Jet flavor tagging with Deep Learning using Python

2nd Developers@CERN Forum

Marie Lanfermann, Tobias Golling

marie.lanfermann@unige.ch

University of Geneva

31st May 2016

Outline

Introduction

Workflow and Data Handling

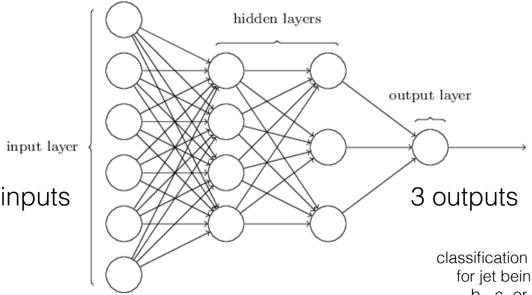
- Preparation
- Training
- Plotting
- Making the NN accessible in any C++ framework

Essential Python code

- Converting data

Approach

Start: MC sample



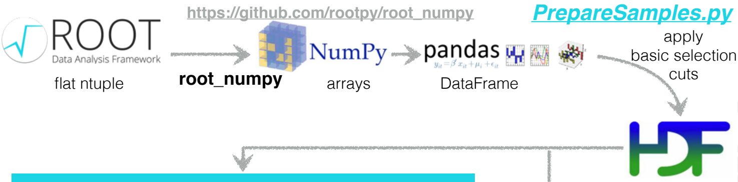
~30 inputs

3 outputs

classification probabilities
for jet being either a
b-, c- or light-jet

End: NN configuration for classification problem

Data preparation - part 1



https://github.com/rootpy/root_numpy

PrepareSamples.py

step	resulting change
determine training and testing set separation (variable via argument)	+ new pandas.DataFrame storing index lists
2D (eta, pT) reweighting (distribution shape via argument)	+ new pandas.Series to add to DataFrame
default variable resetting	+ new pandas.Series check variables & pandas.DataFrame update predefined or calculated on valid values

btag_nn_inputs.py

sample-specific definitions:

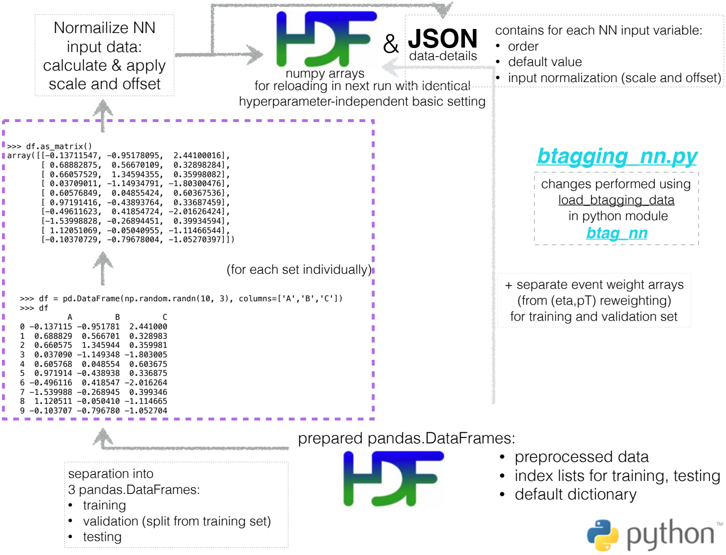
- binning
- input variable details (lists of strings) incl. calculation of new defaults

changes performed using functions defined in python module *btag_nn*

prepared pandas.DataFrames:

- preprocessed data
- index lists for training, testing
- default dictionary

Data preparation - part 2



Normailize NN input data: calculate & apply scale and offset

& JSON data-details

numpy arrays
for reloading in next run with identical hyperparameter-independent basic setting

contains for each NN input variable:

- order
- default value
- input normalization (scale and offset)

```
>>> df.as_matrix()
array([[-0.13711547, -0.95178095,  2.44100016],
       [ 0.68882875,  0.56670109,  0.32898284],
       [ 0.66057529,  1.34594355,  0.35998082],
       [ 0.03709011, -1.14934791, -1.80300476],
       [ 0.60576849,  0.04855424,  0.60367536],
       [ 0.97191416, -0.43893764,  0.33687459],
       [-0.49611623,  0.41854724, -2.01626424],
       [-1.53998828, -0.26894451,  0.39934594],
       [ 1.12051069, -0.05040955, -1.11466544],
       [-0.10370729, -0.79678004, -1.05270397]])
```

