

Model Validation

Andreas Marfurt
MLOps

Information Technology
02.10.2025

Overview

- Metrics
- Monitoring
- A/B testing
- Debugging deployed models

Metrics

"A system or standard of measurement" (Oxford Languages)

We can measure:

- Model performance (accuracy, ...)
- System performance (requests/second, ...)
- Business objectives (conversion rate, ...)
- Data quantity and quality (label imbalance, ...)

Finding the right metric to optimize is very important for a data science project (remember the introduction slide "ML in production: reality")

Model performance

Most common ML metric:

- Accuracy: $\text{\#correct predictions} / \text{\#total predictions}$

Binary classification:

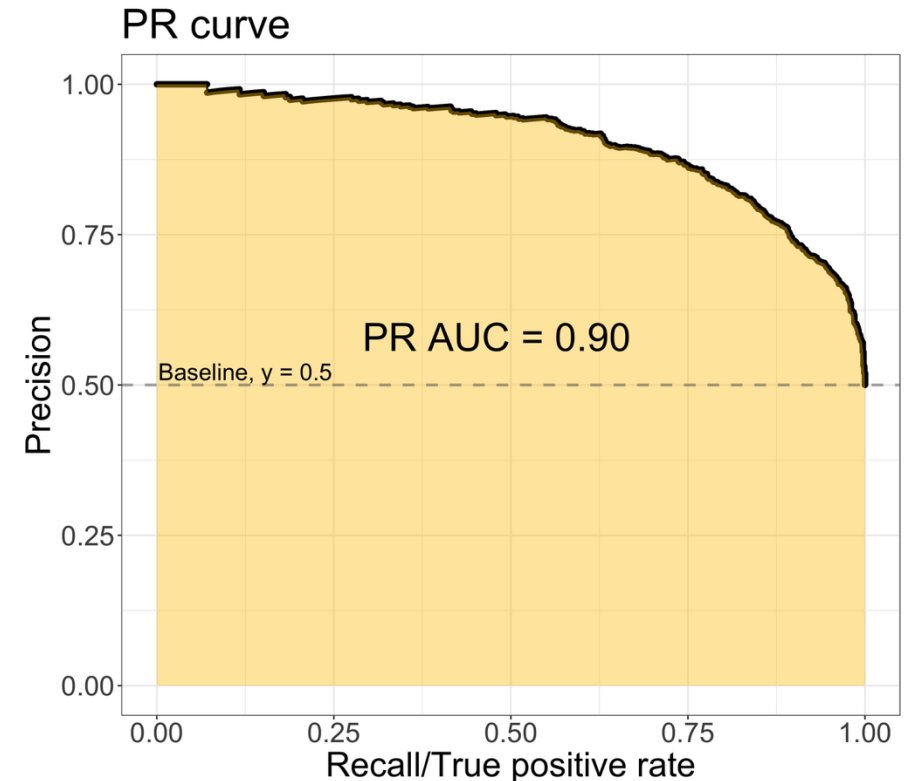
- True positive: label is positive, prediction is positive
- False positive: label is negative, prediction is positive
- True negative: label is negative, prediction is negative
- False negative: label is positive, prediction is negative

$$\rightarrow \text{Accuracy} = (\text{TP} + \text{TN}) / (\text{TP} + \text{FP} + \text{TN} + \text{FN})$$

Model performance

Precision-recall (for detection/retrieval problems)

