

22EF...

# Exploring the State of Machine Learning for Biological Data

---

Edmund Miller

July 26th, 2023

# Introduction

---

## About me

- Phd Candidate @ University of Texas at Dallas
- nf-core maintainer

## About me

- Phd Candidate @ University of Texas at Dallas
- nf-core maintainer
- Been Machine Learning curious since around 2017

## About me

- Phd Candidate @ University of Texas at Dallas
- nf-core maintainer
- Been Machine Learning curious since around 2017
- Never took Linear Algebra

# About me

- Phd Candidate @ University of Texas at Dallas
- nf-core maintainer
- Been Machine Learning curious since around 2017
- Never took Linear Algebra
- I had some questions

## About me

- Phd Candidate @ University of Texas at Dallas
- nf-core maintainer
- Been Machine Learning curious since around 2017
- Never took Linear Algebra
- I had some questions
- What are the trade offs of all these packages?

## About me

- Phd Candidate @ University of Texas at Dallas
- nf-core maintainer
- Been Machine Learning curious since around 2017
- Never took Linear Algebra
- I had some questions
- What are the trade offs of all these packages?
- How do we load biological data? (Personal interest in genomics)

# Exploring the State of Machine Learning for Biological Data

*Be the PR you want to see in the repo*

- Michael Lingelbach

# Exploring the State of Machine Learning for Biological Data

*Be the PR Talk you want to see ~~in the repo~~ at Ju-  
liacon*

# *Goal of most Biologists*

- We're not trying to create novel machine-learning models
- We're trying to apply these models in novel ways
- Then make biological inferences

# Old overview

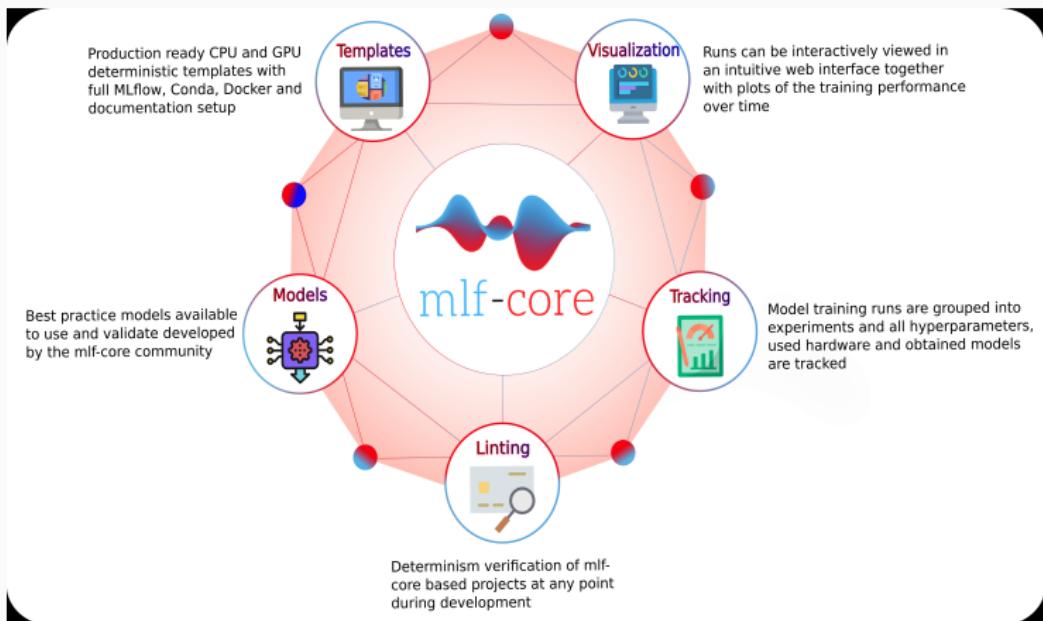
- Ability to call Python and R packages
- What are all of these different ML Packages?
- Loading biological file formats
- Some pretty basic toy examples

Then I had a thought...

# Then I had a thought...

What if I reproduce some analyses and recount what I learned along the way?

