VIEWING MACHINE LEARNING THROUGH A SOFTWARE ENGINEERING LENS

Tom Dietterich

Distinguished Professor (Emeritus)
Oregon State University

Oregon State University

Outline

- Defining Correctness
- Runtime Monitoring
- Uncertainty and Rejection
- Design, Modularity, Debugging, Evolution
- System Issues

Defining Correctness

- The training examples define correct input-output behavior at individual points
- Additional ideas:
  - Smoothness
  - Monotonicity with respect to one or more input variables

Invariant Regions

- Define a convex region in the input space
  - e.g., sphere surrounding a test case
- Apply specialized SMT solver to verify that the output of the neural network is constant in this region
  - Ehlers: Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks (2017; arXiv 1705.01320)
- Similar work:
  - Wong & Kolter (2018): Provable defenses against adversarial examples via the convex outer adversarial polytope
  - Katz et al. (2017): Reluplex

Outline

- Defining Correctness
- Runtime Monitoring
- Uncertainty and Rejection
- Design, Modularity, Debugging, Evolution
- System Issues

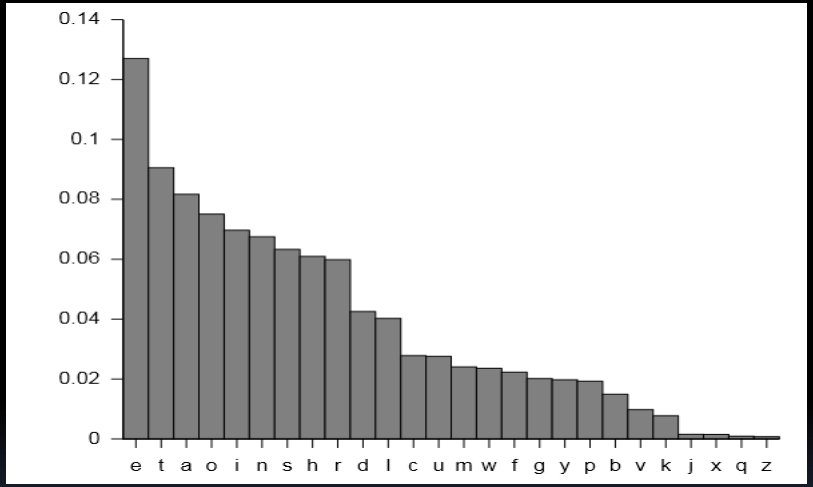
Runtime Monitoring

- Runtime threats to correctness
  - Class frequency shift
  - Covariate shift
  - Decision boundary shift
  - Novel classes

Monitoring Methods

- Monitor distribution of predicted classes
- There are methods for adjusting classifiers to changes in the class distribution

Letter frequencies in English



Credit: Nandhp, Wikipedia

Covariate Shift

- The decision boundaries between classes $P(y|x)$ do not change, but the data distribution $P(x)$ changes
- Methods are available to retrain the classifier on the original training data to compensate

Decision Boundary Shift

- Requires some form of feedback to detect
- Has not received much study

Open Category Recognition

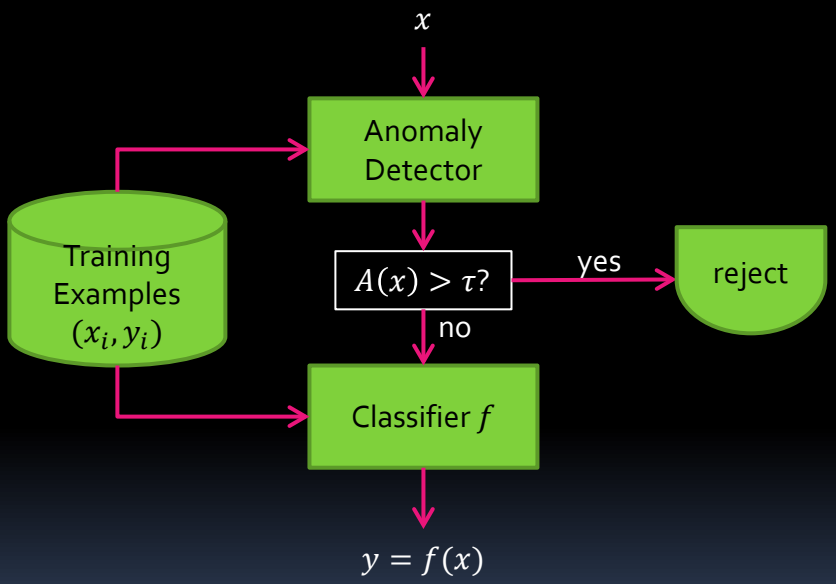
- Train on K categories
- Test queries may belong to novel categories
- Must detect them

