University of Stuttgart
Germany

Institute of Software Engineering (ISTE)
Empirical Software Engineering (ESE)

Software Engineering for AI-Based Systems

Versioning, Deployment & Monitoring

Justus Bogner, Stefan Wagner

Image: https://govdatadownload.netapp.com/2018/05/infographic-ai-based-systems-effectiveness-military-adoption/

Learning Objectives

The goals for today are:

- to understand the basic functionality of and techniques for
  - versioning in ML systems
  - deployment in ML systems
  - monitoring and experimentation in ML systems

Versioning

Versioning Basics [1]

- In incremental software engineering, versioning is used to label the system state at different points in time, e.g., with Git
- Terminology:
  - *Revisions*: sequential states of the software over time; one revision follows a previous one; more recent ones have higher numbers
  - *Variants*: versions that exist in parallel, e.g., the same component for different countries; stored in different branches or files in a repository
  - *Versions*: general term that can refer to both revisions and variants
  - *Releases*: selected revisions that are explicitly announced / published (sometimes with special names)
- Frequently used scheme: semantic versioning with major, minor, and patch version, e.g., 0.1.0 or 1.3.25

[1] https://ckaestne.medium.com/versioning-provenance-and-reproducibility-in-production-machine-learning-355c48665005

Versioning in Machine Learning [1][2]

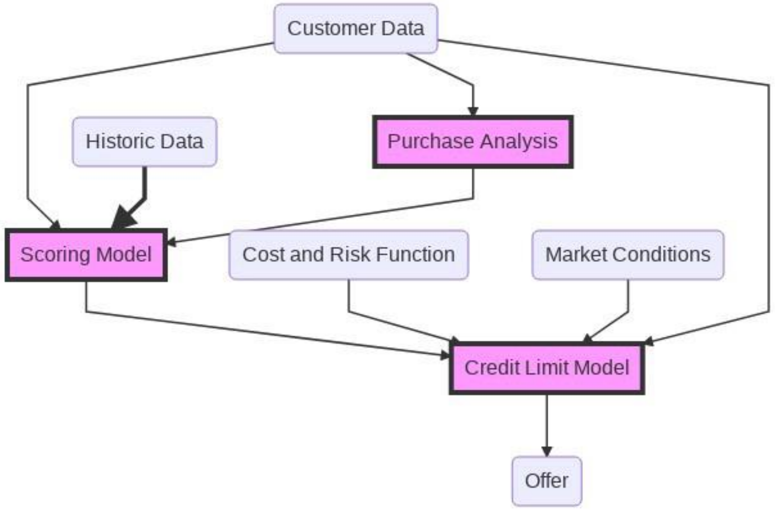
For ML systems, the basic principles stay the same, but:

- Data scientists may not be as familiar / disciplined with versioning, e.g., in notebooks
- Versioning of data sets?
- Versioning of models?
- Versioning of ML pipelines?
- Versioning in ML is important for
  - *Provenance*
  - *Reproducibility*

[1] https://ckaestne.medium.com/versioning-provenance-and-reproducibility-in-production-machine-learning-355c48665005
[2] https://ml-ops.org/content/mlops-principles#versioning

Provenance [1]

- Definition: "the chronology of the ownership, custody or location of a historical object" [2]
- Purpose: "to provide contextual and circumstantial evidence for its original production or discovery" [2]
- Important in ML systems for
  - *Debugging* (e.g., critical prediction errors)
  - *Auditing* (e.g., establishing accountability)
- At a smaller scale, documentation per version can be done manually
- At a larger scale, tool support for automatic provenance is needed (e.g., Google's Goods)



[1] https://ckaestne.medium.com/versioning-provenance-and-reproducibility-in-production-machine-learning-355c48665005
[2] https://en.wikipedia.org/wiki/Provenance

Data and Model Provenance [1]

- **Data provenance** (sometimes referred to as data *lineage*)
  - Where did the data come from (internal, external, which sources, etc.)?
  - When was the data collected (time range)?
  - Who was involved in collection and cleaning?
  - What processing steps did occur?
  - …
- **Model provenance** (feature provenance is a subset)
  - What training data was used?
  - What feature extraction code and hyperparameters were used?
  - What ML library version was used?
  - …
- Effective model provenance requires data provenance

[1] https://ckaestne.medium.com/versioning-provenance-and-reproducibility-in-production-machine-learning-355c48665005

Reproducibility [1][2]

- The term reproducibility is sometimes used with slightly different notions
- In science, we differentiate between *reproducibility* and *replicability*
  - Reproducibility: repeat an experiment with minor differences and, in general, receive similar results
  - Replicability: repeat an experiment or analysis in the exact same way and receive the exact same results
- For ML, most people mean the notion of *replicability* when they say “reproducibility”
- What hinders reproducibility? **Nondeterminism**, i.e., executing the same activity several times with different results

[1] https://ckaestne.medium.com/versioning-provenance-and-reproducibility-in-production-machine-learning-355c48665005
[2] https://ml-ops.org/content/mlops-principles#reproducibility

Reproducibility in ML [1][2]

- Model inference is (usually) deterministic, i.e., same model input → same prediction output
- Sources of nondeterminism in ML:
  - Random number generation during training (e.g., hyperparameters of neural networks, splitting of data sets, etc.)
  - Merging in distributed training depends on timing differences between nodes (e.g., *distribution strategy* pattern from Architecture 3)
  - Some more complex model architectures like random forests make explicit use of randomness
- Try to eliminate nondeterminism as much as possible by setting random seeds
- Additional hindrance: insufficient provenance → explicit documentation / versioning

[1] https://ckaestne.medium.com/versioning-provenance-and-reproducibility-in-production-machine-learning-355c48665005
[2] https://ml-ops.org/content/mlops-principles#reproducibility

Versioning Data Sets [1] (1)

- Most version control systems like Git are optimized for a large number of (smaller) source files → not for large datasets
- Different basic strategies are possible
  - **Store entire datasets:** with each modification, a new copy of the entire dataset is stored; can lead to high storage demand but is convenient for accessing individual versions; easy to implement with clear file naming convention in a Git repository
  - **Store deltas between datasets:** with each modification, only store changes between the versions; tools like `diff` and `patch` help with identifying changes; efficient if most changes are small but may require lots of modifications for restoring a previous version

[1] https://ckaestne.medium.com/versioning-provenance-and-reproducibility-in-production-machine-learning-355c48665005

Versioning Data Sets [1] (2)

- More advanced ore specialized strategies:
  - **Store offsets in append-only datasets:** in such datasets, you only need the file size (offset) to reconstruct any version; simplest form of versioning but can only be used for append-only data, e.g., log files, event streams, etc.
  - **Version individual records:** each record (e.g., tuple, JSON object, etc.) has a version number; especially useful with key-value databases (version per key); can also be implemented with one file per record via Git, but needs more effort
  - **Version pipelines to recreate derived datasets:** don’t store derived datasets but instead version the exact pipeline that created it; when necessary, reproduce it by re-running the pipeline; requires deterministic transformations as well as versioning of the original data and pipeline steps
- Choose a strategy based on dataset type, change volume, and change frequency

