LIBTwinSVM Documentation

Release 0.1.0

Mir, A.

Aug 16, 2023

Contents:

This is the LIBTwinSVM ‘s documentation.

CHAPTER 1

Usage Examples

1.1 User Interface

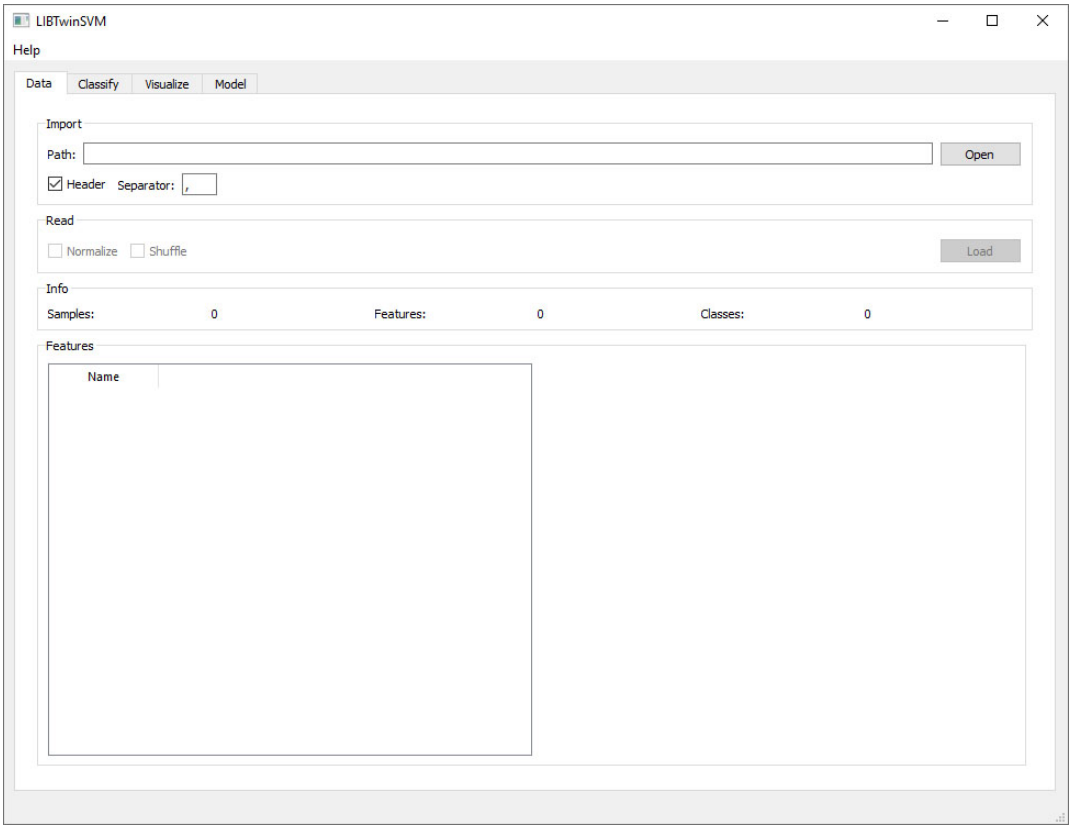
1.1.1 An example of classification using the GUI

In this section we have provided an easy step-by-step *Usage Example* which it shows how the GUI of the LIBTwinSVM works. For more information on the application and its features, go to this link.

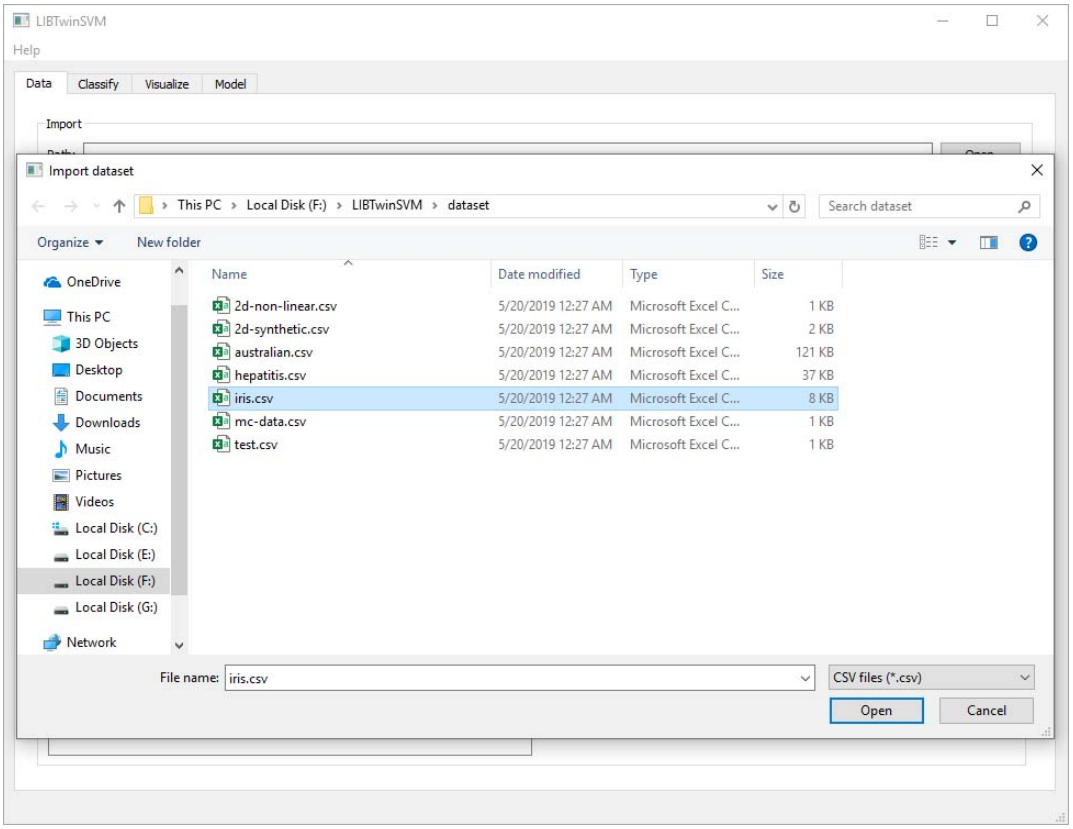
Step 1: Data Import

To use the application, the first step is to import the data. In the following section, we have shown how to import data and how t