(for each set individually)

```
>>> df = pd.DataFrame(np.random.randn(10, 3), columns=['A','B','C'])
>>> df
          A         B         C
0 -0.137115 -0.951781  2.441000
1  0.688829  0.566701  0.328983
2  0.660575  1.345944  0.359981
3  0.037090 -1.149348 -1.803005
4  0.605768  0.048554  0.603675
5  0.971914 -0.438938  0.336875
6 -0.496116  0.418547 -2.016264
7 -1.539988 -0.268945  0.399346
8  1.120511 -0.050410 -1.114665
9 -0.103707 -0.796780 -1.052704
```

btagging_nn.py

changes performed using load_btagging_data in python module **btag_nn**

+ separate event weight arrays (from (eta,pT) reweighting) for training and validation set

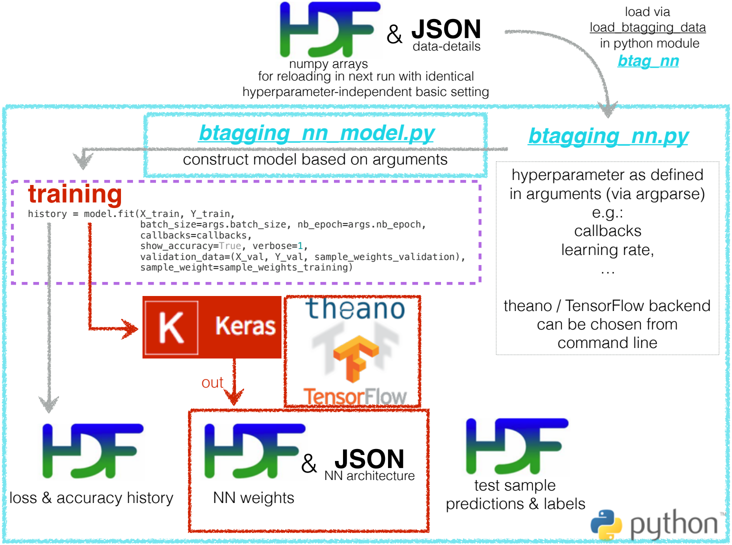
prepared pandas.DataFrames:

- preprocessed data
- index lists for training, testing
- default dictionary

separation into 3 pandas.DataFrames:

- training
- validation (split from training set)
- testing

Training



& JSON
data-details

numpy arrays
for reloading in next run with identical
hyperparameter-independent basic setting

load via
load_btagging_data
in python module
btag_nn

btagging_nn_model.py

construct model based on arguments

btagging_nn.py

hyperparameter as defined
in arguments (via argparse)
e.g.:
callbacks
learning rate,
…

theano / TensorFlow backend
can be chosen from
command line

training

```
history = model.fit(X_train, Y_train,
                    batch_size=args.batch_size, nb_epoch=args.nb_epoch,
                    callbacks=callbacks,
                    show_accuracy=True, verbose=1,
                    validation_data=(X_val, Y_val, sample_weights_validation),
                    sample_weight=sample_weights_training)
```