BBC

Technology

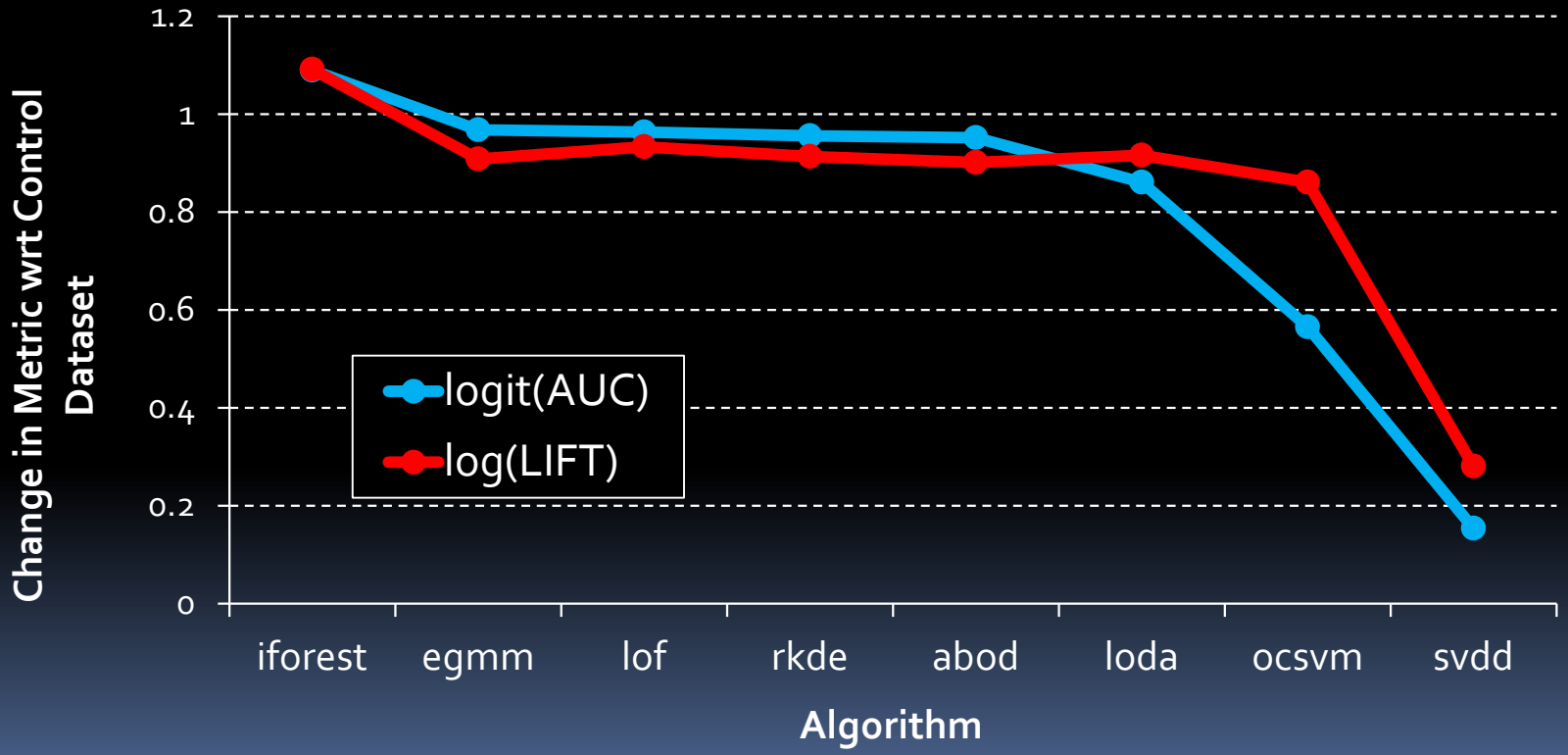
Volvo's driverless cars 'confused' by kangaroos