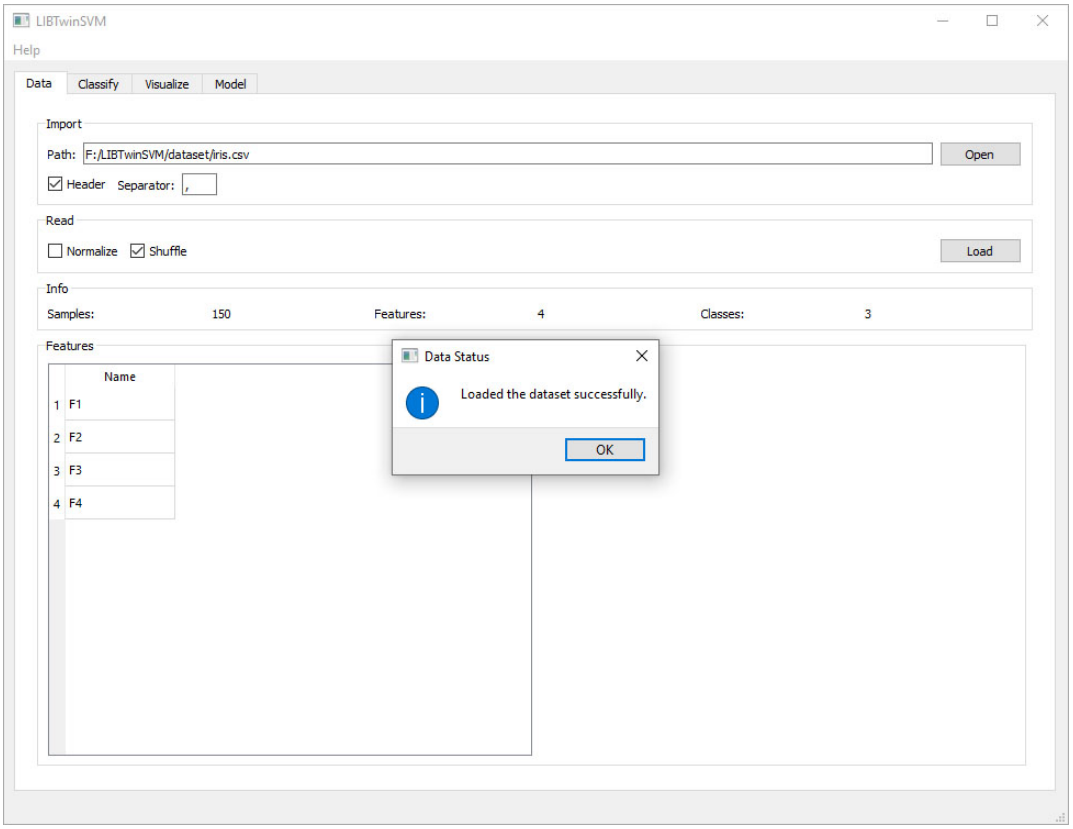
1. By default, the application starts on the Data tab.



2. By clicking on the **Open** button in the **Import** box, the **File Explorer** will be open and you can then choose the file you want to train.



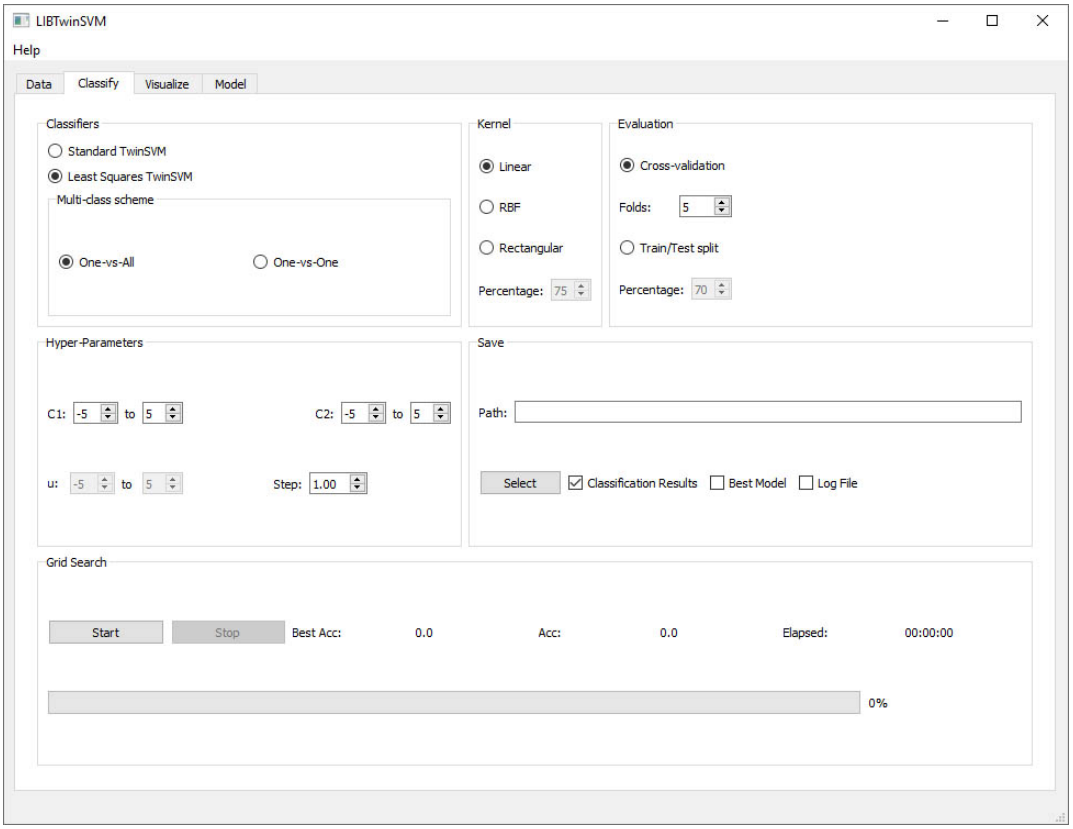
3. You can choose the file content separator if it is other than **comma**.
4. In **Read** box in **Data** tab, you can choose how you want the data to be processed by the application. You may want to **Normalize** or **Shuffle** the data.
5. Then you must click on **Load** Button, and the data will be imported and also displayed in the feature box.



