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/>

# DVC EXAMPLE

```
stages:
  features:
    cmd: jupyter nbconvert --execute featurize.ipynb
    deps:
      - data/clean
    params:
      - levels.no
    outs:
      - features
    metrics:
      - performance.json
  training:
    desc: Train model with Python
    cmd:
      - pip install -r requirements.txt
      - python train.py --out ${model_file}
```

# MLFLOW, MODELDB, NEPTUNE, TENSORBOARD, WEIGHTS & BIASES, COMET.ML

- Instrument pipeline with *logging* statements
- Track individual runs, hyperparameters used, evaluation results, and model files

## Listing Price Prediction

Experiment ID: 0


Artifact Location: /Users/matei/mlflow/demo/mlruns/0

Search Runs:

Filter Params:

Filter Metrics:

4 matching runs

					Parameters		Metrics		
	Time	User	Source	Version	alpha	l1_ratio	MAE	R2	RMSE
<input type="checkbox"/>	17:37	matei	linear.py	3a1995	0.5	0.2	84.27	0.277	158.1
<input type="checkbox"/>	17:37	matei	linear.py	3a1995	0.2	0.5	84.08	0.264	159.6
<input type="checkbox"/>	17:37	matei	linear.py	3a1995	0.5	0.5	84.12	0.272	158.6
<input type="checkbox"/>	17:37	matei	linear.py	3a1995	0	0	84.49	0.249	161.2

Matei Zaharia. [Introducing MLflow: an Open Source Machine Learning Platform](#), 2018

# MODELDB EXAMPLE

```
from verta import Client
client = Client("http://localhost:3000")

proj = client.set_project("My first ModelDB project")
expt = client.set_experiment("Default Experiment")

# log the first run
run = client.set_experiment_run("First Run")
run.log_hyperparameters({"regularization" : 0.5})
run.log_dataset_version("training_and_testing_data", dataset_ver
model1 = # ... model training code goes here
run.log_metric('accuracy', accuracy(model1, validationData))
run.log_model(model1)

# log the second run
run = client.set_experiment_run("Second Run")
```

# GOOGLE'S GOODS

- Automatically derive data dependencies from system log files
- Track metadata for each table
- No manual tracking/dependency declarations needed
- Requires homogeneous infrastructure
- Similar systems for tracking inside databases, MapReduce, Sparks, etc.

# ASIDE: VERSIONING IN NOTEBOOKS WITH VERDANT

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



Further reading: Kery, M. B., John, B. E., O'Flaherty, P., Horvath, A., & Myers, B. A. (2019, May). [Towards effective foraging by data scientists to find past analysis choices](#). In Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (pp. 1-13).

# 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

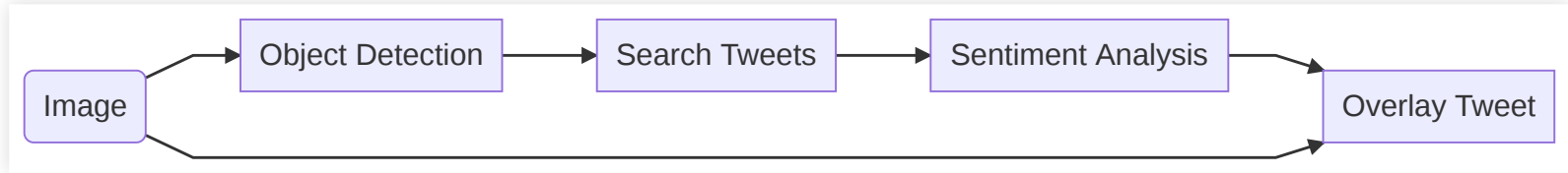
- Version everything
- Record every model evaluation with model version
- Append only, backed up

**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?**

```
<date>,<model>,<model version>,<feature inputs>,<output>  
<date>,<model>,<model version>,<feature inputs>,<output>  
<date>,<model>,<model version>,<feature inputs>,<output>
```

# LOGGING FOR COMPOSED MODELS



*Ensure all predictions are logged*

# DISCUSSION

What to do in movie recommendation scenarios? And how?



# 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, nonexperimental, partial, retest, sequential, statistical, varied, virtual)

Juristo, Natalia, and Omar S. Gómez. "[Replication of software engineering experiments](#)." In Empirical software engineering and verification, pp. 60-88. Springer, Berlin, Heidelberg, 2010.

# PRACTICAL REPRODUCIBILITY

- Ability to generate the same research results or predictions
- Recreate model from data
- Requires versioning of data and pipeline (incl. hyperparameters and dependencies)

# NONDETERMINISM

- 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



# FIXING MODELS

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



# CHASING BUGS

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

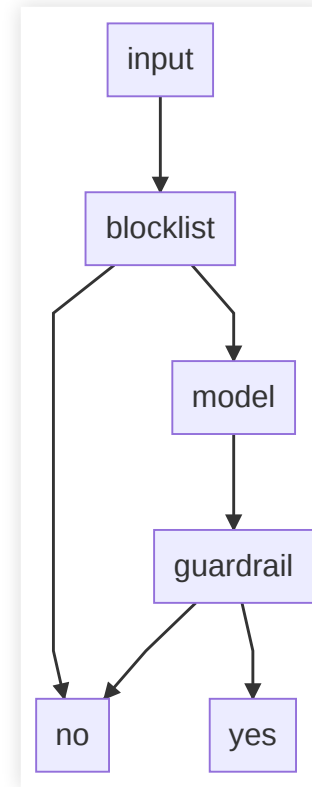
# PARTITIONING CONTEXTS

- 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



# SUMMARY

- Provenance is important for debugging and accountability
- Data provenance, feature provenance, model provenance
- Reproducibility 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