[1] https://ckaestne.medium.com/versioning-provenance-and-reproducibility-in-production-machine-learning-355c48665005

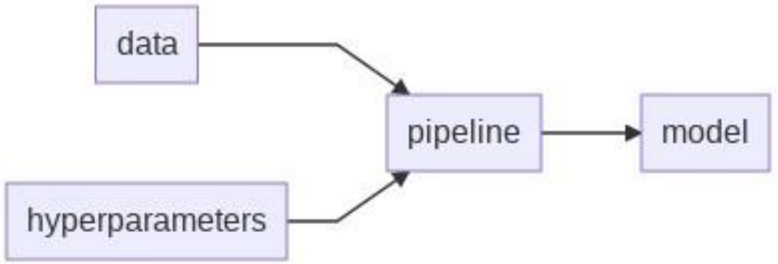
Versioning Models [1]

- Most ML models could be serialized as parameters, i.e., text-based
- But: even small changes to data or hyperparameters can lead to very different models
- Delta analysis is not very efficient and useful
- Therefore: usually versioned as large binary files
- Most version control systems support this, e.g., Git, but also databases
- Similar to “storing entire data set” strategy
- Trained models can be large, but are usually much smaller than datasets
- In some cases, model size can be reduced without losing much performance
- Still, lots of storage required with frequent retraining

[1] https://ckaestne.medium.com/versioning-provenance-and-reproducibility-in-production-machine-learning-355c48665005

Versioning Pipelines [1]

- Models are created through a pipeline that takes in data and hyperparameters
- The pipeline consists of different steps, e.g., cleaning, transformation, feature extraction, etc.
- Each step needs to be explicitly versioned
  - Code
  - Configuration files (e.g., hyperparameters)
- External libraries need to be versioned as well
  - Either use a package manager (e.g., `pip`, `npm`, etc.)
  - Or commit all dependencies to the repository (requires more space; avoid if possible)
  - Don’t use wildcards in library versions!
- Alternative: provide a Docker container with all dependencies per pipeline step
- Dockerfiles are versioned as well



[1] https://ckaestne.medium.com/versioning-provenance-and-reproducibility-in-production-machine-learning-355c48665005

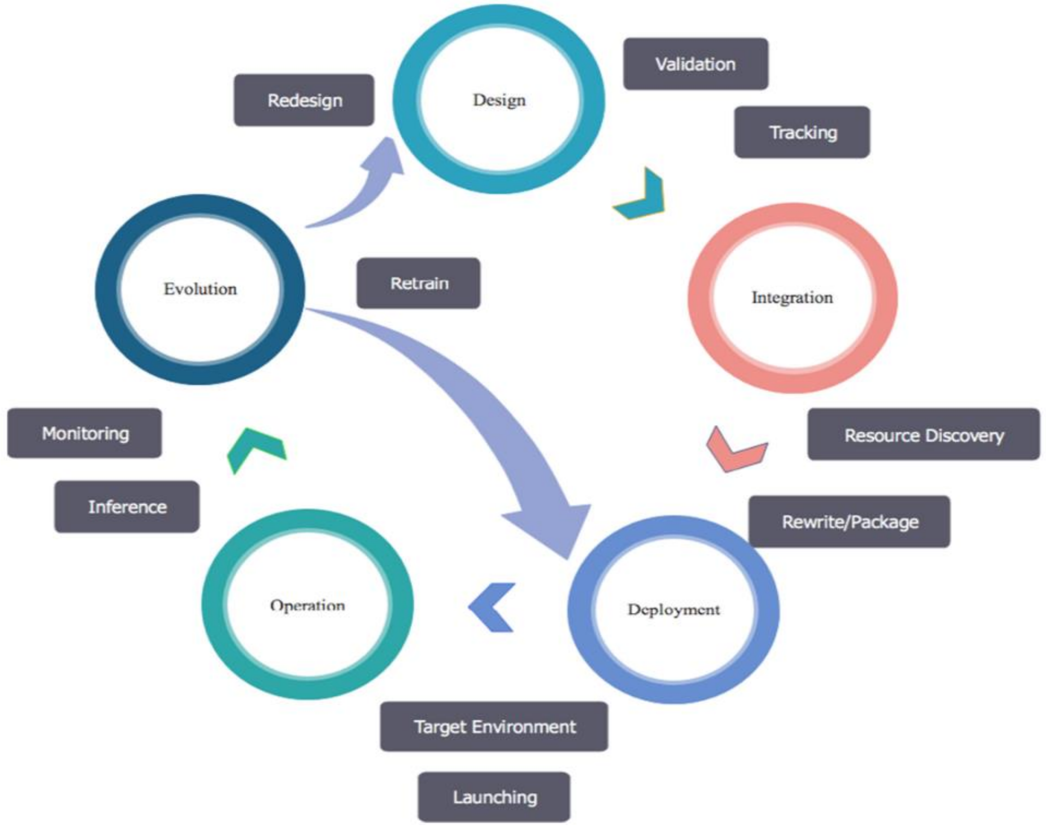
Summary

- In machine learning, versioning is mostly important for *provenance* and *reproducibility*
- Both support debugging, experiment analysis, and auditing
- Versioning should be applied to *datasets*, *ML models*, and *ML pipelines*
- Versions should reference each other, e.g., model v63 was built from dataset v15, pipeline v7, and hyperparameter configuration v19
- Many tools have been developed to help with versioning, provenance, and experiment tracking
  - DVC (https://dvc.org)
  - MLflow (https://mlflow.org)
  - ModelDB (https://github.com/VertaAI/modeldb)
  - For versioning notebooks: Verdant, nbdime, or jupyterlab-git

[1] https://ckaestne.medium.com/versioning-provenance-and-reproducibility-in-production-machine-learning-355c48665005

Deployment

General ML Deployment Process [1]



- Design
- Integration
- Deployment
- Operation
- Evolution

[1] M. M. John, H. Holmström Olsson, and J. Bosch, "Architecting AI Deployment: A Systematic Review of State-of-the-Art and State-of-Practice Literature," in Software Business. ICSOB 2020. Lecture Notes in Business Information Processing, Springer, Cham, 2021.

Before Deploying: Model Delivery Formats [1][2]

- Before deploying a trained model, it is usually exported in some file format
- Ideally, this is a self-contained file without dependencies (needed for, e.g., a *stateless serving function*)
- Most ML frameworks provide custom formats for this
  - scikit-learn: Pickle Format (`.pkl`)
  - TensorFlow: TF Serving Format (`.pf`) or SavedModel (`.pb`)
  - Keras: HDF (`.h5` or `.hdf5`)
  - PyTorch: Torch Script (`.pt`)
- This is convenient, but creates coupling to the framework
- Usually also coupled to Python (except TensorFlow)
- In the R language: custom functions to export / import → `saveRDS()` and `readRDS()`
- But what if I want to use my scikit-learn model in a stream processor on Apache Spark (*portability*)?

```
from keras.models import load_model

# creates an HDF5 file 'my_model.h5'
model.save('my_model.h5')

# loads a compiled model
model = load_model('my_model.h5')
```

