Manual

PaREnTranslatorPlugins

Introduction

The PaREnTranslatorPlugins module is a spin-off from the ExCoLib. Check out the full folder from the svn in order to get it all.

PaREnTranslatorPlugins are stripped down functionality for importing and exporting data into different formats. It closely orients its functionality around the needs of the PaREn Translator class found in PaREnDataSources module.

PaREn Translator

The PaREn Translator class maintains a set of data in a consistent way and makes frequent manipulations easy. The information stored in the translator are the feature vectors, the names of the features (name of columns), the labels for each of the vectors and a probability matrix together with its header description. These are the almost always required data. The translator can be missing some of the fields. You can separate them and join them together easily. Translators use the Convert() method to move between RAM (as NumPy array) and other formats. The other formats are supplied by the PaREnTranslatorPlugins.

Unfortunately, not all file formats are capable of representing all of the information needed to restore a translator from file. The result is that some formats support importing and exporting at different levels of fildelity. The plugins know this in advance of course and have flags that indicate, roughly what kind of capability they have.

PaREnTranslatorPlugins

Installation

Download PaREnTranslatorPlugins (Separated) and invoke

```
python setup.py install
```

This will use distutils, in order to make PaREnTranslatorPlugins globally visible. Through this global visibility you can always invoke:

import PaREnTranslatorPlugins

from PaREnTranslatorPlugins import *

General Description

Available Formats

Currently the following plugins are supported out of the box:

Name	Import	Export
R	None	Multiple data, complete
CSV	Multiple data, complete	Multiple data, complete
XML	Multiple data, complete	Multiple data, complete
MAT	Single data, complete	Single data, complete
SVM	None	Single data, restricted

The SVM plugin supports options for specifying which labels are considered +1 and which -1 and whether the other labels are exported as +0 or not exported.

Classes

The PaREnTranslatorPlugins module consists of several variables and classes that do the job of exporting and importing. The class, which is instantiated only once is the registry class TranslatorPlugins. The variable representing the registry is TranslatorPluginRegister.

```
>>import PaREnTranslatorPlugins
>>PaREnTranslatorPlugins.TranslatorPluginRegister
<PaREnTranslatorPlugins.TranslatorPlugins instance at 0x11ee4d0>
```

The registry provides you with three important methods.

- Import()
- Export()
- PrintAllFormatsInfo()

The Plugin class is the base class for all plugins. Currently PaREnTranslatorPlugins supports RPlugin, MATPlugin, SVMPlugin, XMLPlugin and CSVPlugin out of the box.

Functions

There also exist four functions:

- PluginInfo()
- CreateData()

- Import()
- Export()

Basically, these functions expose TranslatorPluginRegister's functions in a short to type way.

Data

The functions use a specific data format in order to communicate with the Translator. In fact a plugin can import/export multiple translators into/from a single file. Therefore the outer data structure is a list. The inner structure is a dictionary with keywords fixed by convention. These keywords are

- Vectors
- Features
- Labels
- Probabilities
- ClassIDs

Example:

```
Data = [ { “Vectors”:ArrayWithVecs, “Features”:ListWithNames,
“Labels”:ListWithClassIDs, “Probabilities”:ArrayWithProbabilities,
“ClassIDs”:ListWithHeaderForProbabilities} ]
```

Detailed Description

CLASS TranslatorPlugins

The class TranslatorPlugins is instatiated only once to an object known as the TranslatorPluginRegister. Every newly instantiated Plugin uses the Add() method in order to advertise itself. The Remove() method is never called. Originally the __del__ destructor was supposed to do this, but because of a few specialties with Python __del__ is never called. Even if it was called, the object would not be freed from memory.

```
#============================================================================================================
#CLASS TranslatorPlugins
#This class is responsible for handling all the formats registered by the programmer. It maintains the
#record of import-export plugins
#============================================================================================================
class TranslatorPlugins :
        #============================================================================================
        #Class TranslatorPlugins
        #CONSTRUCTOR
        #============================================================================================
        def __init__( self ):
```

```
        ...

#============================================================================================
#Class TranslatorPlugins
#DESTRUCTOR
#============================================================================================
def __del__( self ) :
        ...

#============================================================================================
#Class TranslatorPlugins
#Add()
#Search if the plugin is already registered or not. If it is already registered,
#it will generate an error. Otherwise will add the plugin in self.MyRegistered
#Parameter                          : PluginInfo     (Contains information about a plugin)
#============================================================================================
def Add( self, PluginInfo ):
        ....
#============================================================================================
#Class TranslatorPlugins
#Remove()
#Removes a plugin from self.MyRegistered.
#Parameter                          : PluginInfo     (Contains information about a plugin)
#============================================================================================
def Remove( self, PluginInfo ):
        ...
#============================================================================================
#Class TranslatorPlugins
#Import()
#Imports data using a given stream (Stream) from a particular format (Name). If the format
#is not registered it generates an error. If the format is registered but the import is not
#supported a warning is generated by class Plugin.
#Parameter                          : Name   (Name of the format importer)
#Stream                             : Stream (File name or an object pointing to a file)
#Options                            : Dictionary with options
#============================================================================================
def Import( self, Name, Stream, Options={} ):
        ...

#============================================================================================
#Class TranslatorPlugins
#Export()
#Exports data (Data) using a given stream (Stream) to a particular format (Name). If the
#format is not registered it generates an error. If the format is registered but the export is
```

```
#not supported a warning is generated by class Plugin.
#Name                           : STRING. Name of the format importer
#Stream                         : Stream (File name or an object pointing to a file)
#Data                           : LIST OF DICTS (Data to be exported to file)
#=============================================================================================
def Export( self, Name, Stream, Data, Options={} ):
        ...
#=============================================================================================
#Class TranslatorPlugins
#PrintAllFormatsInfo()
#Prints the name and availability of Import and Export Capabilities of a format
#=============================================================================================
def PrintAllFormatsInfo( self ):
        ...
```

