

EVRI-thing You Need to Know About Deploying Models with Model_Exporter

Bob Roginski

Eigenvector Research, Inc.
Manson, WA USA



Why Deploy Models?

Numerous objectives for models:

- Understand processes from relevant measurements
- Troubleshoot issues
- Characterize interactions
- Use for future events or measurements
 - Often times – greatest rate of return
 - Successful implementation requires:
 - models that are robust over time
 - an effective means of updating them when necessary

Applying A Model

In simplest terms the process of applying a model to new data is akin to following a recipe

- *numerical parameters*, new data \leftrightarrow ingredients
- *mathematical operations* \leftrightarrow steps



Model_Exporter

Model_Exporter creates a file that contains:

- Numerical values and (simple) mathematical steps needed to apply a preprocessing and model to new data
- Translation allows very fast model application
- Results include typical model outputs:
Predictions, scores, diagnostics, contributions
- Output file can be interpreted in a number of different environments and integrated into 3rd party applications.

Methods & Outputs

Model_Exporter supports a wide variety of

- Model types
- Preprocessing methods

Outputs

- Model specific
- Up to the user which to use

Example: PCA with Autoscale Preprocessing

\mathbf{x} is a single new sample

Autoscale: Need **means** and **standard deviations** of variables

Autoscale operations:

$$\mathbf{x} = (\mathbf{x} - \mathbf{m}) ./ \mathbf{sd}$$

PCA: Need **loads** and **tconbasis**

PCA script operations:

```
scores = x * loads
Tcon   = scores * tconbasis
Xhat   = scores * loads'
```

...etc.

Constants from
the model:

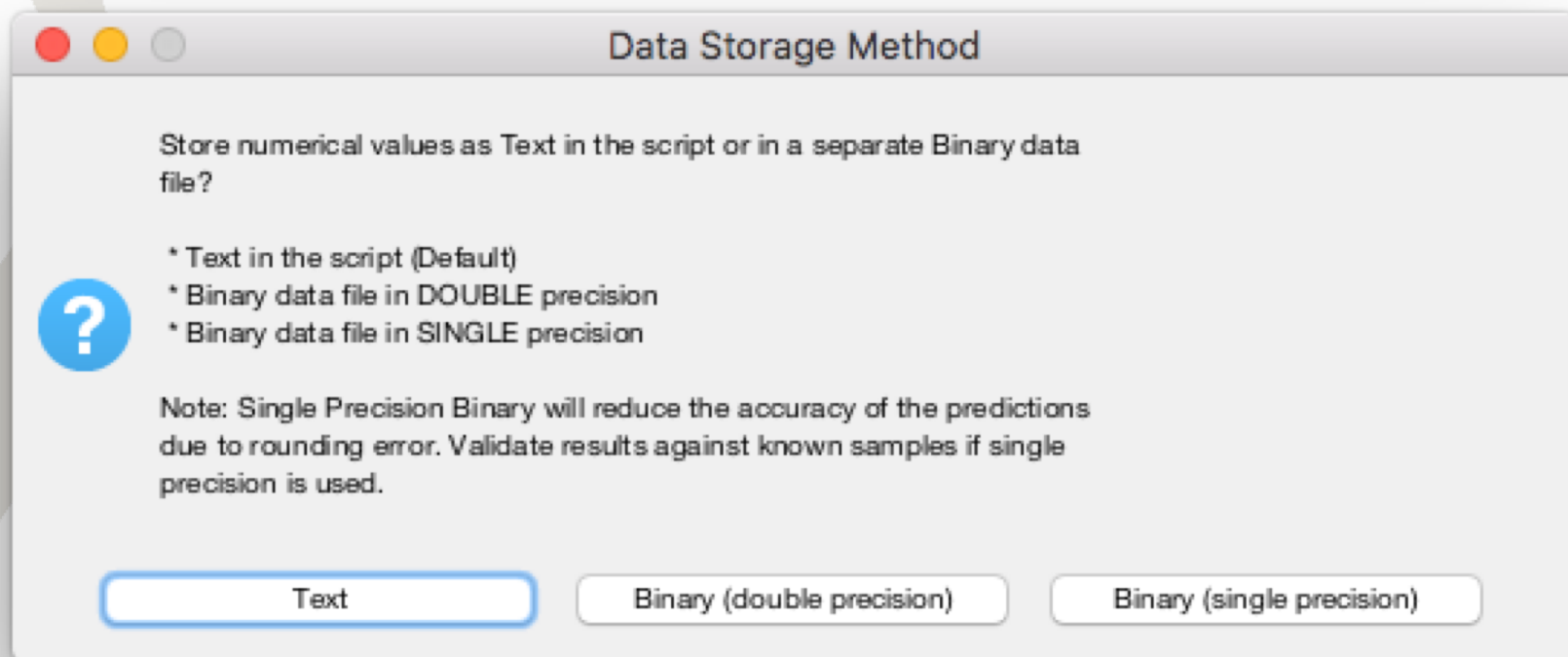
```
m
sd
loads
tconbasis
reslim
tsqlim
```

