IPPP/16/72

HepStore: The Awakening

Peter Schichtel[1]

[1] *Institute for Particle Physics Phenomenology, Durham University, UK**

A framework to trade (hence the name) your phenomelogy analysis in a preserved way.

CONTENTS

I. Installation 3

II. The HepStore – Framework 3
A. hepstore.docker 3
1. Example 5
B. hepstore.plot 5
1. Data Format 5
2. Example 6
C. hepstore.school 6
1. Classifiers 6
2. Tuning 6
3. Training 6
4. Testing 6
5. Example 6
D. hepstore.analysis 6
1. Example 7

III. hepstore-eas: produce and anlyse Extended Air Showers the HepStore way 7
A. List 7
B. Generation 7
1. Nucleon Modle 7
C. Shower 7
D. Observables 7
E. Example 8
1. A User Analysis 8
2. Comparing Herwig 7 di-jet production with internal Corsika QCD model 8

IV. Contributions and Requests 8

V. Conclusions 8

A. The hepstore.docker family 8
1. hepstore-herwig 8
a. Code 8
b. Dockerfile 9

* peter.schichtel@durham.ac.uk

2. hepstore-corsika 9
   a. Code 9
   b. Dockerfile 9
3. hepstore-hepmc2corsika 9
   a. Code 9
   b. Dockerfile 9

I. INSTALLATION

The `HepStore` – Framework is written solely in `Python 2.7`. To install the package you need 'pip' [].

pip install (--upgrade) (--user) git+https://github.com/PeterSchichtel/hepstore.git

The arguments in parenthesis are optional. Furthermore, if you want to make use of the **Monte Carlo reproducability features** in `HepStore` you need a working Docker [] installation.

II. THE HEPSTORE – FRAMEWORK

In the following we will outly the modules hepstore is composed of. We will introduce their usage and motivation.

A. hepstore.docker

There has been tremondous effort in the experimantal community to achieve high scientific standards of reproducibility. Namely the ReCast and Reana frame works. However, one should not think that no such efforts have been undertaken in the phenomelogical community. There is MadAnalysis5 which has the ability to recast LHC analyses. There is, of course, Rivet, which comes along with verified phenomenolgical and experimental analyses. These tools provide building blocks to perform, save and redo an analysis. In addition the MC community develops ever simpler installation routines for their code packages, including interfaces to third party software. So, why bother with recasting respectively reproducing phenomenoligical results? Anybody who has ever tried to redo what they did five years ago might already know the answer: library missing, code missing, data missing, does not compile anymore, does not run anymore, 32-bit what?, not supported, what was the random seed again, which version of package X was used, etc. pp. So part of this work will be to show a recipe to solve this problem (once and for all, hopefully).

The short answer is Docker. For those who know Docker and are ready to critisize this solution and those who do not know Docker a more lengthy argument follows. Docker is a program which allows to containerize applications. This means once a Docker container is produced, the whole state of the system is fixed. This imidiately clears all of the points raised above. One cannot loose code or a library anymore. It's all in the docker image. The docker image itself can be stored locally, of course, as well as on the docker webpage. There it can be made accessible. This means anybody can reproduce your analysis in the exact state it was developed and used. However, this work wants to be more than just an advertisement for Docker. There is an other issue with MC production and analysis, which has to be considered for proper reproducibility. That is the steering and interaction of the analysis steps. This is what hepstore.docker provides. First there is an easy to use interface to any docker image, which can be used from within `Python 2.7`.

```
# module: hepstore.docker.interface

# imports for this module
import docker
import os
```

```
# the actual interface class
class DockerIF(object):

    # cosntructor
    def __init__( self,
                  image="GENERATOR",
                  version="TAG",
                  verbose=False ):
        # save arguments
        self.IMAGE  = image
        self.TAG    = version
        self.name   = "%s:%s" % (self.IMAGE,self.TAG)
        self.verbose = verbose
        # connect to docker
        self.client = docker.from_env()
        # check if docker image exist otherwise try to pull
        if not len(self.client.images.list(name=self.IMAGE))>0:
            print "--%s: pull image" % self.name
            self.client.images.pull(self.IMAGE,tag=self.TAG)
            pass
        pass

    # actual command run interface
    def run(self,args,directory=None,verbose=False):
        # construct docker container name
        containername = "%s:%s" % (self.IMAGE,self.TAG)
        # see if there is a directory to be mounted from host machine
        try:
            folder   = os.path.realpath(directory)
            volume   = {folder: {'bind': '/UserVolume', 'mode': 'rw'}}
            pass
        except Exception:
            volume   = {}
            pass
        # run the command on the container
        container = self.client.containers.run( containername,
                                                command=args,
                                                volumes=volume,
                                                working_dir='/UserVolume',
                                                detach=True )
        # print the container output on screen
        for i,line in enumerate(container.logs( stdout=True, stderr=True, stream=True)):
            if self.verbose:
                print "%i: %s" % (i,line.strip())
                pass
            pass
        # remove container
        container.remove()
        pass

    pass # DockerIF
```