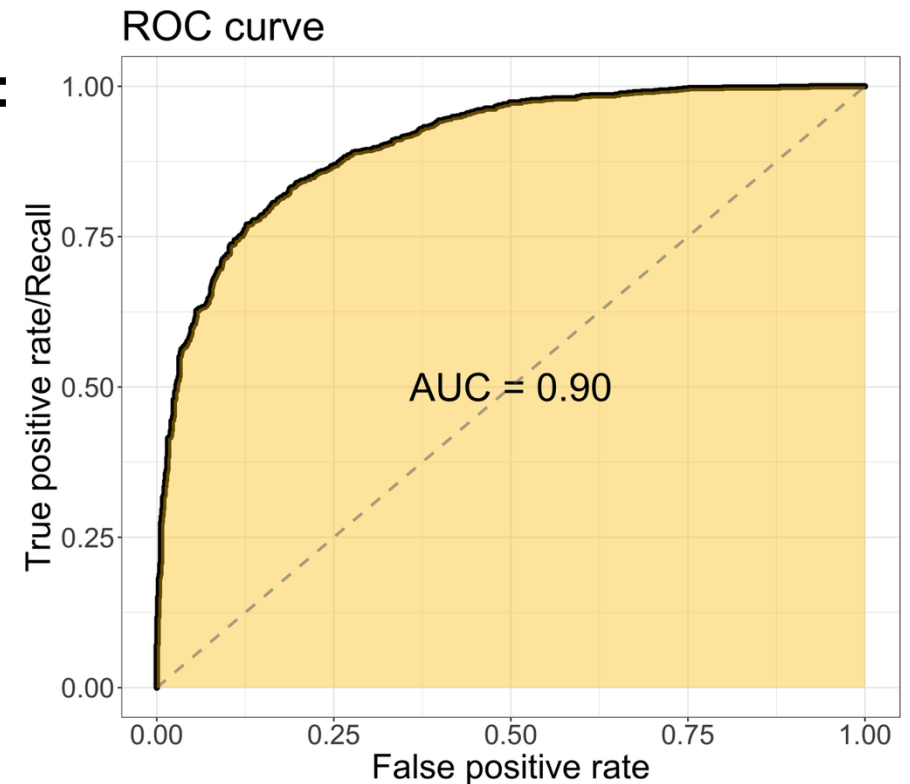
- Precision = $TP / (TP + FP)$
- Recall = $TP / (TP + FN)$
- F1/F1-Score/F-Score = $2PR / (P + R)$
- Precision-recall curve:
Shows tradeoff in precision when detecting more positive examples
- Area under curve (AUC):
as close to 1 as possible



Model performance

- True positive rate = sensitivity = recall = $TP/(TP+FN)$
- False positive rate = fall-out = $FP/(FP+TN)$
- Receiver operating characteristic (ROC) curve:
As you classify more negative data points, how many of the positives do you correctly detect
- Area under ROC curve is a metric for this tradeoff

Caution: ROC is less meaningful when labels are unbalanced → use precision-recall instead (personally, I have better intuitions for PR)



Model performance

- Intrinsic vs. extrinsic evaluation (representations)
 - Model-free vs. model-based evaluation
 - Domain-specific evaluations (FID for generated images, BLEU for translations, ...)
 - Performance relative to baselines/other models
 - Human evaluation
- Be careful not to rely on a single number, evaluate different aspects of a model's predictions

Practical model performance

- Regression tests: Can the model still solve the previous test cases after an update?
 - Same as in software engineering

Practical model performance

- Perturbation tests: Does the model still perform well under perturbations of the inputs?
 - Images: Change in lighting
 - Audio: Background noise
 - Text: Spelling errors
- Data augmentation with noisy inputs

