Aston University
Birmingham

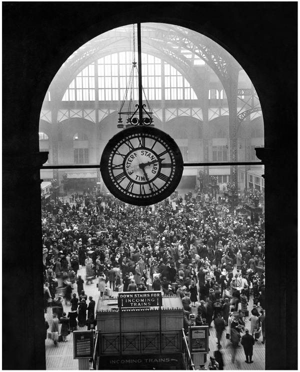
The role of models@run.time in self-explanation in the era of Machine Learning

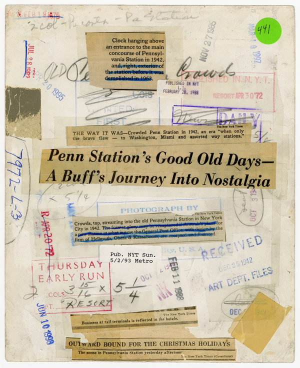
Antonio Garcia, Juan Marcelo Parra-Ullauri and Nelly Bencomo

14th International Workshop on Models@run.time
September 17th, 2019

Introduction

Machine learning is everywhere!



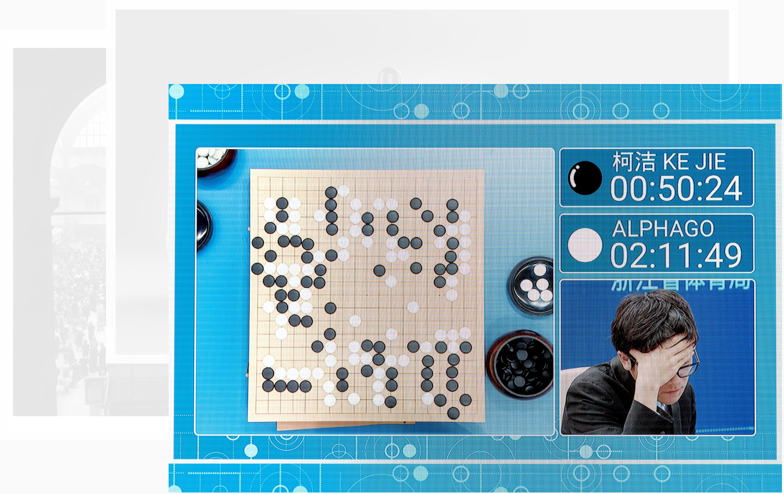


Annotating 100+ years of photos at the New York Times (GCP)

Machine learning is everywhere!

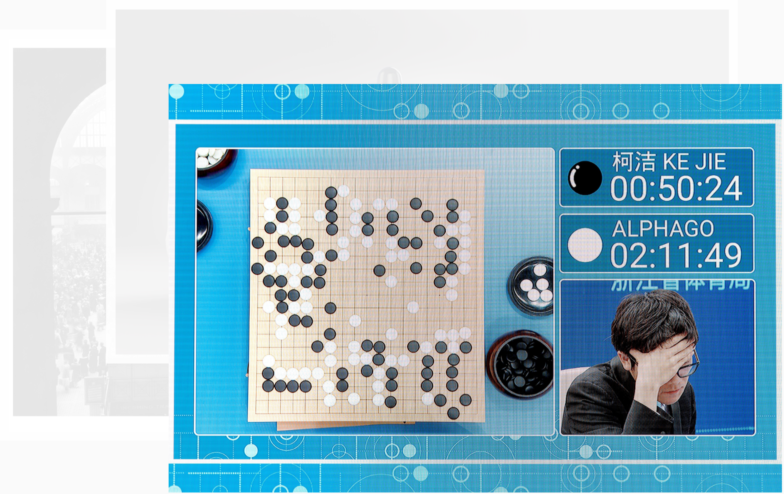
Running an automated ride-hailing service in Metro Phoenix (Waymo)

Machine learning is everywhere!



Beating world-level experts at Go and Starcraft (AlphaGo $\rightarrow$ AlphaZero)

Machine learning is everywhere!



Machine learning is going to take over the world!

