MIMOSAS: Multisource Input Model Output Security Analysis Suite

Installation Guide and Reference Manual

J. Zhao, B.L. Goldblum, C. Stewart
University of California, Berkeley

September 28, 2019

1 General Description

A supervised machine learning pipeline, MIMOSAS (Multisource Input Model Output Security Analysis Suite), has been developed for classification of multimodal data to inform nuclear security and proliferation detection scenarios. MIMOSAS provides an end-to-end data processing workflow, from data ingestion and pre-processing to model training and test set classification. The pipeline is specified via an input deck, making workflow customization effortless, and the framework is modular allowing for the easy addition of new learning algorithms. In the current build, the user selects from decision tree, random forest, and feed-forward neural network classifiers to train customizable models with built-in cross validation methods for hyperparameter optimization and feature selection. Trained model outputs are stored with the associated metadata for rapid deployment. These can be applied in supervised classification to assess previously unseen data or for further training as new observations are added to the existing data set. MIMOSAS provides the capability to fuse a wide range of data sources (e.g., radiation, environmental, acoustic, seismic, imagery, etc.) to make, confirm, and correlate machine learning predictions for nuclear security applications.

2 Customization

MIMOSAS is designed to be modified and expanded according to user specifications. Customization tips will appear throughout the documentation as follows:

```
[CUSTOMIZATION] This is a tip for how to customize MIMOSAS.
```

3 Architecture and Overview

MIMOSAS is organized with a main function that initiates the program, a parameter parser, a data processing utility, and a set of modular classifiers:

- `main.py`
- `parameter_parser.py`
- `preprocessing.py`

- decision_tree.py [modular classifier]
- random_forest.py [modular classifier]
- feed_forward_nn.py [modular classifier]
- source.config

The modular classifiers are stored in the `algorithms` folder. A configuration file is used to set the workflow and configure the parameters and initial settings. If a configuration file is not specified, `default.config` will be generated from `source.config` upon execution. DO NOT modify the `source.config` file unless you are attempting customization. To specify your workflow, modify the `default.config` file with your desired settings. When training is complete, trained models and runtime logs will be saved in a `saved_models` directory with a name based on the classifier type and run date and time. To save the output, ensure that you have write permissions in the directory. The output also includes a duplicated configuration file with model specifications for enhanced reproducibility.

```
[CUSTOMIZATION] New classifiers should be added in separate .py files to facilitate the
inclusion of additional dependencies with minimal interference.  The source.config file
should be modified when adding new argument functionality.
```

The repository includes the following additional files and folders:

- README.md
- Doxyfile
- LICENSE.txt
- mimosas.txt
- sample_data/sample_input_data.csv
- sample_data/sample_background_data.csv
- saved_models/

The `README.md` is rendered in Markdown on the MIMOSAS repository landing page: `https://github.com/nonproliferation/mimosas`. The `Doxyfile` is used to generate software reference documentation (See Sec. 9). The `LICENSE.txt` file specifies the license governing MIMOSAS and is reproduced herein in Sec. 10. The `mimosas.txt` file provides the ASCII art viewable during MIMOSAS execution.

The `sample_data/` directory contains example input data. The `sample_input_data.csv` is an example input dataset that illustrates the expected data format. Similarly, the `sample_background_data.csv` is an example dataset that illustrates the expected data format for use in the background subtraction algorithm described in Sec. 7.4. These sample input files can be used for execution of the workflow specified in the auto-generated `default.config` file.

Within the `saved_models` directory, the following three folders are present: `decision_tree`, `random_forest`, and `feed_forward_n`. Each contains a `Sample_Session` with representative output in Sec. 5. These can also be used to execute the `default.config` using a previously-trained model. Additional information on different modes of execution is available in Sec. 6.

4 Setup

Python 3 is required to run MIMOSAS, along with the Python packages listed in Table 1. In addition, the following tools are recommended:

- `git`, to pull code from the repository, and
- `doxygen`, to compile documentation.

Package	Version
`numpy`	1.17.2
`scipy`	1.3.1
`pandas`	0.25.1
`scikit-learn`	0.21.3
`tensorflow`	1.14.0
`protobuf`	3.9.2
`Keras`	2.3.0

Table 1: MIMOSAS Python package dependencies, including the recommended version.

5 Execution

Once you have cloned the MIMOSAS repository on your local machine, you can run the software by executing the following command:

```
python main.py
```

This generates a default configuration file (`default.config`). To run the software using a custom configuration file, execute:

```
python main.py --config custom.config
```

Edit the `custom.config` file with your desired workflow, specifications, and input parameters. Details on how to structure the configuration file are provided in Sec. 6. Upon execution, the `saved_models` will be populated with a separate folder for each classifier. Runs are organized according to the day and time of execution and the output consists of three files:

- `model.pkl` - the trained model,
- `training.log` - a log file that includes train and test scores, and
- `used_config.config` - a copy of the configuration file used to generate this output.