Practical model performance

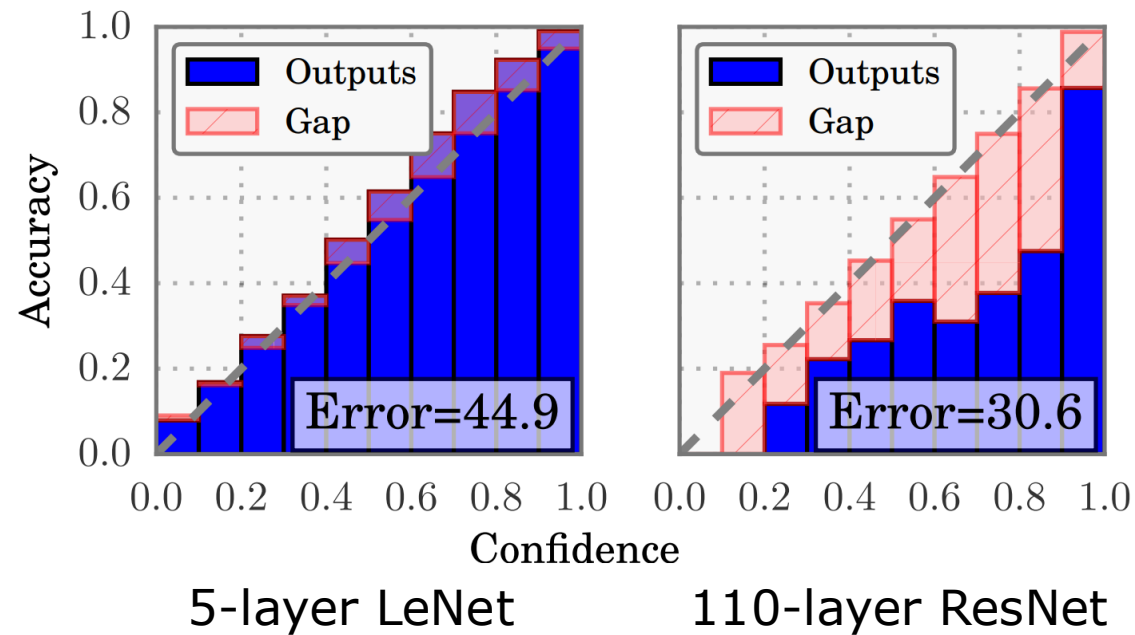
- Invariance tests: Model predictions should be invariant under changes to some unrelated input feature
 - Example: Change in name should not influence credit score
 - If it does, why could the model have picked this up?
 - How can we remove the correlation? Is removing an input feature enough?
 - What other features do we want the model to be invariant to?
- It can be hard to get rid of all correlations of inputs with gender, race, and other attributes that we do not want to discriminate against

Practical model performance

- Directional expectation tests: Some changes in input features should have predictable consequences in predictions
 - Increasing the area shouldn't decrease the price of a house

