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

Abstract

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.

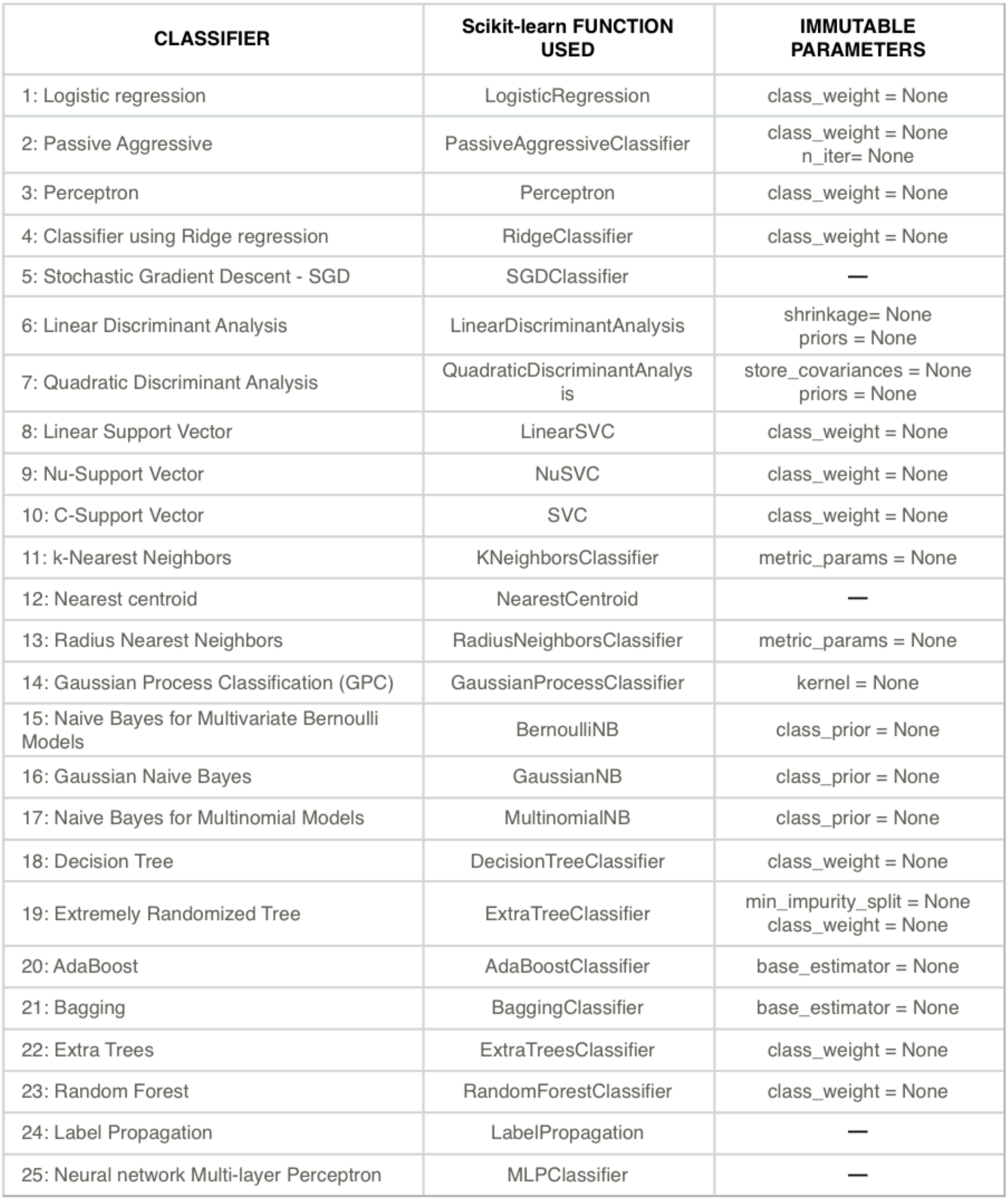
SUPPLEMENTARY CONTENTS

TABLE S1: Classification Algorithms 3

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

ATTACHMENTS: SUPPLEMENTARY FILES (https://github.com/gmiaslab/manuals/tree/master/ClassificaIO/Supplementary%20Files)

	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_TestingResult.csv*	Example ClassificaIO testing result using logistic regression
7.	*S7_TrainValidationResult.csv*	Example ClassificaIO validation result using logistic regression



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 are presented in additional file.

G. Mias Lab

ClassificaIO

Machine Learning for Classification Graphical
User Interface User Manual 1.0.5 (01/2018)

Summary:

ClassificaIO is an open-source Python graphical user interface (GUI) for supervised machine learning classification for the scikit-learn module (Pedregosa, et al., 2011). 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.

Prerequisites:

ClassificaIO is a Python library with the following external dependencies: nltk ≥ 3.2.5, Tcl/Tk ≥ 8.6.7, Pillow ≥ 4.3, pandas ≥ 0.21, numpy ≥ 1.13, scikit-learn ≥ 0.19.1. ClassificaIO requires Python version 3.5 or higher and we recommend using the Spyder integrated development environment (IDE) in Anaconda Navigator (https://www.anaconda.com/download/) on Mac OS High Sierra (10.13) and Microsoft Windows 10 or higher.