Example: ARCH Data Set

- Native American artifacts
 - Well known data set of obsidian samples from four known quarries*
Samples of unknown origin
 - XRF measurements of 10 different metals
- PCA model built on samples of known origin
 - export to .m
 - archPCA.m
 - export to .py
 - archPCA.py

**Anal. Chem.; 1972; 44(13); 2176-2180*

Data Storage Format



Data Storage Method

Store numerical values as Text in the script or in a separate Binary data file?

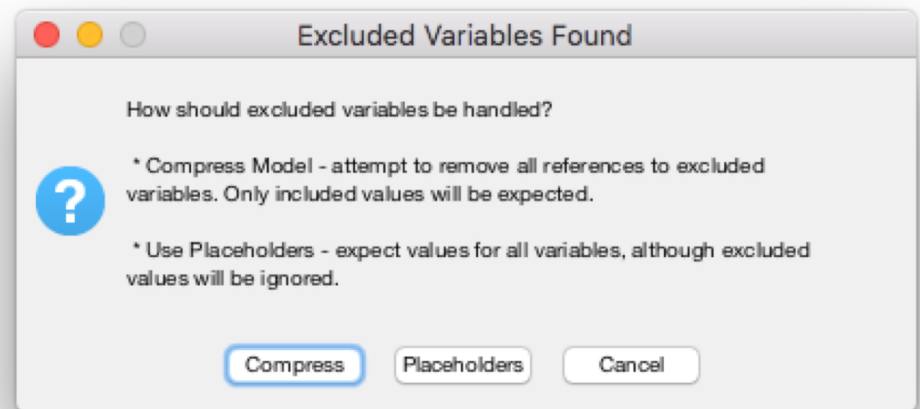
- * Text in the script (Default)
- * Binary data file in DOUBLE precision
- * Binary data file in SINGLE precision

Note: Single Precision Binary will reduce the accuracy of the predictions due to rounding error. Validate results against known samples if single precision is used.

Text Binary (double precision) Binary (single precision)


Excluded Variables

- Two options
 - compressed
 - archPCA_NoY_comp.m
 - placeholders
 - archPCA_NoY_full.m
- Responsibility – error trapping
 - More onus on the developer




XML Format

- Self-describing text file containing
 - parameters
 - operations
- Structurally “similar” to .m file
- Same names used for operations
- But . . .



XML File Format

The XML files output by Model Exporter are not directly usable. They can be used in any programming language for which an interpreter is available to translate the model's XML representation into a form which can be used by that programming language. The Model Exporter software package includes two such interpreters, one for .NET languages and one for Java. For more details see [Model_Interpreter](#). The input variable x should be a vector, representing a single sample, and the output will be a prediction for this one sample.