1. Example

As an example, we construct our own Herwig 7 interface the code can be found in App. ... This code can be called with 'hepstore-herwig' and will behave identical to the 'Herwig' command once Herwig7 would be installed on your machine. Let's say that for any reason you need another version of Herwig. As you can see from the code fragment the docker image and the docker file can be found on the docker webpage under user 'peterschichtel/herwig:7.0.4'. you can use the dockerfile found there and generate your own version of Herwig, say 'yourname/herwig:7.1.0', by altering it correspondingly .Now you can produce MC events with the interface given here: 'hepstore-herwig –repository yourname –generator-version 7.1.0 build yourRunCard.in'. So far so good. Now you'd like to use the 'rivet-mkhtml' command to plot these things you computed. However, there is no special hepstore-rivet-mkhtml you could use. Note that rivet here is a pseudonyme for any command that is not included in the hepstore.docker family which you might need. In this sepcific case the solution is very simple. Copy the above code into a new file rivet-mkhtml.py. In the last command change 'Herwig' to 'rivet-mkhtml' (this is possible because the herwig installation comes along with a rivet installation). use chmod to make the python file executable and just run './rivet-mkhtml.py –repository yourname –generator-version 7.1.0 yourListOfRivetCommandsAndFilePathes' as usual, done. If you think that this is just a more complicated way of running your analysis, don't be mistaken. Something happend on the way: you produced a dockerfile as well as a docker image. This preserves your whole internal state of your code and analysis! Your full analysis has become reproducible for the future by providing two files as supplimential material in your publication: your self written 'rivet-mkhtml' interface and the dockerfile. Even better, if you are up to it you can contribute to hepstore by adding your rivet interface to the hepstore.docker family and upload your docker image to the docker webpage. Your complete production and analysis state is now preserved and interfaced for future use, by publishing the your herwig run card and the following bash snippet:

```
#!/usr/bin/env bash
pip install git+https://github.com/PeterSchichtel/hepstore.git
hepstore-herwig --repository yourname --generator-version 7.1.0 read yourRunCard.in
hepstore-herwig --repository yourname --generator-version 7.1.0 run yourRunCard.run
hepstore-rivet-mkhtml --repository yourname --generator-version 7.1.0
    yourListOfRivetCommandsAndFilePathes
```

This example is where `HepStore` takes its name from. You can add and publish your own product for other's to be used when they want to build on or reproduce your research. Furthermore, it i sthe authors hope that the community might provide official docker images to be used as well.

B. hepstore.plot

In the case where you are not able to use the rivet or madanalysis plotting capabilities, `HepStore`provides a simple user interface 'hepstore-plot' which can plot any one or two dimensional representation of your data*. Currently the following kinds of plots are suported: scatter, histogram, errorbar, line, errorband, and contour.

1. Data Format

`HepStore` uses the '.npy' format

* provided in .npy format

2. Example

C. hepstore.school

One of the themes very interesting to particle physiscts and also recieving more and more attention in the phenomenolgical community is machine learning. 'hepstore.school' is an attempt to provide a genric interface to python's sklearn package. our school consists of a teacher, a student and, of course, a book. the teacher advices the student which algorithm to use, where upon the student takes the algorithm from the book and is able to explore and train itself to learn from the data provided.