Step 2: Classifying

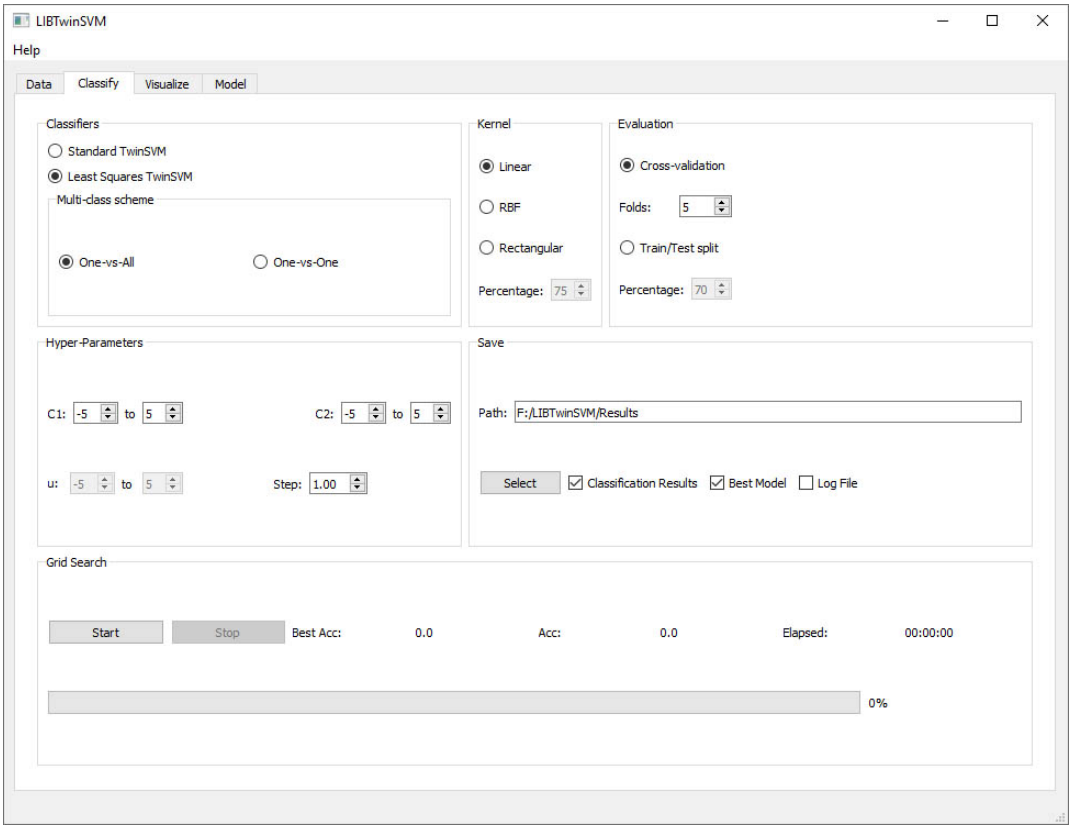
For this section, the data must be already imported and you are going to see how Classify tab works.

1. Switch the tabs to **Classify** tab.

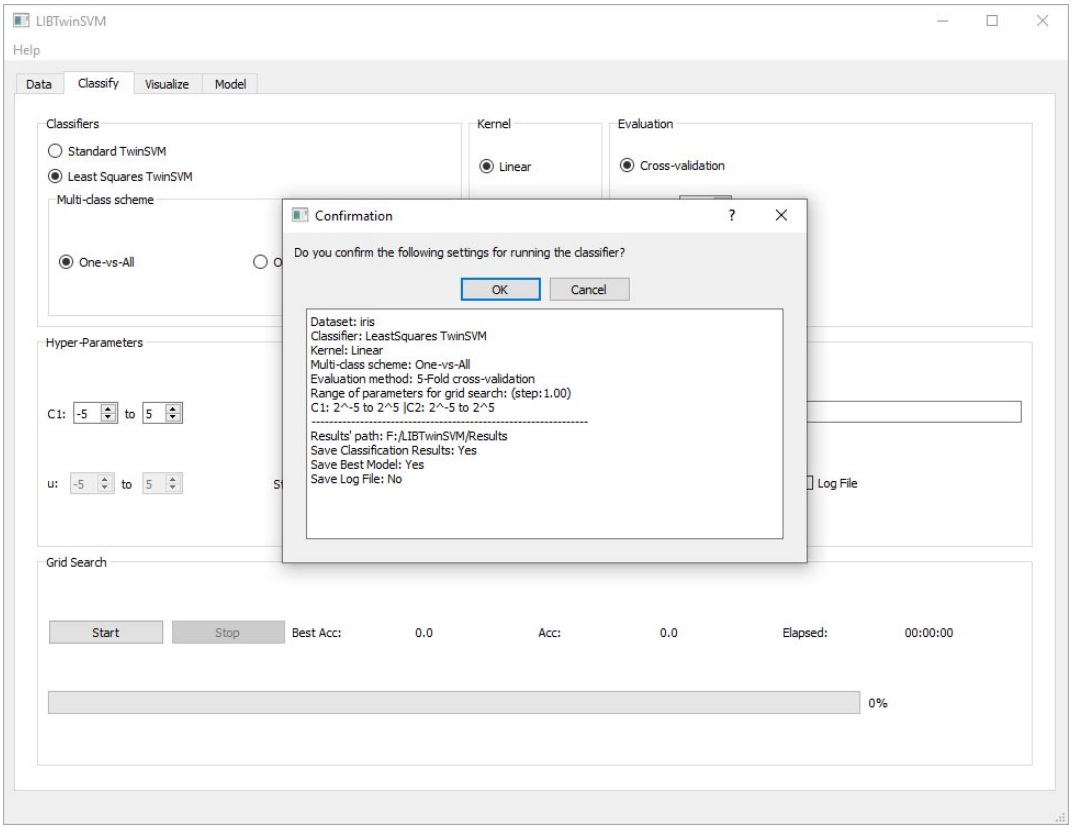


2. Now there is several options to choose such as **Classifiers**, **Kernel** type, **Evaluation** Method and **Hyper Parameters**. You can choose one of them, or leave them with their default values and just go to the next step.