Machine learning can automate existing biases (I)

Source — ProPublica

Crime risk scores

- ProPublica study on 7000 automated risk assessments in Broward County (Florida) with Northpointe tool
- 2x rate blacks wrongly mislabelled "high risk"
- 2x rate whites wrongly mislabelled "low risk"
- Training data may have questions correlated with race

Machine learning can automate existing biases (I)

Source — ProPublica

Crime risk scores

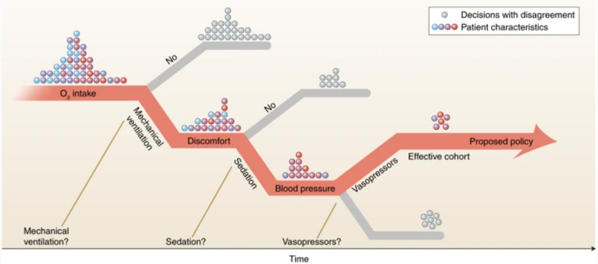
- ProPublica study on 7000 automated risk assessments in Broward County (Florida) with Northpointe tool
- 2x rate blacks wrongly mislabelled “high risk”
- 2x rate whites wrongly mislabelled “low risk”
- Training data may have questions correlated with race

Machine learning can automate existing biases (II)

Recruiting automation

- Reuters reported Amazon had worked on and later scrapped a machine learning-based CV screening system
- Most CVs sent to Amazon are from males (tech industry after all...)
- Algorithm learned to ignore common IT skills (e.g. programming)
- Algorithm favored aggressive language ("executed", "captured")

Machine learning can be inscrutable (I)

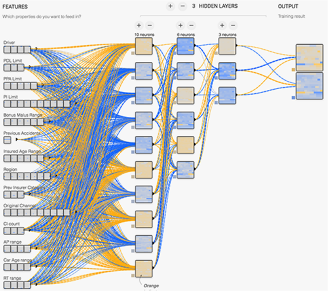


Debbie Maizels/Springer Nature

Nature Medicine guidelines for reinforcement learning

- Guidelines for RL when assist patient treatment decisions
- Concerns about available information, the real sample size for a specific scenario, and "Will the AI behave prospectively as intended?"
- Concludes that "it is essential to interrogate RL-learned policies to assess whether they will behave prospectively as intended"

Machine learning can be inscrutable (II)



Google Cloud whitepaper on TensorFlow at AXA Insurance

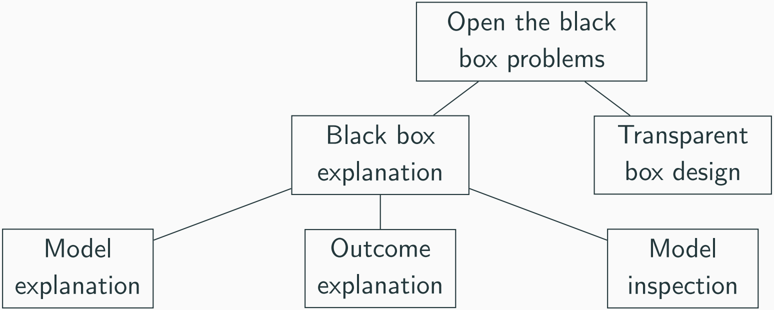
- Assesses clients at risk of "large-loss" car accidents (w/payouts of \$10k+)
- Built neural net with 70 inputs (age range of car/driver, region, premium...)
- 78% accuracy vs 38% (random forest)

Interesting note

"AXA is still at the early stages with this approach — architecting neural nets to make them transparent and easy to debug will take further development — but it's a great demonstration of the promise of leveraging these breakthroughs."

How can runtime models help?

Types of explanations: making AI "interpretable"



R. Guidotti, A. Monreale, S. Ruggieri et al.
A Survey of Methods for Explaining Black Box Models.
ACM Computing Surveys, 51(5) 1–42. January 2019.
http://dx.doi.org/10.1145/3236009

Things to watch out for

When do we need interpretability?

- Whenever there are real consequences from the result!
- Finding cat pictures *vs* deciding if someone "looks like a criminal"

Interpretability of the process

- Different algorithms have different inherent interpretability
- Compare decision trees and neural networks

Interpretability of the data

- Humans can easily follow explanations about texts or images
- Hard to explain conclusions about complex networks, GIS data...

