ClassificaIO: machine learning for classification graphical user interface

Raeuf Roushangar[1,2], George I. Mias[1,2,*]

[1]Department of Biochemistry and Molecular Biology,

[2]Institute for Quantitative Health Science and Engineering,

Michigan State University, East Lansing MI 48824, USA

Abstracts

Background: Machine learning methods and algorithms have been used routinely by scientists in many research areas. However, significant statistical and programing expertise are typically required to use these methods and algorithms. Thus, an easy-to-use graphical user interface software that enables biomedical researchers wanting to apply machine learning methods in their research in biology is essential, and can facilitate further use of such methodology.

Results: Here we present ClassificaIO, an open-source Python graphical user interface for machine learning classification for the scikit-learn Python module. ClassificaIO aims to provide an easy-to-use interactive way to train, validate, and test data on a range of state-of-the-art classification algorithms. The software enables fast comparisons within and across classifiers, and facilitates uploading and exporting of trained models, and both validated, and tested data results. ClassificaIO is implemented as a Python package and is available for download and installation through the Python Package Index (PyPI) (http://pypi.python.org/pypi/ClassificaIO) and it can be deployed using the "import" function once installed. The software is distributed under an MIT license and source code is available for download (for Mac OS X, Linux and Microsoft Windows) through PyPI and GitHub (http://github.com/gmiaslab/ClassificaIO), and at https://doi.org/10.5281/zenodo.1133266.

Conclusions: ClassificaIO facilitates the use of machine learning algorithms through a graphical user interface (GUI), and can help biomedical and other researchers with broad machine learning

background to use machine learning, and apply it to their research in a simple and interactive point-and-click way.

SUPPLEMENTARY CONTENTS

TABLE S1: Classification Algorithms 3

SUPPLEMENTARY INFORMATION: ClassificaIO Software Manual..... 4

ATTACHMENTS: SUPPLEMENTARY FILES (Files S1-7 available with manuscript).

	File Name	***Description***
1.	*S1_Iris_Dependent_DataSet.csv*	Iris data set (150 data points)
2.	*S2_Iris_Target.csv*	Iris Target data set (150 labels)
3.	*S3_Iris_Testing_DataSet.csv*	Iris Testing data set (150 data points)
4.	*S4_Iris_FeatureNames.csv*	Example Iris features (2 features: sepal length and petal width)
5.	*S5_LogisticRegression_IrisTrainedModel.pkl*	Example ClassificaIO trained model using logistic regression
6.	*S6_IrisTrainValidationResult.csv*	Example ClassificaIO testing result using logistic regression
7.	*S7_IrisTestingResult.csv*	Example ClassificaIO validation result using logistic regression

All files below (S8-S14) are referenced in the manual, and are available online at GitHub (due to journal file size restrictions):
(https://github.com/gmiaslab/manuals/tree/master/ClassificaIO/Supplementary%20Files)

8.	*S8_GPL570_GSE99039_Dependent_DataSet.csv*
9.	*S9_GPL570_GSE99039_Target.csv*
10.	*S10_Y_ChromosomeGenes_FeatureNames.csv*
11.	*S11_GPL570_GSE18781_Testing_DataSet.csv*
12	*S12_LinearSVC_GPL570_GSE99039TrainedModel.pkl*
13	*S13_GPL570_GSE99039TrainValidationResult.csv*
14	*S14_GPL570_GSE18781TestingResult.csv*

CLASSIFIER	Scikit-learn FUNCTION USED	IMMUTABLE PARAMETERS
1: Logistic regression	LogisticRegression	class_weight = None
2: Passive Aggressive	PassiveAggressiveClassifier	class_weight = None n_iter= None
3: Perceptron	Perceptron	class_weight = None
4: Classifier using Ridge regression	RidgeClassifier	class_weight = None
5: Stochastic Gradient Descent - SGD	SGDClassifier	—
6: Linear Discriminant Analysis	LinearDiscriminantAnalysis	shrinkage= None priors = None
7: Quadratic Discriminant Analysis	QuadraticDiscriminantAnalysis	store_covariances = None priors = None
8: Linear Support Vector	LinearSVC	class_weight = None
9: Nu-Support Vector	NuSVC	class_weight = None
10: C-Support Vector	SVC	class_weight = None
11: k-Nearest Neighbors	KNeighborsClassifier	metric_params = None
12: Nearest centroid	NearestCentroid	—
13: Radius Nearest Neighbors	RadiusNeighborsClassifier	metric_params = None
14: Gaussian Process Classification (GPC)	GaussianProcessClassifier	kernel = None
15: Naive Bayes for Multivariate Bernoulli Models	BernoulliNB	class_prior = None
16: Gaussian Naive Bayes	GaussianNB	class_prior = None
17: Naive Bayes for Multinomial Models	MultinomialNB	class_prior = None
18: Decision Tree	DecisionTreeClassifier	class_weight = None
19: Extremely Randomized Tree	ExtraTreeClassifier	min_impurity_split = None class_weight = None
20: AdaBoost	AdaBoostClassifier	base_estimator = None
21: Bagging	BaggingClassifier	base_estimator = None
22: Extra Trees	ExtraTreesClassifier	class_weight = None
23: Random Forest	RandomForestClassifier	class_weight = None
24: Label Propagation	LabelPropagation	—
25: Neural network Multi-layer Perceptron	MLPClassifier	—

Table S1. A list of all 25 classification algorithms, their corresponding scikit-learn functions, and immutable (unchangeable) parameters with their default values.

ClassificalO

Machine Learning for Classification Graphical User Interface User Manual 1.1.2 (05/2018)

Summary:

ClassificaIO is an open-source Python graphical user interface (GUI) for supervised machine learning classification for the scikit-learn module [1]. ClassificaIO aims to provide an easy-to-use interactive way to train, validate, and test data on a range of classification algorithms. The GUI enables fast comparisons within and across classifiers, and facilitates uploading and exporting of trained models, and both validated, and tested data results.

Dependencies:

ClassificaIO is a Python package with the following external dependencies:

- Tkinter ≥ 8.6.7, Pillow ≥ 5.1.0, pandas == 0.22.0, numpy ≥ 1.14.3, scikit-learn ≥ 0.19.1, and scipy ≥ 1.1.0

Prerequisites:

ClassificaIO requires Python version 3.6 or higher and can be used on Mac OS X High Sierra, Linux (tested on Ubuntu), and Windows 10 operating systems. To avoid any system errors, crashes, and crude fonts, we recommend to not install ClassificaIO using integrated environment package installers – i.e. native installation of ClassificaIO is highly encouraged using pip. In case you do not have pip installed, you must install it first.

