

# Machine Learning In Scilab Using Jupyter Kernel

## **User Guide**

### **1. Introduction**

Machine learning has been implemented in all major software platforms and languages (C++, Java, Python, MATLAB), with each having it's individual set of libraries to support and improve existing numerical analysis and computation.

This toolbox aims to provide machine learning capabilities through the Scilab console, for enhancing the users' ability to perform complex applications.

For ML in Scilab, it was decided to follow an integration approach which would make the existing python ml libraries available in Scilab. This toolbox aims to implement a bridge between the python ml libraries and the scilab console through the jupyter kernel approach.

Due to the possibility that the datasets or computing power wouldn't necessarily be available on the local machine, a Jupyter kernel approach has been adopted. This essentially means, if the user has the configuration details of the jupyter kernel running on a remote machine, the user can connect to it through Scilab for executing ML scripts.

Users without much prior machine learning experience can directly start utilizing the toolbox, as it also provides template scripts of major machine learning algorithms.

ML libraries adapted from python to Scilab-

#### 1. Scikit-learn

Target Users –

- i. Anyone who wants to make use of machine learning in Scilab for solving other complex problems
- ii. Anyone who doesn't have much experience or has beginner level knowledge in machine learning
- iii. Python native users who want to use existing machine learning libraries from python in Scilab, without much configuration

## 2. Requirements (Getting Started)

The toolbox requires the user to have the following softwares and packages to be installed:

- [Scilab 6.0.0](#) with scipython toolbox
- [Python 2.7](#) with numpy and scipy
- Jupyter\_client, notebook
- Ipython kernel

On the remote server side

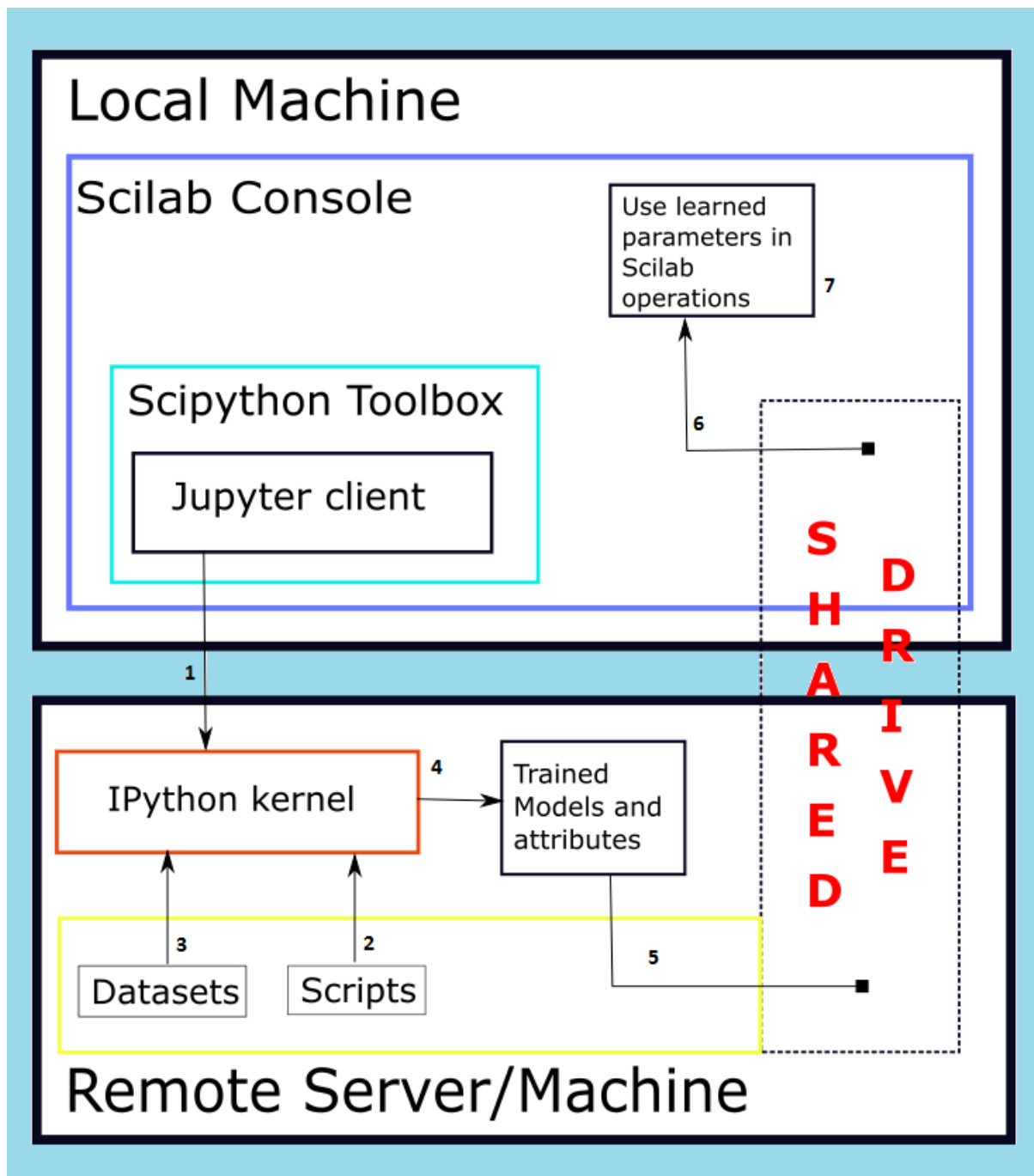
- Python 2.7 with numpy and scipy
- Jupyter notebook
- Ipython
- [Scikit-learn](#)
- Datasets