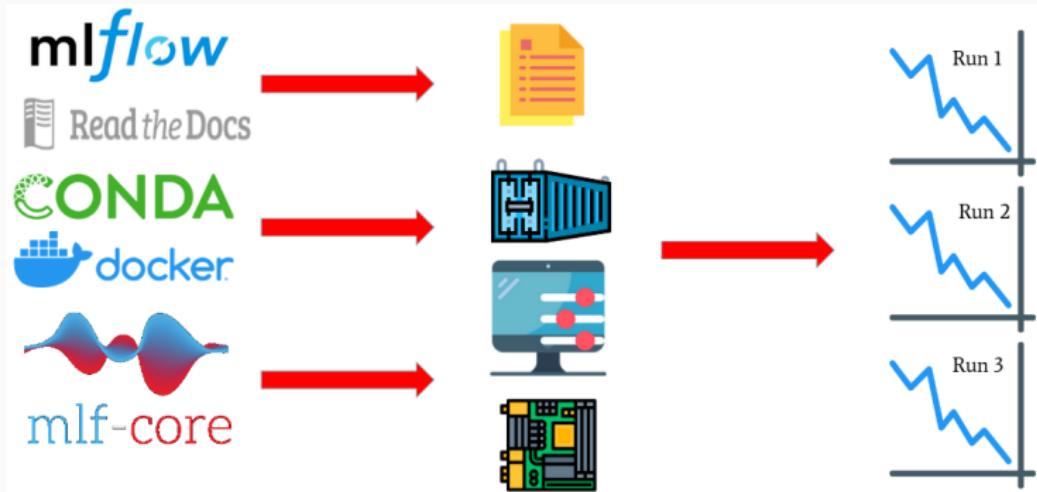
# mlf-core - Overview



1

<sup>1</sup><https://doi.org/10.1093/bioinformatics/btad164>

# mlf-core - Summary



# mlf-core - Concept

```
# Install mlf-core
$ pip install mlf-core

# Get an overview of all commands
$ mlf-core --help

# Check out all available templates
$ mlf-core list

# Get started and create your first project
$ mlf-core create
```

# mlf-core - Why do we care about Reproducibility?

- why does reproducibility matter in science?
  - compared to selling ads, chatbots
- Lives are at stake

lcep

---

# lcep - Overview

- Classifying cancerous liver samples from gene expression data.
- RNA-seq data
- Nice warm up, purpose was to demonstrate [creating a python package using mlf-core](#), and using the package in [a Nextflow pipeline](#)
- [Repo Link](#)

# lcep - Dataloading - mlf-core

```
def load_train_test_data(train_data, test_data):
    X_train, y_train, train_gene_names,
    ↪   train_sample_names =
    ↪   parse_tpm_table(train_data)

    # Convert to Numpy Arrays
    X_train_np = np.array(X_train)

    # Convert from Numpy Arrays to XGBoost Data
    ↪   Matrices
    dtrain = xgb.DMatrix(X_train_np,
    ↪   label=y_train)

    training_data = Dataset(X_train_np, y_train,
    ↪   dtrain, train_gene_names,
    ↪   train_sample_names)
```

# lcep - Dataloading

```
train_url =
    "https://github.com/mlf-core/lcep/raw/master/data/t...
test_url =
    "https://github.com/mlf-core/lcep/raw/master/data/t...

train_data =
    DataFrame(CSV.File(download(train_url)))
test_data =
    DataFrame(CSV.File(download(test_url)))
```

- Note the lack of need to dance around with np arrays and XGBoost Data Matrices
  - The Julia XGBoost wrapper handles the conversion from DataFrames to DMatrix

# lcep - Dataloading

Row	Gene ID	Gene Name	
3	ENSG00000004975	missing	1\bce80114-27b0-4318-9
4	ENSG00000005339	missing	
5	ENSG00000005884	missing	
7	ENSG00000006451	missing	34.9154

# lcep - Data Cleaning - python

- Probably could have used pandas

```
def parse_tpm_table(input):  
    X_train = []  
    y_train = []  
    gene_names = []  
    sample_names = []
```