Download and installation:

ClassificaIO can be installed using pip (https://pypi.python.org/pypi/pip) in the terminal:

```
$ pip install ClassificaIO
```

You can also install it directly from the main GitHub repository using:

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

In case you do not have pip installed, you must install it first. Or you can obtain and install ClassificaIO by downloading or cloning ClassificaIO source code from ClassificaIO GitHub repository (https://github.com/gmiaslab/ClassificaIO)

Getting started:

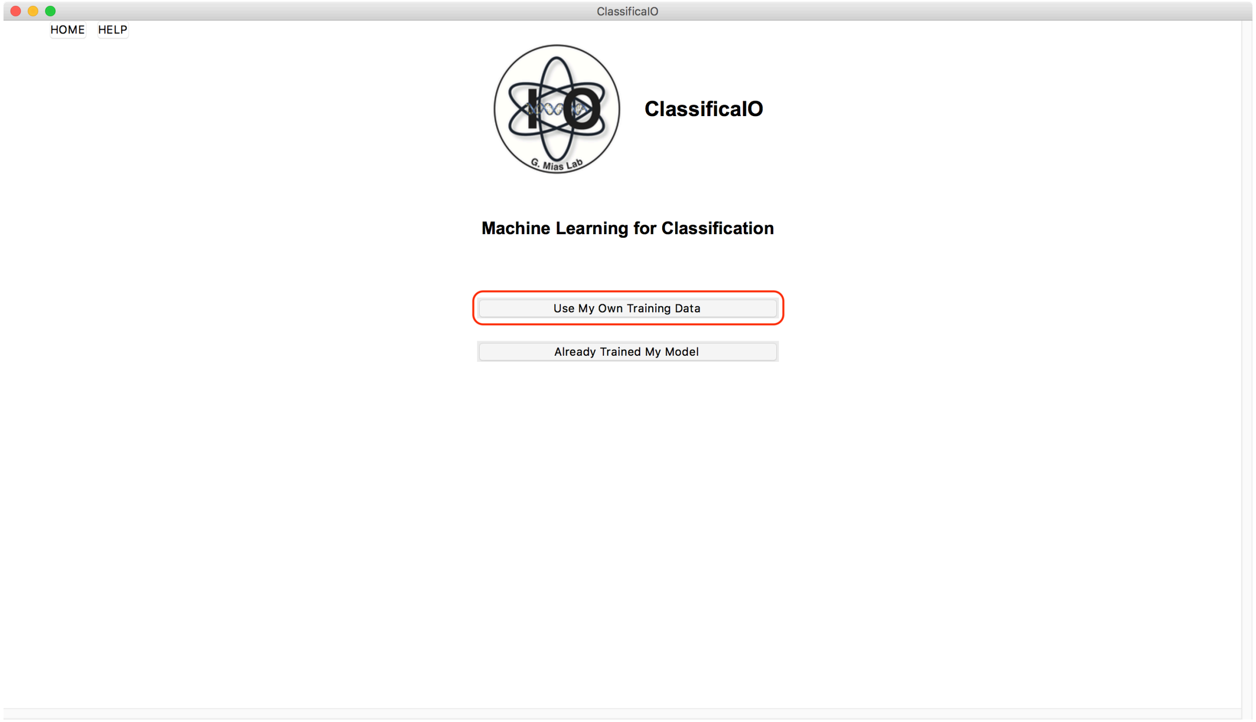
Please Note:

- ClassificaIO supports comma-separated values (CSV) input files only.
- In this document we use the machine learning Iris dataset (Anderson, 1935; Fisher, 1936) (150 data points) as an example, 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.

After installing ClassificaIO, please run it using the “import” function in Python:

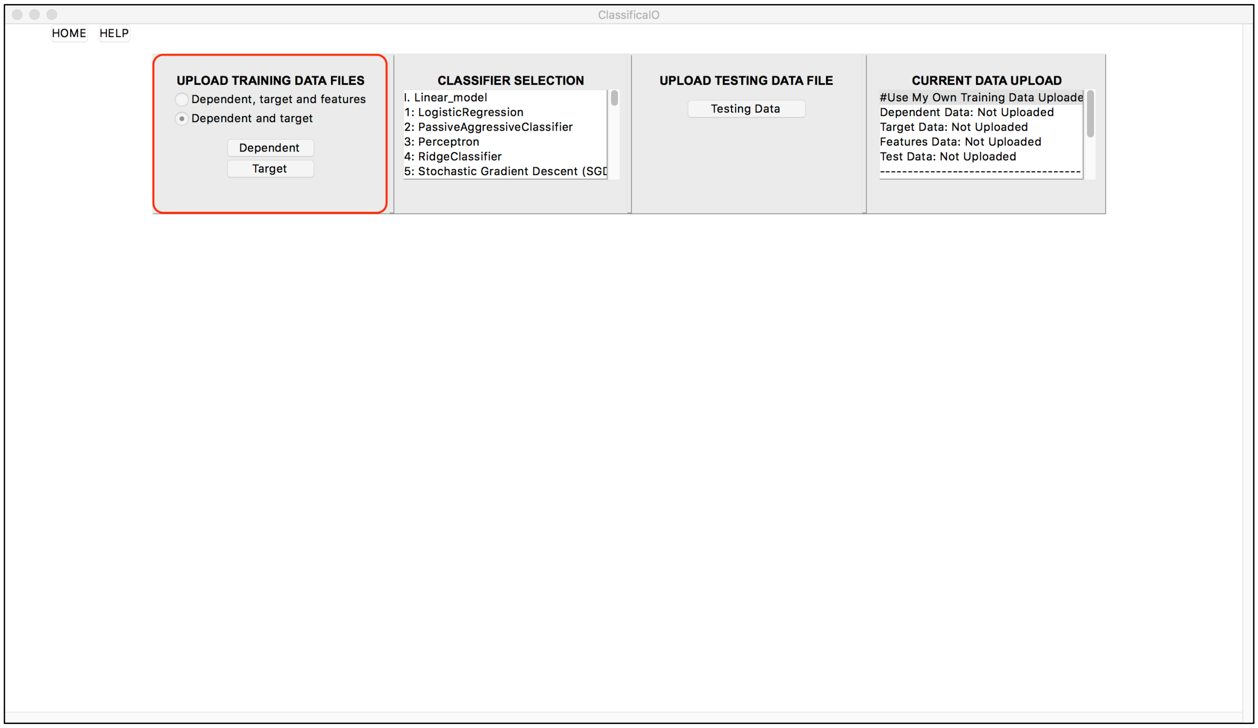
```
>>> from ClassificaIO import ClassificaIO
>>> ClassificaIO.gui()
```

Once ClassificaIO’s main window appears on your screen, you can click on ‘Use My Own Training Data’ button and start your new supervised machine learning classification project.