Size of the explanations

- In an emergency, I can't read 100 pages!
- In a plane crash post-mortem, we need *all* the details

Runtime models for transparent boxes

Rule learning

- Organizing the rules for sufficiently large systems while they are accurate *and* concise is the hard part
- Beyond decision trees: Bayesian rules, "interpretable decision sets", linear models, predictive association rules, etc.
- The runtime models would consist of these rules, plus the way in which they were learned

Prototype selection

- Learn set of protoypes for various equivalence classes in the input space within the training set
- Explanation = here are my prototypes, here are their labels, I apply X strategy to decide (K-means, K-medoids)
- Runtime models would preserve the prototypes, the process for their selection, and trace how the prototypes are used

Runtime models for explaining the process

Generating close-enough interpretable mimics

- Train the NN/decision forest/SVM as usual
- Later, create an interpretable model that mimics the original one as close as possible (e.g. decision tree, ruleset)
- Runtime models could be involved in a loop here, where the NN trains a little, the mimicking model evolves, and users are kept in the loop about the training

Summarizing system evolution

- Suppose the system goes through a finite number of states
- Is there a periodicity to the evolution to the system?
- We can keep a runtime state transition model to incrementally build a baseline of typical system evolution

Runtime models for explaining the outcome

Example from neural networks: saliency masks

We can visualize which parts of the image led to each label for an image, and how confident the network was about it.

How does this translate to runtime models?

- We need to represent what the system perceived, what it thought about what it saw, and how confident it was about its next decision
- Essentially, *decision provenance*
- The history of the model becomes important once more!

Runtime models for inspecting properties

Neural network approaches

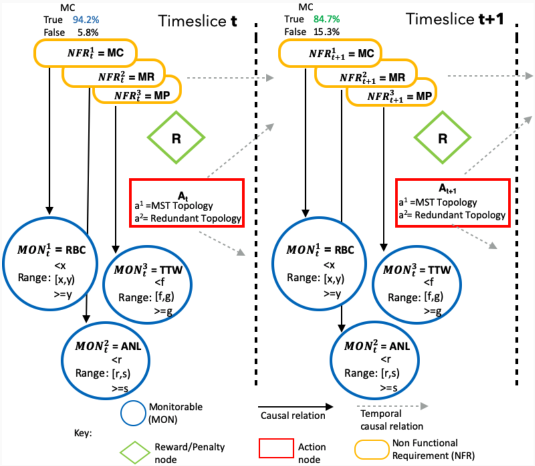
- Cortez, Embrecht: find feature importance (e.g. pH level vs probability of high-quality wine)
- Statistical approaches also exist for general black boxes (partial dependence plot)
- Activation maximization: generate image that highly activates the network (what is each neuron looking for?)

Going back to runtime models

- Do we have approaches to run these sensitivty analyses?
- Model checking is common in MDE for this: does it scale to real-world systems?
- Can we inspect for "softer" desirables, e.g. fairness?
- We could query the history of our runtime models to test desirable properties about its evolution

Example system: RDM

Remote Data Mirroring system



Our current version of RDM

Key points about RDM

- Self-adaptive system
- Switches network between Minimum Spanning Tree and Redundant
- Balances 3 non-functional reqs.:
  - Maximization of Reliability
  - Minimization of Cost
  - Maximization of Performance

Is RDM a transparent box?

Our RDM uses Partially Observable Markov Decision Processes

- Underlying state cannot be directly observed
- Indirectly observed based on three metrics:
  - Range of Bandwidth Consumption (RBC, low is best for MC/MP)
  - Total Time for Writing (TTW, low is best for MC/MP)
  - Active Network Links (ANL, high is best for MR)
- RDM uses Bayesian inference + tree-based lookahead to estimate satisficement of NFRs, then applies reward table to make decision
- Recent versions allow for automated tweaking of the reward table

Is RDM a transparent box?

Our RDM uses Partially Observable Markov Decision Processes

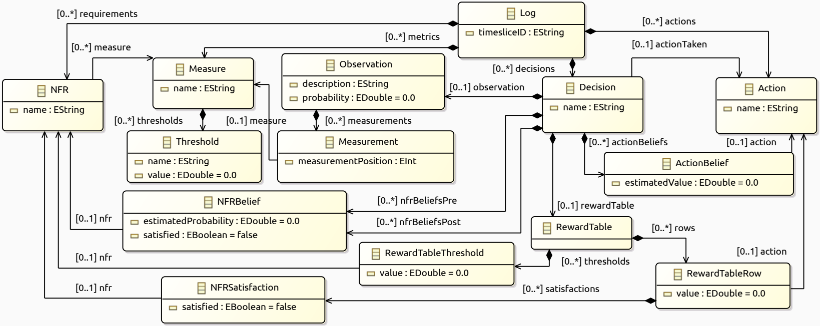
- Underlying state cannot be directly observed
- Indirectly observed based on three metrics:
  - Range of Bandwidth Consumption (RBC, low is best for MC/MP)
  - Total Time for Writing (TTW, low is best for MC/MP)
  - Active Network Links (ANL, high is best for MR)
- RDM uses Bayesian inference + tree-based lookahead to estimate satisficement of NFRs, then applies reward table to make decision
- Recent versions allow for automated tweaking of the reward table

Is this transparent?