Model Exporter Interpreter

The Model_Exporter Interpreter consists of two main versions, one for Microsoft .NET and one for Java. Both of these versions provide classes to apply Model_Exporter models to new data. The Interpreter can be used in any of these environments and includes all source files:

- Java
- Microsoft .NET environments (VS2012 or later)
- Generic C#-based compilers

The following describes the classes implemented and their use. The classes include the ModelInterpreter, Workspace, and Matrix classes.

The ModelInterpreter class allows a software developer to indicate an exported model they want to make predictions with and the data that they want to make predictions from. They can then apply the model and access the prediction results. The other two classes (Matrix and Workspace) are used by ModelInterpreter and are available to the software developer to manage numerical data.

The source code for these classes is supplied with Model_Exporter and can be used, compiled, and redistributed without restriction. Note that such redistribution permission does **not** permit the user to redistribute the Model_Exporter product itself - only the interpreter.

Contents [\[hide\]](#)

- 1 [Microsoft .NET Interpreter](#)
 - 1.1 [ModelInterpreter Class](#)
 - 1.1.1 [Constructors](#)
 - 1.1.2 [Methods](#)
 - 1.1.3 [Properties](#)
 - 1.1.4 [Examples](#)

Deployment Scenarios

- .m format
 - use natively in Matlab without the need for additional toolboxes
 - protect the content by p-coding
 - use in “Matlab-like” platforms such as Octave
 - some minor code modifications may be necessary
 - LabView
 - MathScript Node
 - compile .m code directly using Matlab Compiler
 - create C code for compilation using Matlab Coder

entire data set (75 X 10)



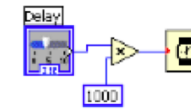
current row

Arch Data



current response

plot of current row



loop delay

loop counter

record number

"stop" pushbutton



```
1 m = [ 1:59.4 247.226666667 41.64 666.52 397.733333333 44.8
2 s = [ 319.650357972 111.316759299 17.1390497702 274.00554
3 %-----
4 % Script Operations
5 x = minus(x,m);
6 x = rdivide(x,s);
7
8 %-----
9 %Step - sequence #2
10 %PCA Prediction
11 %-----
12 % Constants
13 loads = [ 0.408274345863 0.353715185704 0.342334743961 0.3
14 -0.044605850287 0.361749309734 -0.0527230798975 0.42274
15 -0.0872808179825 -0.0616297685265 0.268976401793 -0.1309
16 -0.2353322478 -0.023133849208 0.554419361741 -0.24400015
17 tconbasis = [ 0.178152297608 0.154345167785 0.14937926227
18 -0.0309402818812 0.250922817106 -0.0265706951707 0.2532
19 -0.003926926901 -0.0592615560003 0.250640596561 -0.12590
20 -0.25704555228 -0.025268300633 0.605573746759 -0.2665132
21 reslim = 1.82148950093;
22 tsqim = 10.4257147422;
23 sum = [ 1*ones(1,10) ];
24 c2 = 2;
25 %-----
26 % Script Operations
27 scores = mtimes(x,loads);
28 Tcon = mtimes(scores,tconbasis);
```

model application

scores

reduced Q 2

reduced T-squared 2

Deployment Scenarios cont'd

- .py format
 - take advantage of the
 - user interfaces available to expedite workflow
 - abundance of preprocessing tools
 - in PLS_Toolbox/Solo to build your model
 - export and deploy in Python

Deployment Scenarios cont'd

- XML format
 - useful for embedded platforms
 - supplied interpreters
 - Java
 - .NET


```
1 |
2
3 ModelInterpreter test = new ModelInterpreter("plsexample.xml");
4
5 //display some of the common model information
6 Console.WriteLine("Model type: " + test.modeltype);
7 Console.WriteLine("Expected Data Size:" + test.inputDataSize);
8
9 //the variable inmatrix contains new data for model application
10 //apply model
11 test.inputdata = inmatrix; //assign input data to object
12 test.apply(); //apply model
13
14 //Typical outputs for a PLS or other regression model:
15 Console.WriteLine("yhat = " + test.results.getVar("yhat"));
16 Console.WriteLine("T2 = " + test.results.getVar("T2"));
17 Console.WriteLine("Q = " + test.results.getVar("Q"));
18
```

Some Metrics (C#)

Thanks Donal O'Sullivan!