Training data input:

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’ radio butoon.

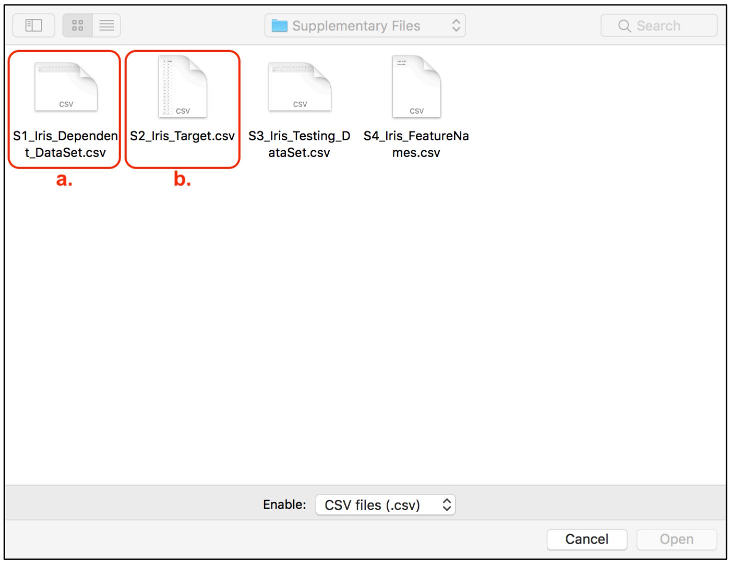


○ Dependent, target and features
● Dependent and target

```
I. Linear_model
1: LogisticRegression
2: PassiveAggressiveClassifier
3: Perceptron
4: RidgeClassifier
5: Stochastic Gradient Descent (SGD
```

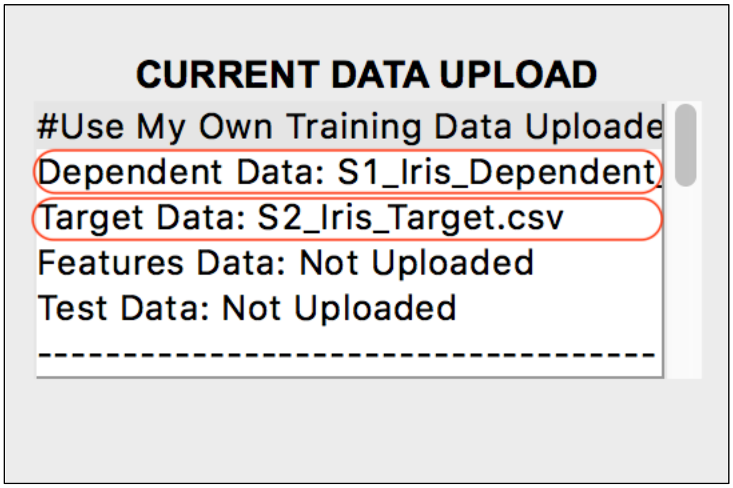
```
#Use My Own Training Data Uploade
Dependent Data: Not Uploaded
Target Data: Not Uploaded
Features Data: Not Uploaded
Test Data: Not Uploaded
-----------------------------------
```

To begin uploading 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.



Enable: CSV files (.csv)

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.



CURRENT DATA UPLOAD

#Use My Own Training Data Uploade

Dependent Data: S1_Iris_Dependent

Target Data: S2_Iris_Target.csv

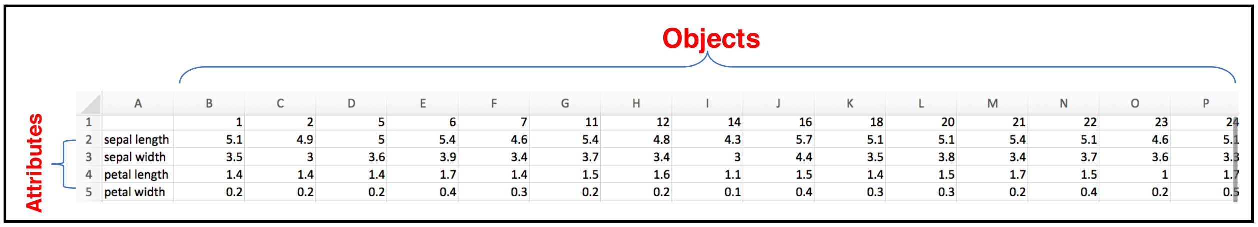
Features Data: Not Uploaded

Test Data: Not Uploaded

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

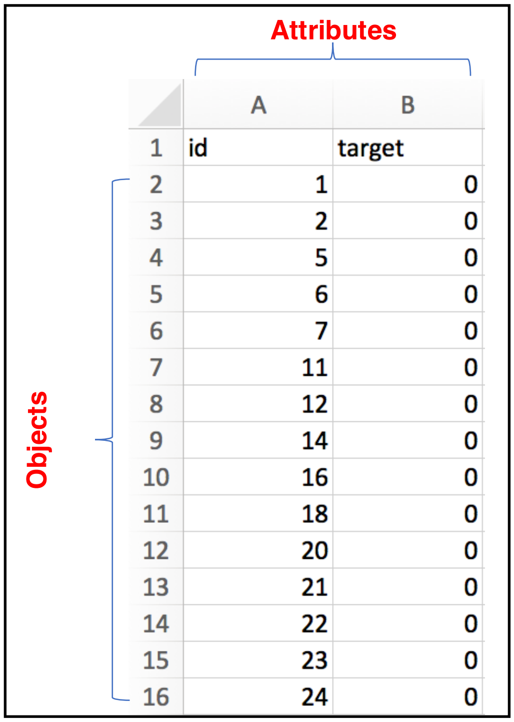
Data format:

Data formats are shown in **Supplementary Figure 3.a** for dependent data and **Supplementary Figure 3.b** for target data.



	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.

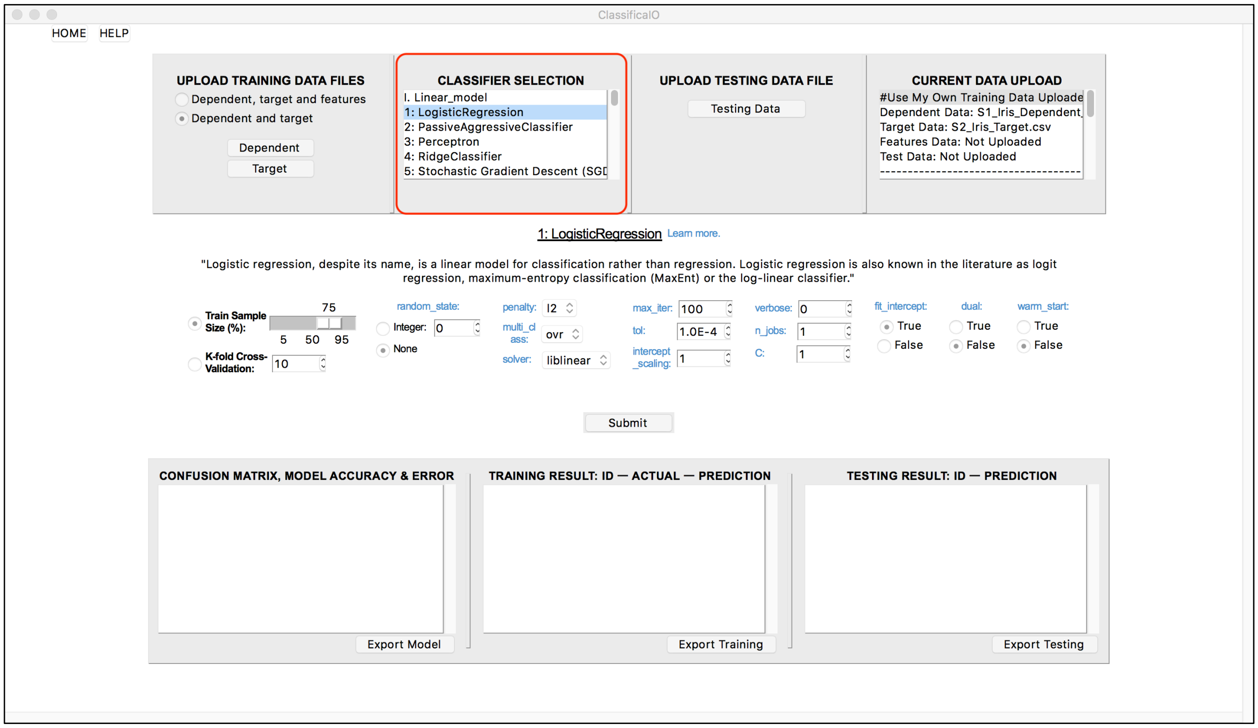


	A	B
1	id	target
2	1	0
3	2	0
4	5	0
5	6	0
6	7	0
7	11	0
8	12	0
9	14	0
10	16	0
11	18	0
12	20	0
13	21	0
14	22	0
15	23	0
16	24	0

Supplementary Figure 3.b. Target Data. Example of partial target data file format.

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.



○ Dependent, target and features
● Dependent and target

I. Linear_model
1: LogisticRegression
2: PassiveAggressiveClassifier
3: Perceptron
4: RidgeClassifier
5: Stochastic Gradient Descent (SGD

#Use My Own Training Data Uploade
Dependent Data: S1_Iris_Dependent_
Target Data: S2_Iris_Target.csv
Features Data: Not Uploaded
Test Data: Not Uploaded

1: LogisticRegression Learn more.

"Logistic regression, despite its name, is a linear model for classification rather than regression. Logistic regression is also known in the literature as logit regression, maximum-entropy classification (MaxEnt) or the log-linear classifier."