- Rule-based system: the overall decision can be traced
- RDM is not visibly exposing its rules and trees, though!
- Transparency requires considering the experience as well

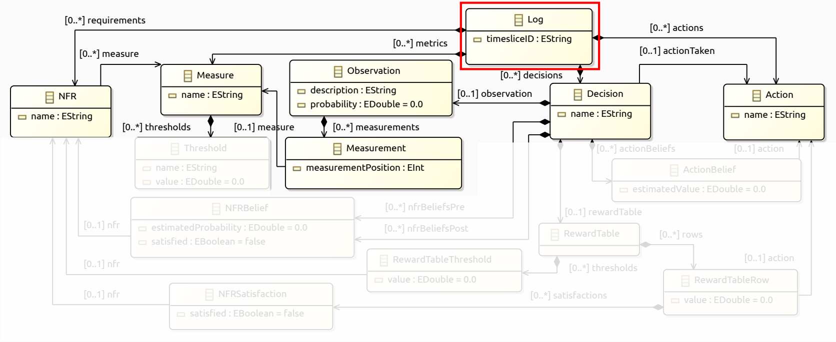
Outcome explanation through dedicated trace models

Existing JSON logs were translated on-the-fly to a trace metamodel:



Outcome explanation through dedicated trace models

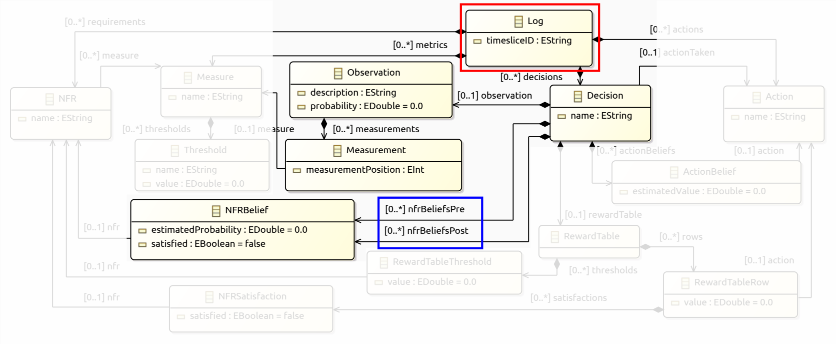
Existing JSON logs were translated on-the-fly to a trace metamodel:



- Log of Decisions, made upon Observations where Measurements have been taken of certain Measures related to NFRs

Outcome explanation through dedicated trace models

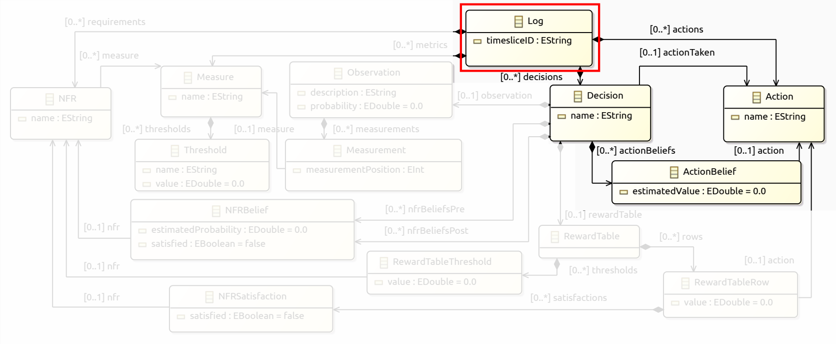
Existing JSON logs were translated on-the-fly to a trace metamodel:



- Log of Decisions, made upon Observations where Measurements have been taken of certain Measures related to NFRs
- Observations result in pre/post-decision levels of belief in the satisficement of the NFRs

Outcome explanation through dedicated trace models

Existing JSON logs were translated on-the-fly to a trace metamodel:



- Log of Decisions, made upon Observations where Measurements have been taken of certain Measures related to NFRs
- Observations result in pre/post-decision levels of belief in the satisficement of the NFRs
- Each Decision picks the Action with the highest estimated value

Model inspection through time-aware querying

Watching for properties as system runs

- Easier to formulate + scalable than automated proofs
- With enough history and scenarios, we gain confidence
- We formulate the property test as query on model history