1. Classifiers

lda,qcd,svc,mlp

2. Tuning

As it is not a prior clear which parameters to chose for an algorithm, 'hepstore-school' provides the switch '--only_explore' which will try to automatically find the best parameters for the given training set. In addition for all numerical parameters it provides cross validation plots to check for over respectively under performance. Note that some classifiers might have a large parameter space and hence can not be tunde in a fully automated way, yet.

3. Training

Training is performed automattically on a 75% subset of the data provided.

4. Testing

Training is performed automattically on a 25% subset of the data provided.

5. Example

D. hepstore.analysis

the hepstore-analysis module is a high level package to perform the usual statistical computations needed in high energy physics. it utilizes hepstore.school to provide signal and background classification and extract statistical quantities such as the maximal poissonian significance, the upper exclusion bound on the signal cross section etc.

still missing: traditional cut analysis

1. Example

III. HEPSTORE-EAS: PRODUCE AND ANLYSE EXTENDED AIR SHOWERS THE HEPSTORE WAY

A. List

'hepstore-eas -L'

As this is a specific air shower framework we organise everything with respect to the following scheme: energy -¿ primary particle -¿ physical process -¿ MC generator -¿ nucleon model -¿ final state the --list argument searches for this structure and lists the following possible contents hard events -¿ attempted showers -¿ showeres -¿ observables

B. Generation

'hepstore-eas -G'

we interface from the hepstore.docker family (more details in appendix) and provide automated runcards for herwig to produce events which may be showerd by corsika

1. Nucleon Modle

we provide two nucleon models 'full' and 'fragmented'

C. Shower

'hepstore-eas -S'

we interface from the hepstore.docker family and provide automated runcards for corsika to either shower events generted with '-G' or produce corsika stand alone showers (internal corsika di-jet production) when using '-C 7.4' (here 7.4 is the version vor the docker image)

D. Observables

'hepstore-eas -A'

we provide our own event class to read corsika produced extended air showers. inspired by rivet we allow for dynamic load of user analysis modules and provide one example for such, constructing ρ_μ and $X_{\max}$. hepstore-eas automatically provides its output in .npy format

E. Example

1. A User Analysis

2. Comparing Herwig 7 di-jet production with internal Corsika QCD model

IV. CONTRIBUTIONS AND REQUESTS

V. CONCLUSIONS

Appendix A: The hepstore.docker family

1. hepstore-herwig

a. Code

```
# module: hepstore.docker.herwig

# imports for this module
import os

##############################################################################
## run the app
##############################################################################
def main(args=None):

    # we need to setup the arg parser
    import argparse
    parser = argparse.ArgumentParser(
        description =
        "This App allows to run the Herwig code out of the box with python 2.7." )

    # specify arguments for versioning
    parser.add_argument( "--repository"   ,
                         type    = str,
                         default = "peterschichtel",
                         help    = "docker repo of the genrator" )
    parser.add_argument( "--generator"    ,
                         type    = str,
                         default = "Herwig",
                         help    = "which generator to run, default Herwig " )
    parser.add_argument( "--generator-version",
                         type    = str,
                         default = "7.0.4",
                         help    = "which version to run, default 7.0.4" )

    # mount a directory
    parser.add_argument( "--directory",
                         type    = str,
                         default = os.getcwd(),
                         help    =
                         "mount this directoy as /UserDirectory (automatic working dir!),
                             default is PWD!" )
```

```
    # verbose stdout
    parser.add_argument( "-v", "--verbose",
                         action = "store_true",
                         help   = "print container stdout" )

    # parse args
    parsed_args, unknown = parser.parse_known_args(args)

    # run the app
    from interface import DockerIF as Herwig
    app=Herwig(
        image    = os.path.join( parsed_args.repository.lower(),
                                 parsed_args.generator.lower() ),
        version  = parsed_args.generator_version,
        verbose  = parsed_args.verbose
    )
    app.run(
        directory = parsed_args.directory,
        args      = [ '/bin/bash',
                      '-c',
                      'source $ACTIVATE && %s ' % " ".join(['Herwig'] + unknown )
        ]
    )

    pass # main
################################################################################

################################################################################
if __name__ == "__main__":
    main()
    pass
################################################################################
```

b. Dockerfile

2. hepstore-corsika

a. Code

b. Dockerfile

3. hepstore-hepmc2corsika

a. Code

b. Dockerfile