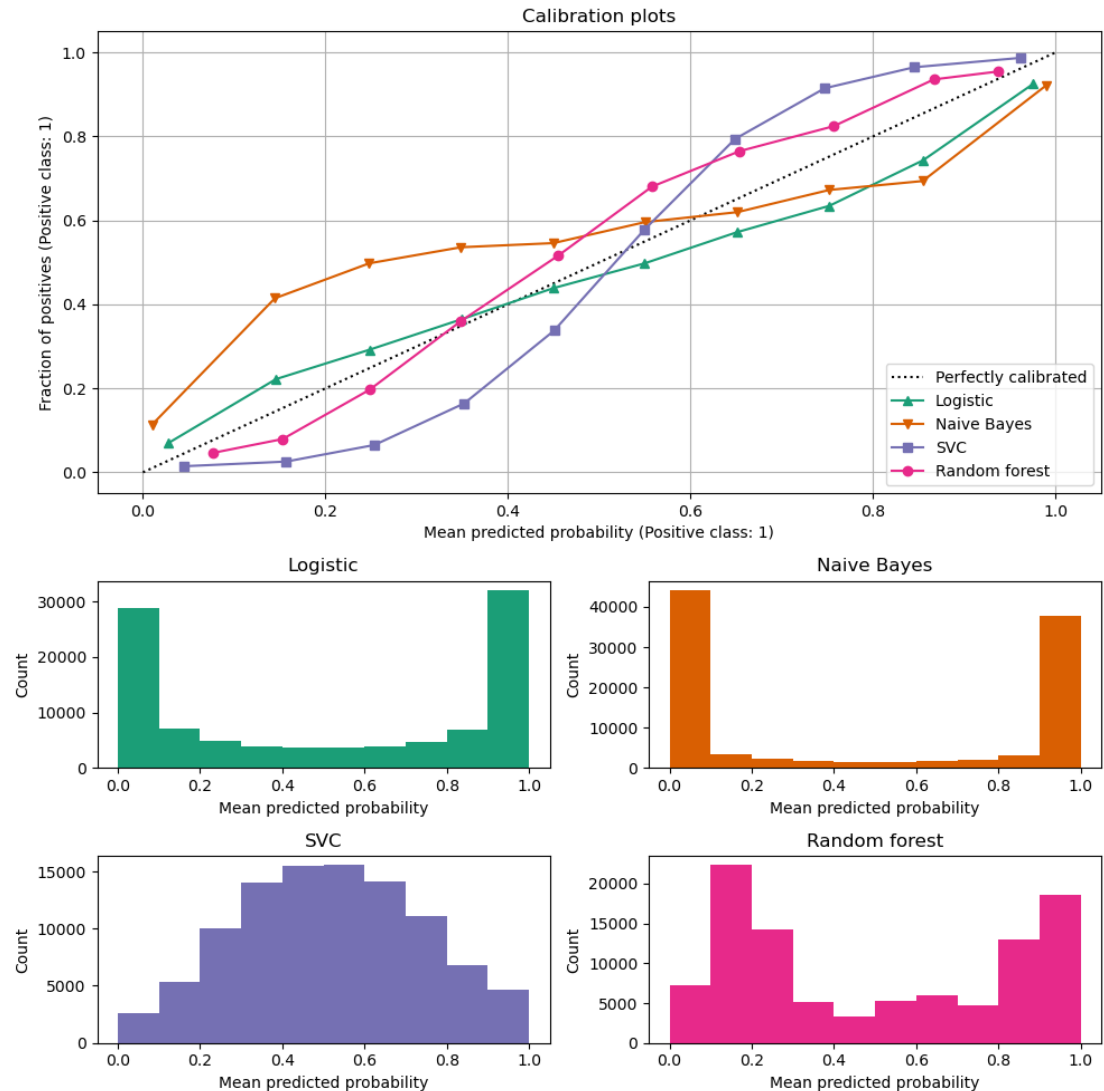
Practical model performance

- Calibration tests: The model confidence should correspond to how often it is correct
 - Predictions with 60% confidence should be correct in 60% of cases



Practical model performance

- Can compare calibration of different methods
- Good: accurate high-confidence predictions (**logistic**)
- Bad: inaccurate low-confidence predictions (**SVC**)



System performance

- Training: Epoch time, validation time, step time, intra step time, inter step time, time spent in I/O, ...
- Hardware utilization: GPU/memory/CPU utilization, free memory, I/O or networking bandwidth used, ...
- Deployed: Requests/second, request latency, request efficiency (how many requests could be batched), ...
- Data pipeline: Queue sizes, request count variability, number of concurrent processors (scaling resources), ...

Business objectives

- Impressions: How many users see my ad?
- Click-through rate: How many users that see my ad also click on it?
- Conversion rate: How many users that visit my website become paying customers/subscribers?
- Average order value: How much do customers spend per order?
- Customer lifetime value: How much does a customer spend in their time with the website?
- Net promoter score: How likely are users to recommend your product?
- Shopping cart abandonment rate: How many users add items to their shopping cart but don't proceed to checkout (in %)?

Data quantity and quality

- Counts of various attributes
- Duplicates
- Missing values
- Invalid values
- Spam
- Label (im)balance
- Biases in the data
- Data distribution shift (more on this later)

Monitoring

Monitoring

- What to monitor
- Possible failures
 - Edge cases and outliers
 - Degenerate feedback loops
 - Data distribution shift
- How to monitor
 - Logs
 - Dashboards
 - Alerts
 - Tools
 - Demo

What to monitor

- Define the right metrics
 - Can include user feedback

Goal:

- Detection of a failure ...
 - Measure performance to figure out if there is a failure
- ... vs. figuring out what went wrong
 - Monitor individual inputs/predictions for outliers, runtime, NaNs, etc.

What to monitor

- Define the right metrics
 - Can include user feedback

Goal:

- Detection of a failure ... ← Outputs more useful
 - Measure performance to figure out if there is a failure
- ... vs. figuring out what went wrong ← Inputs more useful
 - Monitor individual inputs/predictions for outliers, runtime, NaNs, etc.

