# Machine Learning Models and Dataset Versioning

# $ whoami

- Open source contributor DVC, CloudCV,
- FOSSASIA OpenTechNights Winner
- ML intern @Neuroplex
- ML and AI enthusiast
- Final year Compute scinece student @ MEC

# Outline

- Why ML is different from software engineering?
- Problems on using git
- Problems with git-LFS
- Current workflow for ML models management
- Use case as application- Cats vs Dogs model
- Conclusion

# What is ML?

- It provides an systems the ability to automatically learn and improve from experience without being explicitly programmed.
- (supervised, unsupervised, RNN)

# How ML is different?

Data science as different from software

as software was different from hardware.

- Nick Elprin

The rise of Software engineering required inventing new processes like version control, code review, agile, to help teams work effectively. The rise of AI & Machine Learning is now requiring new processes, ..

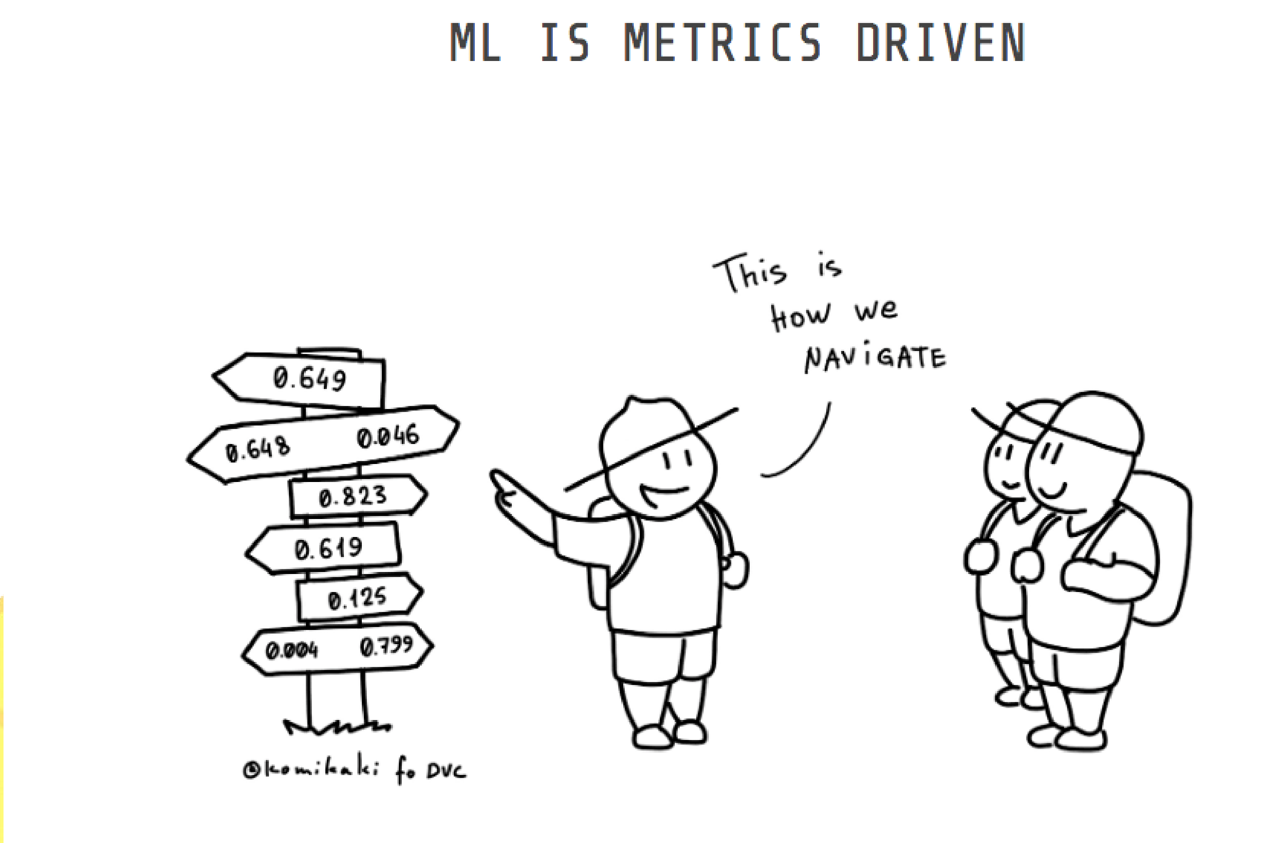
- Andrew Ng

# Key Differences

- ML is metrics driven
- Enormous size of data and models
- Sharing of Models with teams
- Special purpose hardware
- Pipelines
- Difference in best practises

