Inextricably Linked: Reproducibility and Productivity in Data Science and AI

Mark Coleman
mark@dotscience.com
@mrmrcoleman

dotscience


MACHINE LEARNING
TODAY IS LIKE SOFTWARE
ENGINEERING IN THE LATE 90'S
CHANGE MY MIND
imgflip.com

dotscience

Who am/was I?

- VP Marketing - dotscience ← Now
- Co-organized first ServerlessConf - Brooklyn
- CNCF Marketing chairperson
- Co-organized first DockerCon EU
- CI/CD consultant
- Devops early fan
- C++ programmer
- C embedded systems ← Then

Let's compare

Data Science/ML/AI

and

Software Dev/DevOps

Not long ago, software dev was a bit of a mess

- Work split across silos
  - Development
  - Testing
  - Operations
- Caused huge amounts of pain

dotscience

90s Software Development

- Without version control, life is hard
  - You email zip files of source code
- Two people change the same files?
  - Your work gets clobbered

dotscience

90s Testing

- "Works on my machine"
- Email, USB stick, or shared drive → separate testing team
- High latency between breakage & knowing
  - Lost valuable context by time to fix
  - A slow & frustrating cycle

90s Operations

- Throw release candidates over the wall to Ops
- They drop a WAR file onto a Tomcat server
- Dev & test failed to account for NFR
  - Ops can't fix it
- Monitoring is sketchy, users find bugs
  - SSH into the production box
  - Process skipped during outage, introduces more bugs
- Everyone is sad

dotscience

How did we ship anything with all this mess?

- Slowly!
- Release cycles are weeks or months
- Bad tooling & process?
  - Choose **SPEED** or **SAFETY** but not both
- Most companies were forced to choose **SAFETY**

What's this have to do with reproducibility?

- Software is iterative
- Try something → figure out what happened → learn → try something else
- How do we figure out what happened?
  - Reproduce all the variables & see what changed
- When bad tooling stops us reproducing an environment development grinds to a halt

dotscience

Things got a lot better in 20 years!

dotscience

The Joel Test

1. Do you use source control?
2. Can you make a build in one step?
3. Do you make daily builds?
4. Do you have a bug database?
5. Do you fix bugs before writing new code?
6. Do you have an up-to-date schedule?
7. Do you have a spec?
8. Do programmers have quiet working conditions?
9. Do you use the best tools money can buy?
10. Do you have testers?
11. Do new candidates write code during their interview?
12. Do you do hallway usability testing?

https://www.joelonsoftware.com/2000/08/09/the-joel-test-12-steps-to-better-code/

dotscience

Destructive vs Constructive Collaboration

- Destructive = making copies
  - No source of truth
  - Divergence occurs instantly
- Constructive = single source of truth
  - Multiple branches, try different ideas
  - Diff & merge enables reconciliation
- Version control enables constructive collaboration

Ubiquitous Version Control

- Sane people use version control
- Developers collaborate effectively
- Testing teams can too
- Even Ops uses version control now – GitOps!

dotscience

Continuous Integration

- Version control enables CI
- CI enables fast feedback
  - React to failures when we can still remember what we changed (minutes not weeks)
- Platform for tested versioned artifacts
  - Deploy into CD pipeline

Continuous Delivery & Observability

- A single traceable way to get a tested change in development to production
- DevOps = ops can collaborate in same way that dev & test teams do with CI
- Application level observability & monitoring allows deep dive into root causes

What has all this achieved?

- --> Version control enabled reproducibility & collaboration
- --> This unlocks Continuous Integration & Continuous Delivery
- --> Add some Observability & Monitoring...
- --> You get ***both SPEED and SAFETY!***

dotscience

How is AI doing in 2018?

- Been talking to dozens of data science & AI teams
- Data science & AI seems to be where software development was in the late 90s :'(

dotscience

In retrospect if we had been able to save the versions or have gone back in time to see how he got his learning rates it would have avoided a lot of questions from the auditors.

Two of the data scientists who worked on that particular model have left and gone to other companies. You want to be able to see what they did and how they did it and not that <u>it's gone when they're gone</u>.