Adapted from https://keras.io/api/saving/model_saving_and_loading/

[1] https://ml-ops.org/content/three-levels-of-ml-software#ml-model-serialization-formats
[2] A. Burkov, Machine Learning Engineering. True Positive Inc., 2020, Chapter 8, Model Deployment

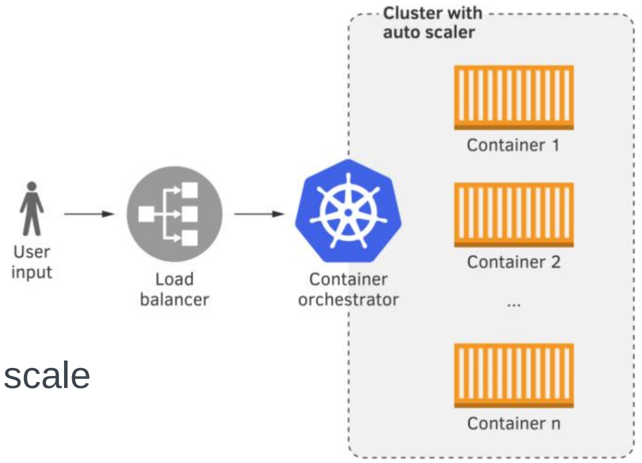
Technology-Agnostic Model Delivery Formats [1][2]

- Several language- and framework-agnostic model exchange formats have been proposed for **portability**
- Predictive Model Markup Language (PMML), `.pmml`
  - XML-based format from the Data Mining Group (DMG)
  - First developed in 1997, last update in 2019
  - Support is limited, especially for most popular ML Python frameworks (list of supported software)
- Portable Format for Analytics (PFA), `.json`
  - JSON-based format as a replacement for PMML, also from the DMG (2015)
  - Also limited support, needs a PFA-enabled runtime environment like Hadrian or Titus
- Open Neural Network eXchange (ONNX), `.onnx`
  - Open format for model exchange based on a computational graph model
  - Originally from the PyTorch team, now part of the Linux Foundation
  - Endorsed by major companies, supports the majority of popular ML frameworks

[1] https://ml-ops.org/content/three-levels-of-ml-software#ml-model-serialization-formats
[2] A. Burkov, Machine Learning Engineering. True Positive Inc., 2020, Chapter 8, Model Deployment

Types of Deployment [1][2]

- After integrating the exported model into an ML component: deploy it to the target location
- Targets for server-centric environments:
  - On-premise, e.g., onsite data center (models don't leave your organization, fast local response times)
  - Cloud (least management overhead, great scalability, response times can be higher)
  - Edge (can be complex to manage, fastest response times for clients in the respective zones)
- Targets for the host / platform:
  - Deploying ML components on a virtual machine
  - Deploying ML components as Docker containers
  - Deploying ML components as Serverless functions
- VMs, containers, and functions all can be scaled
- But: the more lightweight, the faster to deploy and the easier to scale



[1] A. Burkov, Machine Learning Engineering. True Positive Inc., 2020, Chapter 8, Model Deployment
[2] https://ml-ops.org/content/three-levels-of-ml-software#deployment-strategies

Containerization for Machine Learning [1]

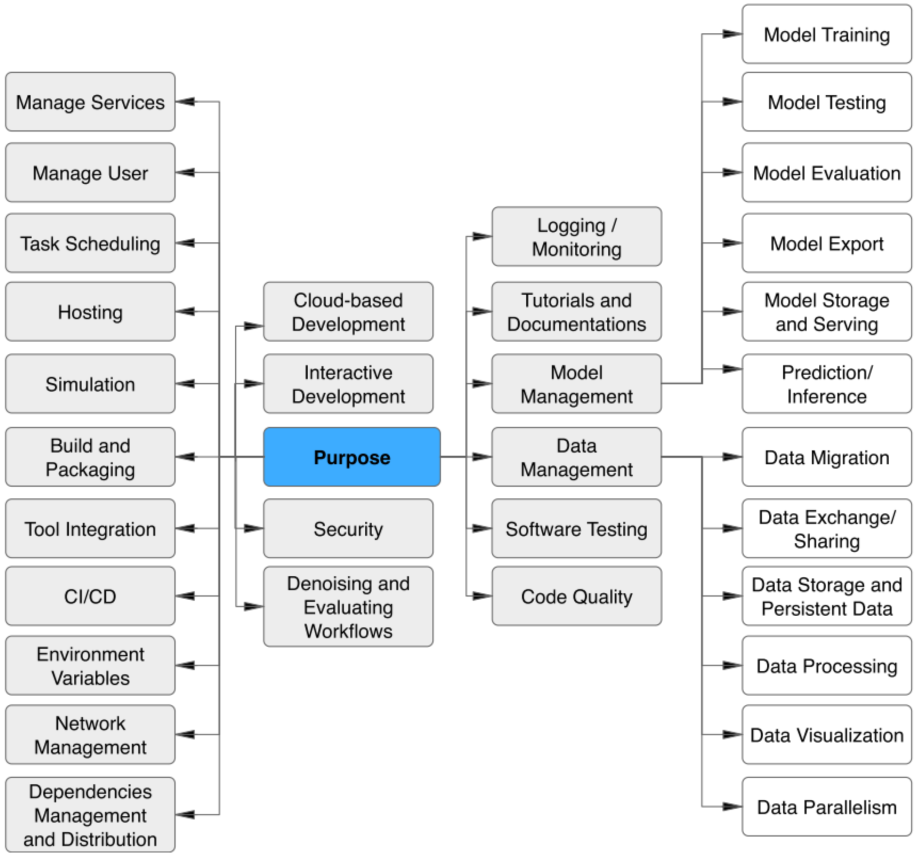
- Containers provide a nice middle ground regarding encapsulation, size, vendor lock-in, and scalability
- Docker is still the de-facto standard for containers
- A recent study [1] analyzed the usage of Docker for ML projects
- They analyzed 406 open-source projects on GitHub and Docker Hub
  - 42% applications for end-users
  - 23% MLOps / AIOps services to support deployment and operations, e.g., CI/CD
  - 16% toolkits for development, including AutoML
  - 15% DL frameworks base images or example use cases
  - 3% trained ML models
  - 1% documentation or tutorials
- Often, more than 1 Dockerfile → varied usage

[1] M. Openja, F. Majidi, F. Khomh, B. Chembakottu, and H. Li, “Studying the Practices of Deploying Machine Learning Projects on Docker,” in The International Conference on Evaluation and Assessment in Software Engineering 2022, 2022, vol. 1, no. 1, pp. 190–200.

Purpose of Using Docker for ML [1]

- They identified 21 different purpose categories
- Some frequent examples include:
  - Logging and monitoring
  - Cloud-based development
  - Interactive development
  - Model management
  - User management
  - CI/CD
- As base images, mostly OS, ML platforms, language runtimes, or DL frameworks were used

→ Docker provides encapsulation and portability for numerous ML use cases



[1] M. Openja, F. Majidi, F. Khomh, B. Chembakottu, and H. Li, “Studying the Practices of Deploying Machine Learning Projects on Docker,” in The International Conference on Evaluation and Assessment in Software Engineering 2022, 2022, vol. 1, no. 1, pp. 190–200.

