mlsquare Documentation

Release unknown

MLSquare Foundation

Apr 11, 2019

Contents

1 Getting Started! **3**

2 Contents **5**

2.1 Installation Guide 5
2.2 User Guide 6
2.3 Developer Guide 7
2.4 API Reference 8
2.5 License 8
2.6 Contributors 8
2.7 Creating a Issue 9
2.8 Changelog 9
2.9 Supported Modules and Models 9

3 External links **11**

4 Indices and tables **13**

MLSquare is an open source developer-friendly Python library, designed to make use of Deep Learning for Machine Learning developers.

Note: `mlsquare` python library is developed and maintained by MLSquare Foundation

In the first version we come up with **Interoperable Machine Learning [IMLY]**. *IMLY* is aimed to provide every Machine Learning Algorithm with an equivalent DNN Implementation.

CHAPTER 1

Getting Started!

Setting up `mlsquare` is simple and easy

1. Create a Virtual Environment

```
virtualenv ~/.venv
source ~/.venv/bin/activate
```

2. Install `mlsquare` package

```
pip install mlsquare
```

3. Import `dope()` function from `mlsquare` and pass the `sklearn` model object.

```
>>> from mlsquare.imly import dope
>>> from sklearn.linear_model import LinearRegression
>>> from sklearn.preprocessing import StandardScaler
>>> from sklearn.model_selection import train_test_split
>>> import pandas as pd

>>> model = LinearRegression()
>>> data = pd.read_csv('./datasets/diabetes.csv', delimiter=",",
                       header=None, index_col=False)
>>> sc = StandardScaler()
>>> data = sc.fit_transform(data)
>>> data = pd.DataFrame(data)

>>> X = data.iloc[:, :-1]
>>> Y = data.iloc[:, -1]
>>> x_train, x_test, y_train, y_test =
    train_test_split(X, Y, test_size=0.60, random_state=0)
>>> m = dope(model)

>>> # All sklearn operations can be performed on m, except that the␣
↪underlying implementation uses DNN
```

(continues on next page)

(continued from previous page)

```
>>> m.fit(x_train, y_train)
>>> m.score(x_test, y_test)
```

Note: For a comprehensive tutorial please do checkout this link

CHAPTER 2

Contents

2.1 Installation Guide

This guide describes how to install *mlsquare*

On this page

- *Setting up a virtual environment*
- *Installing the `mlsquare` package*
- *Testing the installation*

2.1.1 Setting up a virtual environment

The recommended way to install `mlsquare` is to use a virtual environment created by `virtualenv`. Setup and activate a new virtual environment like this:

```
$ virtualenv envname
$ source envname/bin/activate
```

If you use the `virtualenvwrapper` scripts, type this instead:

```
$ mkvirtualenv envname
```

2.1.2 Installing the `mlsquare` package

The next step is to install `mlsquare`. The easiest way is to use *pip* to fetch the package from the Python Package Index (PyPI). This will also install the dependencies for Python.

```
(envname) $ pip install mlsquare
```

Note: Installation via `pip` installs the stable version in your environment. To install the developer version checkout the package source from GitHub and run `python setup.py install` from the directory root. Note that developer version is not stable and there are chances that code will break. If you are not sure about it, we suggest you use the stable version.

2.1.3 Testing the installation

Verify that the packages are installed correctly:

```
(envname) $ python -c 'import mlsquare'
```

If you don't see any errors, the installation was successful. Congratulations!

Next steps

Now that you successfully installed HappyBase on your machine, continue with the *User Guide* to learn how to use it.

2.2 User Guide

This user guide explores the MLSquare API and should provide you with enough information to get you started. Note that this user guide is intended as an introduction to MLSquare, not to Keras or SkLearn or any other packages in general. Readers should already have a basic understanding of the packages they were using and its API.

While the user guide does cover most features, it is not a complete reference guide. More information about the MLSquare API is available from the *API documentation*.

On this page

- *Importing the `mlsquare` module*
- *Load `dope()` method into the enviroment*
- *Transpiling an existing model using* dope

2.2.1 Importing the `mlsquare` module

To start using the package, we need to import the module into the python enviroment.

```
>>> import mlsquare
```

If the above command doesn't result in any errors, then the import is successful

Note: To use `mlsquare` you need *Python* 3.6 or higher

2.2.2 Load dope() method into the enviroment

dope() is the base function, that returns an implementation of a given model to its DNN implementation. Once a model is dope'd, users will be able to use the same work flow as their initial model on the dope'd object.

```
>>> from mlsquare.imly import dope
```

2.2.3 Transpiling an existing model using *dope*

To demonstrate dope(), we will transpile sklearn LinearRegression and use the sklearn operations on the transpiled model.

```
>>> from sklearn.linear_model import LinearRegression
>>> model = LinearRegression()
>>> m = dope(model)

# Dope maintains the same interface as the base model package
>>> m.fit(x_train, y_train)
>>> m.score(x_test, y_test)
```

Note: dope() function doesn't support all the packages and the models in the package. A list of supported packages and models is available at the *Supported Modules and Models*

2.3 Developer Guide

mlsquare is open source. Developers can contribute to the module by making contributions.

1. To contribute to the development, you can create a branch by checkout the source from GitHub.
2. Include proper test cases for the feature.
3. Raise a pull request against the master branch

2.3.1 How to raise a pull request

Bug Fix

1. **Description about the bug that was been fixed** This pull request fixes #issue_number
2. **How it was fixed** Problem & Solution
3. **How to verify it** Steps or Code to verify the fix

New Feature

1. **Description about the feature that was been fixed** The pull request adds the functionality
2. **How it was done** Description about the solution
3. **How to verify it** Steps or Code to verify the feature

2.4 API Reference

`mlsquare.imly.`**`dope`**(*model*, ***kwargs*)

Transpiles a given model to it's DNN equivalent.

Parameters

- **`model`** (*`class`*) – The primal model passed by the user that needs to be transpiled.
- **`using`** (*`str`*) – Choice of type of "model transpilation" you want your model to undergo.
- **`accepts None and 'dnn' as values.`** (*`Currently`*) –

1. None: Returns the model as it is.
2. dnn (default): Converts the model to it's DNN equivalent.

- **`best`** (*`bool`*) – Whether to optmize the model or not.
- **`**kwargs`** (*`dict`*) – Dictionary of parameters mapped to their keras params.

Returns The transpiled model.

Return type model (class)

2.5 License

2.6 Contributors

- Soma S. Dhavala <soma@mlsquare.org>
- Shakkeel Ahmed <shakkeel@mlsquare.org>
- Ravi S. Mula <ravi@mlsquare.org>

We hear your feedback!

If you notice any issues during the usage the package `mlsquare`, please create an issue on GitHub. Before creating any issue, please check if the same issue was already created by any other user.

2.7 Creating a Issue

1. To create a new issue, navigate to the github page of the project and create an issue from the issues column
2. Include a short title
3. Include the error generate
4. Include the steps to reproduce it.

2.8 Changelog

2.8.1 Version 0.1

- Support for Linear & Logistic Regression from Sklearn

2.9 Supported Modules and Models

As of the current release, `mlsquare` supports the following models from the below modules

- **`sklearn`**
  - `LinearRegression`
  - `LogisticRegression`

We are working supporting more models and modules, however if you would like us to add any module, please write to us at info[at]mlsquare.org

CHAPTER 3

External links

- Online documentation (Read the Docs)
- Downloads (PyPI)
- Source code (Github)

CHAPTER 4

Indices and tables

- genindex
- search

Index

D

`dope()` (*in module mlsquare.imly*), 8