## 3. Supported Machine learning Algorithms

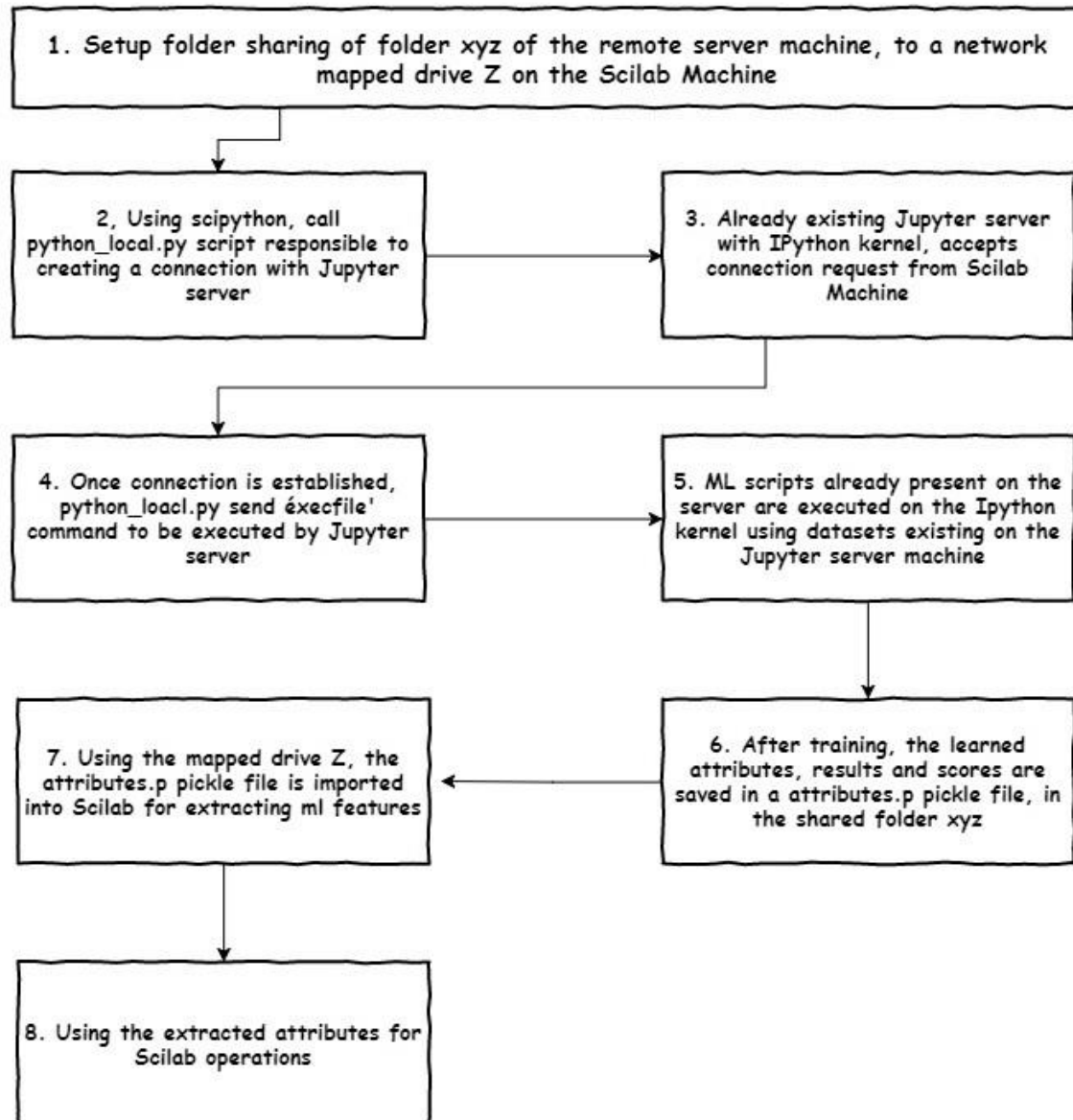
Currently the machine learning toolbox in Scilab is at a very early stage. It has been developed to support the scikit-learn python ml library. As of now it has the set of most commonly used ml algorithms as listed below:

- i. Linear regression
- ii. Multiple regression
- iii. Logistic regression
- iv. Support Vector Machine
- v. Decision trees classification
- vi. k-means clustering
- vii. hierarchical clustering
- viii. Naive bayes classification
- ix. Decision trees clustering
- x. kernel SVM
- xi. Ridge regression
- xii. Random forest classification
- xiii. Kernel Ridge regression

#### 4. Logical Flow Diagram



## 5. Workflow Diagram



## 6. Procedure

Ensure that you have all software requirements satisfied on both the Scilab machine and the remote server side (for Jupyter approach)

- I. Selection of Machine learning algorithm
  - a. Define the type of machine learning problem to be solved (regression, classification, clustering). You could use this [reference](#)

- b. From the given catalogue of ml algorithms select the one which suitably satisfies your particular need

## II. Setup Folder Sharing

- a. Follow steps mentioned [here](#)
- b. From your local machine, open Networks → Shared Folder → Map Network Drive

## III. Sharing the connection file

- a. Ensure that the Ipython kernel is running on the remote machine  
`ipython kernel - ip =client_machine_ip`
- b. Copy the kernel-<pid>.json file from(C:/Users/UserName/AppData/Roaming/Jupyter/runtime) the remote machine to the shared drive

## IV. Edit the python local.py script

- a. Add the paths of source file and data-sets (if needed) to be copied on the shared folder drive
- b. To accommodate the ml script of your need inside the  
`km.execute( 'execfile( "ml_script.py" )' )`
- c. Specify the path of the Jupyter kernel's configuration file inside the python\_local.py script

## V. Modify the ML Template script

- a. Edit the ML script of your choice (like linear regression, logistic regression, SVM) for specifying the path of the datasets involved
- b. Add or remove any learned attributes of the ml algorithm in use, in the pickle commands.

## VI. Executing the ml script

- a. Once everything is setup, run the corresponding Scilab script from the scilab console  
`exec linear_regression.sce`
- b. All the attributes are loaded as elements of the pickle file imported inside Scilab for further usage.

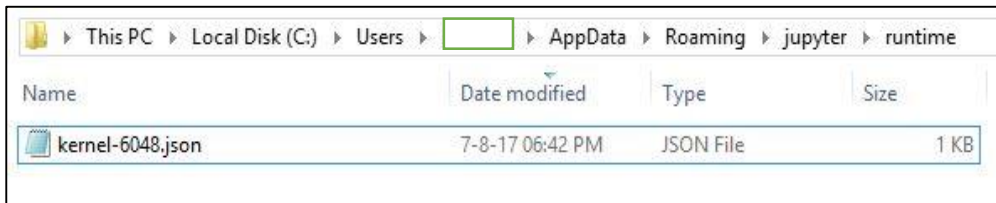
## 7. Sample Usage Tutorial

Here we are considering a sample example of linear regression through the ML toolbox. Users can utilize these steps as a reference for other applications.

It's being considered that the user has already setup a network shared drive and mapped it to Drive X.

- i. From the remote server/machine the IPython kernel connection file needs to be copied to Drive X.