Deployment Strategies for Production (1) [1]

- There are several ways you can approach deploying ML components to production
- **Single deployment:**
  - Immediately override the existing running component(s)
  - Benefits: simplest form that is very easy to implement, no additional nodes necessary
  - Disadvantages: potential downtime and risk (e.g., model issues affect all users), rollback takes time
- **Silent deployment:**
  - Deploy components to new nodes so that old and new versions run in parallel
  - All requests are routed to both versions, but only responses from the old version are returned to users
  - Responses from the new version are collected for some time and analyzed for potential issues
  - If responses look good, the new version can replace the old one (e.g., single deployment)
  - Benefits: doesn’t expose users to potential prediction issues, enough time for analysis
  - Disadvantages: more nodes needed, no real user reactions for new version (bad for some use cases)

[1] A. Burkov, Machine Learning Engineering. True Positive Inc., 2020, Chapter 8, Model Deployment

Deployment Strategies for Production (2) [1]

- **Canary deployment:**
  - Deploy the new version to a few new nodes and route a small portion of the traffic to it (e.g., 5%)
  - Analyze the user exposure to the new version for issues until you feel confident to use it for the rest
  - Benefits: provides a balance between exposing only a few users to potential prediction issues while getting real user reactions, does not need a lot of new nodes
  - Disadvantages: not suitable for spotting rare issues (e.g., a bug that affects <10% of users)
- **Blue / green deployment:**
  - Create the same number of nodes running the new version (green) as for the old one (blue)
  - Once the new nodes are ready, switch all traffic to the new version (green)
  - Briefly analyze the user reactions and switch traffic back to blue in case of major issues
  - Benefits: no downtime, lots of user reactions to analyze, efficient rollback
  - Disadvantages: requires double the nodes, exposes all users to potential bugs (at least for some time)

[1] A. Burkov, Machine Learning Engineering. True Positive Inc., 2020, Chapter 8, Model Deployment

Summary

- All major ML frameworks provide export formats for integration into an ML component
- Technology-agnostic formats exists, ONNX seems to be most promising (good for portability)
- Deployments can target
  - On-premise, cloud, edge
  - VMs, containers, stateless functions
- Docker provides encapsulation and portability for a wide variety of ML use cases
- Different strategies to deploy to production
  - Single deployment, silent deployment, canary deployment, and blue / green deployment
  - All have advantages and disadvantages
  - Can also be sequentially combined, e.g., first silent, then canary, and finally blue / green deployment

Monitoring & Experimentation

Monitoring for Machine Learning [1][2]

- Monitoring your ML system in production provides insights into its current usage and performance (*observability*)
- Intelligence telemetry management from (Hulten, 2019) is part of this, but you can monitor much more
- Many general tools and services are available for monitoring (e.g., Prometheus, Loki, Dynatrace, Datadog, etc.)
- Within MLOps, monitoring is usually combined with
  - Dashboards for visualizations (e.g., Kibana, Grafana, etc.)
  - Alerting, e.g., via email, instant messaging, or support desk ticketing systems
  - Automatic retraining via triggers
  - Online experimentation

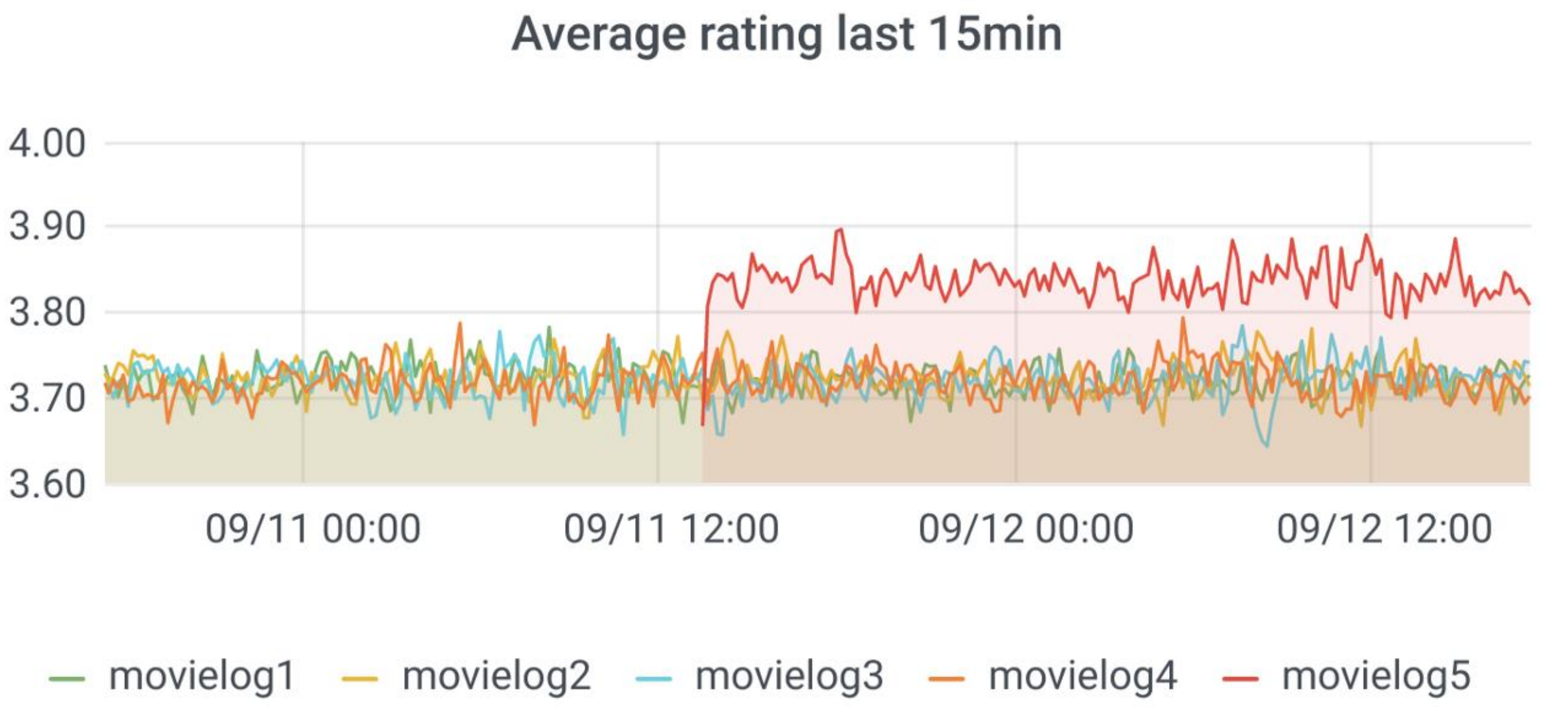
[1] A. Burkov, Machine Learning Engineering. True Positive Inc., 2020, Chapter 9, Model Serving, Monitoring, and Maintenance
[2] https://ckaestne.medium.com/quality-assurance-in-production-for-ml-enabled-systems-4d1b3442316f

Basic Monitoring Data for ML [1][2]

- Model usage
  - Model servings per hour (compare to yesterday)
  - Model servings per day (compare to last week)
  - …
- Minimum and maximum prediction values (potential outliers)
- Prediction values over a given timeframe
  - Median
  - Mean
  - Standard deviation
- Latency of predictions
- Memory and CPU usage when performing predictions