Failure: Edge cases and outliers

- Edge cases: Usually in-distribution, but model performs badly
- Outliers: Out-of-distribution data points
- Strategies
 - Don't predict input examples outside a certain range
 - Suggest transformation to in-distribution data point (e.g. spelling correction)
 - Regularization during training, model should generalize to unseen inputs

Failure: Degenerate feedback loops

- Predictions influence model input
 - Algorithmic trading engine predicts prices and buys/sells stocks
 - Language model is trained on internet text, is also used to write text on the internet
 - Recommender systems that collect implicit user feedback (clicks, views, time spent on website) and get reinforced by user actions
 - Showing content that outrages people gets more interaction on social media
→ even more sensationalist content is shown
- Not easy to detect
 - Can measure recommendation success normalized by frequency an item is shown (or the rank where it is shown in a list)

Failure: Data distribution shift

- Input data distribution can change rapidly when a system interacts with users
 - News events: pandemic, plane crash, new technology, ...
 - Internet/social media trends
 - Language: New terms being invented
 - Vision: Special occasions, carnival, snow, ...
- Can be temporary or permanent
- Monitor data quality and model performance metrics
 - Decide if model needs to be updated or even fully retrained/finetuned

Types of data distribution shifts

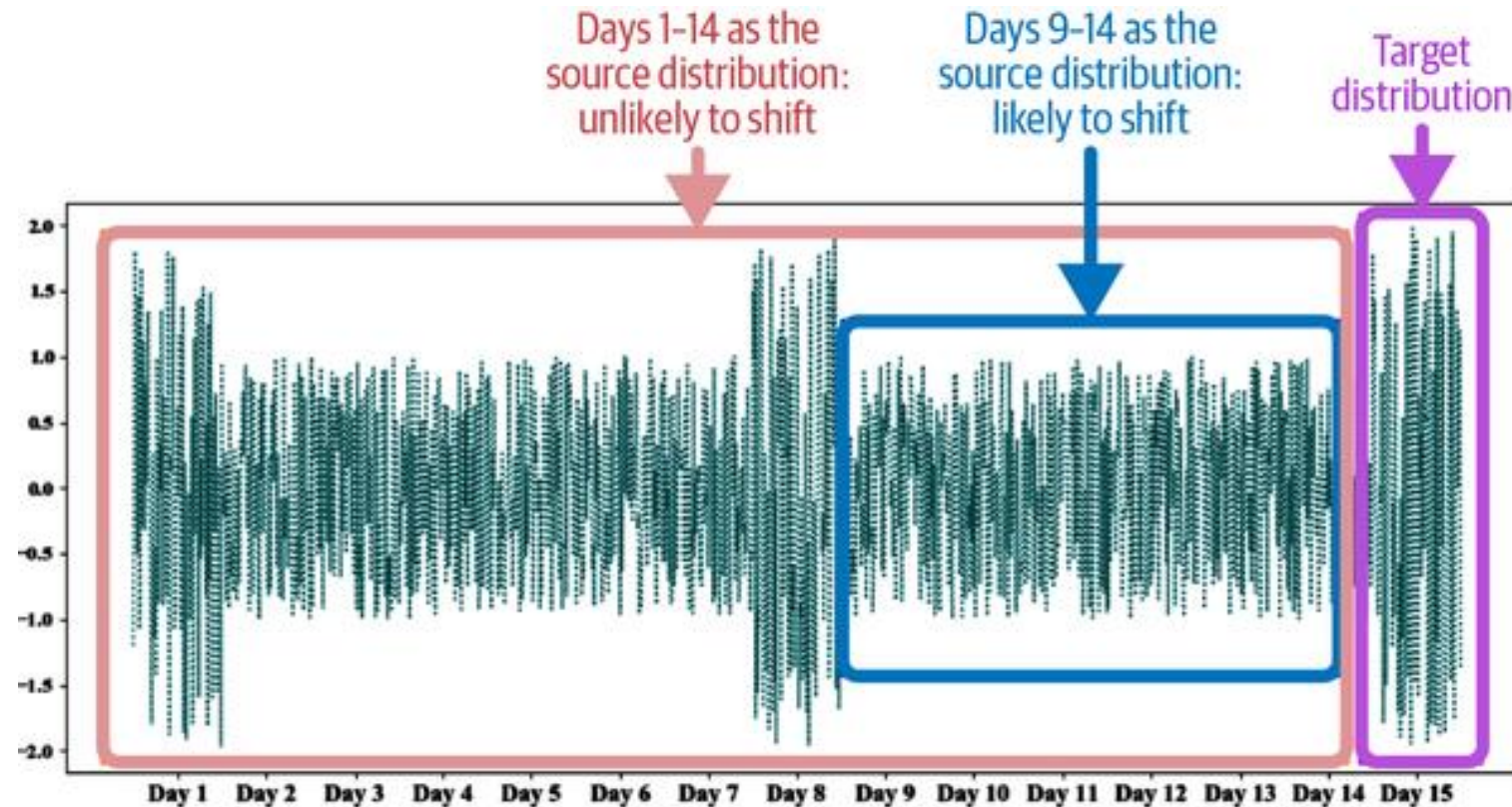
- Covariate shift: Input data changes
 - Data selection process changes: Marketing, new regulations (e.g. data privacy laws), Safari blocks third-party cookies, political movements, ...
- Label shift: Outputs change
 - Can follow from covariate shift: If input changes, outputs change as well
 - Label schema change, introduction of new categories
- Concept drift: Association between outputs and inputs changes
 - Mostly a result of behavior change of users
 - Often cyclic/seasonal
 - Power consumption on weekdays vs. weekends
 - Flight ticket prices on school holidays
 - Ice cream consumption in winter vs. summer