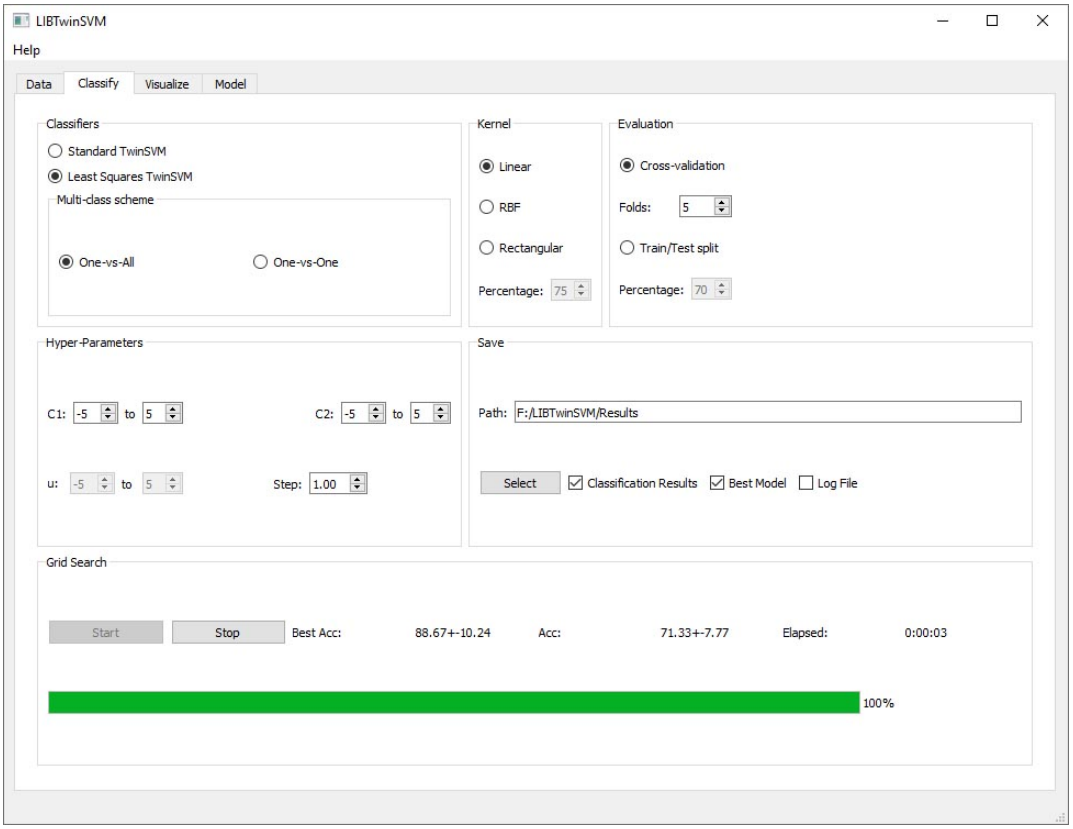
3. There is a save result box on that page. Before starting the training, you have to select a path so the application saves the final results. You can also check **Best Model** if you want the application to save the trained model and **Log FIle** if you need the application logs. (Note that for using the trained pretrained model, refer to Model Example.)



4. Now It is time to *Classify*. By Clicking on the **Run!** button, you can see a **Confirmation** message, pops up on the screen just to check if everything is exactly the way you had set before training.



5. After checking you click OK if everything is the way you want and it takes a few seconds to several minutes (depends on the data size) to be done!



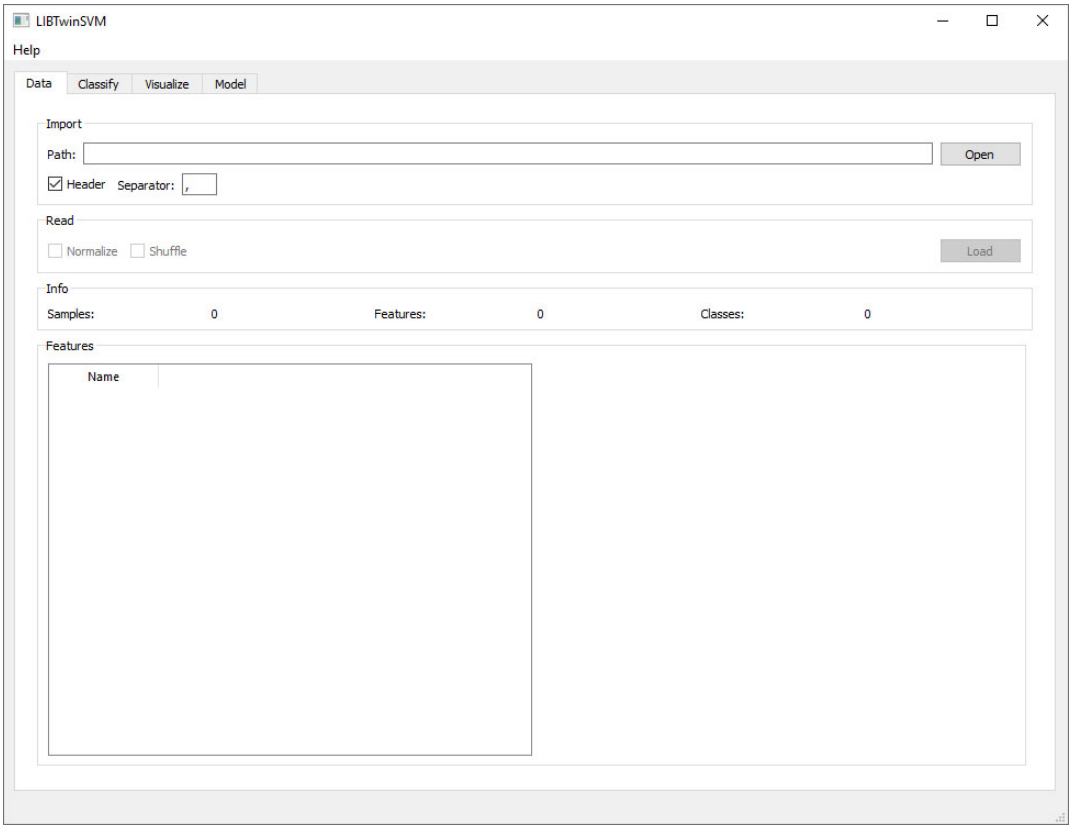
1.1.2 An example of Visualisation using the GUI

In this section we have provided an easy step-by-step *Usage Example* of Visualisation with LIBTwinSVM. For more information on the application and its features, go to this link.