Installation Instructions:

1. Mac or Windows

To install the current release use pip in the terminal:

```
$ pip install ClassificaIO
```

Alternatively, you can install directly from github using:

```
$ pip install git+https://github.com/gmiaslab/ClassificaIO/
```

2. Linux

First install the current release of tkinter and pip:

```
$ sudo apt-get install python3-tk
$ sudo apt-get install python3-pip
```

To install the current ClassificaIO realease use pip:

```
$ pip3 install ClassificaIO
```

Alternatively, you can install directly from github using:

```
$ pip install git+https://github.com/gmiaslab/ClassificaIO/
```

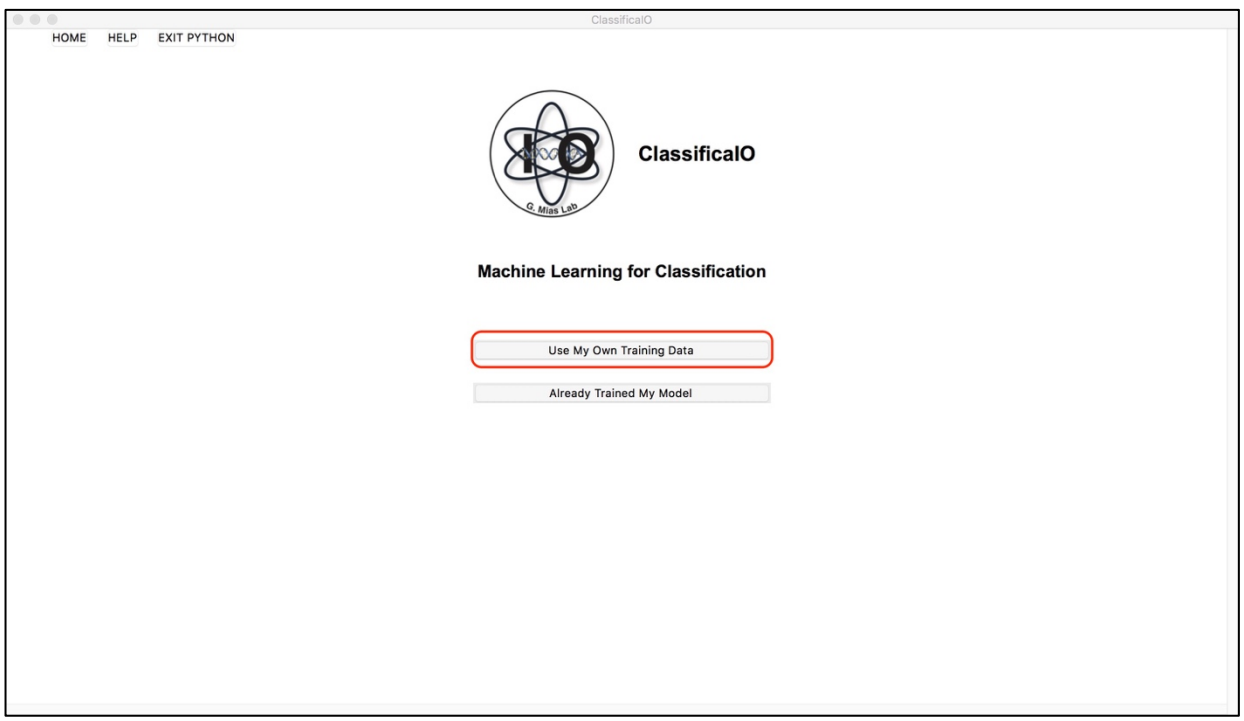
Getting started:

Please Note:

- ClassificaIO supports comma-separated values (CSV) input files only.
- In this document we use the machine learning Iris dataset [2, 3] (150 data points) as an example to illustrate the interface, to demonstrate model training, validation, and testing, as well as the data formats that ClassificaIO relies on.
- In the classification example below, we use 70% of the Iris dataset (105 data points) for model training and 30% (45 data points) for model testing.
- You can download the Iris and the processed gene expression datasets, as well as all files used in this manual from: https://github.com/gmiaslab/manuals/tree/master/ClassificaIO/Supplementary%20Files

After installing ClassificaIO, please run it from the terminal using Python:

```
$ python3
>>> from ClassificaIO import ClassificaIO
>>> ClassificaIO.gui()
```

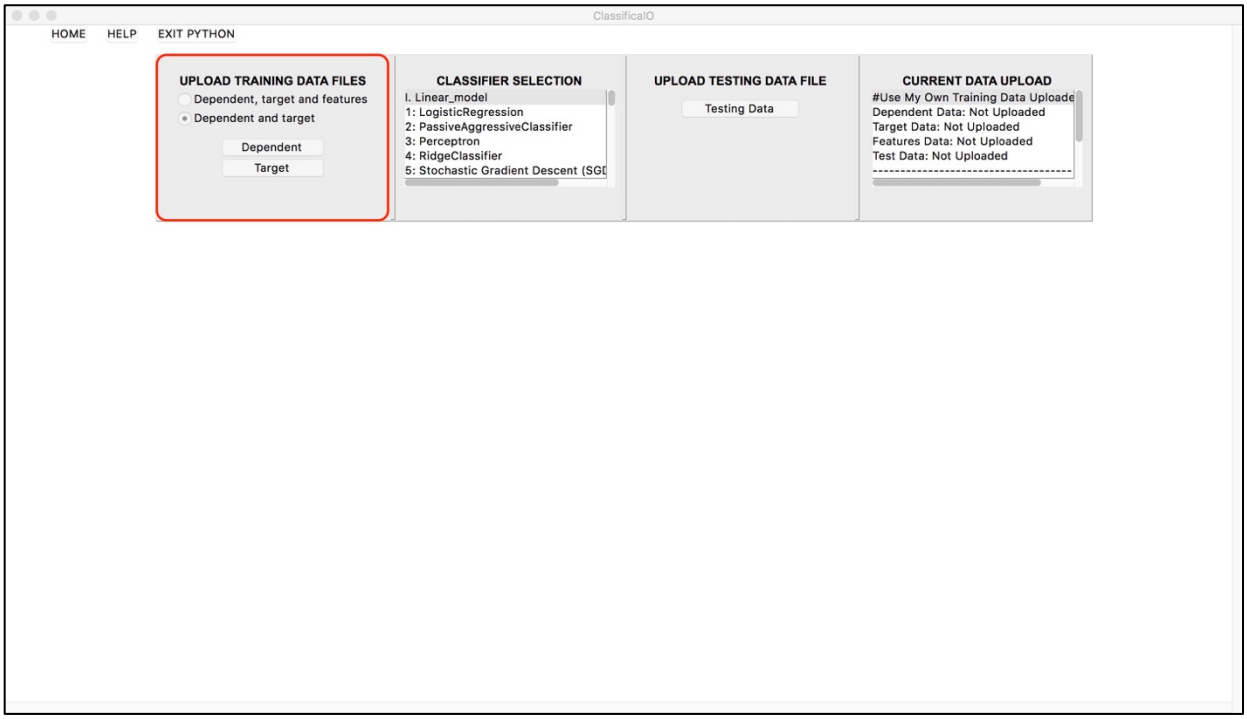


We note here the name is case sensitive (i.e. the 'IO' is capitalized). Once ClassificaIO's main window appears on your screen, you can click on 'Use My Own Training Data' button and start your supervised machine learning classification project.