Anomaly Detection



Liu, S., Garrepalli, R., Dietterich, T. G., Fern, A., & Hendrycks, D. (2018). Open Category Detection with PAC Guarantees. *ICML 2018*.

Algorithm Comparison



Outline

- Defining Correctness
- Runtime Monitoring
- Uncertainty and Rejection
- Design, Modularity, Debugging, Evolution
- System Issues

Uncertainty and Rejection

- Given runtime query x_q and prediction $\hat{y} = f(x_q)$
- What is the probability that $\hat{y}$ is correct?
- If it is too low, raise an exception

- We can't answer this for a single point, but we can provide calibrated probabilities and rejection functions

Calibrated Probabilities

- When the classifier says "$\hat{y}$ is correct with probability 0.8", it is right 80% of the time…
  - on a labeled calibration data set C
- We have good methods for producing calibrated probabilities

Niculescu-Mizil, A., & Caruana, R. (2005). Predicting good probabilities with supervised learning. *Proceedings of the 22nd International Conference on Machine Learning ICML '05*, (2005), 625–632

Rejection Functions

- Given:
  - Training data $(x_1, y_1), \ldots, (x_N, y_N)$
  - Target accuracy level $1 - \epsilon$
  - Learn a classifier f and a rejection rule r
- At run time
  - Given query x_q
  - If $r(x_q) < 0$, REJECT (raise exception)
  - Else classify $f(x_q)$
- Guarantee accuracy of $1 - \epsilon$

Cortes, C., DeSalvo, G., & Mohri, M. (2016). Learning with rejection. *Lecture Notes in Artificial Intelligence*, *9925 LNAI*, 67–82.

Outline

- Defining Correctness
- Runtime Monitoring
- Uncertainty and Rejection
- Design, Modularity, Debugging, Evolution
- System Issues

Machine Learning is Immature

- Until recently ML was a purely academic pursuit
- Researchers focused entirely on accuracy
- We ignored all other engineering considerations
  - maintainability and evolution
  - assurance (testing, debugging)
  - modularity and systems issues

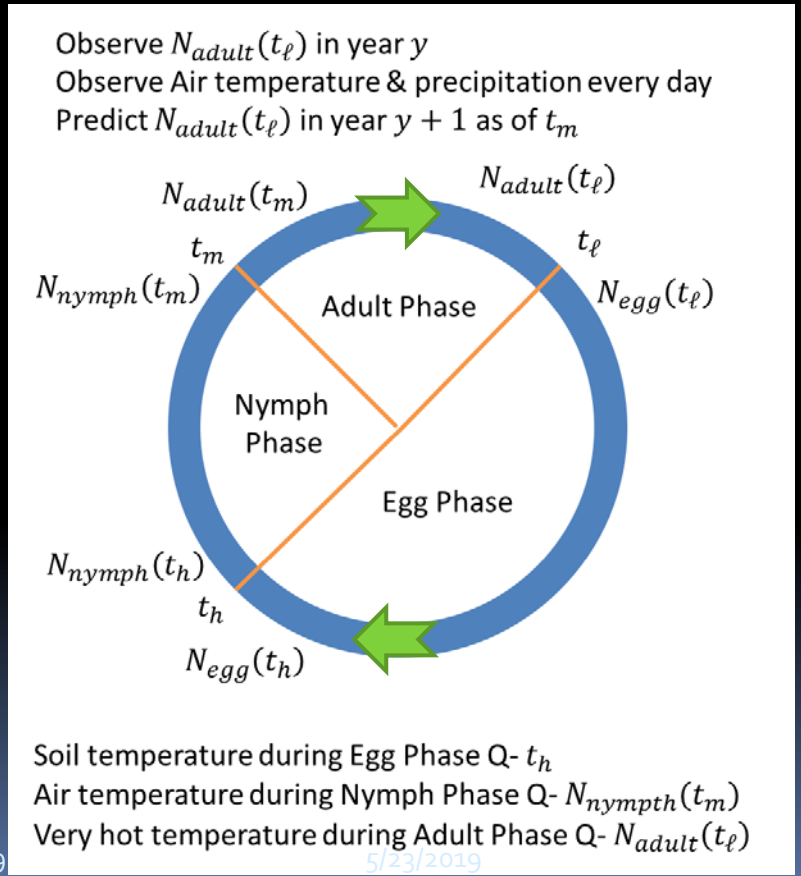
Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., … Young, M. (2014). Machine Learning: The High-Interest Credit Card of Technical Debt. *NIPS 2014*, 1–9.