● Train Sample Size (%): 75 (5 50 95)
○ K-fold Cross-Validation: 10

random_state: ○ Integer: 0 ● None

penalty: l2
multi_class: ovr
solver: liblinear

max_iter: 100
tol: 1.0E-4
intercept_scaling: 1

verbose: 0
n_jobs: 1
C: 1

fit_intercept: ● True ○ False
dual: ○ True ● False
warm_start: ○ True ● False

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

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**

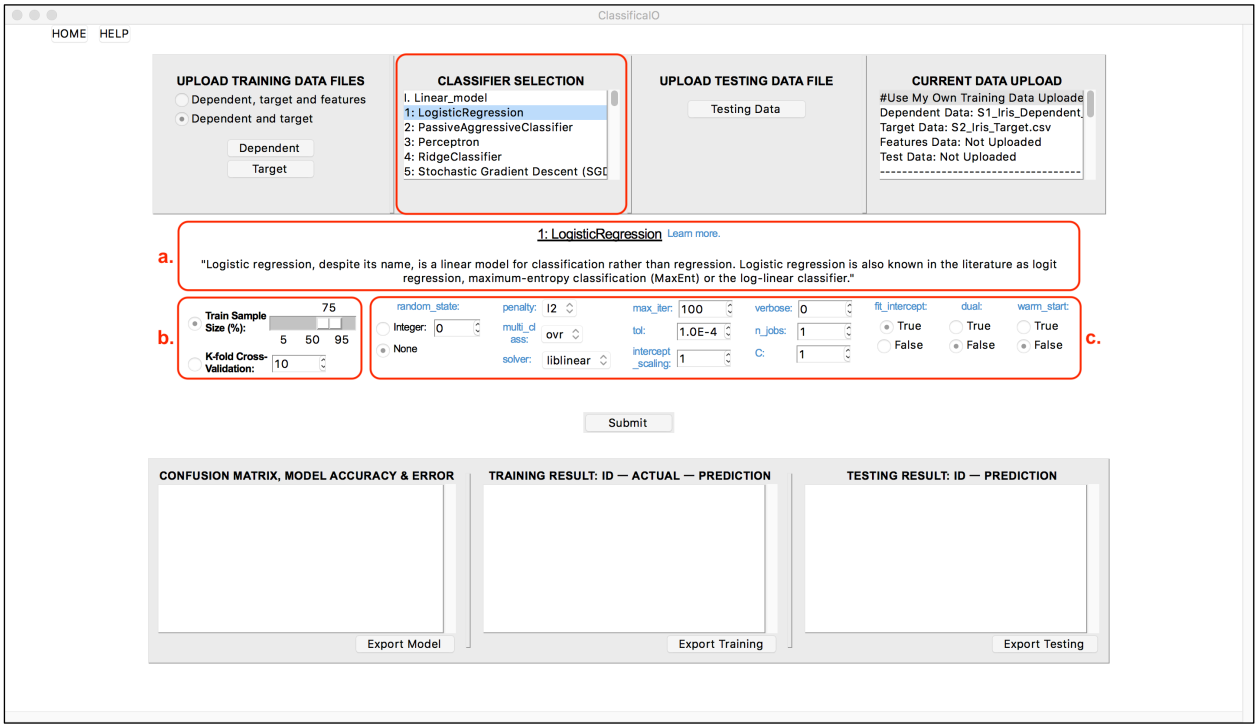
24:LabelPropagation.

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 hyperlink "learn more" in blue, which, once clicked, opens an external web-browser to the scikit-learn documentation for the selected classifier.
- **Supplementary Figure 4.b**: Easy 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



○ Dependent, target and features
● Dependent and target

I. Linear_model
1: LogisticRegression
2: PassiveAggressiveClassifier
3: Perceptron
4: RidgeClassifier
5: Stochastic Gradient Descent (SGD

#Use My Own Training Data Uploade
Dependent Data: S1_Iris_Dependent_
Target Data: S2_Iris_Target.csv
Features Data: Not Uploaded
Test Data: Not Uploaded

1: LogisticRegression Learn more.

"Logistic regression, despite its name, is a linear model for classification rather than regression. Logistic regression is also known in the literature as logit regression, maximum-entropy classification (MaxEnt) or the log-linear classifier."

● Train Sample Size (%): 75 (5 50 95)
○ K-fold Cross-Validation: 10

random_state: ○ Integer: 0 ● None
penalty: l2
multi_class: ovr
solver: liblinear
max_iter: 100
tol: 1.0E-4
intercept_scaling: 1
verbose: 0
n_jobs: 1
C: 1
fit_intercept: ● True ○ False
dual: ○ True ● False
warm_start: ○ True ● False

Supplementary Figure 4. Selected Logistic Regression Classifier. a. Classifier definition is displayed, together with, **b,** the. training methods with 'Train Sample Size(%)' method selected, and **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, dependent, and target data 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 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.