“

One model failed for 3 months and we lost an immeasurable amount of money!

“

After the last audit I was surprised by how many problems in the audit we could have solved by keeping <u>PAPER LOGS</u>. But if we ask our data scientists to do this <u>they will leave</u>!

“

We keep our data scientist teams small and in the same room so they can track their summary statistics by talking to each other and remembering

Destructive collaboration is commonplace

- Shared drives for training data
- Notebooks emailed or slacked between team members
- Scant manual documentation
- Data wrangles go unrecorded

dotscience

Testing of models is rare

- Automated testing of models is rare
- CI systems uncommon
- "Testing" is more often done manually by an individual in an untracked Jupyter environment

dotscience

Deployment is manual

- Models often “thrown over the wall”
- Left in production to rot until somebody notices
- No real monitoring, especially challenging with retraining & model drift
- Haven't seen much continuous delivery

dotscience

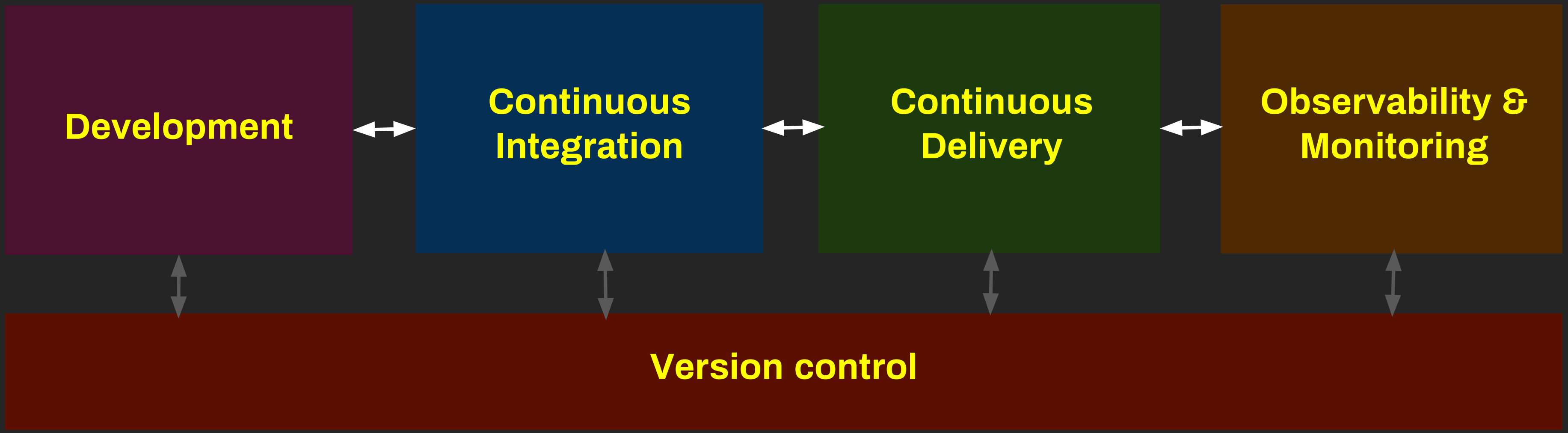
How do we ship anything with all this mess?

- Inappropriate tooling makes us choose between **SPEED** and **SAFETY**
- Therefore
  - AI/ML projects being shipped slowly with meticulous docs
  - AI/ML projects being shipped unsafely
    - not tracked, not auditable
    - no single source of truth for what made it into prod & how
    - siloed in peoples' heads...

dotscience

How do we get AI & ML & etc out of the 90s?

Version control is fundamental & enabling in the lifecycle



How do we version control?

- Versioned data, environment, code: notebooks + parameters
- Metrics tracking: parameters ↔ summary statistics (e.g. accuracy, business metrics)
- Diff & merge for notebooks, data, code
- Forks, pull requests & comment tracking
- Enables:
  - Creativity & collaboration
  - Audit & reporting

How do we continuously integrate?