Design: Supporting Feature Engineering

- Build a domain model
- Annotate with data collection model
- Annotate with prediction goal
- Reason about what is needed to achieve the prediction goal
- Introduce summaries, proxies, and auxiliary data sources as needed

Example: Grasshopper Infestations

- To predict $N_{adult}(t_m)$ need to know $N_{nymph}(t_m)$
- To predict $N_{nymph}(t_m)$ need to know $N_{nymph}(t_h)$ and weather during Nymph phase [observed]
- To know Nymph phase, need to know t_h
- To know t_h need to know soil temperature
- This is unobserved
  - Proxy: Air temperature? [observed]
- To known $N_{egg}(t_h) = N_{egg}(t_\ell)$ need to know $N_{adult}(t_\ell)$ [observed]



Engineered Features

- Air heat degree days (HDD) during Egg Phase
  - air temperature summary; soil temperature proxy
- Egg Phase ends (t_h) when HDD exceeds threshold
  - “maturation pattern”
- Air cold degree days (CDD) during Nymph Phase
  - air temperature summary
- Number of adults previous year
- t_ℓ = September 1
- t_m = June 1

Modularity

- Sculley, et al.: if ML mixes data from multiple sources, it couples those sources and inhibits future change
- Idea: Add regularization term to penalize the number of distinct data sources in an ML model
  - Existing technique: Block Lasso regularization

Evolution With Partial Retraining

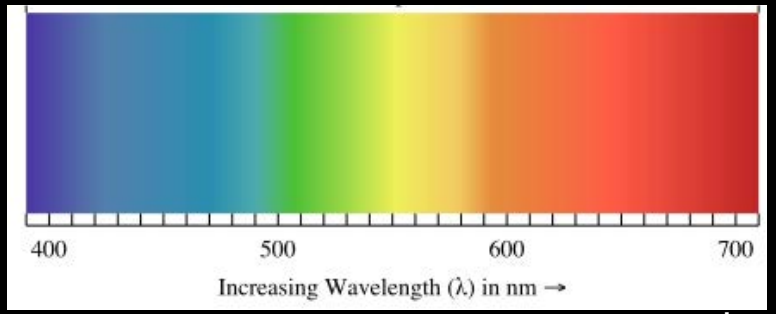
- Semantically-meaningful features should be described (e.g., using a measurement ontology)
- When features are updated or extended, definitions can be compared to determine which features are identical, similar, or unrelated
- This can direct the transfer of learned weights vs. re-learning weights

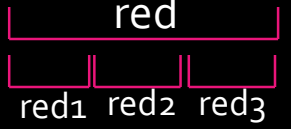
Example: Splitting “red”

- “red”: (625,740) nm
  - Learned weight +1.83

- “red1”: (625-660) nm
- “red2”: (660-700) nm
- “red3”: (700-740) nm
  - Transfer weights of
    - 0.557, 0.634, 0.634
  - Gives the same result
  - Subsequent learning can adjust these weights

Credit: Philip Ronan





Assurance

- Open Question: How can we test and debug ML systems?
  - Often, the programmer lacks the knowledge to judge whether particular program outputs are correct
  - Crowd sourcing?
  - Subject Matter Experts?
  - Labeled assurance data?

