Hidden Technical Debt in Machine Learning Systems

Makhmud Shaban, 171

What does “technical debt” actually mean?

- ML models are easy to deploy but hard to maintain
- Technical debt is a metaphor for underlying codebase maintain costs
- Technical debt can be paid by
    - refactoring code
    - improving unit tests
    - reducing dependencies
    - improving documentation
- ML code has a special capacity for technical debt
- ML technical debt is on a **system** level which makes it hard to detect it

Complex Models Erode Boundaries

- We need to enforce boundaries
- Traditional code - encapsulation and modular design
- ML code - ???
- Main culprit: **ML is used when desired behavior cannot be effectively expressed in software logic without dependency on external data**

Entanglement

- Everything in ML systems is co-dependent
- Example: change distribution of one feature - weights or mportance of others may change as well
- CACE principle: Changing Anything Changes Everything

Solutions:

- Isolate ensembles
- Detect changes in predicting behavior

Correction Cascades

- We have problem A and model M_a
- Slightly different problem A’
- Model M_a’ takes M_a as input and learns a correction
- Cascade of such models -> hard to maintain and detect problems
- Improvement deadlock

Solutions:

- Change model M_a itself
- Learn from scratch

Undeclared Consumers

- Something that relies on our system (for example, reading our logs)
- We don’t know about it
- Possible feedback loops
- Change in API -> death

Solutions:

- Access restriction

Data Dependencies Cost More than Code Dependencies

- *Dependency debt* in traditional code - code dependencies (detectable via static analysis by compilers and linkers)
- *Dependency debt* in traditional code - data (hard to detect)

Unstable Data Dependencies

- Occurs when input features of one system == output features from another
- They can be *unstable*, meaning that they can change over time (i.e distribution)
- Changes in other systems affect our ML system

Solutions:

- Have a *versioned copy* of signal provider code

Underutilized Data Dependencies

- Input signals that does not contribute much to system’s performance
- No improvement, but can make model vulnerable to change
- Hard to transfer code to new domains (we require many obsolete features)
- Examples:
    - Legacy features
    - Bundled features
    - eps-features
    - Correlated features

Solutions:

- Detect via leave-one-out feature evaluation

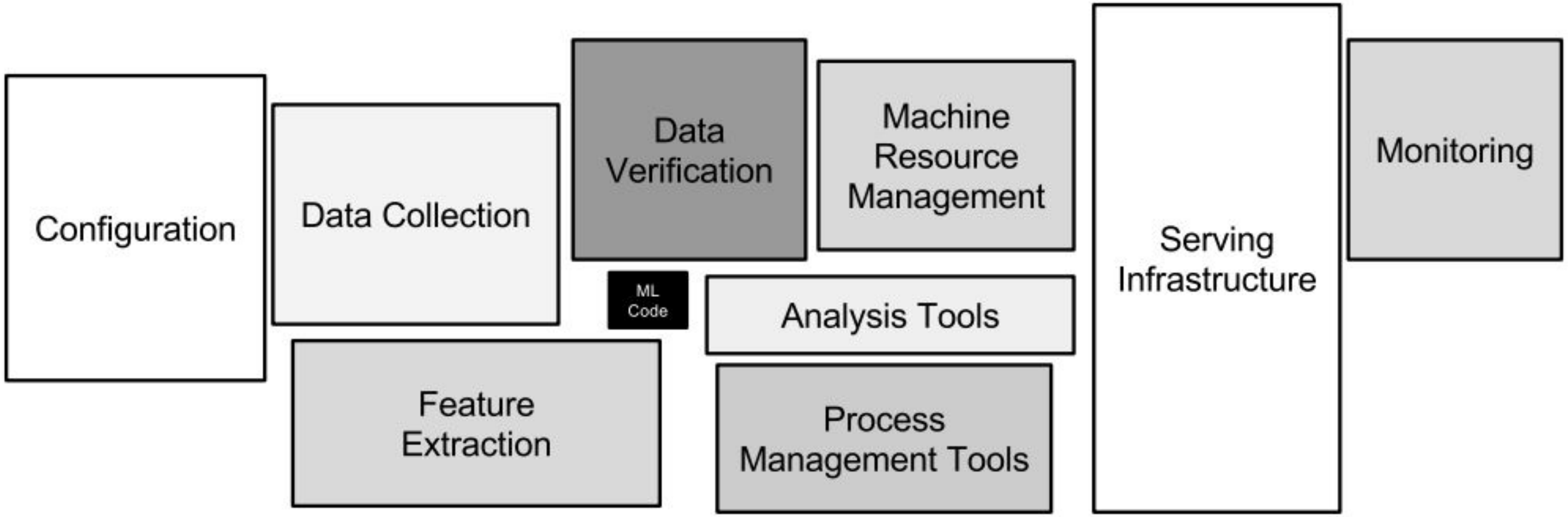
Feedback loops

- Feedback loop - when model output influences its own behavior
- Makes it difficult to predict behavior before release
- Direct loops
- Hidden loops (interaction of two systems)

Example of hidden loop: two recommendation systems (one for products to show, another for reviews) - improvement of one affects clicks on the other, and vice versa

Example of hidden loop on unrelated systems: two trading agents

See the size of ML Code part?



ML-System Anti Patterns

- Glue code (for example, converting to PASCAL VOC)
- Pipeline Jungles (scrape, prepare data, etc…)
- Data Experimental Codepaths (conditional codepaths)
- Abstraction debt (no analogue of good abstractions, such as Map-Reduce)
- Common Smells
  - Data Type Smell
  - Multiple Language smell

Dealing with Changes in External World

- Fixing Thresholds in Dynamic Systems (manually)
- Monitoring and Testing
- Sanity check: distribution of predicted labels == observed labels
- Set action limits on model predictions
- Monitor “data providers”

Other areas

- Data testing debt
- Reproducibility debt
- Process management debt
- Cultural debt (strict division of engineering and research)

Conclusion

- Technical debt is a useful metaphor (:
- There are hidden troubles we should be aware of (like a checklist)
- Monitor and test!

References

- D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-François Crespo, Dan Dennison. Hidden Technical Debt in Machine Learning Systems. NIPS 2015

Вопросы

1. Приведите два пункта, объясняющих, почему сложные модели "стирают границы" и кратко их поясните (два из трех, описанных в плане).
2. Опишите любую из описанных зависимостей от данных и приведите способ борьбы с ней.
3. Опишите любые два антипаттерна в ML системах.