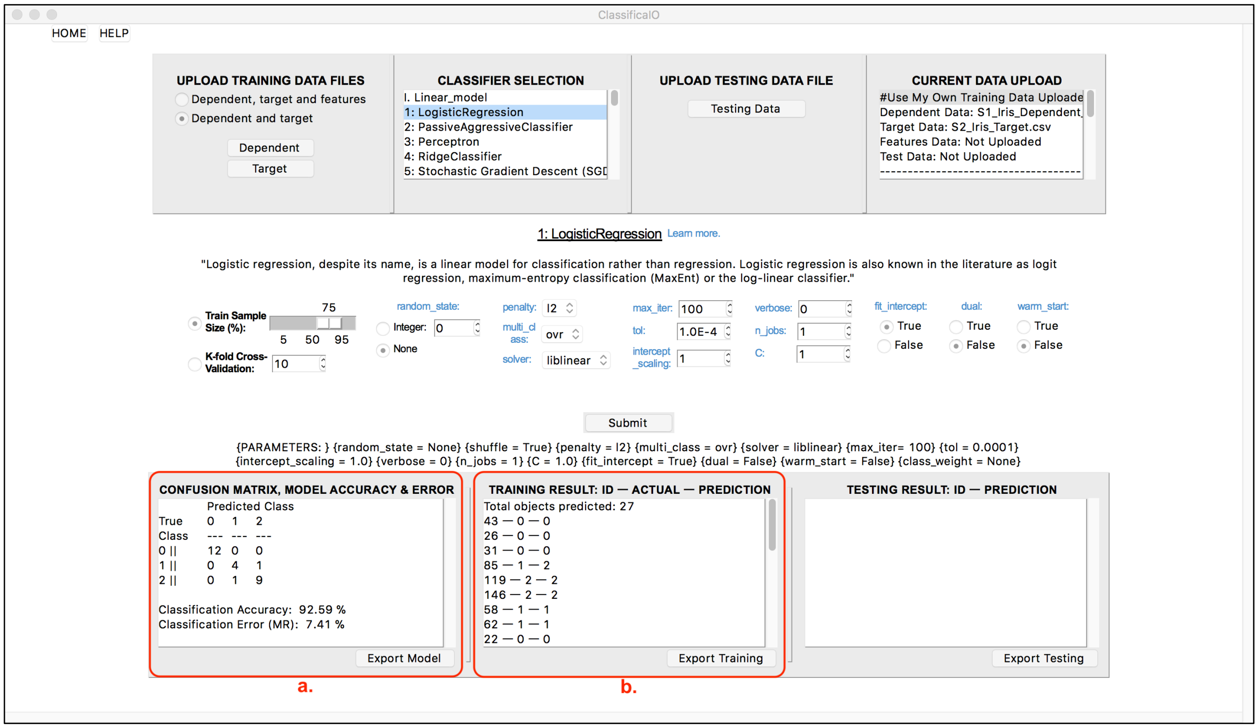
After clicking ‘submit’, our selected classifier, ‘LogisticRegression’ for this example, is trained using the loaded training data, ‘Dependent and Target’ for this example.

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 function, 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 function 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 first value, actual target value is displayed 2nd, and predicted target value third.



○ Dependent, target and features
● Dependent and target

```
I. Linear_model
1: LogisticRegression
2: PassiveAggressiveClassifier
3: Perceptron
4: RidgeClassifier
5: Stochastic Gradient Descent (SGI
```

```
#Use My Own Training Data Uploade
Dependent Data: S1_Iris_Dependent_
Target Data: S2_Iris_Target.csv
Features Data: Not Uploaded
Test Data: Not Uploaded
-----------------------------------
```

1: LogisticRegression

"Logistic regression, despite its name, is a linear model for classification rather than regression. Logistic regression is also known in the literature as logit regression, maximum-entropy classification (MaxEnt) or the log-linear classifier."

● Train Sample Size (%): 75 (5 50 95)
○ K-fold Cross-Validation: 10

random_state: ○ Integer: 0 ● None

penalty: l2
multi_class: ovr
solver: liblinear

max_iter: 100
tol: 1.0E-4
intercept_scaling: 1

verbose: 0
n_jobs: 1
C: 1

fit_intercept: ● True ○ False
dual: ○ True ● False
warm_start: ○ True ● False

{PARAMETERS: } {random_state = None} {shuffle = True} {penalty = l2} {multi_class = ovr} {solver = liblinear} {max_iter= 100} {tol = 0.0001} {intercept_scaling = 1.0} {verbose = 0} {n_jobs = 1} {C = 1.0} {fit_intercept = True} {dual = False} {warm_start = False} {class_weight = None}

CONFUSION MATRIX, MODEL ACCURACY & ERROR

```
        Predicted Class
True    0    1    2
Class   ---  ---  ---
0 ||    12   0    0
1 ||    0    4    1
2 ||    0    1    9

Classification Accuracy:  92.59 %
Classification Error (MR):  7.41 %
```

TRAINING RESULT: ID — ACTUAL — PREDICTION

```
Total objects predicted: 27
43 — 0 — 0
26 — 0 — 0
31 — 0 — 0
85 — 1 — 2
119 — 2 — 2
146 — 2 — 2
58 — 1 — 1
62 — 1 — 1
22 — 0 — 0
```