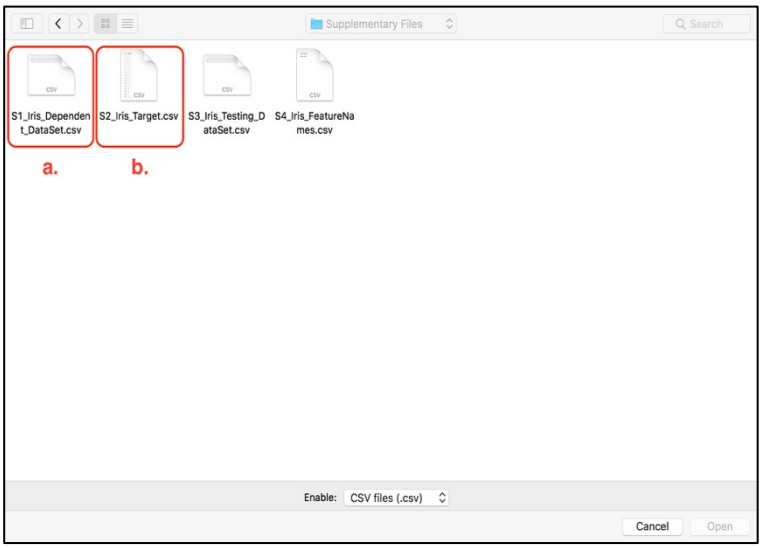
Iris dataset prediction using a logistic regression classifier:

<u>Training data input:</u>

You first need to make a selection (either ‘Dependent and Target’ or ‘Dependent, Target and Features’) from the ‘UPLOAD TRAINING DATA FILES’ panel to upload training data files. For this example, we select the ‘Dependent and Target’ button.

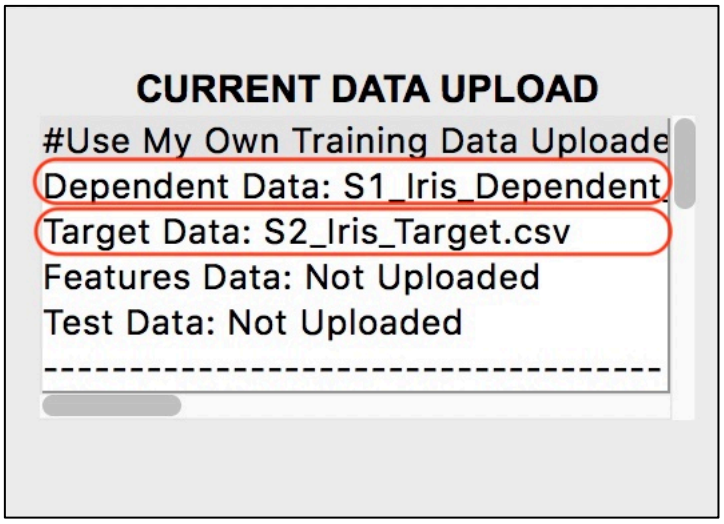


To begin uploading data files, click the corresponding buttons in the 'UPLOAD TRAINING DATA FILES' panel: a file selector directs you to upload both, dependent data file (**Supplementary Figure 1.a**) and target data file (**Supplementary Figure 1.b**). Once a file is uploaded to ClassificaIO, the file name and directory are automatically saved in the 'CURRENT DATA UPLOAD' panel (**Supplementary Figure 2**). This updatable log allows for tracking current data files in use, and maintains a history of all files uploaded to the software.



Supplementary Figure 1. Graphical Control Element Dialog Box. a. Dependent data file selected for upload. **b.** Selected target data file to upload. N.B. each file selection has to be done one at a time.

Supplementary Figure 2. Current Data Upload Panel. Both dependent and target data file names shown (red boxes). Scroll down for uploaded data files directories.



Data format:

Data formats are shown in **Supplementary Figure 3.a** for dependent data and **Supplementary Figure 3.b** for target data. The dependent data represent the data on which the model will depend on for learning and the target data is the annotation, i.e. what is going to be predicted. In this example, the dependent data have 4 rows and 105 columns. For the dependent data, each row is an attribute (also known as feature) and each column is an object (also known as an observation or a sample). Thus, the header row enumerates the objects, and the header column names the attributes. The values in the file represent the measurement made for each of the objects(columns) for each of the attributes (rows).

Objects

Attributes

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1		1	2	5	6	7	11	12	14	16	18	20	21	22	23	24
2	sepal length	5.1	4.9	5	5.4	4.6	5.4	4.8	4.3	5.7	5.1	5.1	5.4	5.1	4.6	5.1
3	sepal width	3.5	3	3.6	3.9	3.4	3.7	3.4	3	4.4	3.5	3.8	3.4	3.7	3.6	3.3
4	petal length	1.4	1.4	1.4	1.7	1.4	1.5	1.6	1.1	1.5	1.4	1.5	1.7	1.5	1	1.7
5	petal width	0.2	0.2	0.2	0.4	0.3	0.2	0.2	0.1	0.4	0.3	0.3	0.2	0.4	0.2	0.5

Supplementary Figure 3.a Dependent Data. Example of partial dependent data file format. Testing data (not shown) uses the same format.

For the target data, we have 105 rows and 2 columns (note: for the target data, the rows correspond to the objects and column is the class per object). The values in the "ids" column in the target data must much the Objects header row in the dependent data, the columns headers must match, otherwise an error will occur. Hence, the number of columns (i.e. objects) in the dependent data must also match the number of rows in the target data (i.e. each object has a unique "id" and must be assigned a target class for training). Finally, the "target" column in the target data must be numerically-valued.

Attributes

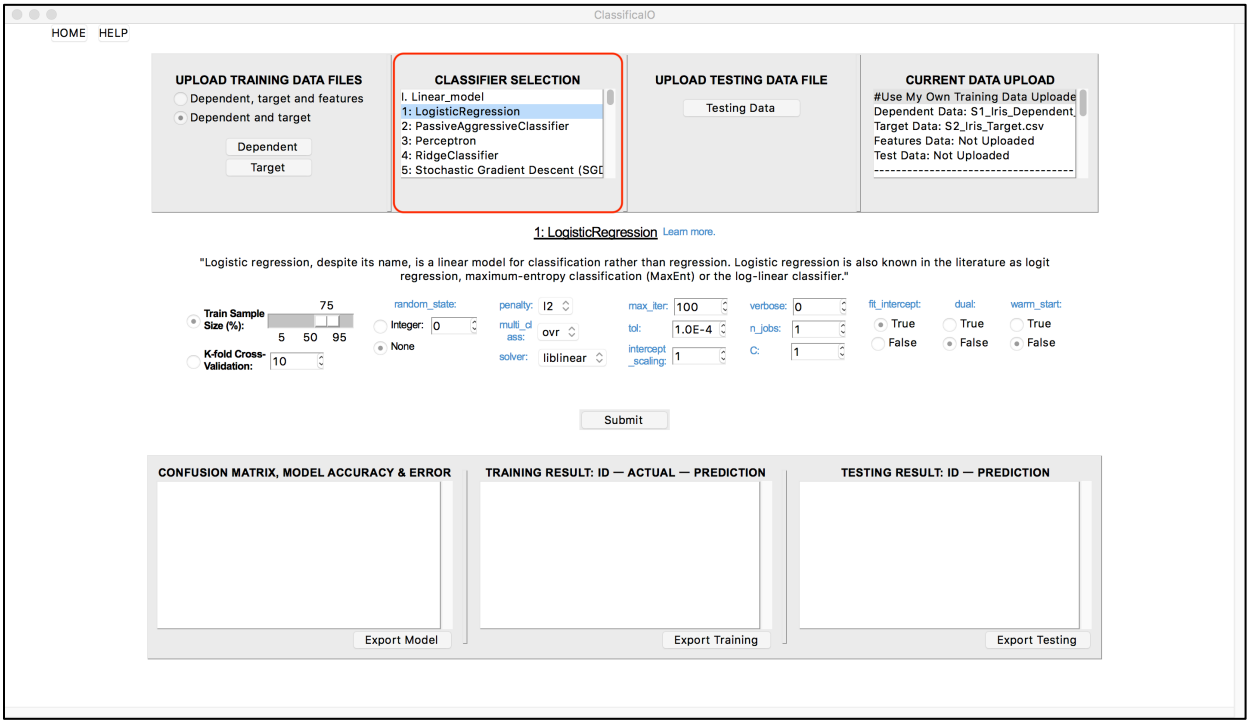
Objects

	A	B
1	id	target
2	1	0
3	2	0
4	5	0
5	6	0
6	7	0
7	11	0
8	12	0
9	14	0
10	16	0
11	18	0
12	20	0
13	21	0
14	22	0
15	23	0
16	24	0