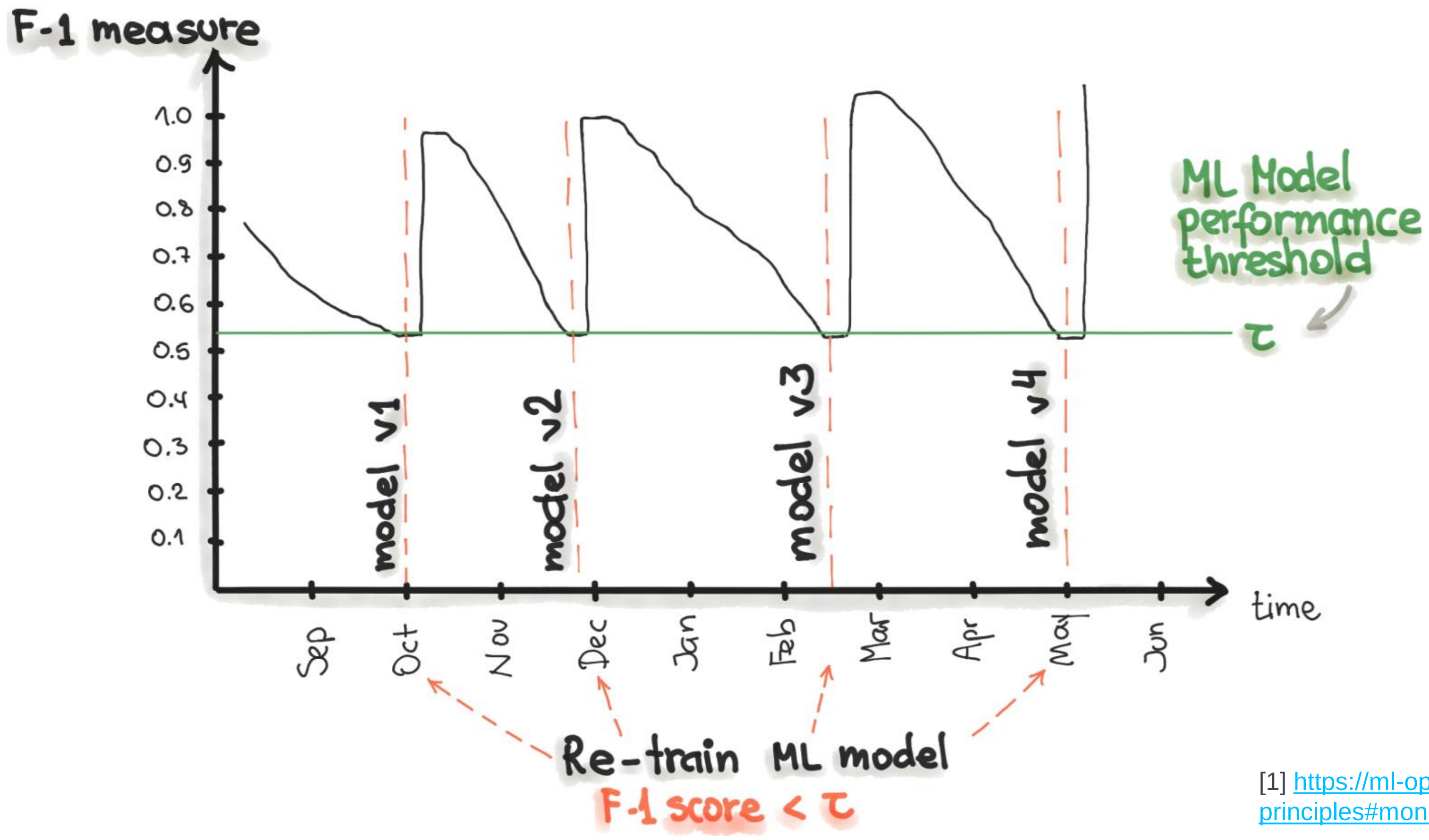
[1] A. Burkov, Machine Learning Engineering. True Positive Inc., 2020, Chapter 9, Model Serving, Monitoring, and Maintenance
[2] https://ckaestne.medium.com/quality-assurance-in-production-for-ml-enabled-systems-4d1b3442316f

Monitoring Example: Subpopulation Behavior [1]



[1] https://ckaestne.medium.com/quality-assurance-in-production-for-ml-enabled-systems-4d1b3442316f

Monitoring Example: Model Performance Decay [1]



[1] https://ml-ops.org/content/mlops-principles#monitoring

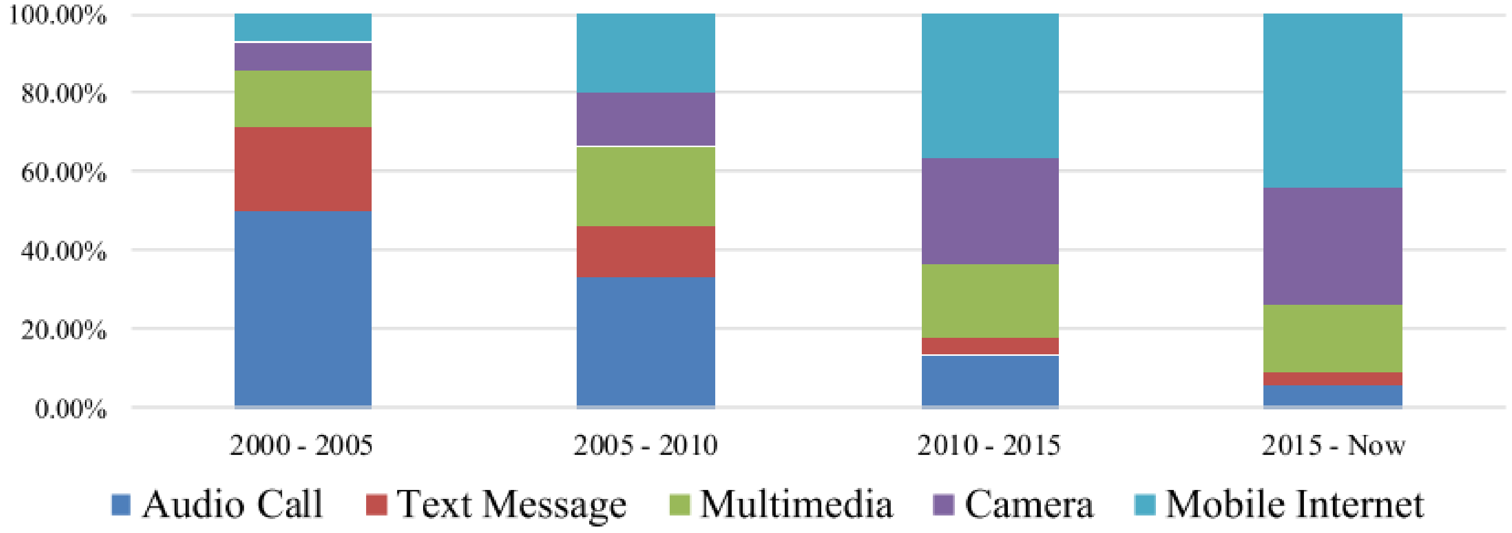
What Monitoring Should Detect: Drift [1][2]

- Our world and the people in it are constantly changing
- The input and output in ML systems changes as well → *drift* (sometimes also *shift*)
- Terminology varies, but frequent terms are *concept drift* [2], *data drift*, or *model drift* [1]
- Careful: refer sometimes to the same concept, sometimes to different facets of drift
- Drift can have a strong impact on ML model performance (*model decay*)
- Model performance monitoring is often unfeasible or delayed (manual labeling needed)
- Model performance monitoring alone will not reveal the *reasons* for the change
- Monitoring of the different data distributions in your ML system will provide more insights
- Retraining the model with new data may not always be the best solution!