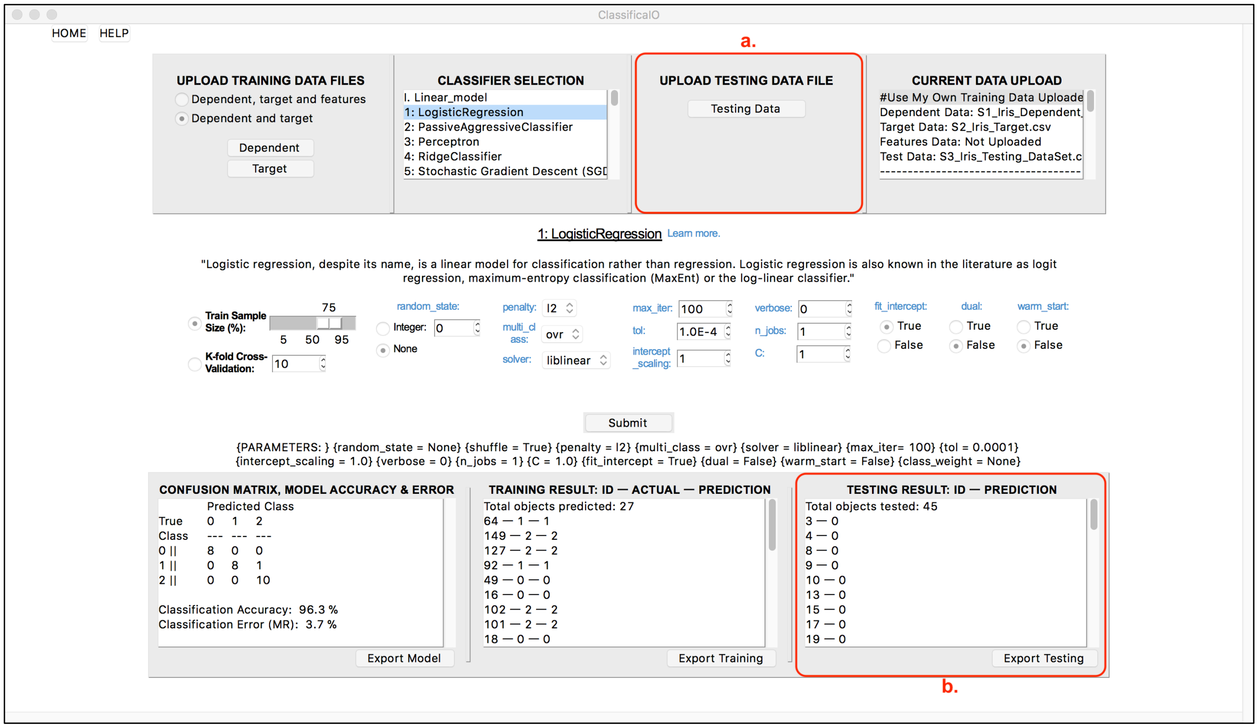
TESTING RESULT: ID — PREDICTION

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 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 first, and the corresponding predicted target value displayed after it, separated by a hyphen.



○ Dependent, target and features
● Dependent and target

```
I. Linear_model
1: LogisticRegression
2: PassiveAggressiveClassifier
3: Perceptron
4: RidgeClassifier
5: Stochastic Gradient Descent (SGD
```

```
#Use My Own Training Data Uploade
Dependent Data: S1_Iris_Dependent_
Target Data: S2_Iris_Target.csv
Features Data: Not Uploaded
Test Data: S3_Iris_Testing_DataSet.c
-----------------------------------
```

1: LogisticRegression

"Logistic regression, despite its name, is a linear model for classification rather than regression. Logistic regression is also known in the literature as logit regression, maximum-entropy classification (MaxEnt) or the log-linear classifier."

● Train Sample Size (%): 75 (5 50 95)
○ K-fold Cross-Validation: 10

random_state: ○ Integer: 0 ● None

penalty: l2
multi_class: ovr
solver: liblinear

max_iter: 100
tol: 1.0E-4
intercept_scaling: 1

verbose: 0
n_jobs: 1
C: 1

fit_intercept: ● True ○ False
dual: ○ True ● False
warm_start: ○ True ● False

{PARAMETERS: } {random_state = None} {shuffle = True} {penalty = l2} {multi_class = ovr} {solver = liblinear} {max_iter= 100} {tol = 0.0001} {intercept_scaling = 1.0} {verbose = 0} {n_jobs = 1} {C = 1.0} {fit_intercept = True} {dual = False} {warm_start = False} {class_weight = None}

```
         Predicted Class
True     0    1    2
Class    ---  ---  ---
0 ||     8    0    0
1 ||     0    8    1
2 ||     0    0    10

Classification Accuracy:  96.3 %
Classification Error (MR):  3.7 %
```

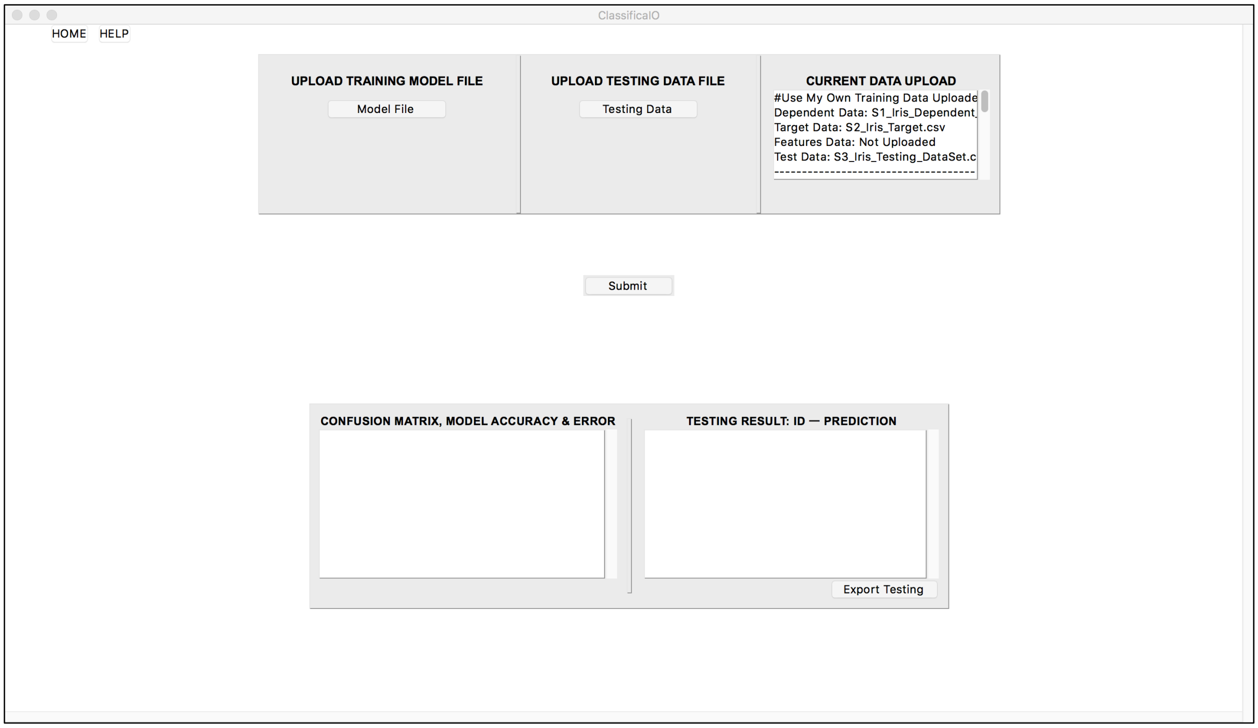
```
Total objects predicted: 27
64 — 1 — 1
149 — 2 — 2
127 — 2 — 2
92 — 1 — 1
49 — 0 — 0
16 — 0 — 0
102 — 2 — 2
101 — 2 — 2
18 — 0 — 0
```

```
Total objects tested: 45
3 — 0
4 — 0
8 — 0
9 — 0
10 — 0
13 — 0
15 — 0
17 — 0
19 — 0
```

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.