- What do automated tests look like for models?
  - Not always binary like software – probabilistic
  - Pick some inputs / outputs & put triggers on them
  - If it goes > N stddev, fail tests
  - Also test NFR & unit/integration tests on code
- When issues are reported with a model, convert issues to tests
  - This way, CI provides "guide rails" for faster & more confident development

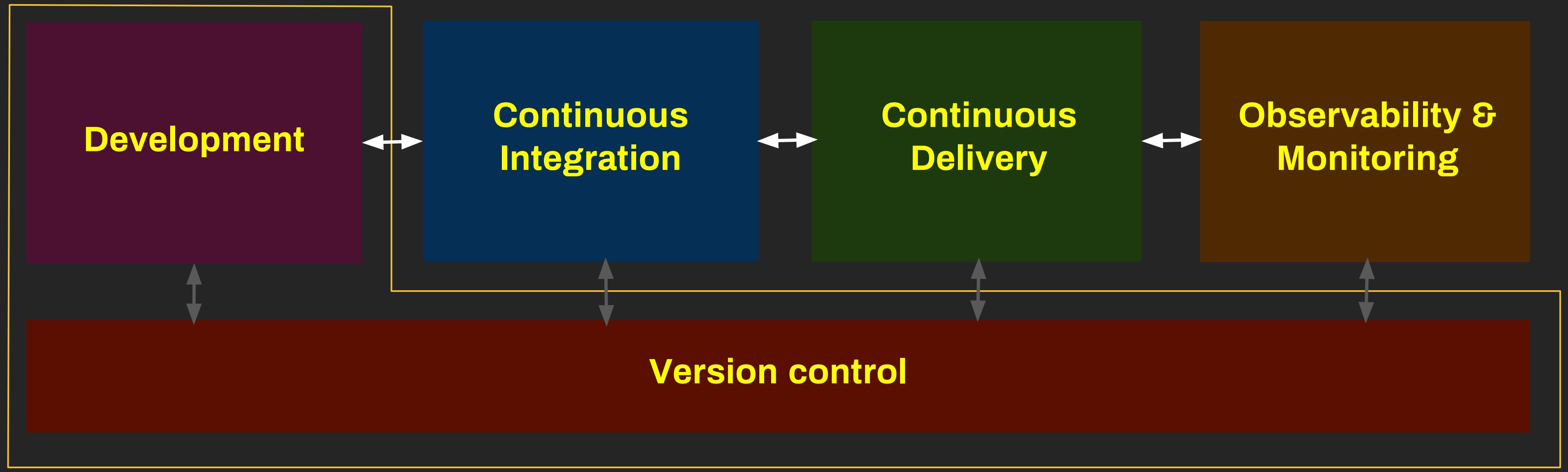
How do we continuously deliver?

- Triggers: when code changes or data changes
- Automatically run code and model tests
- If tests pass, automatically deploy to production
  - Champion Challenger, A/B
  - Minimize time between breakage & knowing
  - Minimize MTTR not MTBF, fast rollback
- From decisions made in production, be able to track back perfectly
  - See lineage of model development right down to individual parameter tweakings - who/what/when/why

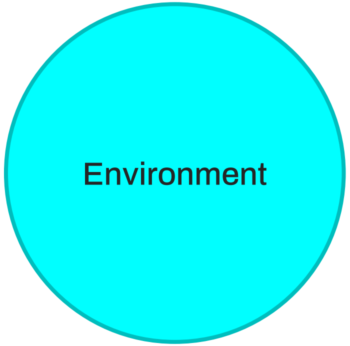
How do we solve observability?

- Once model is in production, track model health with same metrics used in development
  - Single source of truth for dev/prod metrics
    - See model drift
  - If model health < X, page a human
- Automatic retraining can happen periodically when new data is available
- CI & CD gives us confidence to ship quickly

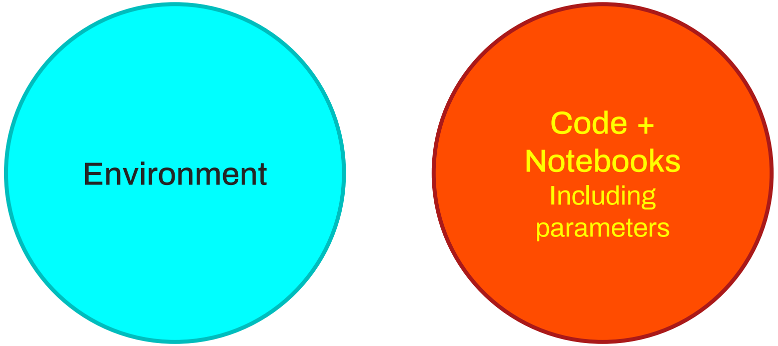
So that's the big vision… where do we start?