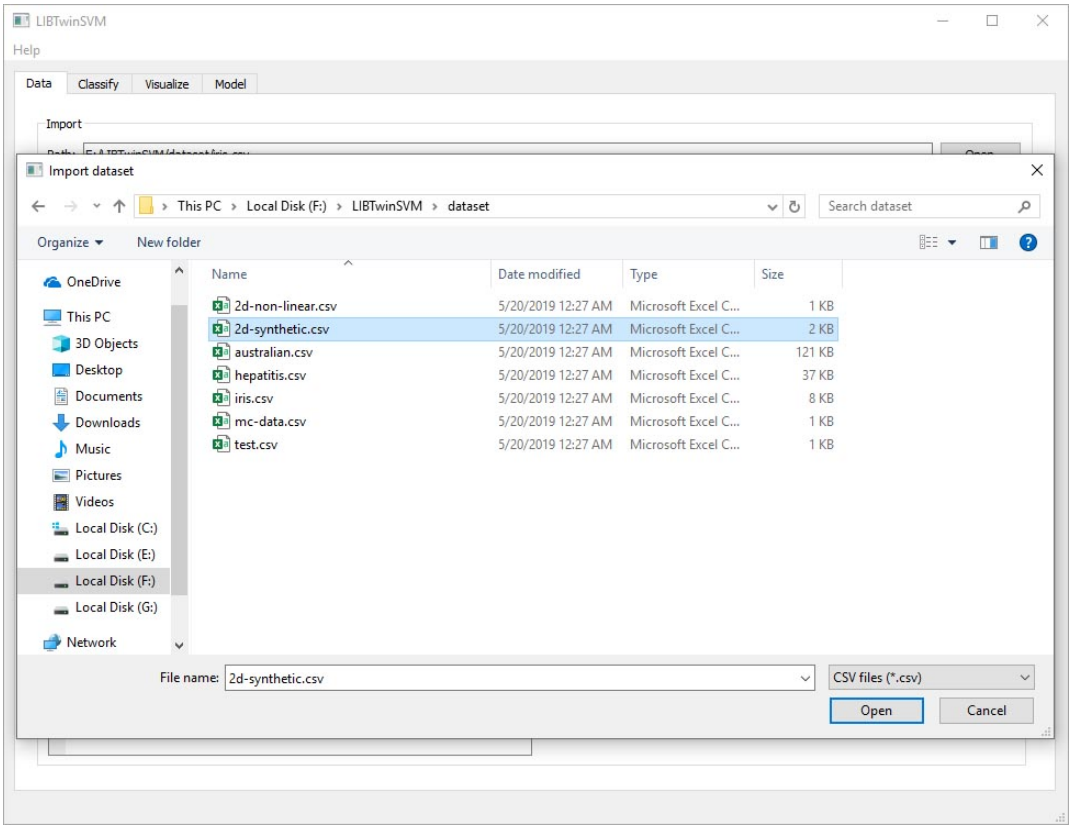
Step 1: Data Import

To visualize a model, the first step is to import the data. In the following section, you have shown how to import data and how to

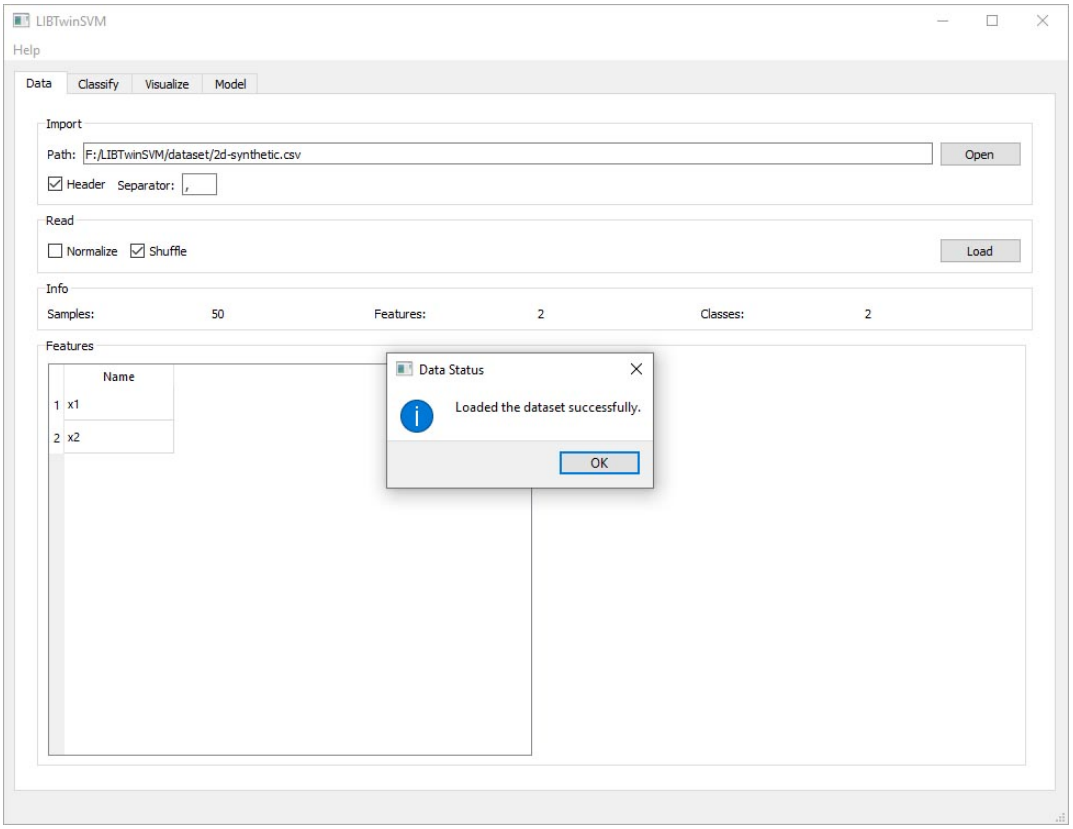
1. By default, the application starts on the Data tab.



2. By clicking on the **Open** button in the **Import** box, the **File Explorer** will be open and you can then choose the file you want to train.