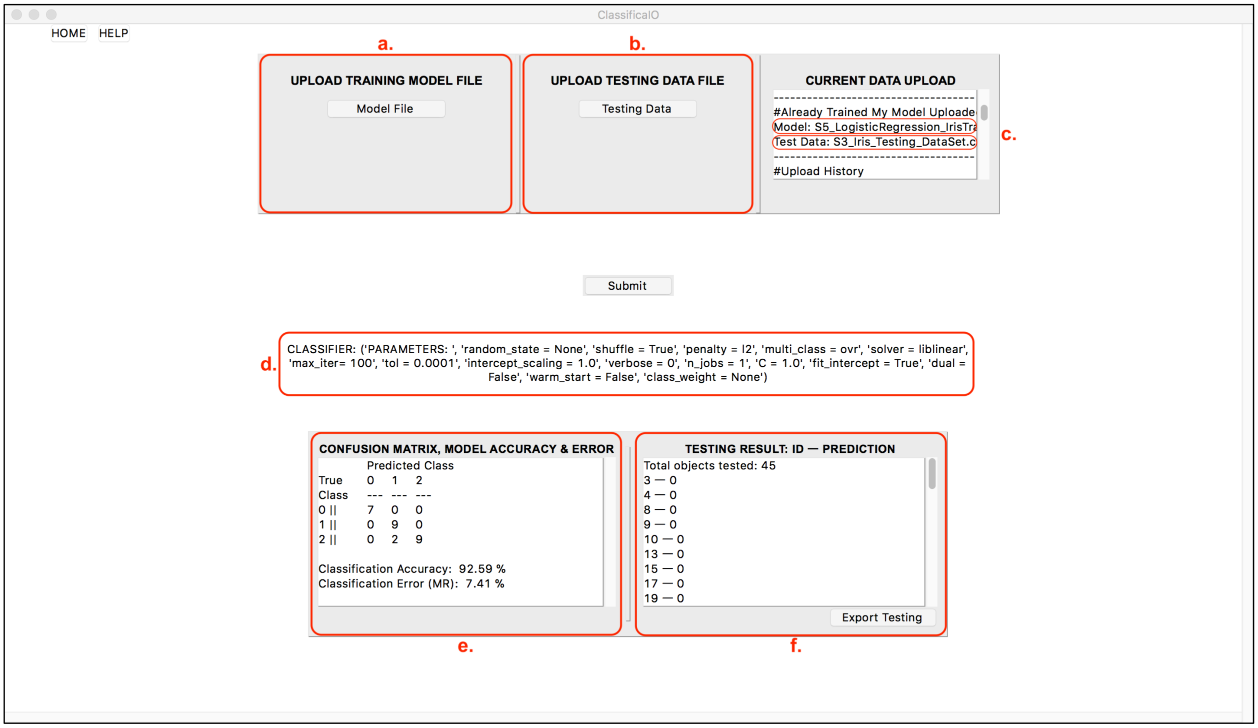


```
#Use My Own Training Data Uploade
Dependent Data: S1_Iris_Dependent_
Target Data: S2_Iris_Target.csv
Features Data: Not Uploaded
Test Data: S3_Iris_Testing_DataSet.c
------------------------------------
```

ClassificalO model input:

You will need to upload ClassificalO 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 ClassificalO 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 ClassificalO 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 first, followed by a hyphen and the predicted value displayed right after it.



```
----------------------------------------
#Already Trained My Model Uploade
Model: S5_LogisticRegression_IrisTra
Test Data: S3_Iris_Testing_DataSet.c
----------------------------------------
#Upload History
```

```
CLASSIFIER: ('PARAMETERS: ', 'random_state = None', 'shuffle = True', 'penalty = l2', 'multi_class = ovr', 'solver = liblinear',
'max_iter= 100', 'tol = 0.0001', 'intercept_scaling = 1.0', 'verbose = 0', 'n_jobs = 1', 'C = 1.0', 'fit_intercept = True', 'dual =
False', 'warm_start = False', 'class_weight = None')
```

```
        Predicted Class
True    0    1    2
Class   ---  ---  ---
0 ||    7    0    0
1 ||    0    9    0
2 ||    0    2    9

Classification Accuracy:  92.59 %
Classification Error (MR):  7.41 %
```

```
Total objects tested: 45
3 — 0
4 — 0
8 — 0
9 — 0
10 — 0
13 — 0
15 — 0
17 — 0
19 — 0
```

Supplementary Figure 7. ‘Already Trained My Model’ window a. Upload ClassificalO 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 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 more details on the export data file formats, see the Supplementary data files).

References

Anderson, E. The Irises of the Gaspe peninsula. *Bulletin of American Iris Society* 1935;59:2-5.

Fisher, R.A. The use of multiple measurements in taxonomic problems. *Ann Eugenic* 1936;7:179-188.

Pedregosa, F.*, et al.* Scikit-learn: Machine Learning in Python. *J Mach Learn Res* 2011;12:2825-2830.