Quantify distribution shifts

- Compare your training data distribution to that of recently collected data
- Use measures for distributional similarity
 - Kullback-Leibler divergence/Jensen-Shannon divergence
 - Maximum mean discrepancy (MMD)
 - Kolmogorov-Smirnov test (for one-dimensional data)
- Library for drift detection: [Alibi Detect](#)
- Analyze data closely: Could there be an adversarial attack to poison your data? Is the data drift explainable?

Quantify distribution shifts

- The reference data distribution can depend on the time scale:



Addressing distribution shifts

- Update model on detected performance degradation
- Update on detected distribution shift (necessary?)
- Update periodically every N days/weeks/months

Updating strategies

- Finetuning the model
 - Train on new data points
 - Add in some fraction of your previous training data to avoid "catastrophic forgetting" (from multi-task learning, model unlearns what it previously knew)
 - Interleave old and new data, don't train on one first and then on the other
- Retraining the model from scratch
 - If data distributions are too far apart
 - Potentially sample data, with bias towards more recent data
 - When is the cutoff point where data is considered too old?
 - Automated training/finetuning pipelines help you here
 - Use regression testing with old test cases
 - Add test cases for new data

How to monitor: Logs

- Sequence of events
 - Tricky when there are multiple components running concurrently
- Details of event
 - Request time, user input, model predictions
 - Make it possible (and easy) to reconstruct an error
- Log in real time
- Use standardized tools
- Logs need to be searchable (use e.g. [ElasticSearch](#))

Dashboards

- Visualizations help humans quickly understand:
 - How healthy a system is
 - Correlations/patterns in inputs/outputs
- Visualizations help non-experts understand, too
- Too many visualizations can crowd out the important information
 - "Dashboard rot"

Alerts

Alert consists of:

- Alert policy: Defines the condition for an alert
 - E.g. performance under threshold, response time too long
- Notification channels
 - Alert dashboards
 - Automatic emails/instant messenger notifications
 - Product: [PagerDuty](#)
- Description: Details about the alert
 - Problem description
 - Severity
 - Potentially time horizon until resolution

Alerts

- Handling the alert
 - Runbook with routine procedures and operations that the system administrator or operator carries out ([Wikipedia](#))
 - Document problem, actions taken, resolution
 - Faster resolution next time
 - Potentially adapt processes/models to avoid the problem in the future
- Alert fatigue
 - Too many alerts desensitize people
 - Demoralizing: Getting a call in the middle of the night for something irrelevant/not in your responsibility

Tools

- [Grafana](#)
- [Prometheus](#)
- [Datadog](#)
- [Amazon CloudWatch](#)