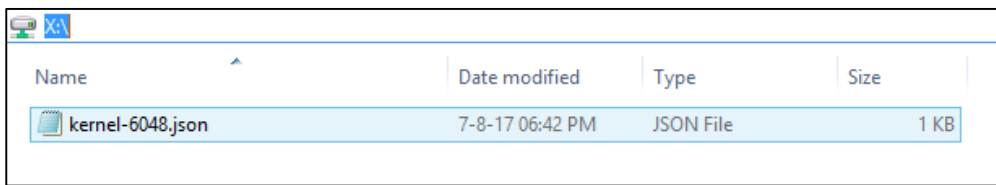
Source



The screenshot shows a Windows File Explorer window with the address bar path: This PC > Local Disk (C:) > Users > [username] > AppData > Roaming > jupyter > runtime. The file list contains one entry: kernel-6048.json, which is a JSON File, 1 KB in size, and was last modified on 7-8-17 at 06:42 PM.

Name	Date modified	Type	Size
kernel-6048.json	7-8-17 06:42 PM	JSON File	1 KB

Destination



The screenshot shows a Windows File Explorer window with a network drive icon in the address bar. The file list contains one entry: kernel-6048.json, which is a JSON File, 1 KB in size, and was last modified on 7-8-17 at 06:42 PM.

Name	Date modified	Type	Size
kernel-6048.json	7-8-17 06:42 PM	JSON File	1 KB

- ii. Open the python\_local.py script and add paths of the following 3 files:
  - a. ML script to be run on server
  - b. Datasets to be transferred to server
  - c. Kernel.json

```
import jupyter_client
from shutil import copyfile

# For copying files from local machine to remote server/machine

# For python script file
copyfile('F:/linear_regression.py', 'X:/linear_regression.py')
# For datasets involved
copyfile('F:/Salary_Data.csv', 'X:/Salary_Data.csv')

# Path of the kernel connection file
cf="X:/kernel-6048.json"

# Setup up a blocking kernel client using kernel connection file
km=jupyter_client.BlockingKernelClient(connection_file=cf)

# load the connection settings
km.load_connection_file()

# execute any python commands on remote IPython kernel
km.execute('execfile("linear_regression.py")')
```

- iii. Edit the linear regression template script to use the dataset of your choice and adapt it to your need

```
# Simple Linear Regression
# Importing the libraries
import numpy as np
import pandas as pd
import pickle

# Importing the dataset
dataset = pd.read_csv('Salary_Data.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random

# Feature Scaling
"""from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
sc_y = StandardScaler()
y_train = sc_y.fit_transform(y_train)"""

# Fitting Simple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)

# Predicting the Test set results
y_pred = regressor.predict(X_test)
coef_ = regressor.coef_
intercept_ = regressor.intercept_

# Score of prediction model
from sklearn.metrics import r2_score
score = r2_score(y_test, y_pred)

pickle.dump([y_pred, score, coef_, intercept_], open("attributes.p", "wb"))
```