There is an Asymmetry in Testing

Traditional Software

- Programmer writes general procedures that focus on common cases
- Testing is based on point tests and fuzzing which reveal corner cases

ML-based Software

- Learning algorithms are trained on points (training examples)
- Maybe testing/verification should focus on (fragments of) general procedures?

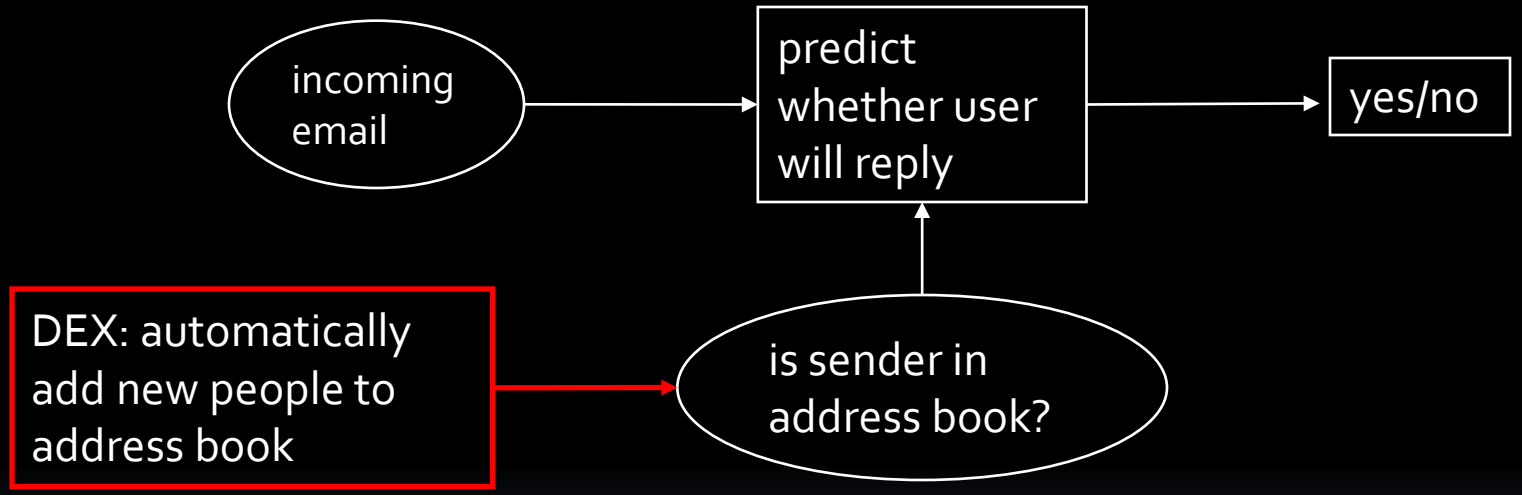
Outline

- Defining Correctness
- Runtime Monitoring
- Uncertainty and Rejection
- Design, Modularity, Debugging, Evolution
- System Issues