The importing and exporting is done via the Import() and Export() methods, respectively:

```
f = open( “something.csv”, “w” )
TranslatorPluginRegister.Export( “CSV”, f, Data, Options )
f.close()
```

CLASS Plugin

The Plugin class defines the interface that is understood by the registry. It is very simple. It only provides four methods:

- Import()
- ImportOptions()
- Export()
- ExportOptions()

Whenever a new plugin is created, it first must advertise itself: (EXAMPLE)

```
#============================================================================================================
#CLASS RPlugin
#============================================================================================================
class RPlugin( Plugin ):

        #============================================================================================
        #CLASS RPlugin
        #CONSTRUCTOR
        #============================================================================================
        def __init__( self ):
                MyCanImport = Plugin.NoCapability;
                MyCanExport = Plugin.MultiCompleteData;
                NameOfPlugin = “R”;
                Plugin.__init__( self, NameOfPlugin, MyCanImport, MyCanExport );

        …


#the following codes are to be used for import and export direction:
NoCapability        = 0;      #can't do anything for the direction
MultiCompleteData   = 1;      #can read (write) multiple data sets with all data fields (like XML)
SingleCompleteData  = 2;      #can read (write) just a single data set with all data fields (like MAT)
MultiRestrictedData = 3;      #can read (write) multiple data sets, but not all fields
SingleRestrictedData= 4;      #can read (write) only a single data set and not with all fields (like SVM)
```

In order to make the module really visible, you must create an object of your class. If you added an additional plugin to the PaREnTranslatorPlugins module, then you should add this object to the following section:

```
#============================================================================================
#PLUGIN CREATION
#============================================================================================
_XML = XMLPlugin()
_CSV = CSVPlugin()
_SVM = SVMPlugin();
_MAT = MATPlugin();
_RRR = RPlugin();
_??? = MyPlugin()
```

If your plugin is not created in the PaREnTranslatorPlugin module, you must make sure, that your plugin object is long lived , since the unregistration doesn't work.

Functions

PluginInfo()

Prints a list on the terminal, so that one knows what kind of plugins are available.

CreateData()

Creates a list with a single dictionary in it. The dictionary is given the right keys, so that you don't have to remember them yourself. All parameters are optional. You supply them, as they are availbale. Note, that if you supply Features, you also must support Vectors and that if you support Probabilities, then you also must provide ClassIDs.

```
#============================================================================================================
#FUNCTION CreateData( Name, Vectors, Labels, Probabilities, ClassIDs )
#RETURNS LIST OF DICTS
#This class packages the parameters into a single dictionary understood by the plugins.
#In order to store more than one translator in the file, you can do the following:
#Data = CreateData(...) + CreateData(...) + ...
#
#Name             : OPTIONAL STRING. Name of the translator
#Vectors          : OPTIONAL 2D ARRAY. Rows are feature vectors.
#Labels           : OPTIONAL LIST. Contains labels for each row
#Probabilitis     : OPTIONAL 2D ARRAY. Probability of each feature vector to belong to a specific class
#ClassIDs         : MANDATORY TO PROBABILITIES. LIST. Identifies the headers (label) for each column in
#                   Probabilities.
#============================================================================================================
def CreateData( Name=None, Vectors=None, Features=None, Labels=None, Probabilities=None,
                  ClassIDs=None ):

        ...
```

Import()

The Import function is a simple way to get data read into your experiment:

```
Data = PaREnTranslatorPlugins.Import( 'file.mat' );
```

or

```
Data = PaREnTranslatorPlugins.Import( 'file.csv', Options = {...} );
```

if you need to specify some options. This function will only work, when file extension matches the plugin name!

Export()

The Export function is a simple way to get your data out on the way:

```
PaREnTranslatorPlugins.Export( “file.xml”, Data )
```

or

```
PaREnTranslatorPlugins.Export( “file.xml”, Data, Options = {...} )
```

if you need to specify some auxiliary options. Be aware, that only those data get exported that are actually supported by the file format, other fields get lost without warning.

Data

Vectors: 2D numpy array

Labels: List where length = height of Vectors

Features: List where length = width if Vectors

Probabilities: 2D numpy array where height = height of Vectors

ClassIDs: List wheren length = width of Probabilities