So you want to do reproducible data science/AI/ML?



So you want to do reproducible data science/AI/ML?



So you want to do reproducible data science/AI/ML?

Versioned Data

Environment

Code + Notebooks
Including parameters

How?

Pinning down environment

- In the DevOps world, Docker has been a big hit.
- Docker helps you pin down the execution environment that your model training (or other data work) is happening in.
- What is Docker?

Pinning down code & notebooks

- Developers have been version controlling their code for a while now.
- Git is the default choice

Challenges with git in data science

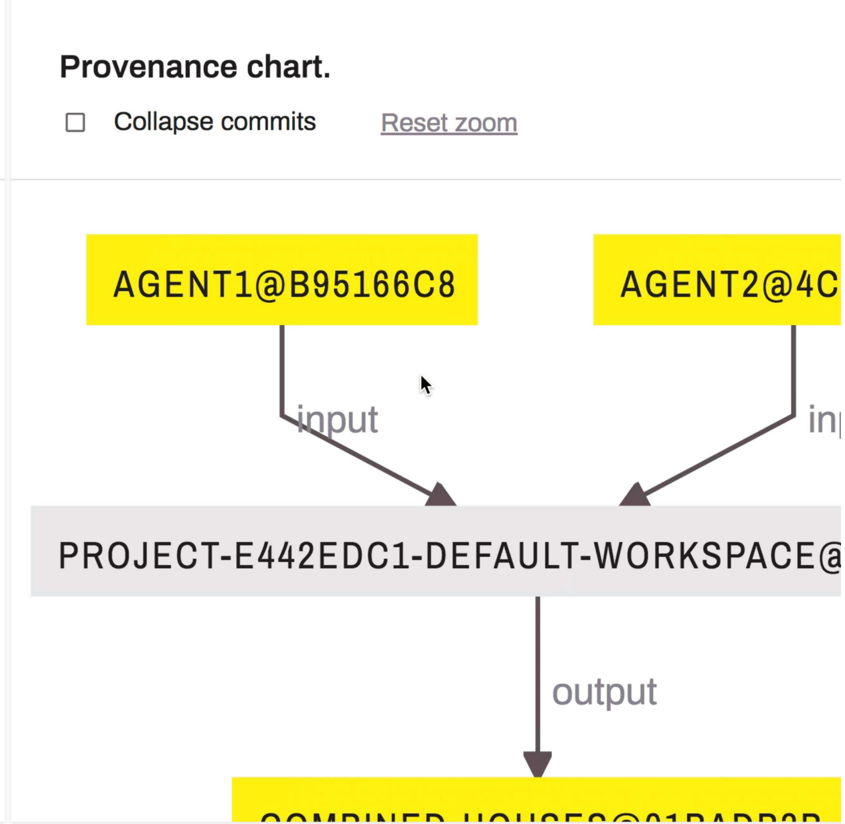
Lets you track versions of your code and collaborate with others by commit, clone, push, pull…

Problems:

- In data science, it's not natural to commit every time you change anything, e.g. while tuning parameters
- But you generate important results while you're iterating
- git doesn't cope with large files, data scientists often mingle code & data
- diffing and merging Jupyter notebooks not easy

Proposal: a new version control & collaboration system for AI

- Use Dotmesh with ZFS
  - "Git for data"
  - Handles large data atomically & efficiently
  - Deal with terabyte workspaces
- Track metrics/stats & params
- Track lineage & provenance
- Next:
  - Diff & merge notebooks
  - Enable pull requests



If you aren’t already dealing with this you most likely will be soon...

I need your help 🙏

mark@dotscience.com

@mrmrcoleman

@getdotmesh

dotscience.com/try

Questions?