Approaches: balance between flexibility and storage efficiency

- We turn the model history into a temporal graph and use a time-aware model query language (we have a dialect of EOL)
    - We have a presentation on Thursday about this!
- We turn the query into a set of Complex Event Processing patterns on model changes, trying to find evidence/violations on the fly

Example: TrafficLight

- Suppose we have a TrafficLight with two attributes:
  - `count` with the number of vehicles that passed in the last 5 seconds.
  - `color` of the light (red, yellow, green).
- We want to check that across all intervals when the light was yellow, no more than 5 vehicles passed.
- The query in our EOL dialect looks like this:

```
tl.when(v | v.color = 'yellow'
            and (v.prev.isUndefined() or v.prev.color <> 'yellow'))
  .always(v | v.unscoped.sinceThen
              .before(v | v.color <> 'yellow' or v.next.isUndefined())
              .versions.collect(v | v.count).sum() <= 5)
```

- `.when(...)` finds the moments when the light becomes yellow.
- `.always(...)` escapes the `when` scope, sets out an interval from then until a color change or the end of the history, and then sums the `count` over all versions in the interval

Other potential types of model inspection

What-if scenarios

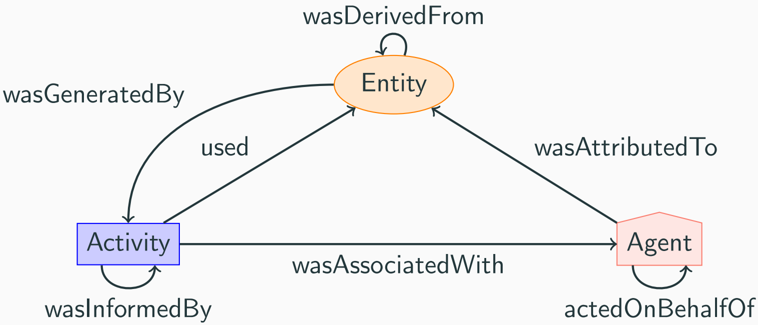
- What if this particular scenario happened now?
- How would our runtime model evolve?
- Conceptually, we have our base timeline, and what-if timelines
- Essentially, this allows for a form of sensitivity analysis

Domain-specific inspections

- Inspection of neural nets has received a lot of attention
- Can we design domain-specific inspections for our runtime models?
- Do we have to design our notations in a specific way to allow for it?

Where next?

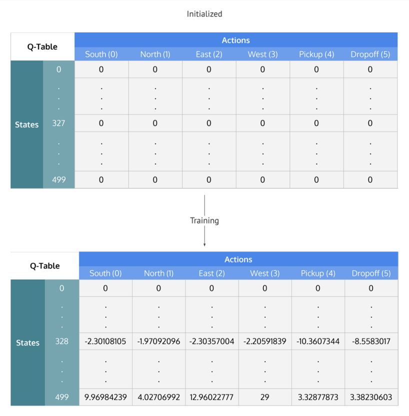
Designing models@run.time systems as transparent boxes



- We have base architectures for self-adaptive systems (e.g. MAPE-K)
- Can use architectures to make our SAS transparent boxes by design
- Example: capturing system evolution with a W3C PROV graph
- This structure is independent from our particular ML algorithm
- *Runtime models as algorithm-agnostic metadata*

Extracting approximation models from ML models

Source — LearnDataSci (CC-BY-SA 4.0)



Initialized

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	327	0	0	0	0	0	0
	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	499	0	0	0	0	0	0

Training

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	328	-2.30108105	-1.97092096	-2.30357004	-2.20591839	-10.3607344	-8.5583017
	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	499	9.96984239	4.02706992	12.96022777	29	3.32877873	3.38230603

- You have an ongoing training process (e.g. Q-learning)
- Can you make sense of the matrix? 500 $\times$ 6 numbers to watch!
- Could extract decision forest that approximates the matrix
- Users can watch how the decision forest changes
- *Runtime models as the interpretable versions of an ML model undergoing training*

Using requirement/process models to inspect ML approaches

ML approaches bring new desirable properties

- We want it to be *fair* (not biased with minority populations)
- We want it to be consistent
- It must preserve privacy, have a certain accuracy, follow a recommended cross-validation process...

Runtime models to supervise ML training process

- Training itself is not just about defining the process, but also defining our requirements on a "successful" trained ML model
- How do we specify "fairness", though?
- How do we check this at runtime, during training?
- We'll have to check the Friday talks on models + ML/AI!

Thank you!

Questions?

@antoniogado / a.garcia-dominguez@aston.ac.uk