6 Configuration File

The configuration file consists of sections that contain arguments, illustrated below.

```
[SECTION_1]
ARGUMENT_1=100
ARGUMENT_2=200
```

```
ARGUMENT_3=300
ARGUMENT_4=400
ARGUMENT_5=500

[SECTION_2]
ARGUMENT_1=100
ARGUMENT_2=200
ARGUMENT_3=300
ARGUMENT_4=400
ARGUMENT_5=500
```

`[CUSTOMIZATION] Arguments are loaded as string variables and can be sensitive to extra spaces and line breaks. Additional processing (e.g., type casting, split, etc.) may be required when adding additional arguments.`

The sections enabled in the MIMOSAS configuration file are outlined in Table 2.

Section	Description	Required? (Y/N)
`[MAIN]`	General MIMOSAS operation and workflow settings	Y
`[TRAINING_DATA]`	Data import, preprocessing, and split options for training. The filepath for the labeled training dataset is specified in this section.	Y
`[TEST_DATA]`	Import, preprocessing, and split options for the test data. The filepath for the test dataset is specified in this section. The same dataset may be used for training and testing, but the user is cautioned to specify the appropriate split parameters.	Y
`[DECISION_TREE]`	Supervised classification model in the form a tree structure	N
`[RANDOM_FOREST]`	Supervised classification model ensemble comprised of multiple decision trees	N
`[FEED_FORWARD]`	Supervised feed-forward neural network classification model	N

Table 2: Sections supported in the MIMOSAS configuration file.

The arguments for the `[MAIN]`, `[TRAINING_DATA]`, `[TEST_DATA]`, `[DECISION_TREE]`, `[RANDOM_FOREST]`, and `[FEED_FORWARD]` sections are described in Tables 3, 4, 5, 6, 7, and 8, respectively.

Argument	Options	Description
`Save =`	`True, False`	If `True`, save model after training
`Verbose =`	`True, False`	If `True`, print and log verbose output
`Mode =`	`Train, Test`	`Train` performs model training and cross validation. `Test` uses a previously trained model specified in the model path of the [TEST] section for classification. `Train, Test` performs model training, cross validation, and tests.

Table 3: Arguments supported in the [MAIN] section of the MIMOSAS configuration file.

Argument	Options	Description
`Data =`	`String`	Path to labeled data.
`Background_Data =`	`String`	Path to background data. If `background_correction` is not specified in `Data_Options`, leave blank. For proper functionality, data and background data must have the same units.
`Background_Correction_Cols =`	`Dict`	`key, value` pairs of string literals where `key` is the column in `Data` that will be corrected by the column `value` in `Background_Data`. If `background_correction` is not specified in`Data_Options`, leave blank.
`Split_Fraction =`	`Float` $\in [0, 1]$	Fraction of data to go into the training set.
`Split_Seed =`	`Int`	Seed used for random shuffling of data prior to split.
`Data_Options =`	`standardize, minmaxscale, remove_outliers, background_correction`	Preprocessing options applied prior to shuffle and split. These are detailed in Sec. 7.
`Cols_To_Use =`	`String`	Column headings separated by commas indicating variables to use as input features.
`Label_Col =`	`String`	Column heading indicating variable to use as label.
`Cols_To_Standardize =`	`String`	Column headings separated by commas for variables to be standardized prior to input.
`Cols_To_MinMaxScale =`	`String`	Column headings separated by commas for variables to be min-max scaled prior to input.
`Remove_Outlier_Cols =`	`String`	Column headings separated by commas for variables to be used to reject outlier data.
`Outlier_MADs_Threshold =`	`Float`	Number of Mean Absolute Deviations (MADs) about the mean of features in `Remove_Outlier_Cols` outside which a sample is flagged for removal.
`Device_ID_Col =`	`String`	Column header indicating which device (if, e.g., the data were gathered by an array of sensors) collected each sample. Used to perform preprocessing tasks on a by-device basis. If only one device was used, leave blank.

Table 4: Arguments supported in the [TRAINING_DATA] section of the MIMOSAS configuration file.