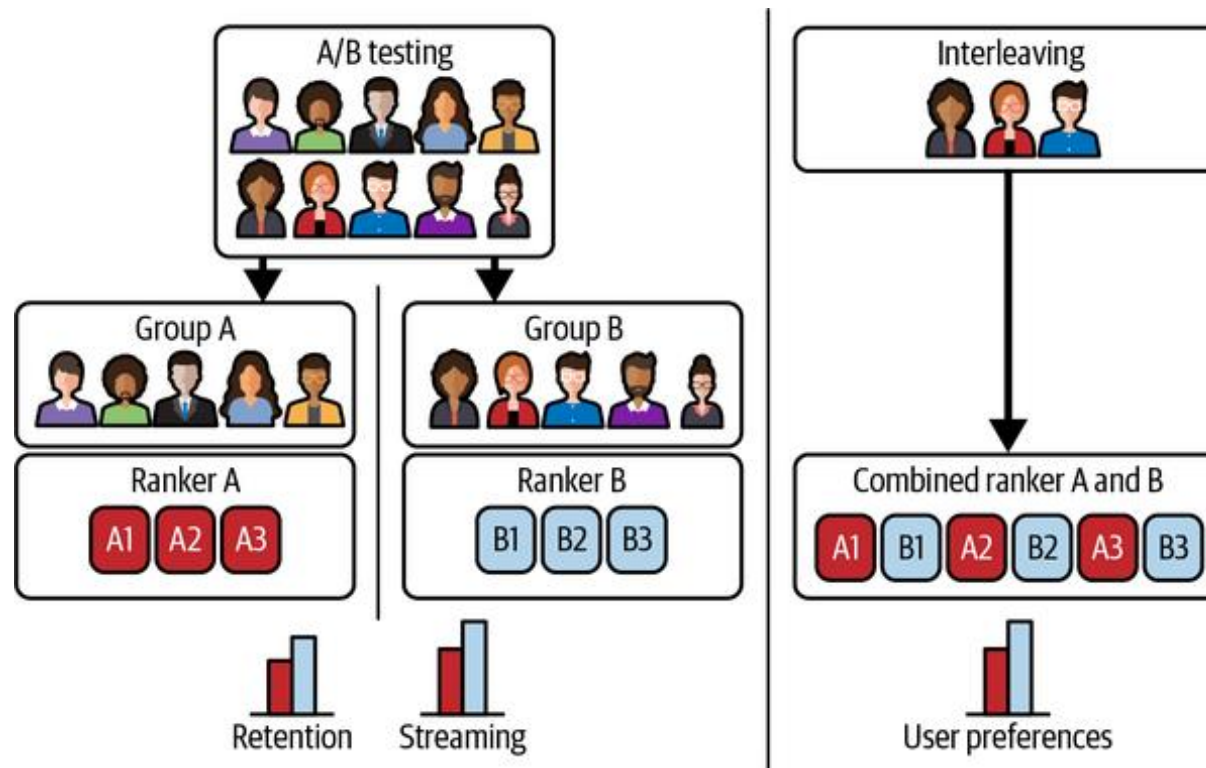
AWS CloudWatch Demo

A/B testing

- Old system A, new system B
- Route a fraction of traffic to new system
 - Randomly, do not introduce a selection bias here
- Compare how A and B perform (include business metrics)
- Alternatives:
 - Interleaved recommendations/ranking
 - Show both predictions of system A and B to the user
 - See which recommendations get selected
 - Shadow testing
 - Run all incoming requests through systems A and B, but only show predictions of system A to the user
 - Check how well system B's predictions would have performed

A/B testing

A/B testing vs. interleaved recommendations



Debugging deployed models

- Detecting problems
- Debugging with the scientific method
- Explaining model predictions

Detecting problems

- Monitoring: track, measure, and log different metrics to figure out *when* something goes wrong
- Observability: set up the system in a way that you can more easily detect *what* went wrong
- Check common issues
 - Data quality
 - Data distribution shift
 - New usage patterns

Debugging with the scientific method

Step 0: Try to reproduce the bug with a minimal example, will make debugging much easier

1. Determine expected behavior
2. Identify deviation from expected behavior
3. Form a hypothesis as to what caused the deviation
4. Run an experiment that tests this hypothesis
5. Check if experiment confirms hypothesis
 - a. Yes: Fix the problem
 - b. No: Go back to step 3 with updated information

Explaining model predictions

- Attribution methods
 - Gradient attribution
 - Saliency maps
 - [Collection of relevant papers by me \(written in Sep 2020\)](#)
- Explain importance/contribution of inputs to predictions
 - [SHAP](#) (all ML models)
 - [LIME](#) (all ML models)
 - [DecompX](#) (NLP)
- More details in lecture on interpretability/explainability