# lcep - Data Cleaning - python

```
with open(input, "r") as file:  
    all_runs_info =  
        next(file).split("\n")[0].split("\t")[2:]  
    for run_info in all_runs_info:  
        split_info = run_info.split("_")  
        y_train.append(int(split_info[0]))  
        sample_names.append(split_info[1])  
    for line in file:  
        splitted =  
            line.split("\n")[0].split("\t")  
        X_train.append([float(x) for x in  
            splitted[2:]])  
        gene_names.append(splitted[:2])  
  
X_train = [list(i) for i in zip(*X_train)]
```

# lcep - Data Cleaning - Julia

```
function clean_data(input)
    # Drop any rows that are 0s
    input_zeros = input[findall(x -> x != 0,
                                names(input)), :]
    # Drop Gene Name col
    input_id = input_zeros[!, Not(2)]
    # Flip the dataframe
    input_flip = rename(permutedims(input_id,
                                    "Gene ID"), "Gene ID" => :status)

    # The 1_s(cancer) and 0_s(normal) are the
    # labels
    # Split status column by _ and take the first
    transform(input_flip, :status => ByRow(x ->
                                              parse(Float64, split(x, "_")[1])) =>
                                              :status)

end
```

# lcep - Training

```
booster = xgb.train(param, training_data.DM,
                     dict_args['max_epochs'],
                     evals=[(test_data.DM, 'test')],
                     evals_result=results)

test_predictions =
    np.round(booster.predict(test_data.DM))

train = (clean_train_data[:, 2:end],
         clean_train_data.status)
bst = xgboost(train; num_round=1000, param...)

test_predictions = predict(bst, clean_test_data)
```

# GPUs

```
using CUDA
X = cu(randn(1000, 3))
y = randn(1000)

dm = DMatrix(X, y)
XGBoost.isgpu(dm) # true

xgboost((X, y), num_rounds=10) # no need to use
↪ `DMatrix`
```

- [ageron Julia notebooks Template](#)
- Automatically downloads the CUDA toolkit for you

# sc-autoencoder

---

## sc-autoencoder - Overview

- 3000 Peripheral blood mononuclear cells (PBMCs) from 10x Genomics
- This time however they started from h5ad
- However they used [scanpy](#) to load and clean the data

# Dataloading - Attempt to replicate scanpy functionality in Julia

```
using Muon

pbmc3k_url =
    "https://raw.githubusercontent.com/chanzuckerberg/cellxgene/develop/datasets/h5ad/pbmc3k.h5ad"

pbmc3k = readh5ad(download(pbmc3k_url))
```

- Muon is a part of `scverse`, the same group that wrote `scanpy`
- But it threw an error

```
ERROR: MethodError: no method matching
       read_dataframe(::HDF5.Dataset)
```

# Hacking on a package

```
pkg> develop --local Muon
```

- Then there's a repo cloned at dev/Muon!
- And added to the Project.toml for tracking

# Hacking on a package

```
pkg> develop --local Muon
```

- Then there's a repo cloned at dev/Muon!
- And added to the Project.toml for tracking
- You can do this in python but here it's Julia all the way down

# Loading Data using PythonCall - CondaPkg.jl

```
julia> using CondaPkg  
pkg> conda add_channel conda-forge  
pkg> conda add scanpy python-igraph leidenalg
```

## CondaPkg.toml

```
channels = ["conda-forge"]  
  
[deps]  
leidenalg = ""  
scanpy = ""  
python-igraph = ""
```

# PythonCall and Pycall are different

- Doesn't have to support as much legacy
  - PythonCall supports Julia 1.6.1+ and Python 3.7+
  - PyCall supports Julia 0.7+ and Python 2.7+.
- Uses CondaPkg by default
- You can use them both at the same time if you needed to for some reason

# Loading Data using PythonCall

using PythonCall

```
sc = pyimport("scanpy")  
  
function preprocessing(adata)  
    sc.pp.filter_cells(adata, min_genes=200)  
    sc.pp.filter_genes(adata, min_cells=3)  
  
    # Normalization and scaling:  
    sc.pp.normalize_total(adata, target_sum=1e4)  
    sc.pp.log1p(adata)
```