[1] B. Wilson, Machine Learning Engineering in Action. Shelter Island, NY: Manning Publications, 2022, Chapter 12
[2] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, “Learning under Concept Drift: A Review,” IEEE Trans. Knowl. Data Eng., 2018.

Example of Concept Drift: Mobile Phone Usage Over the Years [1]

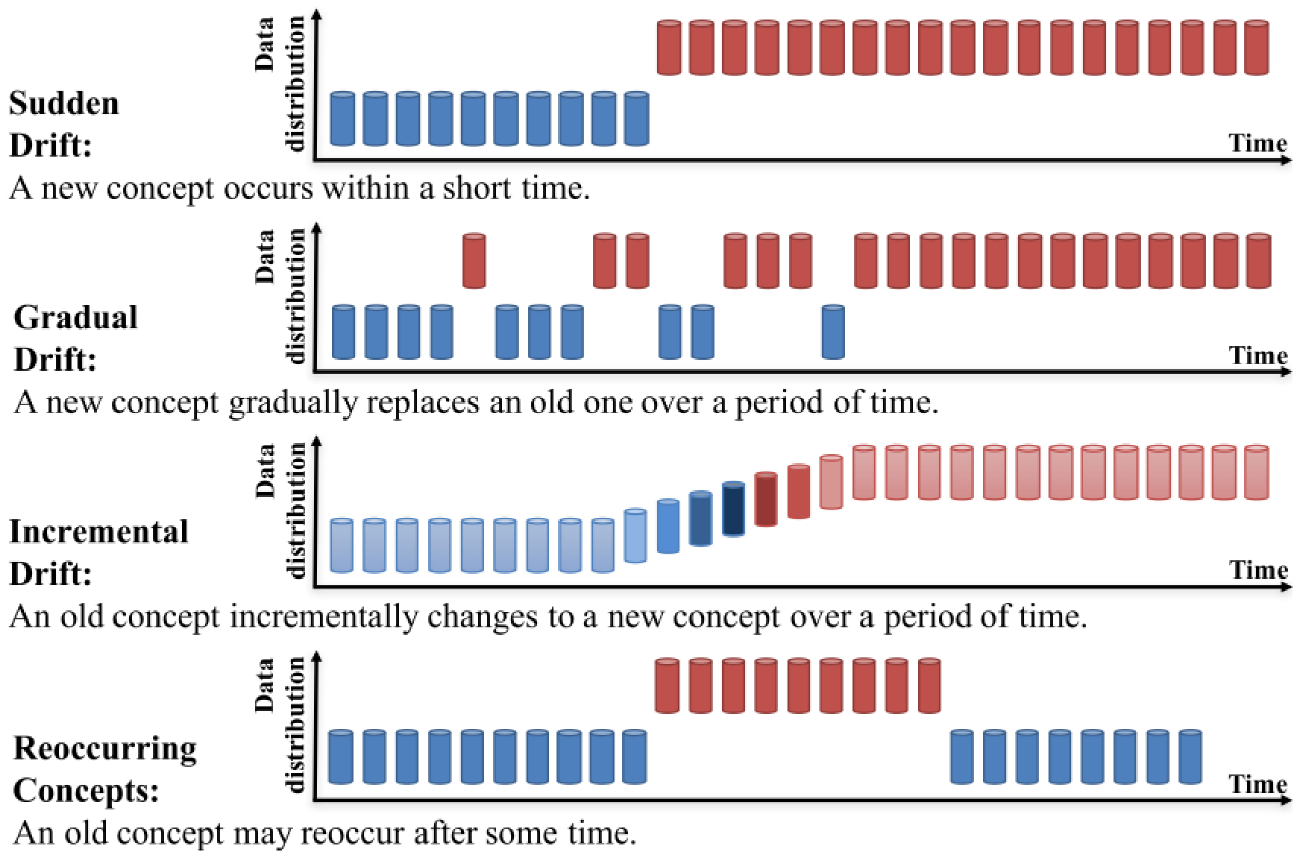
- Usage patterns of mobile phones have changed quite drastically over the years, e.g., concerning calls and internet usage:



- ML models trained on usage data from 2007 would have a hard time with accurate predictions in 2017

[1] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, “Learning under Concept Drift: A Review,” IEEE Trans. Knowl. Data Eng., 2018.

Drift Patterns Over Time [1]



Sudden Drift: A new concept occurs within a short time.

Gradual Drift: A new concept gradually replaces an old one over a period of time.

Incremental Drift: An old concept incrementally changes to a new concept over a period of time.

Reoccurring Concepts: An old concept may reoccur after some time.

[1] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, “Learning under Concept Drift: A Review,” IEEE Trans. Knowl. Data Eng., 2018.

Types of Drift

- **Feature drift** [1]
  - Many models use feature inputs from external sources like APIs, not only requests or internal DBs
  - If these sources change their representation, the model trained on the old one can’t handle this
  - Example: a weather API changes the temperature from Fahrenheit to Celsius
  - Another cause could be changed code for feature extraction that hasn’t been communicated
  - Monitor each feature for distribution changes (especially external ones)
  - Retraining with the new data or changing the feature extraction code can fix this
- **Label drift** (sometimes called prediction drift) [1]
  - Small changes in feature distributions can lead to strong changes in the prediction distribution
  - Example: a model that previously proposed sending a marketing coupon to 20% of customers now does this for 70%
  - The predictions don’t have to be wrong, but they can be challenging for the business nonetheless
  - Monitor model output distributions as well!
  - Retraining can often fix this, but sometimes you need additional business rules or model features

[1] B. Wilson, Machine Learning Engineering in Action. Shelter Island, NY: Manning Publications, 2022, Chapter 12

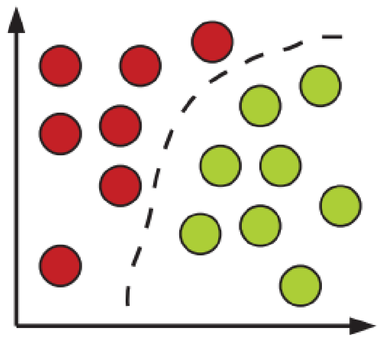
Concept Drift [1][2]