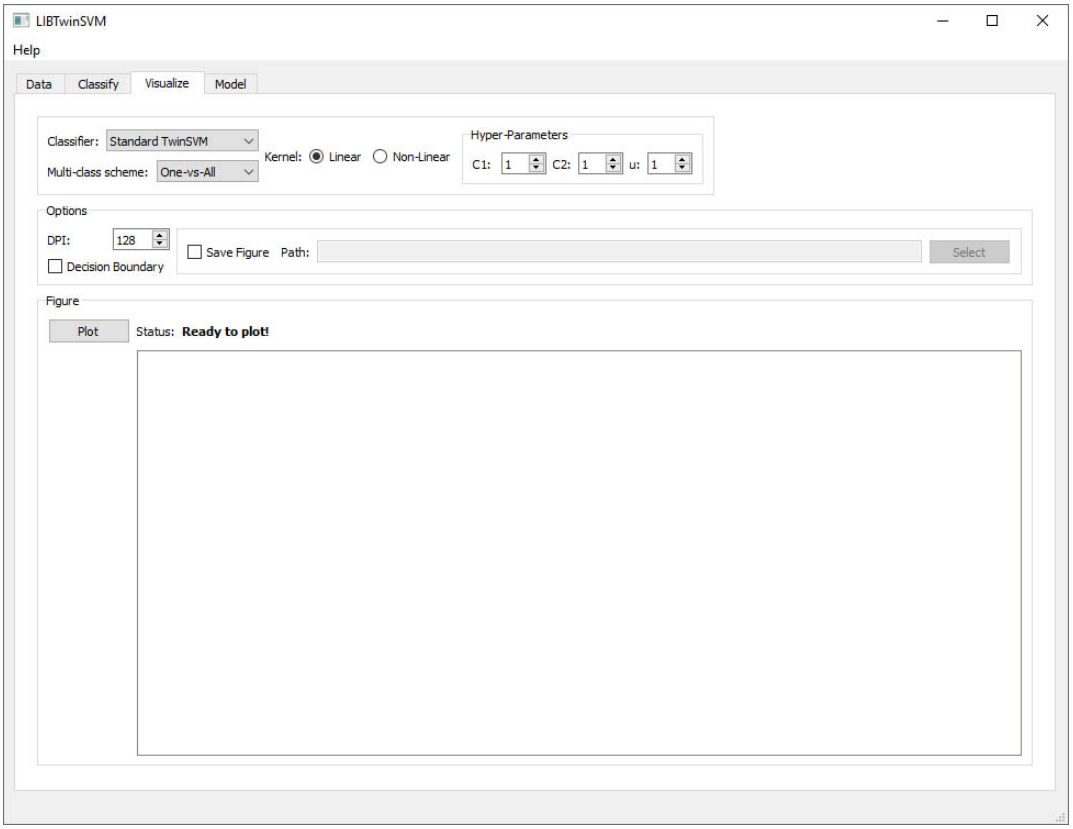
3. You can choose the file content separator if it is other than **comma**.
4. In **Read** box in **Data** tab, you can choose how you want the data to be processed by the application. You may want to **Normalize** or **Shuffle** the data.
5. Then you must click on **Load** Button, and the data will be imported and also displayed in the feature box.



Step 2: Visualisation

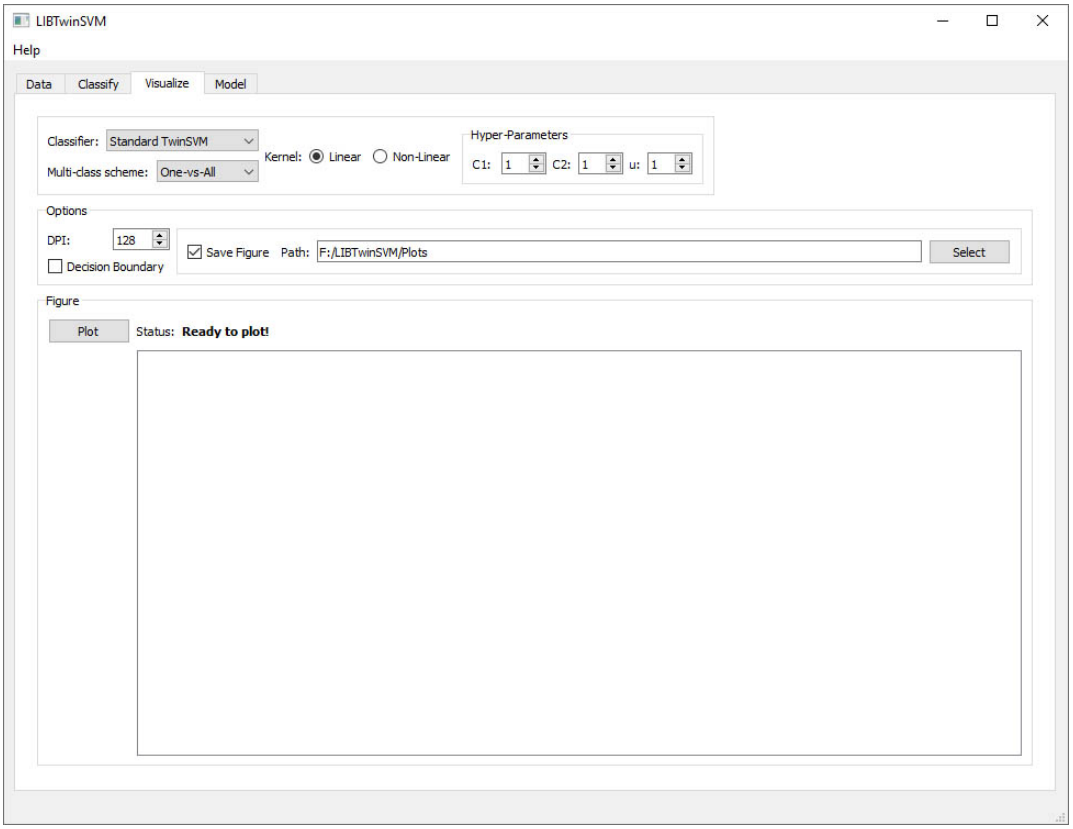
So far, the data is already imported and you are going to see how Visualize tab works.