# ML is metric driven

- We focus on improving metrics like accuracy, AUC, RUC.
- Aim to make SOTA models, rapidly experiment



# Feature-driven Development

- While in software engineering, your team works by building new features.
- Use code-review and agile best practises.

# Huge amount of Data dealed

- People work with lots of data for everything.
- Managing datasets are a problem
- Models – a byproduct :P

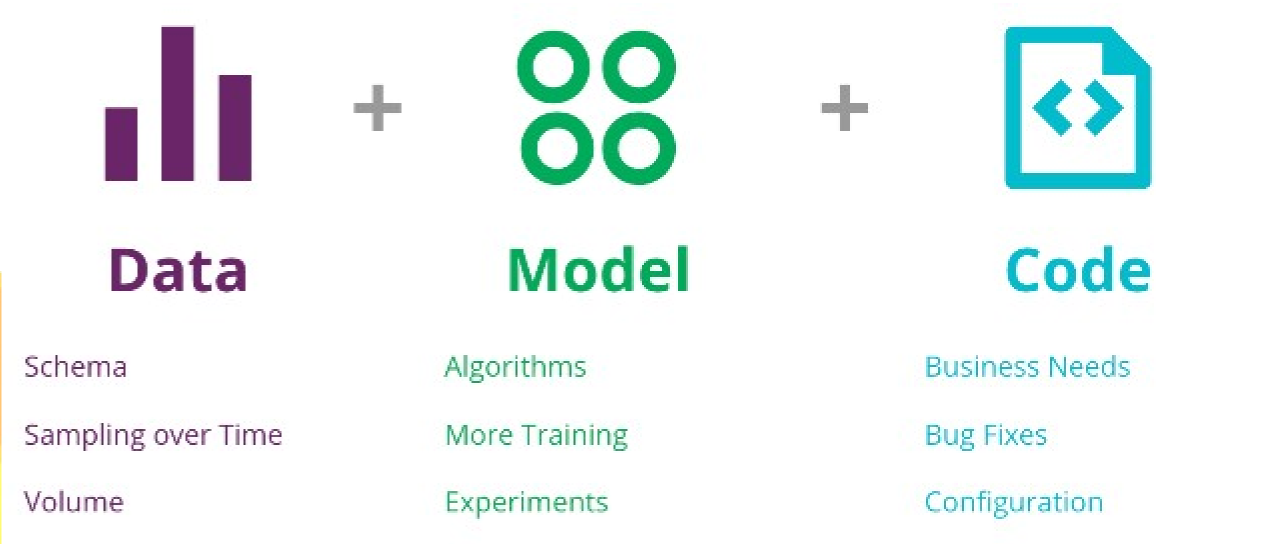
# Huge amount of Data dealed

- People work with TB’s of data for everything.
- Managing datasets are a problem
- amount of data generated for a self-driving car (4 TB/day)

# Ml models need huge resources

- Most software product take a few minutes to run, which team can easily follow.

- Collabrative challenges faced when working with large teams and remotely.



# Pipelines of different stages

- Defining process of steps at each duration like:

  - Input

  - Feature extraction

  - Classification

  - Final output

# Special purpose hardwares

- Use of special TPUs and cloud Computing
- ML researchers have huge computation needs

# Difference in best practises

- Use of Python PEP8 conventions are not always properly practised in jupyter notebooks.
- DataScience = software eng + science&maths

```
from fastai import *
```

# Using git for large dataset

- Trying git for dataset of size of 1GB takes about 8-10 minutes to commit
- .gitignore – ignores Dataset versioning

# Using git-LFS

“Git Large File Storage (LFS) replaces large files such as audio samples, videos, datasets, and graphics with text pointers inside Git, while storing the file contents on a remote server like GitHub.com or GitHub Enterprise.”