Supplementary Figure 3.b. Target Data. Example of partial target data file format where the targets correspond to setosa = 0, versicolor = 1, and virginica = 2. Versicolor and virginica are not visible in this screenshot.

Classifier selection:

Once you have uploaded all required training data files, you can select between 25 different machine learning classification algorithms in the 'CLASSIFER SELECTION' panel.



Here are all classification algorithms in order of appearance in the 'CLASSIFER SELECTION' panel. Immutable (unchangeable) parameters with their default values are also listed for each classifier in the parentheses:

I. Linear_model

1: LogisticRegression. (class_weight = None)
2: PassiveAggressiveClassifier. (class_weight = None, n_iter= None)
3: Perceptron. (class_weight = None)
4: RidgeClassifier. (class_weight = None)
5: Stochastic Gradient Descent (SGDClassifier).

II. Discriminant_analysis

6: LinearDiscriminantAnalysis. (shrinkage= None, priors = None)
7: QuadraticDiscriminantAnalysis. (store_covariances = None, priors = None)

III. Support vector machines (SVMs)

8: LinearSVC. (class_weight = None)
9: NuSVC. (class_weight = None)
10: SVC. (class_weight = None)

IV. Neighbors

11: KNeighborsClassifier. (metric_params = None)
12: NearestCentroid.
13: RadiusNeighborsClassifier. (metric_params = None)

V. Gaussian_process

14: GaussianProcessClassifier. (kernel = None)

VI. Naive_bayes

15: BernoulliNB. (class_prior = None)
16: GaussianNB. (class_prior = None)
17: MultinomialNB. (class_prior = None)

VII. Trees

18: DecisionTreeClassifier. (class_weight = None)
19: ExtraTreeClassifier . (min_impurity_split = None, class_weight = None)

VIII. Ensemble

20: AdaBoostClassifier. (base_estimator = None)
21: BaggingClassifier. (base_estimator = None)
22: ExtraTreesClassifier. (class_weight = None)
23: RandomForestClassifier. (class_weight = None)

IX. Semi_supervised

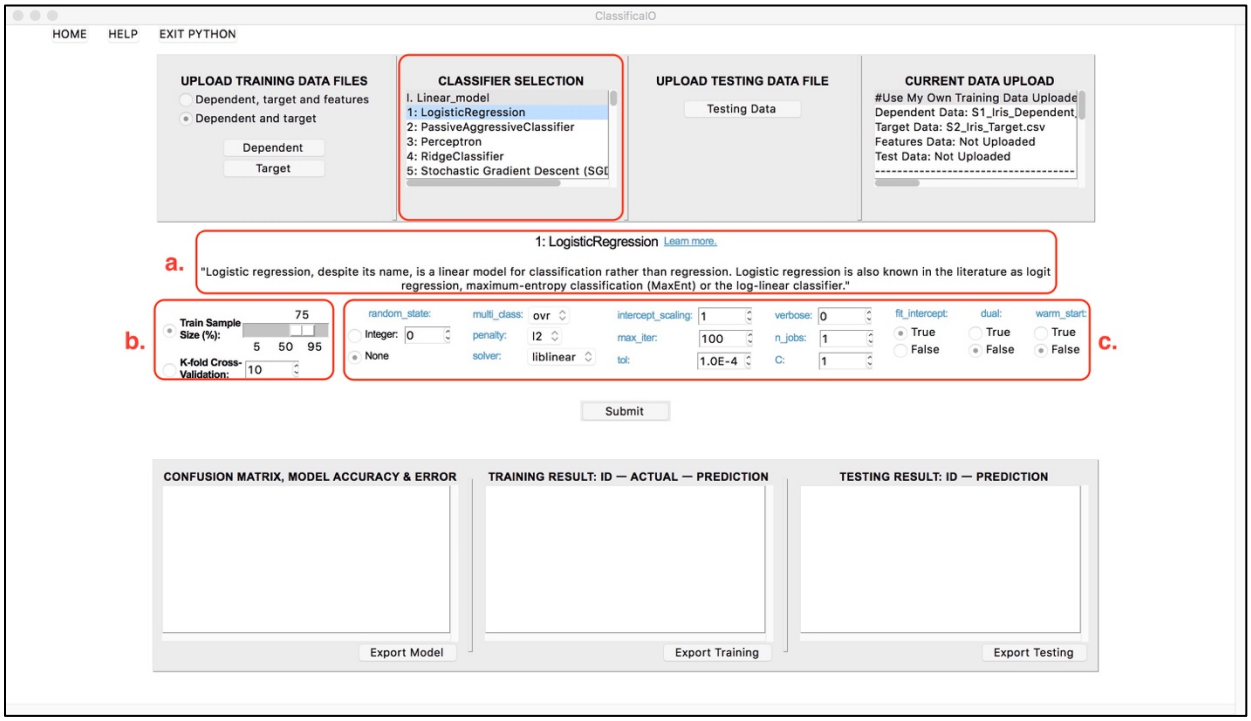
24: LabelPropagation.

X. Neural_network

25: MLPClassifier.

The following will populate once you make a classifier selection:

- **Supplementary Figure 4.a**: The classifier definition with a clickable underlined link “learn more” in blue, which, when clicked opens an external web-browser to the scikit-learn documentation for the selected classifier.
- **Supplementary Figure 4.b**: Interactive way to select between train-validate split and cross-validation methods (radio buttons), which are necessary to prevent/minimize training model overfitting.
- **Supplementary Figure 4.c**: classifier parameters, to provide you with a point-and-click interface to set, modify, and test the influence of each parameter on your data



Supplementary Figure 4. Selected Logistic Regression Classifier. The interface for each selected classifier, has uniform features: a. Classifier definition is displayed, together with an underlined clickable link that reads “Learn more” next to the classifier name. **b.** Training methods with ‘Train Sample Size (%)’ method selected. **c**. The classifier parameters set to their default values.

Model training, evaluation, validation and result output:

You can now click 'submit' to train your classifier using the uploaded training data files 'Dependent and Target' in this example, and evaluate your result. Or, alternatively you can upload testing data first, and then click 'submit' to train and test a classifier on your uploaded data at the same time! For this example, **first**: we train a selected classifier, 'LogisticRegression', using its default parameters, and default train-validate split method 'Train Sample Size (%)', and then, **second**: we upload testing data to test the trained model.