Argument	Options	Description
`Data =`	`String`	Path to test data
`Background_Data =`	`String`	Path to background data. If `background_correction` is not specified in `Data_Options`, leave blank. For proper functionality, data and background data must have the same units.
`Background_Correction_Cols =`	`Dict`	`key, value` pairs of strings where `key` is the column in `Data` which will be corrected by the column `value` in `Background_Data`. If `background_correction` is not specified in `Data_Options`, leave blank.
`Split_Fraction =`	`Float` $\in [0, 1]$	Fraction of data to go into the test set
`Split_Seed =`	`Int`	Seed used for random shuffling of data prior to split. When using the same dataset for train and test, it's important to sequester the test data by using the same random seed.
`Data_Options =`	`standardize, minmaxscale, remove_outliers, background_correction`	Preprocessing options applied prior to shuffle and split. These are detailed in Sec. 7.
`Cols_To_Use =`	`String`	Column headings separated by commas indicating variables to use as input features
`Cols_To_Standardize =`	`String`	Column headings separated by commas for variables to be standardized prior to input
`Cols_To_MinMaxScale =`	`String`	Column headings separated by commas for variables to be min-max scaled prior to input
`Remove_Outlier_Cols =`	`String`	Column headings separated by commas for variables to be used to reject outlier data.
`Outlier_MADs_Threshold =`	`Float`	Number of Mean Absolute Deviations (MADs) about the mean of features in `Remove_Outlier_Cols` outside which a sample is flagged for removal.
`Device_ID_Col =`	`String`	Column header indicating which device (if, e.g., the data were gathered by an array of sensors) collected each sample. Used to perform preprocessing tasks on a by-device basis. If only one device was used, leave blank.

Table 5: Arguments supported in the [TEST_DATA] section of the MIMOSAS configuration file.

Argument	Options	Description
`Classifier_Criterion =`	`entropy, gini`	Splitting criterion metric at each non-terminal node.
`Feature_Selection =`	`True, False`	If `True`, determine feature importance using selected splitting criterion. See Sec. 8.
`Max_Depth =`	`Int`	Integers separated by commas representing maximum tree depth. The maximum tree depth will be optimized using cross validation.
`CV_Folds =`	`Int >1`	Number of groups that the training dataset is split into for k-fold cross validation
`Load_Model_Path =`	`String`	Path to previously trained model. If used in `Train` mode, the specified model will be further trained. Leave blank to train a new model. Required for `Test` only mode.

Table 6: Arguments supported in the [DECISION_TREE] section of the MIMOSAS configuration file.

Argument	Options	Description
`Classifier_Criterion =`	`entropy, gini`	Splitting criterion metric
`Feature_Selection =`	`True, False`	If `True`, determine feature importance using selected splitting criterion. See Sec. 8.
`N_Estimators =`	`Int`	Integers separated by commas representing number of trees. The number of trees will be optimized using cross validation.
`Depth =`	`Int`	Integers separated by commas representing maximum tree depth. The maximum tree depth will be optimized using cross validation.
`CV_Folds =`	`Int >1`	Number of groups that the training dataset is split into for cross validation
`Load_Model_Path =`	`String`	Path to previously trained model. If used in `Train` mode, the specified model will be further trained. Leave blank to train a new model. Required for `Test` only mode.

Table 7: Arguments supported in the [RANDOM_FOREST] section of the MIMOSAS configuration file.

Argument	Options	Description
Feature_Selection =	True, False	If True, determine feature importance using random feature addition and random feature elimination. See Sec. 8.
Features_To_Select =	Int	Number of features to select
Layers =	[list[list[(Int)]]]	List of lists; Each sub-list contains positive integers indicating the number of nodes in the feed-forward neural network's fully-connected dense hidden layers. For example, $[[8, 6, 4], [8, 4]]$ will test networks of two sizes. The first has three hidden layers with 8, 6, and 4 nodes in hidden layers one, two, and three, respectively. The second has two hidden layers with 8 nodes in first and 4 in the second.
Activation_Fns =	[list[(Str)]]	Activation function(s) to use in the networks hidden layers. Default is 'elu,' the Exponential Linear Unit. A full list is available at https://keras.io/activations/. Alternatively, the user can define a function to be used at node activations.
Dropout_Rates =	[list[(Float)]]	Floating point(s) on [0.0, 1.0) indicating the fraction of nodes at each layer that should be made inactive during each training epoch [1].
Loss_Fns =	[list[(Str)]	Function(s) used to calculate loss based on discrepancy between predicted and true outputs of the model during training. This loss is iteratively minimized during training. A full list of options is available at https://keras.io/losses/. Alternatively, the user can define a function to be used to calculate loss.
Optimizers =	[list[(str)]]	Algorithm(s) that define the process of network parameters adjustment during the training process. A full list of options is available at https://keras.io/optimizers/.
Learning_Rates =	[list[(float)]]	Initial learning rate(s) which scale the rate at which network parameters are adjusted during training. Note: Some optimization algorithms dynamically scale learning rates during training.
Max_Epochs =	[list[(int)]]	Positive integer(s) indicating the maximum number of times the training set is fed into the network during training.