# Using git-LFS

- There is a data limit of storing files of 2GB, which is a github limitation.
- Another issue is the ease of placing data files on a cloud storage system (AWS, GCP, etc) as is often required when to run cloud-based AI software.
- Not designed in mind with ML/data science

# Other open source tools

- Data Version Control – Track experiments, data
- ML flow – Metric Tracking, Model Deployment
- Neptune.ml – Experiment Tracking, collabration
- Jovian – Share notebooks, easily reproducible

# Why DVC is a good option?

- It’s open source
- Has support for multiple cloud providers
- Support for:

  a) Pipelines

  b) Versioning ML models

  c) Versioning Datasets

  d) Tracking metrics
- “Made for Data scientist, by the Data scientist”

# Using DVC with a problem

- We are going to work on a Classic Deep Learning problem like cats vs dogs:

  # initialising project

  $ git clone https://github.com/iterative/example-versioning.git

  $ cd example-versioning

# 1. Add the data to dvc

- We use 1000 labelled images of Cats and dogs at first

$ dvc add data → Add datafolder to dvc

$ python train.py

$ dvc add model.h5 → model is add to dvc

# 2. Track changes with git

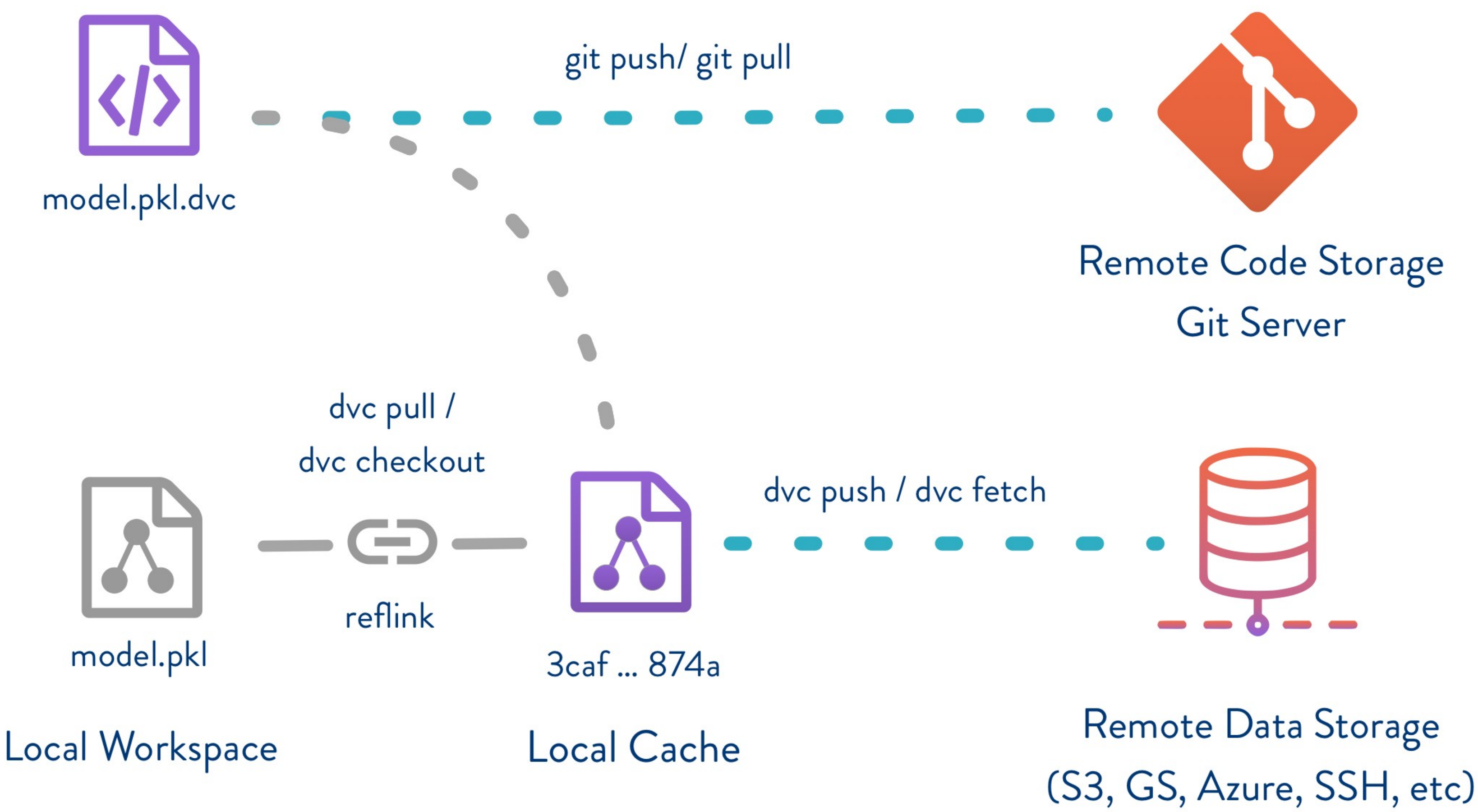
```
$ git add .gitignore model.h5.dvc data.dvc metrics.json

$ git commit -m "First model, trained with 1000 images"

$ git tag -a "v1.0" -m "model v1.0, 1000 images"
```