# Loading Data using PythonCall

```
# Identify highly-variable genes
sc.pp.highly_variable_genes(adata,
    ↪ min_mean=0.0125, max_mean=3,
    ↪ min_disp=0.5, subset=true)
sc.pp.scale(adata, zero_center=true,
    ↪ max_value=3)
x = adata.X
# We don't need Tensorflow because Julia is
    ↪ fast enough I think?
# data =
    ↪ tf.data.Dataset.from_tensor_slices((x,
    ↪ x))
x = pyconvert(Array{Float32}, x)

return [x, x], x
end
```

# TensorFlow Dataset Loading

```
data = tf.data.Dataset.from_tensor_slices((x, x))
```

- Realized once again there's no need to learn yet another library, can just use built-in Julia types

# DataToolkit

- Conda doesn't work well on NixOS
- Exported the x Matrix

```
pkg> add DataToolkit
# }
(.)> init
(sc-autoencoder) data> add pbmc3k
↪ https://huggingface.co/datasets/emiller/pbmc3k/reso
```

# DataToolkit

- Conda doesn't work well on NixOS
- Exported the x Matrix

```
pkg> add DataToolkit
# }
(.)> init
(sc-autoencoder) data> add pbmc3k
→ https://huggingface.co/datasets/emiller/pbmc3k/reso
```

- Better practice would be to use the [Julia loader](#)

# DataToolkit

```
[[pbmc3k]]  
uuid = "2c0e014e-eea8-49e2-9916-6ab5d2df08b3"  
description = "Preprocessed pbmc3k Single Cell  
↪ dataset"  
  
[[pbmc3k.storage]]  
driver = "web"  
url =  
↪ "https://huggingface.co/datasets/emiller/pbmc3k"  
  
[[pbmc3k.loader]]  
driver = "delim"  
delim = "\t"  
dtype = "Float32"
```

# DataToolkit

```
using DataToolkit  
  
dataset = d"pbmc3k"
```

# DataToolkit - Want to learn more?

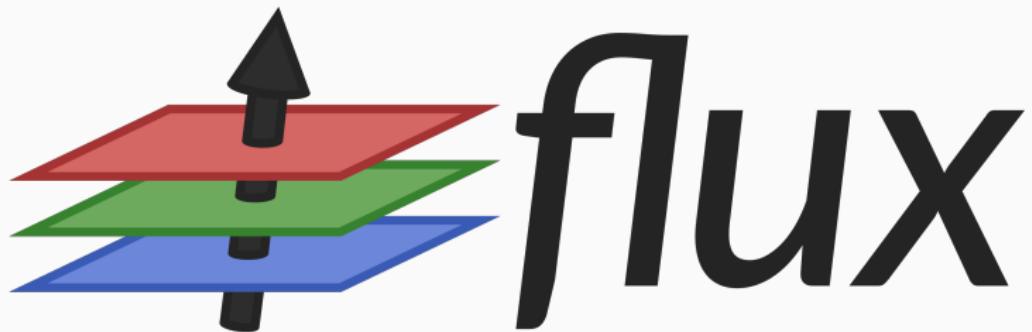


Robust data management made simple: Introducing  
DataToolkit

Timothy Chapman

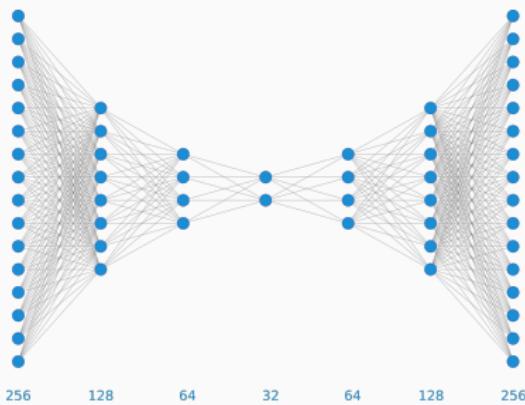
Friday, 07-28, 16:00–16:30 (US/Eastern), 32-123