- iv. Ensure that linear\_regression.sce has the correct path of the shared drive. In this case its 'X:/'

```
pyImport pickle
py = pyBuiltin()
pyImport numpy

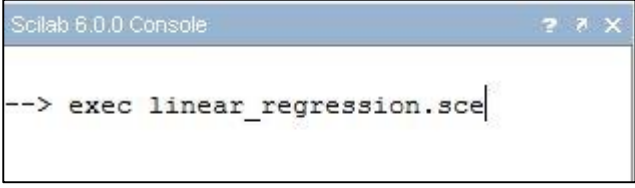
pyExecFile('python_local.py')

sleep(100) // for pickle file generation time lag

// Here X is the network shared drive name
attributes = pickle.load(py.open("X:/attributes.p", "rb"))

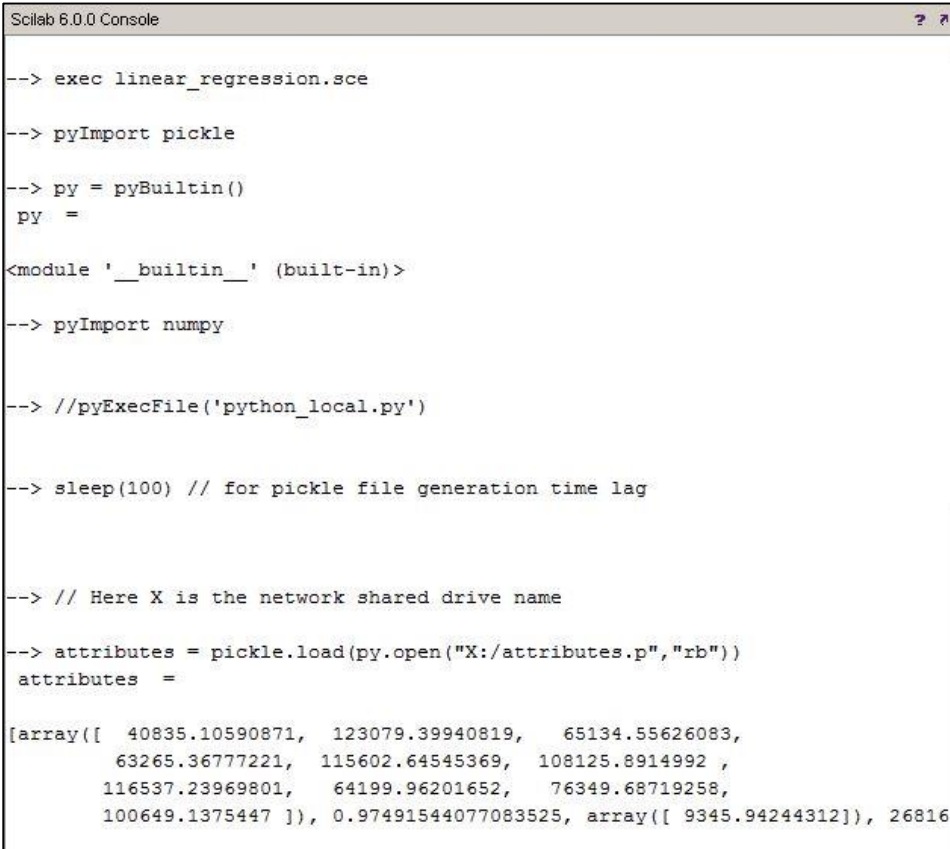
sleep(200)
y_pred = attributes(0)
score = attributes(1)
coef_ = attributes(2)
intercept_ = attributes(3)
```

- v. Once everything is setup, simple run the linear\_regression.sce script in the Scilab console



```
Scilab 6.0.0 Console
--> exec linear_regression.sce
```

- vi. After execution completes you can see that the linear regression's prediction results and learned attributes (coefficient and intercept) have been imported inside the Scilab console



```
Scilab 6.0.0 Console
--> exec linear_regression.sce
--> pyImport pickle
--> py = pyBuiltin()
py =
<module '__builtin__' (built-in)>
--> pyImport numpy
--> //pyExecFile('python_local.py')
--> sleep(100) // for pickle file generation time lag
--> // Here X is the network shared drive name
--> attributes = pickle.load(py.open("X:/attributes.p", "rb"))
attributes =
[array([ 40835.10590871, 123079.39940819, 65134.55626083,
        63265.36777221, 115602.64545369, 108125.8914992 ,
        116537.23969801, 64199.96201652, 76349.68719258,
        100649.1375447 ]), 0.97491544077083525, array([ 9345.94244312]), 26816
```



```

Scilab 6.0.0 Console
--> sleep(200)

--> y_pred = attributes(0) '
y_pred =

    40835.106
    123079.4
    65134.556
    63265.368
    115602.65
    108125.89
    116537.24
    64199.962
    76349.687
    100649.14

--> score = attributes(1)
score =

    0.9749154

--> coef_ = attributes(2)
coef_ =

    9345.9424

--> intercept_ = attributes(3)
intercept_ =

    26816.192

```

Also all the attributes have been added to the workspace, for the user to use for other Scilab operations.

	Name	Value	Type	Visibility
<input checked="" type="checkbox"/>	ans	1x1	Boolean	local
<input type="checkbox"/>	attributes	N/A	_EObj (Mlist)	local
<input type="checkbox"/>	coef_	9.35e+03	Double	local
<input type="checkbox"/>	intercept_	2.68e+04	Double	local
<input type="checkbox"/>	numpy	N/A	_EObj (Mlist)	local
<input type="checkbox"/>	pickle	N/A	_EObj (Mlist)	local
<input type="checkbox"/>	py	N/A	_EObj (Mlist)	local
<input type="checkbox"/>	score	0.975	Double	local
<input type="checkbox"/>	y_pred	10x1	Double	local