# 3. Push changes to cloud

$ dvc push

$ dvc pull

- Data is pushed to remote server from local cache.
- DVC is storage agnostic and support s3, Azure, GCP, hdfs, ssh


git push/ git pull
model.pkl.dvc
Remote Code Storage
Git Server
dvc pull /
dvc checkout
dvc push / dvc fetch
reflink
model.pkl
3caf ... 874a
Local Workspace
Local Cache
Remote Data Storage
(S3, GS, Azure, SSH, etc)

# 4. Adding more labelled data

(downloading 1000 more images of labelled cat and dog images)

```
$ dvc add more_data
$ dvc remove model.h5.dvc
$ python train.py
$ dvc add model.h5
```

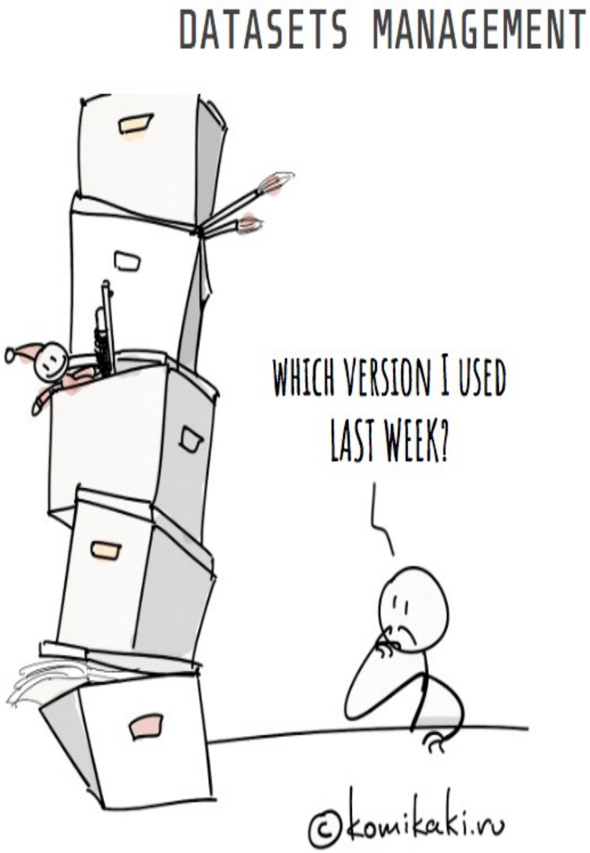
```
$ git add model.h5.dvc data.dvc metrics.json

$ git commit -m "Second model, trained with 2000 images"

$ git tag -a "v2.0" -m "model v2.0, 2000 images"
```