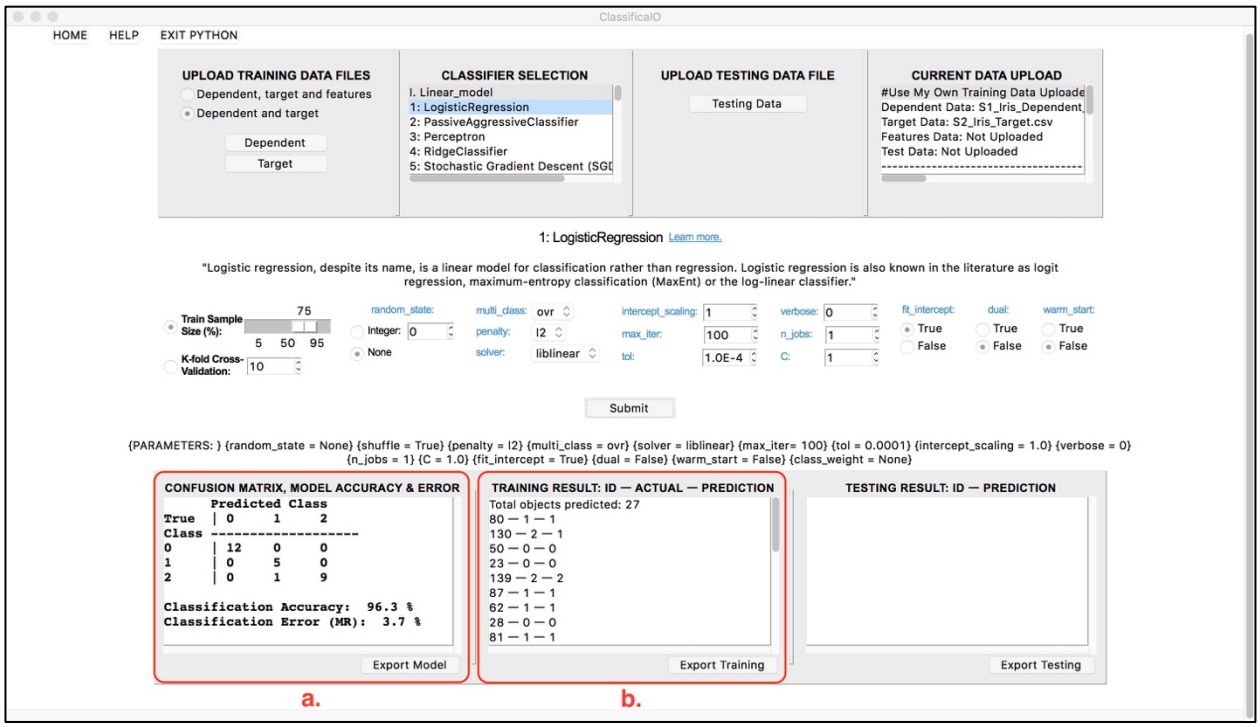
After clicking 'submit', our selected classifier, 'LogisticRegression' for this example, is trained using the loaded training data, 'Dependent and Target' for this example.

Notes
ClassificaIO always shuffles your training data before splitting to eliminate mini batch effects.

Internally, when 'Train Sample Size (%)' method is selected, ClassificaIO uses the scikit-learn *train_test_split* method, to allow for fast training data split into training and validation subsets. With this method the parameter set is *train_size*, which takes the train sample size set by you (e.g. Train Sample Size (%): set to 75% means *train_size* = 0.75 and *test_size* = 0.25).

If the 'K-fold Cross-Validation' method is selected instead, ClassificaIO uses the scikit-learn *cross_val_predict* method where the training data is split into k-sets. The model is trained on k-1 of the folds followed by a validation step on the remaining part of the data. This will be repeated for each of the k-folds.

After training is completed, the confusion matrix, classifier accuracy and error are displayed in the 'CONFUSION MATRIX, MODEL ACCURACY & ERROR' panel (**Supplementary Figure 5.a**). Model validation data results are displayed in the 'TRAINING RESULT: ID – ACTUAL – PREDICTION' panel (**Supplementary Figure 5.b**) with each data point ID is the 1st value, actual target value is displayed 2nd, and predicted target value 3rd , where the predictions correspond to the iris flower species, with 0=setosa, 1=versicolor, and 2=virginica.

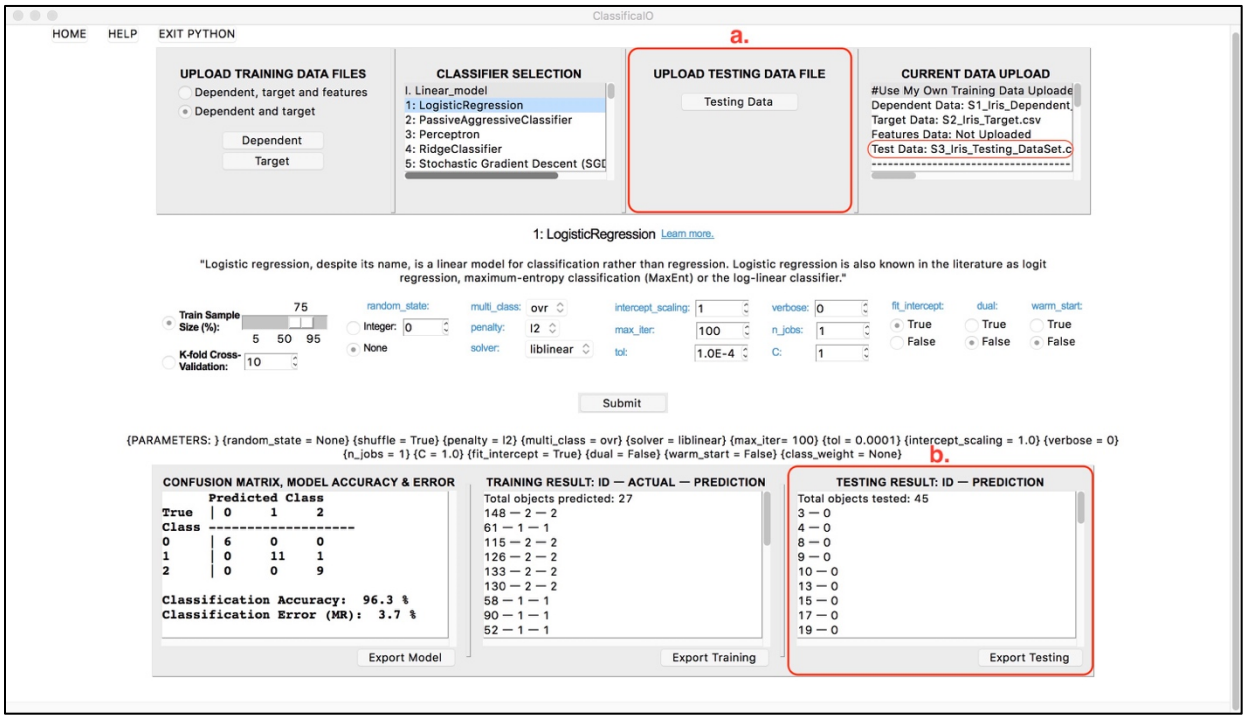


Supplementary Figure 5. Trained Logistic Regression Classifier. a. Trained model using 78 data points (75% of 105 data points), classifier evaluation (confusion matrix, model accuracy and error). **b**. Model validated using 27 data points (25% of 105 data points).