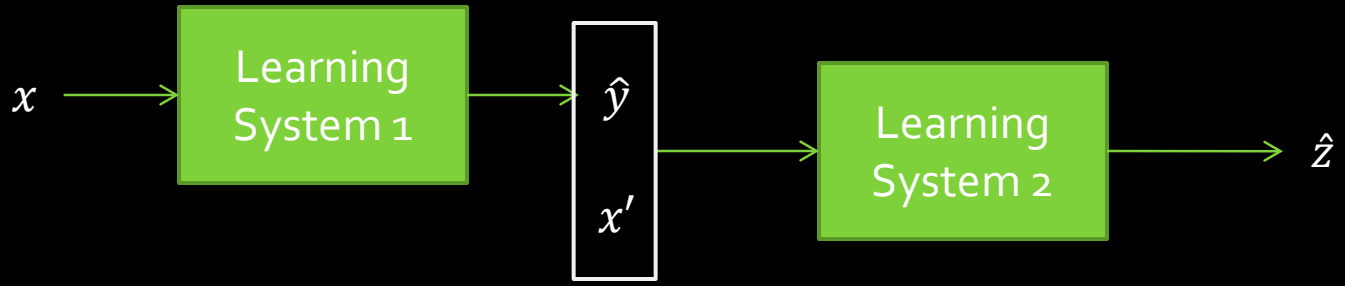
System Issues

- Brittle feature semantics
- Interacting ML components
- Hidden feedback loops

New component makes old feature useless