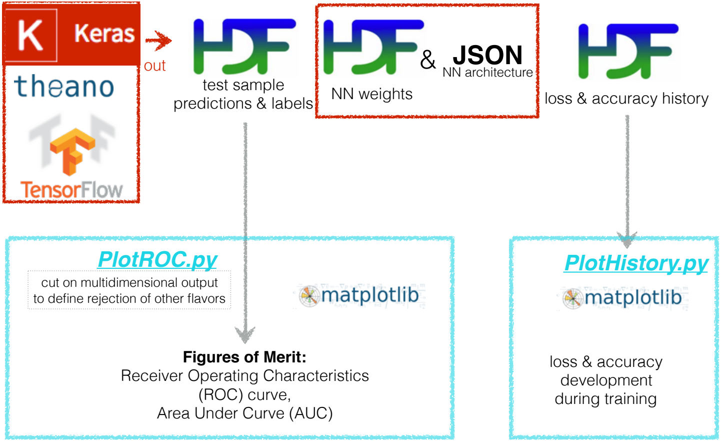
loss & accuracy history

NN weights

& JSON
NN architecture

test sample
predictions & labels

Plotting



test sample predictions & labels

NN weights & JSON NN architecture

loss & accuracy history

PlotROC.py

cut on multidimensional output to define rejection of other flavors

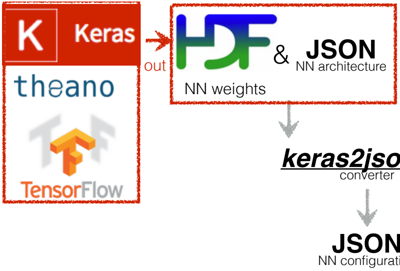
Figures of Merit:
Receiver Operating Characteristics (ROC) curve,
Area Under Curve (AUC)

PlotHistory.py

loss & accuracy development during training

Select final training

Making the NN accessible in any C++ framework



NN weights & **JSON** NN architecture

JSON data-details

contains for each NN input variable:

- order
- default value
- input normalization (scale and offset)

keras2json.py converter

JSON NN configuration

C++ lightweight classes to apply trained NN in any C++ framework

https://github.com/dguest/lwtnn.git

(mainly developed by Dan Guest, UC Irvine)

Converting flat ROOT ntuple into pandas.DataFrame

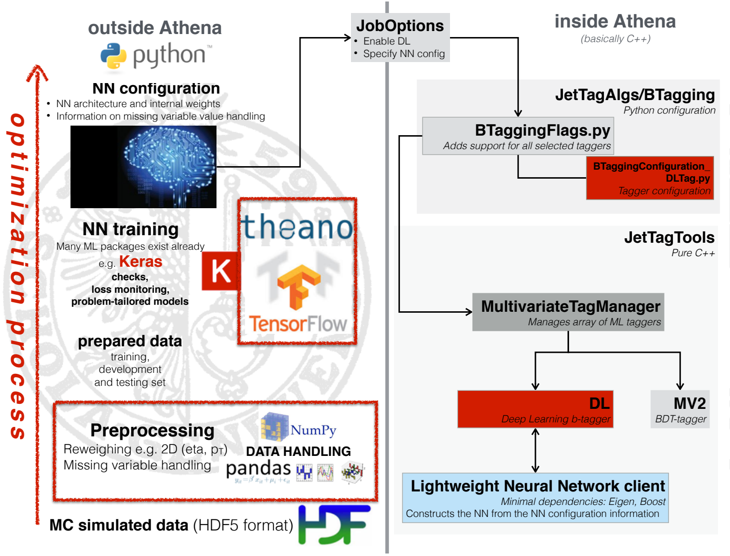
```
import pandas as pd
from numpy.lib.recfunctions import stack_arrays
from root_numpy import root2rec

df = pd.DataFrame(stack_arrays([root2rec(inFile, TTree_name)]))
```

https://github.com/Marie89/BTagging_DL1.git

Backup

How we do it in Athena (ATLAS C++ framework)



outside Athena

python

NN configuration

- NN architecture and internal weights
- Information on missing variable value handling

NN training

Many ML packages exist already

e.g. **Keras**

checks,
loss monitoring,
problem-tailored models

theano

TensorFlow

prepared data

training,
development
and testing set

Preprocessing

Reweighing e.g. 2D (eta, p_T)
Missing variable handling

NumPy

DATA HANDLING

pandas

MC simulated data (HDF5 format)

optimization process

JobOptions

- Enable DL
- Specify NN config

inside Athena

(basically C++)

JetTagAlgs/BTagging

Python configuration

BTaggingFlags.py

Adds support for all selected taggers

BTaggingConfiguration_DLTag.py

Tagger configuration

JetTagTools

Pure C++

MultivariateTagManager

Manages array of ML taggers

DL

Deep Learning b-tagger

MV2

BDT-tagger

Lightweight Neural Network client

Minimal dependencies: Eigen, Boost

Constructs the NN from the NN configuration information