Testing data input and result output:

To test your trained model, first upload the testing data file by clicking the 'Testing Data' button in the 'UPLOAD TESTING DATA FILE' panel (**Supplementary Figure 6.a**). Once clicked, a file selector directs you to upload the testing data file, see **Supplementary Figure 1**. Once testing data is uploaded, the file name is automatically saved in the 'CURRENT DATA UPLOAD' panel (outlined in the red box in the figure) to indicate that your file has been uploaded. The Testing Data file format is the same as for the dependent data file, see **Supplementary Figure 3.a**.

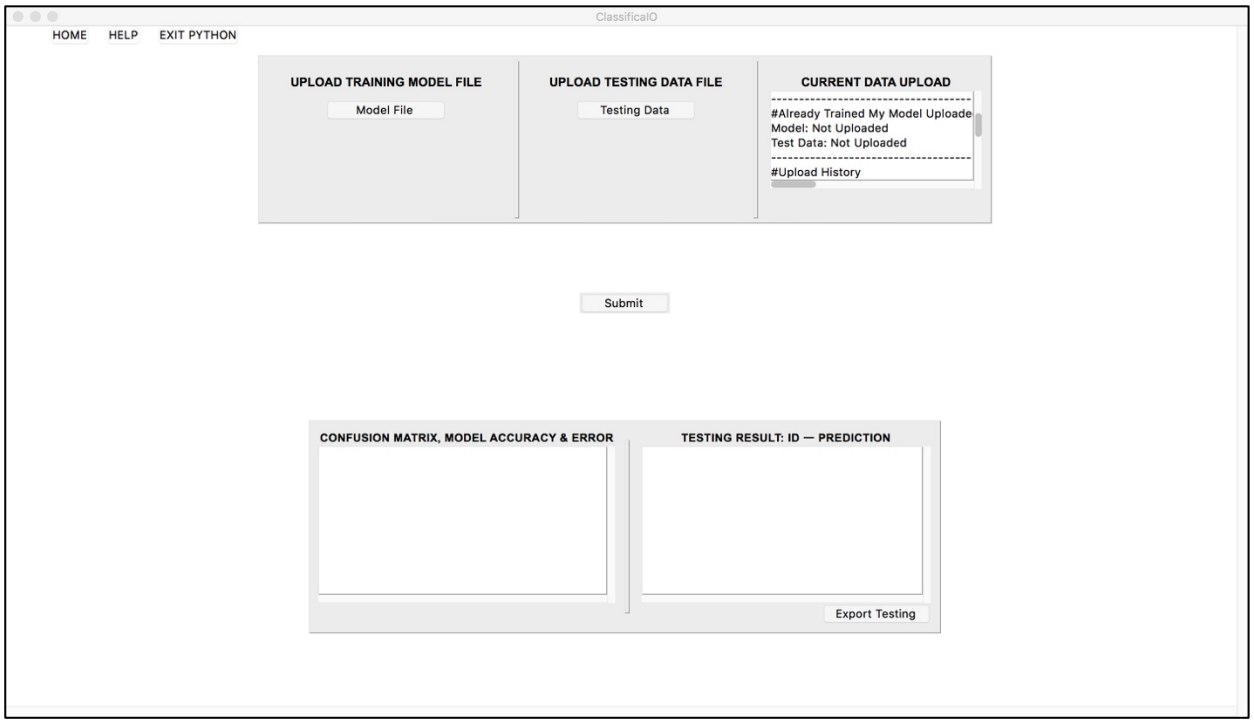
After clicking 'Submit', testing results are displayed in the 'TESTING RESULT: ID – PREDICTION' panel (**Supplementary Figure 6.b**) with each data point ID shown 1st, and the corresponding predicted target value displayed after it 2nd, separated by a hyphen.



Supplementary Figure 6. Tested Logistic Regression Classifier. a. Upload testing data panel. **b**. Model tested using 45 data points.

Result export:

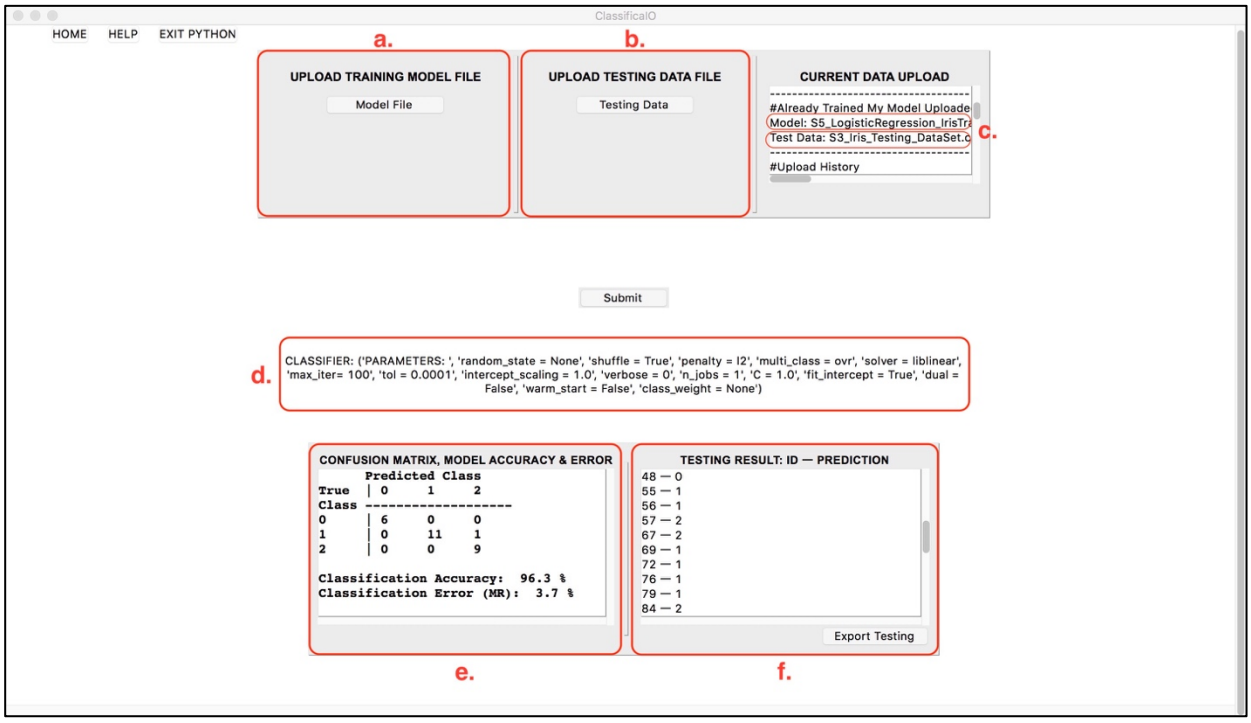
Now you are ready to export your trained model to preserve it for future use without having to retrain. Simply, click the 'Export Model' button (**Supplementary Figure 5.a**) and save your model. Your exported ClassificaIO model can then be used for future testing on new data in the 'Already Trained My Model' window in ClassificaIO, shown below.