Continued on next page

Continued from previous page		
`Batch_Sizes =`	`[list[(int)]]`	Positive integer(s) indicating the number of training set events the network is scored on between each parameter adjustment during the training process. For a training set with N examples, there will be N/Batch_Size batches per epoch.
`CV_Folds =`	`Int`	Number of groups that the training set is split into for cross validation during hyperparameter optimization
`Load_Model_Path =`	`String`	Path to previously trained model. If used in `Train` mode, the specified model will be further trained. Leave blank to train a new model. Required for `Test` only mode.

Table 8: Arguments supported in the [FEED_FORWARD] section of the MIMOSAS configuration file.

7 Preprocessing Options

Several algorithms are included as `Data_Options` for both train and test data preprocessing. These are detailed in turn below.

7.1 Standardize

The `standardize` algorithm adjusts each input feature specified in `Cols_To_Standardize` using Z-score normalization. This is accomplished by individually calculating the mean, μ, and standard deviation, σ, of each input feature distribution. Then, each value, x, is adjusted as follows:

$$x' = \frac{x - \mu}{\sigma}, \tag{1}$$

where x' is the scaled value. The result is an input feature distribution centered about zero and scaled to unit variance.

7.2 Min-Max Scale

The `minmaxscale` algorithm adjusts each input feature specified in `Cols_To_MinMaxScale` individually such that it is in the range between zero and one. This transformation is given by:

$$x' = \frac{x - min}{max - min} \tag{2}$$

where x is the original value, x' is the scaled value, and max and min are the maximum and minimum values in the feature range, respectively. The result is an input feature distribution that ranges from zero to one.

7.3 Remove Outliers

The `remove_outliers` algorithm is used to remove outlier data from the input features specified in `Remove_Outlier_Cols`. Outliers are identified based on the number of mean absolute deviations

(MADs) about the mean of the feature distribution. The MAD is given by:

$$\text{MAD} = \frac{1}{n}\sum_{i=1}^{n} \mid x_i - \mu \mid, \tag{3}$$

where μ is the mean of the input feature distribution, x_i is each individual value in the distribution, and n is the total number of values for a given feature. The parameter `Outlier_MADs_Threshold` is used to specify the number of MADs from the mean a value must be for it to be considered an outlier.

7.4 Background Correction

For the input features listed in `Background_Correction_Cols`, the `background_correction` algorithm piece-wise linearly interpolates background data specified in `Background_Data` and subtracts the interpolated values from the raw input data. The user is cautioned to ensure that the same units are used for `Data` and `Background_Data`.

8 Feature Selection

MIMOSAS includes methods to evaluate feature importance to assist in feature selection. For the decision tree and random forest classifiers, the feature importance is determined based on the decrease in node impurity and the fraction of input samples a feature contributes to in the prediction decision [2]. For the feed-forward neural network, recursive feature elimination [3] and recursive feature addition [4] are used with features ranked according to the Matthews Correlation Coefficient [5]. Users are encouraged to use this output to optimize `Cols_To_Use` in the workflow.

9 Documentation

To compile the software reference documentation, the Doxygen package must be installed [6]. From the main directory, issue the following command:

```
doxygen Doxyfile
```

The folder *docs* contains the HTML documentation source pages. After generation, the documentation can be viewed by opening the *index.html* in the *docs* directory using an HTML browser. The documentation can also be viewed online here: `https://nonproliferation.github.io/mimosas/`.

10 License

Created by Jared Zhao, Bethany L. Goldblum, Christopher Stewart, Alicia Ying-Ti Tsai, Shruthi Chockkalingam, and Pedro Vicente Valdez, Department of Nuclear Engineering, University of California, Berkeley.

11 Acknowledgements and Disclaimer

The project was funded by the U.S. Department of Energy, National Nuclear Security Administration, Office of Defense Nuclear Nonproliferation Research and Development (DNN R&D).

References

[1] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," Journal of Machine Learning Research, 15 (2014) 1929-1958. `http://jmlr.org/papers/v15/srivastava14a.html`

[2] G. Louppe, "Understanding Random Forests: From Theory to Practice," PhD Thesis, U. of Liege, 2014. `https://arxiv.org/abs/1407.7502`

[3] A. Altmann, L. Tolosi, O. Sander, and T. Lengauer, "Permutation importance: a corrected feature importance measure," Bioinformatics, vol. 26, no. 10, pp. 1340 − 1347, 2010.

[4] T. Hamed, "Recursive Feature Addition: a Novel Feature Selection Technique, Including a Proof of Concept in Network Security," Doctoral Dissertation, The University of Guelph, 2017.

[5] B. Matthews, "Comparison of the predicted and observed secondary structure of t4 phage lysozyme," Biochimica et Biophysica Acta (BBA) - Protein Structure, vol. 405, no. 2, pp. 442−451, 1975. `http://www.sciencedirect.com/science/article/pii/0005279575901099`

[6] Doxygen, `http://www.doxygen.nl/`