# What is Dataset versioning?

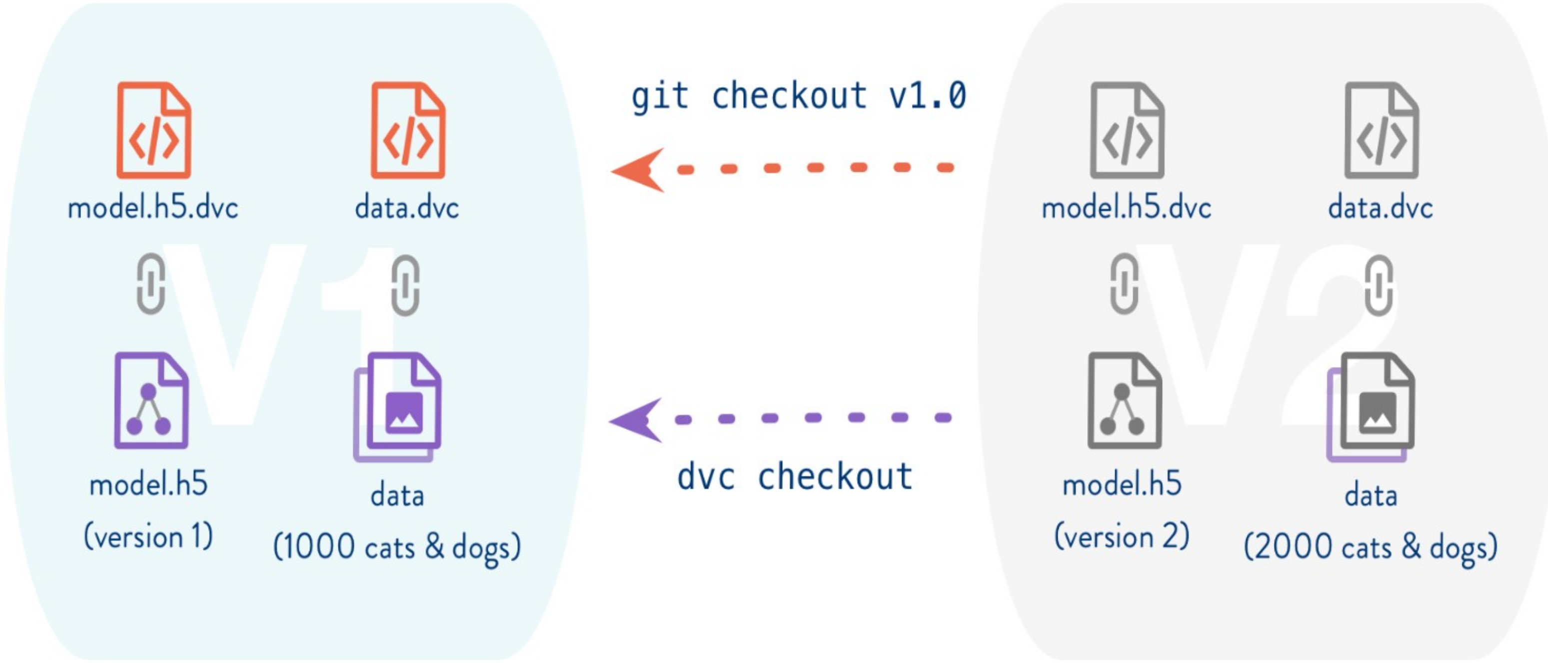
Tracking of data at each time

When working with TB’s of data it’s essential

Different splits for train/test/val



# 5. Switching between versions




$ git checkout v1.0

$ dvc checkout

# 6. Running experiments

$ dvc pull data_folder.dvc

$ dvc run <parameters>

DVC is internally building a dependency graph using dependencies, output and stores the result.

```
dvc run -f modelDvcfile \
        -d train.py -d data \
        -M metrics.json \
        -o model.h5 \
 -o bottleneck_features_train.npy \
        python train.py
```

d for dependency: specify an input file

o for output: specify an output file ignored by git and tracked by dvc

M for metric: specify an output file tracked by git

f for file: specify the name of the dvc file.

# Model versioning

Models = Code + data + hyperparameters

- Everytime our hyperparameters are changed, we can have corresponding DVC files with store the changes
- You can even reproduce the old results

# 7. More Experiments

- I use VGG architecture for version1
- Then resnet for version2
- V2 made much more improv
  Bottle-necks
- Fine tune models

# 8. Iterating over experiments

- $ dvc repro train_v2.dvc
- $ dvc metrics show

# ML Reproducibilty crisis

- Much of ML projects are not reproducible
- Managing the metrics and hyperparameters which changed this module is very important


# Conclusion

# Questions