# Flux



The *Elegant* Machine Learning Stack

# Flux - Model



- Autoencoders are common in scRNA-seq data
  - Denoising of single cell data
  - predict perturbation responses
- mlf-core purpose was to show that **non-deterministic operations** can lead to significant differences in **latent space embeddings**

# Flux - DataLoader

```
train_set = Flux.DataLoader((dataset, dataset),  
    ↵  batchsize=256)
```

# Flux - Experiment Setup

```
device = cpu # where will the calculations be
    ↪ performed?
L1, L2, L3 = 256, 128, 64 # layer dimensions
η = 0.01 # learning rate for ADAM optimization
    ↪ algorithm
batch_size = 100; # batch size for optimization
```

Autoencoders blog post by wildart

# Flux - Build the Model

```
enc1 = Dense(d, L1, relu)
enc2 = Dense(L1, L2, relu)
enc3 = Dense(L2, L3, relu)
dec4 = Dense(L3, L2, relu)
dec5 = Dense(L2, L1, relu)
dec6 = Dense(L1, d)
model = Chain(enc1, enc2, enc3, dec4, dec5, dec6)
↪ |> device
```

Autoencoders blog post by wildart

# Flux - Training

```
opt_state = Flux.setup(Adam(0.001), model)
for data in train_set
    # Unpack this element (for supervised
    # training):
    input, label = data
    # Calculate the gradient of the objective
    # with respect to the parameters within the
    # model:
    loss(A, B) = Flux.mae(model(A),B)
    grads = Flux.gradient(model) do m
        result = m(input)
        loss(result, label)
    end

    Flux.update!(opt_state, model, grads[1])
end
```

# Flux - train!

```
opt_state = Flux.setup(Adam(0.001), model)

loss(A, B) = Flux.mae(model(A),B)

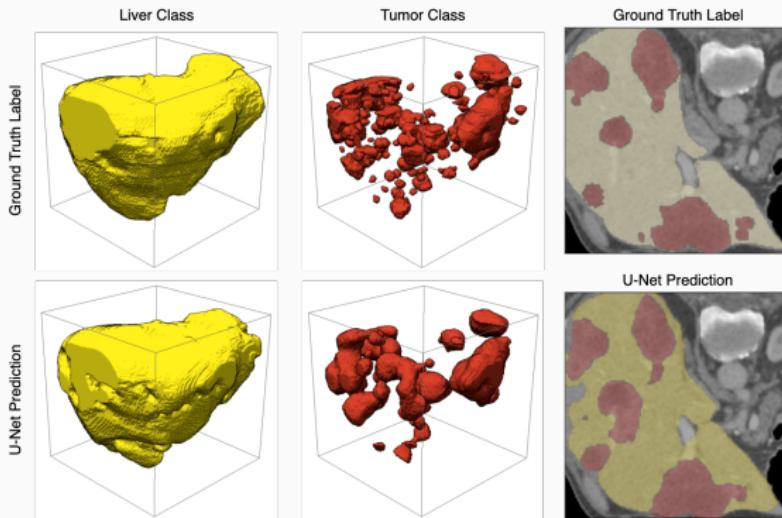
@withprogress Flux.train!(model, train_set,
    opt_state) do m, x, y
    loss(m(x), y)
end
```