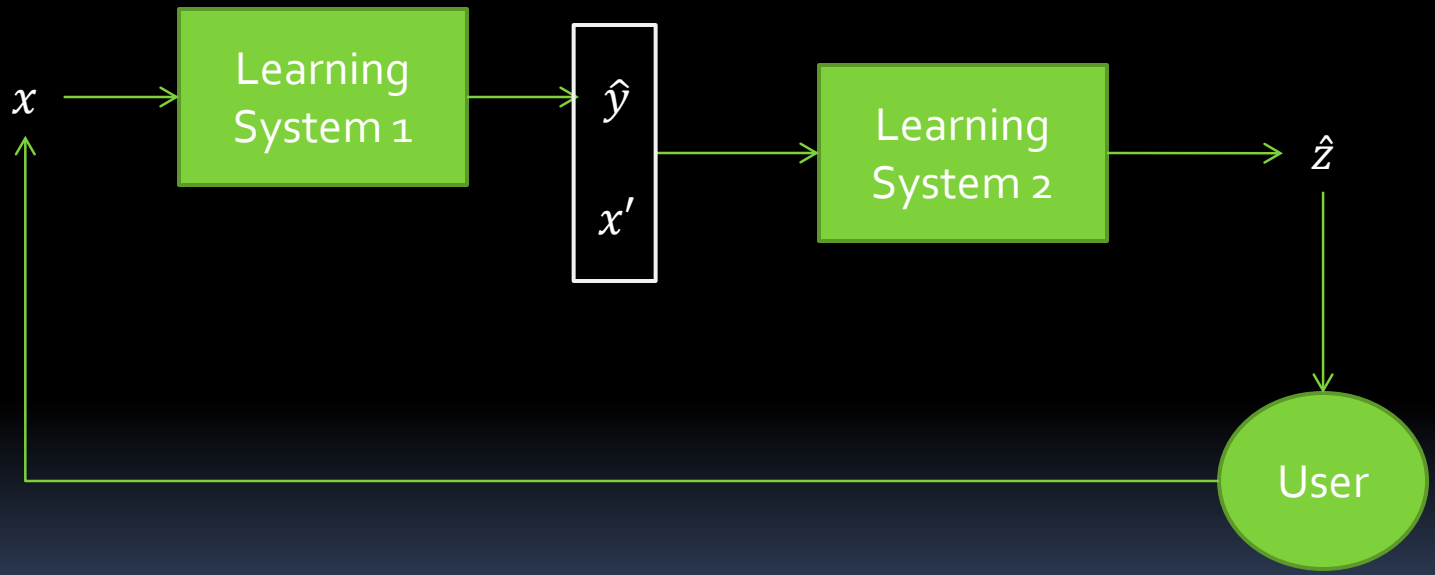


Improvements in LS1 Break LS2



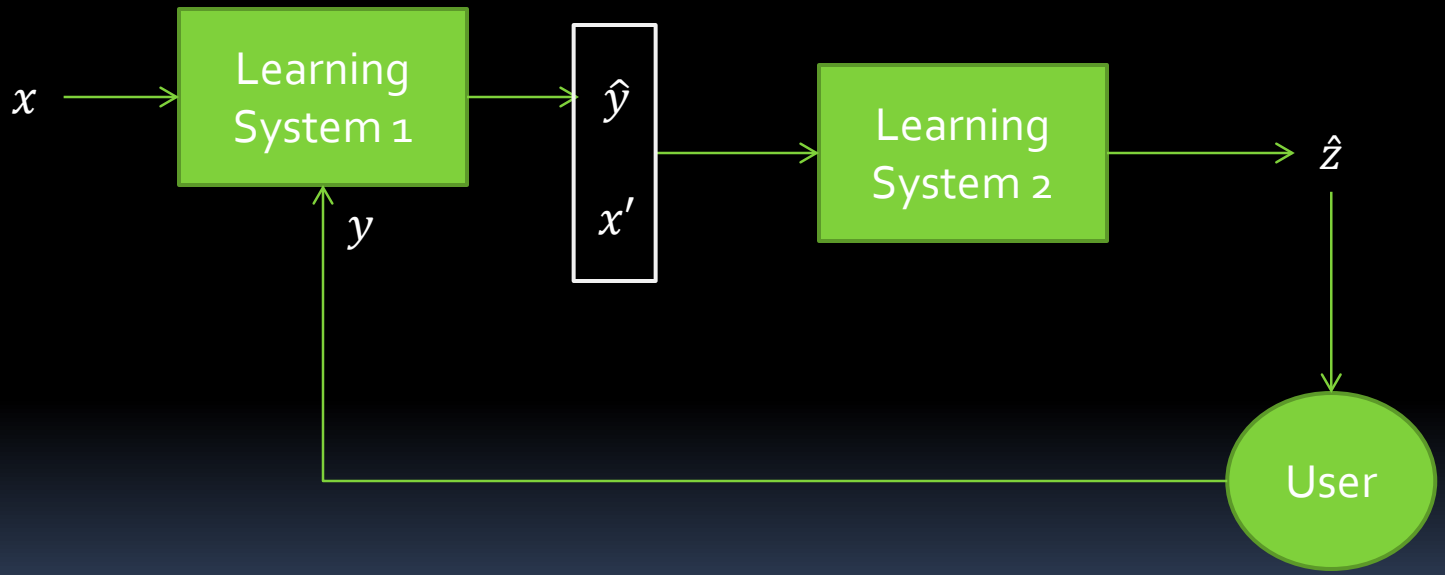
LS1 changes behavior, which violates stationarity assumption of LS2