- “the statistical properties of the target variable, which the model is trying to predict, change over time” [1]
- A new, currently not considered confounding factor may appear as a feature for the model
- Example: changed purchasing behavior due to new economic policies, negative social media trends, etc.
- More formally: the actual probability at time *t* that feature vector *X* and output *y* occur together is different from the probability for the same at time *t+1*, i.e., $P_t(X, y) \neq P_{t+1}(X, y)$
- Hardest type of drift to analyze
    - Model inputs don’t change, new hidden relationships invalidate the predictions
    - Lots of monitoring data is needed for analysis and in many cases new feature engineering experiments

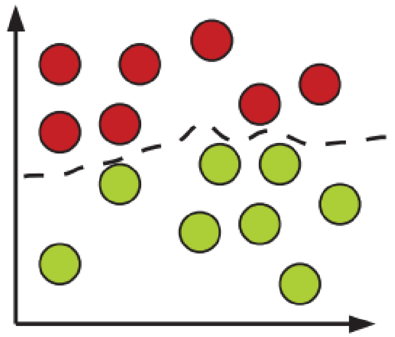
[1] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, “Learning under Concept Drift: A Review,” IEEE Trans. Knowl. Data Eng., 2018.
[2] B. Wilson, Machine Learning Engineering in Action. Shelter Island, NY: Manning Publications, 2022, Chapter 12

Difference between Concept and Data Drift [1]

Original data:

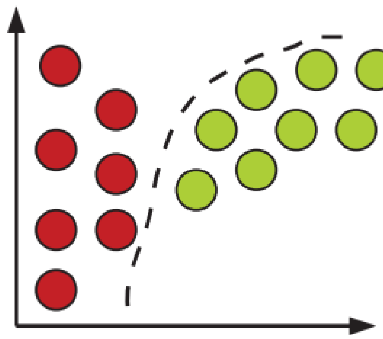


Concept drift:



$p(y|X)$ changes

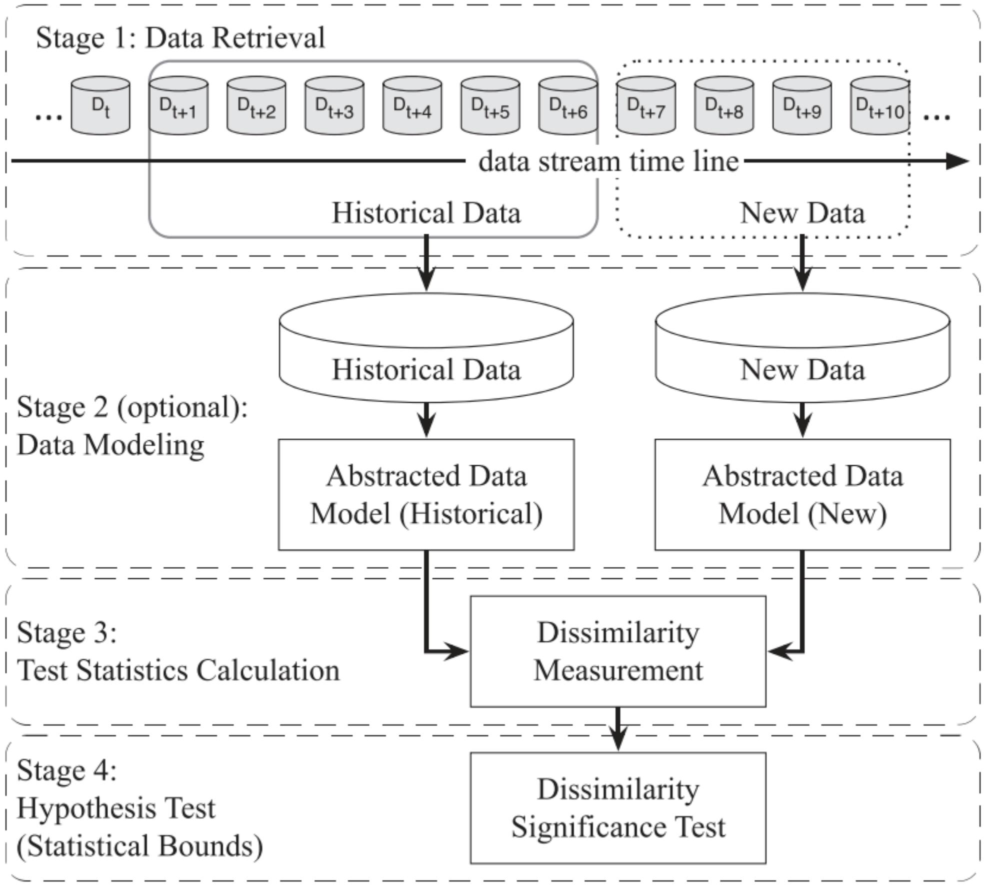
Data drift (also called virtual drift):



$p(X)$ changes, but not $p(y|X)$

[1] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, “A survey on concept drift adaptation,” ACM Comput. Surv., vol. 46, no. 4, pp. 1–37, Apr. 2014, doi: 10.1145/2523813.

A General Framework for Concept Drift Detection [1]


Stage 1: Data Retrieval
… D_t D_{t+1} D_{t+2} D_{t+3} D_{t+4} D_{t+5} D_{t+6} D_{t+7} D_{t+8} D_{t+9} D_{t+10} …
data stream time line
Historical Data
New Data
Stage 2 (optional): Data Modeling
Historical Data
New Data
Abstracted Data Model (Historical)
Abstracted Data Model (New)
Stage 3: Test Statistics Calculation
Dissimilarity Measurement
Stage 4: Hypothesis Test (Statistical Bounds)
Dissimilarity Significance Test

- Windowing is used for measuring dissimilarity
- Different types of algorithms can be used for drift detection
  - Based on error rate
  - Based on data distribution
  - …
- Different statistical tests can be used
  - Chi-square test
  - Kolmogorov–Smirnov test
  - Fisher’s exact test
  - …

[1] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, “Learning under Concept Drift: A Review,” IEEE Trans. Knowl. Data Eng., 2018.

Conclusion: Drift in ML

For effective drift analysis, you need to monitor

- Distribution of raw inputs, e.g., user requests
- Distribution of features
- Distribution of predictions
- Model performance, e.g., with intelligence telemetry (success metrics like click-through rate) or hand labeling

Continuous Experimentation [1]

- Extensive observability and monitoring are prerequisites for testing the effectiveness of different variants
- In SE, this is called **continuous experimentation**: “a software development approach based on experiments with stakeholders consisting of repeated Build-Measure-Learn cycles” [2]
- Continuous experimentation combines
  - Deployment strategies
  - Monitoring strategies
  - The scientific method with controlled experiments and hypothesis testing

[1] https://mlinproduction.com/ab-test-ml-models-deployment-series-08/
[2] B. Fitzgerald and K.-J. Stol, “Continuous software engineering: A roadmap and agenda,” J. Syst. Softw., vol. 123, pp. 176–189, Jan. 2017.