ClassificaIO model input:

You will need to upload ClassificaIO model by clicking the 'Model File' button in the 'UPLOAD TRAINING MODEL FILE' panel (**Supplementary Figure 7.a**). Once clicked, a file selector directs you to upload a ClassificaIO trained model. Also, you will need to upload a testing data file (the testing data file format is the same as explained above), by clicking the 'Testing Data' button in the "UPLOAD TESTING DATA FILE" panel (**Supplementary Figure 7.b**). Once a ClassificaIO model and testing data files are uploaded, files names are automatically displayed in the 'CURRENT DATA UPLOAD' panel (**Supplementary Figure 7.c**).

After clicking 'submit', the uploaded model preset parameters will populate (**Supplementary Figure 7.d**) to show the classifier used to originally train the uploaded model. The confusion matrix, classifier accuracy and error of trained model are then displayed in the 'CONFUSION MATRIX, MODEL ACCURACY & ERROR' panel (**Supplementary Figure 7.e**). Testing data results are displayed in the 'Testing RESULT: ID – PREDICTION' panel (**Supplementary Figure 7.f**) with the data point ID shown 1st, followed by a hyphen and the predicted value displayed right after it.



Supplementary Figure 7. 'Already Trained My Model' window a. Upload ClassificaIO trained model panel. **b**. Upload testing data panel. **c.** Current data upload panel with both model and testing data files names shown (red boxes). **d.** Model preset parameters. **e**. Trained model result and model evaluation (confusion matrix, model accuracy and error). **f**. Model testing result.

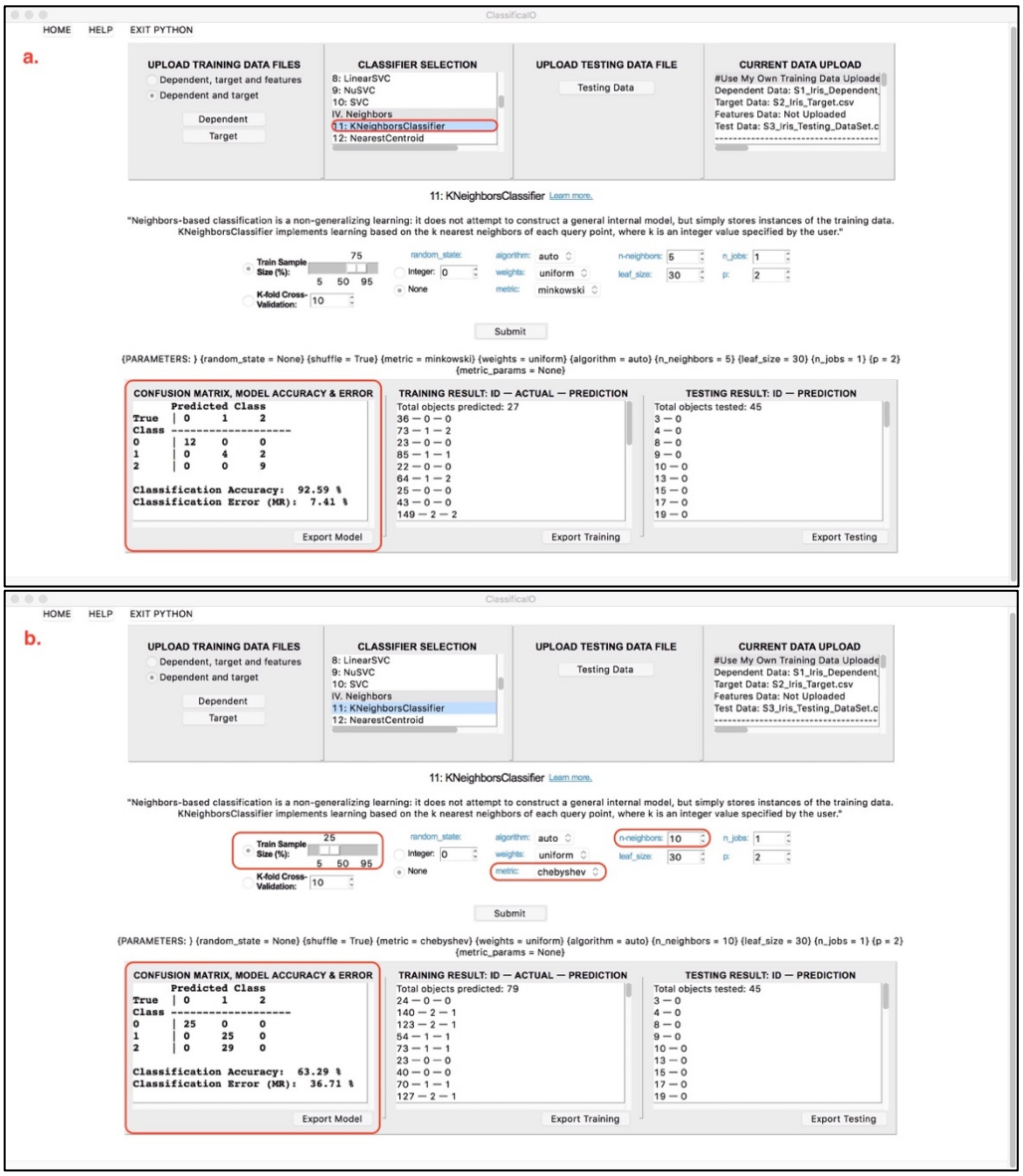
Results Export:

Full results (trained models, and both trained/validated, and tested data) for both windows (**'Use My Own Training Data' and 'Already Trained My Model'**) can be exported as CSV files for later use, e.g. further analysis, publication, sharing, etc. (for an example on the export data file formats, see the **Additional files S6 and S7**).

Additional Examples:

Ex1. Iris dataset prediction using k-nearest neighbors classifier:

Here we follow the same process as above but select k-nearest neighbors classification algorithm 'KNeighborsClassifier' rather than logistic regression. The Classification Accuracy shown here to be 92.59 % with low classification error. When the train Sample Size (%) change to 25% means *train_size* = 0.25 and *test_size* = 0.75, metric set to Chebyshev for distance metric, and n-neighbors to 10 for number of neighbors, the Classification Accuracy goes down to 63.29 % with much higher classification error. Please note that you may get different results when running the program, given the random selection of samples used for training/validation.