1. You switch the tabs to **Visualize** tab.

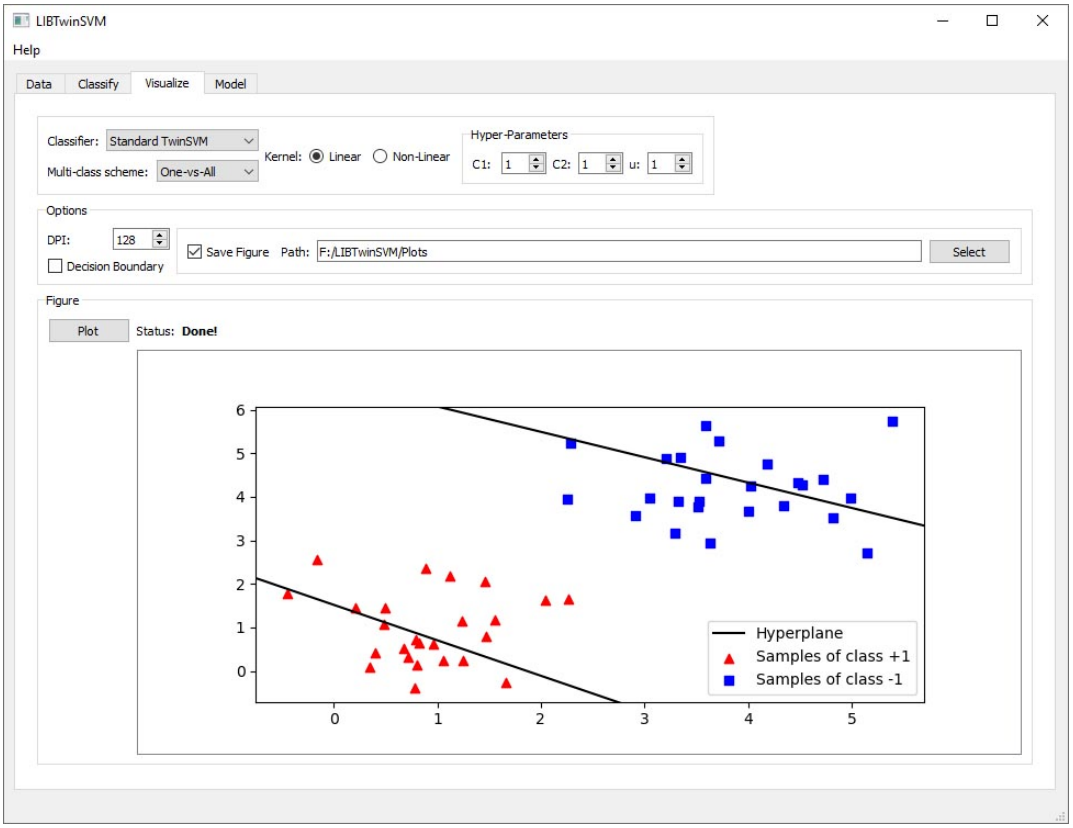


2. Now you have several options to choose or modify such as **Classifier**, **Kernel** type, **Hyper Parameters** and **DPI** which it determines the output plot quality.

3. In the *Options* box in Visualize tab, there is a check-box for saving the figure. By checking that check-box, you need to select a path, so the application can save the final figure.



4. In the final step by Clicking on the **Plot** button, the plot will be displayed in the figure box.



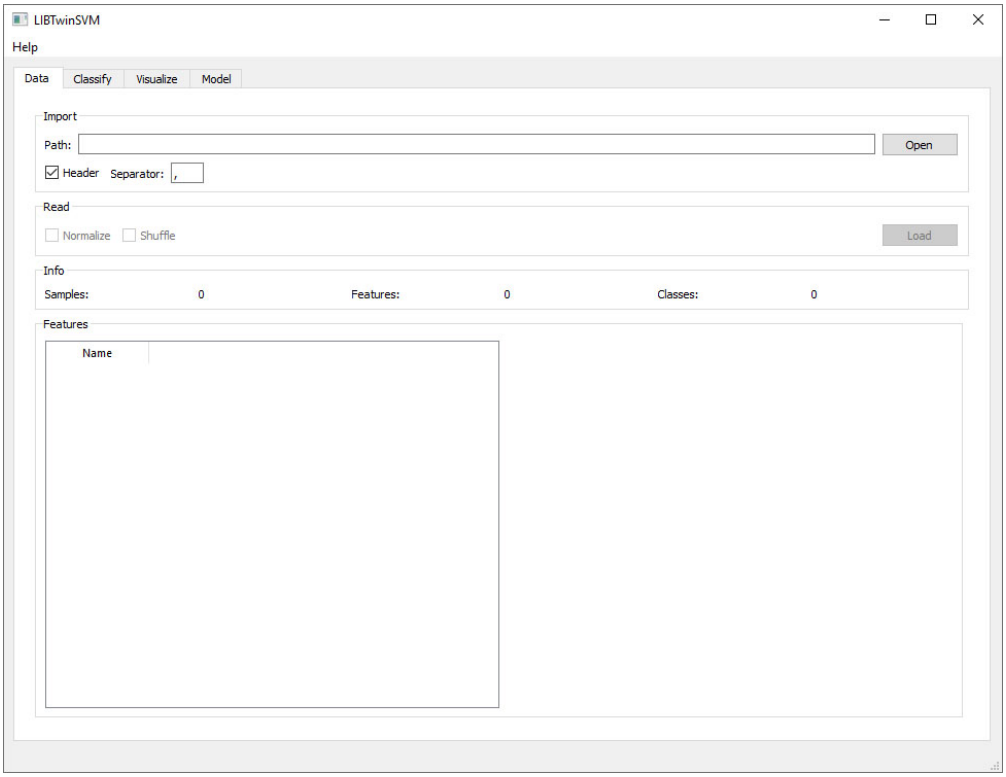
1.1.3 An example of using a pre-trained Model for classification

In this section, we have illustrated how to use a pretrain model, saved by the LIBTwinSVM. Note that, this step requires a pre-trained model file saved in classifying step as a **.joblib** file. If you don't have the previously said file, please refer to [classication usage example](https://libtwinsvm.readthedocs.io/en/latest/examples/GUI/classify.html#).