Hidden Feedback Loops



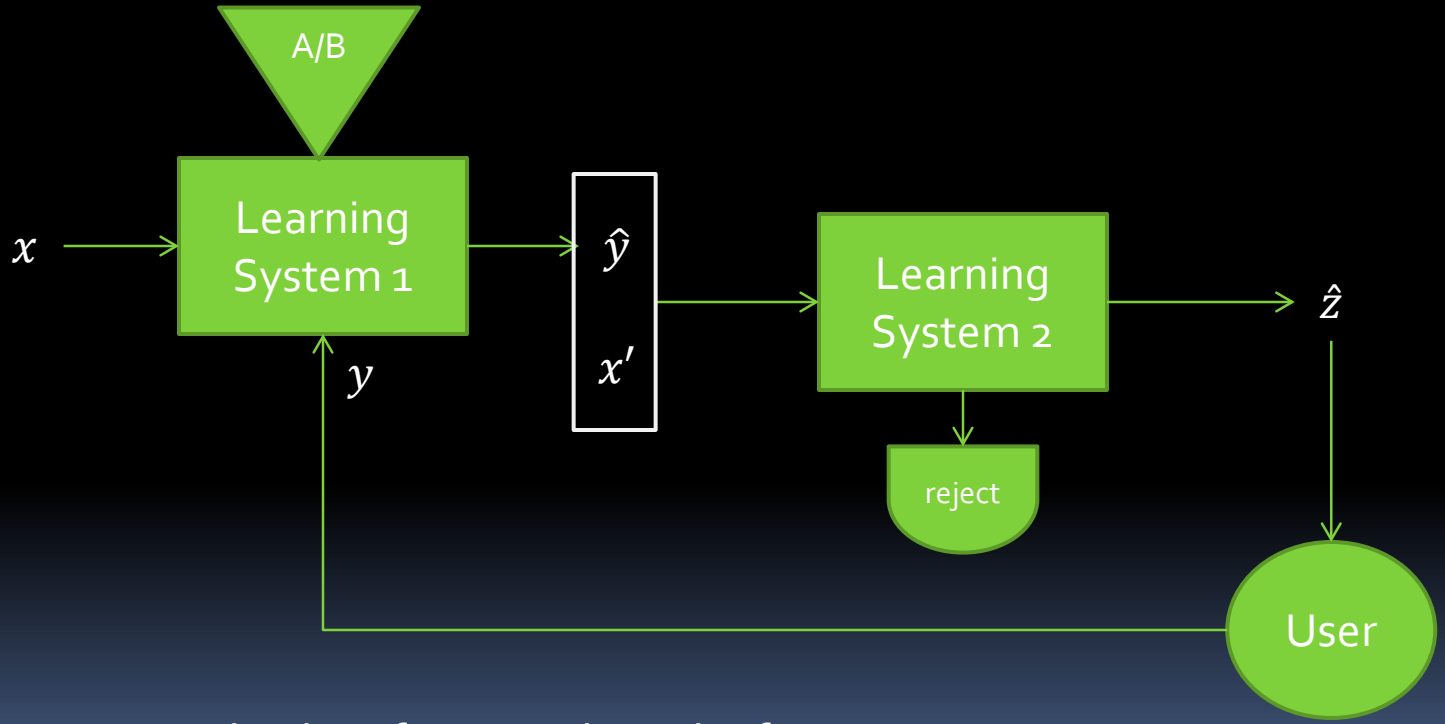
Output of ML system influences its own input (violates iid assumption)

Credit Assignment Problem



User feedback to LS1 is influenced by LS2's behavior

Blocked A/B Test



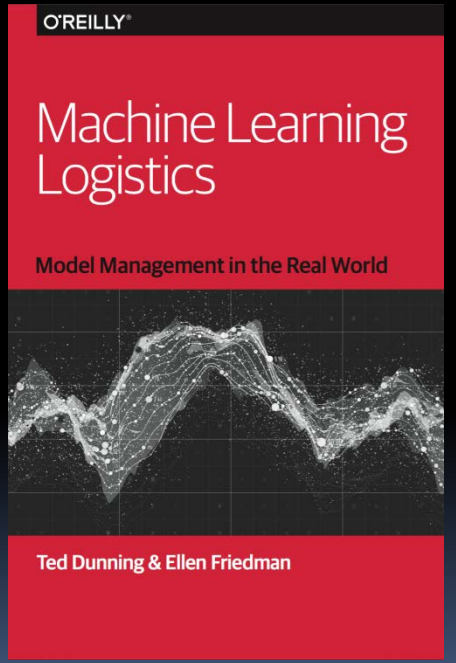
LS2 rejects $\hat{y}$ values from one branch of A/B test

Lesson Learned

- Need global data flow analysis to detect feedback loops
  - including model of user/environment
- Need to coordinate A/B testing
  - testing data needs A/B tag

Model Management

- Cost of ownership of ML systems is very high
- Improved tooling can help reduce this cost



Summary

- Defining Correctness
  - Need ways of expressing desired behavior in addition to training data
- Runtime Monitoring
  - Essential to catch threats to correctness
- Uncertainty and Rejection
  - ML tools for ensuring high accuracy (on average)
- Design, Modularity, Debugging, Evolution
  - Need design methodology
  - Tools for testing, debugging
  - Features should be self-describing to support classifier evolution
- System Issues
  - Interacting subsystems can destroy semantics, create feedback loops

Research Issues

- Online (post-deployment) Learning
  - Deployed Autonomous Testing and Verification?
- Learning from optimization
  - Many of the same issues
  - …and many more