- 1000 cycles
 - worst case => create new object instance and import text file of test data
 - improved case => create object instance and test data once, loop over `object.apply()`
- PCA model
 - 10 variables, autoscale
 - w.c. => 0.970 s
 - i.c. => 0.002 s
- PLS model
 - 401 variables, mean centering
 - w.c. => 4.800 s
 - i.c. => 0.005 s
- PLS model
 - 700 variables, MSC, GLS weighting, mean centering
 - w.c. => 4.800 s
 - i.c. => 0.006 s

Updating Models Without Having to Recompile Code

- .m file format
 - feasible if using binary format to store parameters
- XML format
 - more flexible

Calibration Transfer

Model Centric Calibration Transfer Tool

File Edit Help

Calibration

Master X Block Pre Xfer Post Xfer

Y Block

Slave X Block Xfer

Master Model

Slave Model

Validation

Master X Block

Y Block

Slave X Block

Validation Results

Calibration Transfer Model Types

Direct Standardization (DS)

Piecewise Direct Standardization (PDS)

Double Window Piecewise Direct Standardization (DWPDS)

Spectral Space Transformation (SST)

Calibration Transfer Model Settings

	Min	Step	Max
Window (PDS):	11	2	11
Window 1 (DWPDS):	11	2	11
Window 2 (DWPDS):	21	2	21
Noomp (SST):	1	1	1

	CAL/VAL	PP Index	TransferT...	X Preproc...	Model ID
1					

Model_Exporter supports

- PDS
- DWPDS
- DS
- SST

In Closing

Model_Exporter

- provides a means to export your model in a simple form to a number of target formats
 - Matlab .m
 - Python .py
 - XML
 - interpreters for Java and C# available
- flexibility for the deployment specialist



Comparing Solo_Predictor and Model_Exporter

- **Time:**
 - Solo_Predictor is slower to return results because of socket overhead but response can include predictions for many samples. ~0.4 seconds per call
 - Model_Exporter is fast. Only predicts on one sample per call. ~0.001 seconds per prediction
- **Memory requirements:**
 - ~200 Mb for Solo_Predictor.
 - Model_Exporter memory depends on the target programming language memory footprint (Matlab, Python, C#, VB.NET, Java).
 - This is in addition to the sample data and model memory requirements.
- **Licensing/costs**
 - Exported models have no license costs for the end-user
- **Versatility**
 - Solo_Predictor is full featured while Model_Exporter is not

Which Should I Use?

- Example: process NIR generates spectrum every 30 sec, need results to go to DCS
 - Solo_Predictor (probably)
- Example: need to combine spectroscopic measurements along with process measurements stored in a data historian and apply a model
 - Solo_Predictor (database functionality and multiblock)
 - Model_Exporter if you're willing and able to add in a bunch of additional code

Which Should I Use?

- Example: handheld spectroscopic instrument using Linux and an ARM platform with limited computing resources
 - Model_Exporter
 - export to XML
 - use Java XML interpreter
 - compile using C# XML interpreter
 - export to m-file, use Matlab Coder to convert to ANSI C
- Example: high throughput analysis (>1 Hz)
 - Model_Exporter

Which Should I Use?

- Example: I've built a model that I wish to share with others at my large corporation who don't use PLS_Toolbox or Solo. I also wish to keep them from modifying any aspect of the model
 - Model_Exporter => m-file
 - use `pcode` to obfuscate the code or . . .
 - . . . compile it using Matlab Coder