Continuous Experimentation for Machine Learning [1]

- CE is very popular for testing different ML model variants
- Sometimes called *online experiments* or *online model evaluation*
- Experiments can run only for a few hours, but also over longer periods of time, e.g., for business goals
- Two popular concrete techniques:
  - A/B testing
  - Multi-armed bandit (MAB) testing

[1] https://mlinproduction.com/ab-test-ml-models-deployment-series-08/

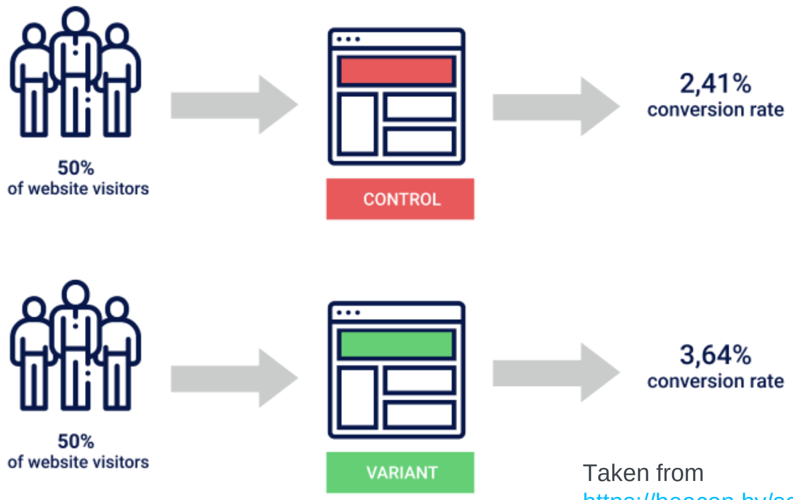
A/B Testing [1][2]

- In a typical A/B experiment (e.g., in medicine or psychology), participants are divided into two groups, e.g., randomly
  - Control
  - Treatment
- In Internet-based systems, we may have a very large number of users
- Only a small percentage may be enough, e.g., 5-10%
- With enough users, we could even run many more variants in parallel
- Great opportunity for evidence-based engineering!

[1] https://mlinproduction.com/ab-test-ml-models-deployment-series-08/
[2] https://ckaestne.medium.com/quality-assurance-in-production-for-ml-enabled-systems-4d1b3442316f

Basic Process for A/B Testing with ML Models [1][2]

- Deploy new variant(s) to new node(s), then route traffic to the new node(s)
- Monitor the model performance
- Use statistics to analyze which version performs best
  - With two versions: t-test, Mann–Whitney U test, etc.
  - With more than two versions: pair-wise tests or ANOVA



Taken from https://beacon.by/scandiweb/ab-testing-that-works-in-2021

[1] https://mlinproduction.com/ab-test-ml-models-deployment-series-08/
[2] https://ckaestne.medium.com/quality-assurance-in-production-for-ml-enabled-systems-4d1b3442316f

The Dilemma of A/B Testing [1][2]

- A/B testing has one potential drawback
  - Many observations may be needed for statistical significance, especially for small effects
  - **But:** exposing many of your users to a suboptimal model is not great
- Ideally, we want to route as few users as possible to the worse ML model variant(s)
- However, we also need sufficient data to reliably identify the best model
- This is called the *exploration-exploitation dilemma*
- How to address this? *Multi-armed bandits* (MAB), a reinforcement learning approach

[1] A. Burkov, Machine Learning Engineering. True Positive Inc., 2020, Chapter 8, Model Deployment
[2] https://vwo.com/blog/multi-armed-bandit-algorithm

Multi-Armed Bandits Evaluation with ML Models [1][2]

- MAB provides optimization for dynamic traffic allocation based on a reward function, e.g., conversion or click-through rate
  - Deploy new variant(s) to new node(s) → “arms”
  - MAB algorithm explores each “arm” and routes traffic to it
  - Based on the reward, MAB adjust the frequency of traffic
  - More profitable “arms” are used more frequently until one survives
- Requires a reasonable and fast reward function
- Downside: not many insights about why the winner is better can be gained

[1] A. Burkov, Machine Learning Engineering. True Positive Inc., 2020, Chapter 8, Model Deployment
[2] https://vwo.com/blog/multi-armed-bandit-algorithm

The ML Test Score for Monitoring (1) [1]

- The ML test score is a checklist for the production-readiness of ML systems created by Google
- It consists of 28 tests in 4 categories (data, model, infrastructure, and **monitoring**)
- **Monitor 1: Dependency changes result in notification**
  - Many ML systems consume data from other sources. Changes in these sources can impact ML models.
  - Solution: subscribe to all announcements for your dependencies and let them know you're using the data
- **Monitor 2: Data invariants hold in training and serving inputs**
  - Input data for ML models may change over time ($\rightarrow$ *drift*, e.g., feature drift)
  - Solution: implement monitoring to compare inputs and computed features to the training data with a schema
- **Monitor 3: Training and serving features compute the same values**
  - When feature extraction differs between training and production, performance suffers (*training-serving skew*)
  - Solution: monitor the feature computation in production and compare the distribution to the training environment

[1] E. Breck, S. Cai, E. Nielsen, M. Salib, and D. Sculley, "The ML test score: A rubric for ML production readiness and technical debt reduction," in 2017 IEEE International Conference on Big Data (Big Data), 2017, pp. 1123–1132.

The ML Test Score for Monitoring (2) [1]

- **Monitor 4: Models are not too stale**
  - The older a model is, the higher the chance that its performance may decay
  - Solution: monitor production models for their age to alert for retraining (thresholds can be derived from tests)
- **Monitor 5: The model is numerically stable**
  - With low quality data, invalid or implausible values can appear during training (e.g., NaN, infinity, etc.)
  - Solution: monitor the training process for such invalid numeric values and alert if thresholds are crossed
- **Monitor 6: The model has not experienced a dramatic or slow-leak regressions in training speed, serving latency, throughput, or RAM usage**
  - At scale, the efficiency of computational performance is very important, and changes can signify problems
  - Solution: monitor standard computational performance metrics and log them per data and model version
- **Monitor 7: The model has not experienced a regression in prediction quality on served data**
  - Input data in production is always newer than the last data a model was evaluated with before deployment
  - Solution: monitor the prediction quality of the deployed model (e.g., using telemetry or manual labeling)

[1] E. Breck, S. Cai, E. Nielsen, M. Salib, and D. Sculley, “The ML test score: A rubric for ML production readiness and technical debt reduction,” in 2017 IEEE International Conference on Big Data (Big Data), 2017, pp. 1123–1132.

Summary

- Monitoring is crucial to analyze the current performance of an ML system
- Basic usage and hardware-related metrics are a good start
- Additionally, more sophisticated monitoring for *drift* is needed
    - Feature drift
    - Label drift
    - Concept drift
- Monitor the distribution for inputs, features, and predictions plus the model performance
- Continuous experimentation can be used to combine deployment and monitoring
    - A/B testing
    - Multi-armed bandit testing
- The ML test score for monitoring can give an indication for production-readiness

Universität Stuttgart
Germany

Vielen Dank!

Dr. Justus Bogner, PostDoctoral Researcher

E-Mail	justus.bogner@iste.uni-stuttgart.de
Telefon	+49 (0) 711 685-88306
Fax	+49 (0) 711 685-88380

Universität Stuttgart
Institute of Software Engineering (ISTE) – Empirical Software Engineering
Universitätsstraße 38
70569 Stuttgart

https://www.iste.uni-stuttgart.de/institute/team/Bogner

https://www.researchgate.net/profile/Justus_Bogner