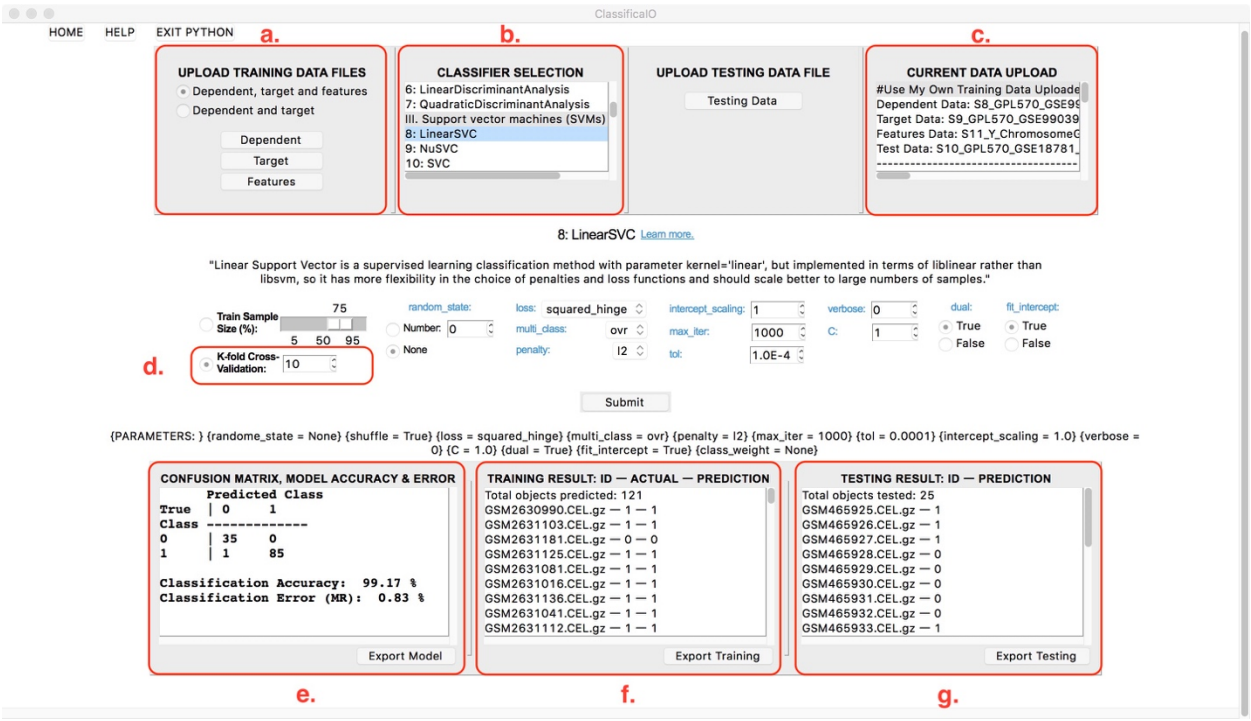
Supplementary Figure 8. a. selected k-nearest neighbors classifier with trained and tested the data using the default parameters values, **b.** same classifier selected with trained and tested data but using different parameters values.

Ex. 2. Gene expression sex prediction using linear support vector classifier:

In this example we use gene expression data from microarrays to predict the sex of the donor who provided the experimental sample. This is useful when the sex of the donor is unknown, and the information is needed for metadata analysis. We use two raw microarray gene expression datasets obtained from the Gene Expression Omnibus (GEO), GSE99039[4], 121 samples for training and GSE18781[5], 25 samples for testing. The data were generated using Affymetrix Human Genome U133 Plus 2.0 Array, (GPL570). Gene expression arrays need to first be processed and normalized: We first preprocessed all datasets (microarray raw CEL files) separately using the RMA (Robust Multi- Array Average) algorithm [6] implemented in R. Preprocessing consisted of performing background noise correction, quantile normalization of the intensities to obtain the same distribution of intensities across arrays, and summarization using median polish procedure to summarize the data in each probe set to obtain a single value corresponding to the gene expression of a probe set. The expression data were logarithmically transformed (base 2 log), and exported as CSV files.

We select 'Dependent, target and features' button from the 'UPLOAD TRAINING DATA FILES' (**Supplementary Figure 9.a**) and select 'LinearSVC' as classifier (**Supplementary Figure 9.b**). We use the buttons to upload the data: GSE32140 data were used for the Dependent Data button (Supplementary File S8), with Target data using values as annotated on GEO (Supplementary File S9), Y chromosomes genes where used for the Features (Supplementary File S10), and GSE18781was used for Testing Data (Supplementary File S11) as an example. The file names are displayed in the 'CURRENT DATA UPLOAD' panel (**Supplementary Figure 9.c**) after selection has been made.

For the classifier, we use the default parameters, and 'K-fold Cross-Validation' set to 10 (**Supplementary Figure 9.d**) to train our model. After training is completed, the confusion matrix, classifier accuracy and error are displayed in the 'CONFUSION MATRIX, MODEL ACCURACY & ERROR' panel (Supplementary Figure 9.e). Model validation data results are displayed in the 'TRAINING RESULT: ID – ACTUAL – PREDICTION' panel (Supplementary Figure 9.f) with each data point ID is the 1st value, actual target value is displayed 2nd, and predicted target value 3rd where Male = 0 and Female = 1. Testing results are displayed in the 'TESTING RESULT: ID – PREDICTION' panel (Supplementary Figure 9.g) with each data point ID shown 1st, and the corresponding predicted target value displayed after it 2nd, separated by a hyphen. Full results (trained models, and both trained/validated, and tested data) were then exported (for more details on the export data file formats, see the Supplementary files S13-15).



Supplementary Figure 9. Trained linear support vector machine Classifier. Trained model using *GSE99039* 121 data points and k-fold cross validation, classifier evaluation (confusion matrix, model accuracy and error). Model validated and tested model using *GSE18781* 25 data points.

Note on Features annotation: To identify the Y chromosomes genes, we used the biomaRt [7] which mapped the Y probes on Affymetrix Human Genome U133 Plus 2.0 Array, GPL570, to the human reference genome (for complete list of all 96 Y chromosome gene ids, see Supplementary file S12).

Attribute

	A
1	224174_at
2	207646_s_at
3	207647_at
4	216842_x_at
5	232402_at
6	211461_at
7	244498_x_at
8	234715_at
9	207918_s_at
10	217160_at
11	234913_at
12	208331_at
13	1565320_at
14	207909_x_at
15	208282_x_at
16	208281_x_at
17	216922_x_at

Supplementary Figure 10. Features Data. Example of partial features data file format where each Affymetrix probe id correspond to a Y chromosome gene.

References

1. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V *et al*: **Scikit-learn: Machine Learning in Python**. *J Mach Learn Res* 2011, **12**:2825-2830.
2. Fisher RA: **The use of multiple measurements in taxonomic problems**. *Ann Eugenic* 1936, **7**:179-188.
3. Anderson E: **The Irises of the Gaspe peninsula**. *Bulletin of American Iris Society* 1935, **59**:2-5.
4. Shamir R, Klein C, Amar D, Vollstedt EJ, Bonin M, Usenovic M, Wong YC, Maver A, Poths S, Safer H *et al*: **Analysis of blood-based gene expression in idiopathic Parkinson disease**. *Neurology* 2017, **89**(16):1676-1683.
5. Sharma SM, Choi D, Planck SR, Harrington CA, Austin CR, Lewis JA, Diebel TN, Martin TM, Smith JR, Rosenbaum JT: **Insights in to the pathogenesis of axial spondyloarthropathy based on gene expression profiles**. *Arthritis Res Ther* 2009, **11**(6):R168.
6. Bolstad BM, Irizarry RA, Astrand M, Speed TP: **A comparison of normalization methods for high density oligonucleotide array data based on variance and bias**. *Bioinformatics* 2003, **19**(2):185-193.
7. Durinck S, Moreau Y, Kasprzyk A, Davis S, De Moor B, Brazma A, Huber W: **BioMart and Bioconductor: a powerful link between biological databases and microarray data analysis**. *Bioinformatics* 2005, **21**(16):3439-3440.