# liver-ct-segmentation

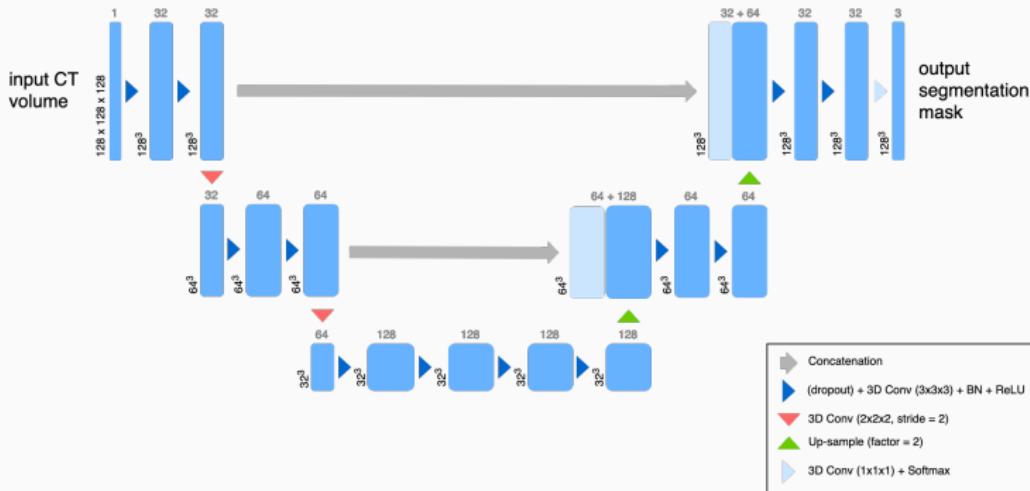
---

# liver-ct-segmentation - Overview

- Liver-tumor segmentation of computed tomography scans using a U-Net model.
- The data in this challenge contains abdomen CT scans with contrast enhancement for liver lesions.



# liver-ct-segmentation - 3D U-Net architecture



# liver-ct-segmentation - Dataset

*The data set for LiTS was collected from 6 medical centres. The CT scans as well as the segmentations are provided as Nifti .nii files.*

- Of course there's a package for that  
[JuliaNeuroscience/NIfTI.jl](#) (It's 9 years old!)

# liver-ct-segmentation - model

- UNet · Julia Packages
  - Written in Flux

```
julia> u = Unet()  
UNet:  
    ConvDown(64, 64)  
    ConvDown(128, 128)  
    ConvDown(256, 256)  
    ConvDown(512, 512)  
    UNetConvBlock(1, 3)  
    ...  
    UNetConvBlock(1024, 1024)  
    UNetUpBlock(1024, 512)  
    ...  
    UNetUpBlock(256, 64)
```

# Conclusion

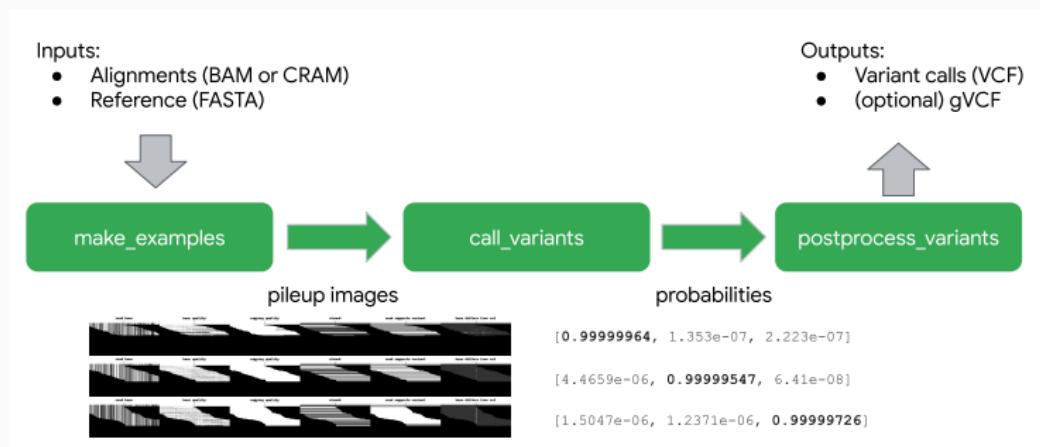
---

# Things we touched upon along the way

- Simplicity of Dataloading
- Hacking on dependencies live
- Calling python packages from Julia
- DataToolkit
- Flux
- There's plenty of packages

# Things I didn't get answers to

- What would loading up BAMs like Deepvariant look like?



# Things I didn't get answers to

- What would FunctionLab/selene look like?
- More Determinism
  - Lux.jl?

# Links

Slides



[link.edmundmiller.dev](http://link.edmundmiller.dev)