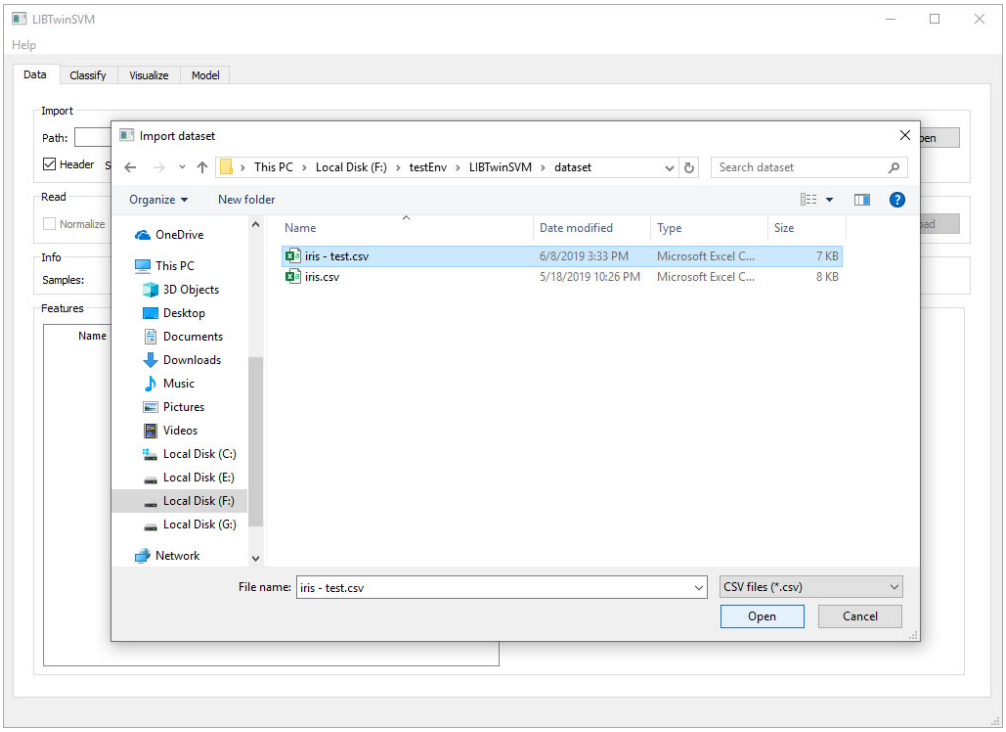
Step 1: Data Import

Please note that, to use a model on test samples, the test data must have the same features as the training data. Below is a step-by-step procudure on how to load your data for reusing a pre-trained model.

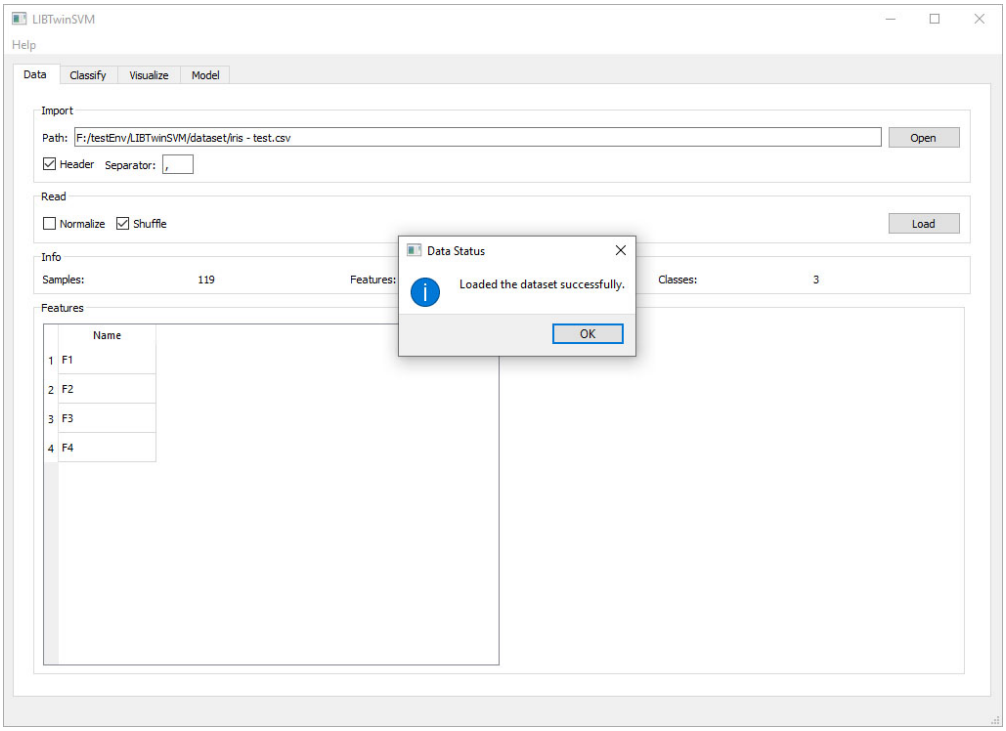
1. By default, the application starts on the Data tab.



2. By clicking on the **Open** button in the **Import** box, the file dialog will be opened and you can then choose the dataset you want to test with a pre-trained model.



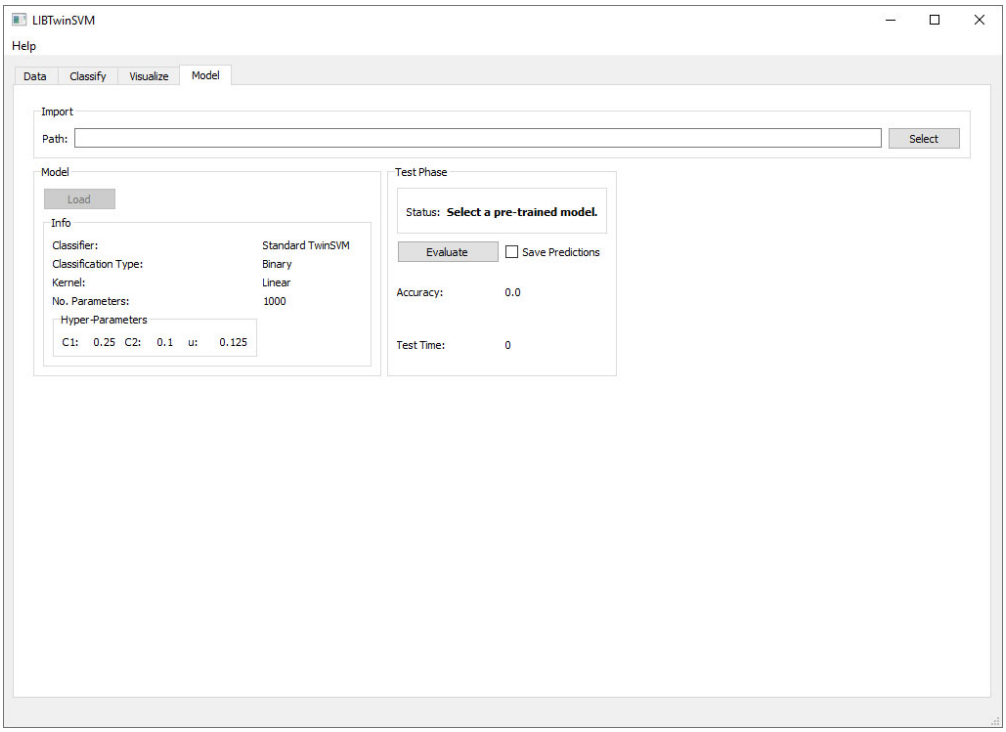
3. You may need to change the column separator if it is other than **comma**.
4. In **Read** box in **Data** tab, you may want to **Normalize** and/or **Shuffle** the data.
5. Then you must click on **Load** Button, and the data will be loaded and also displayed in the feature box.



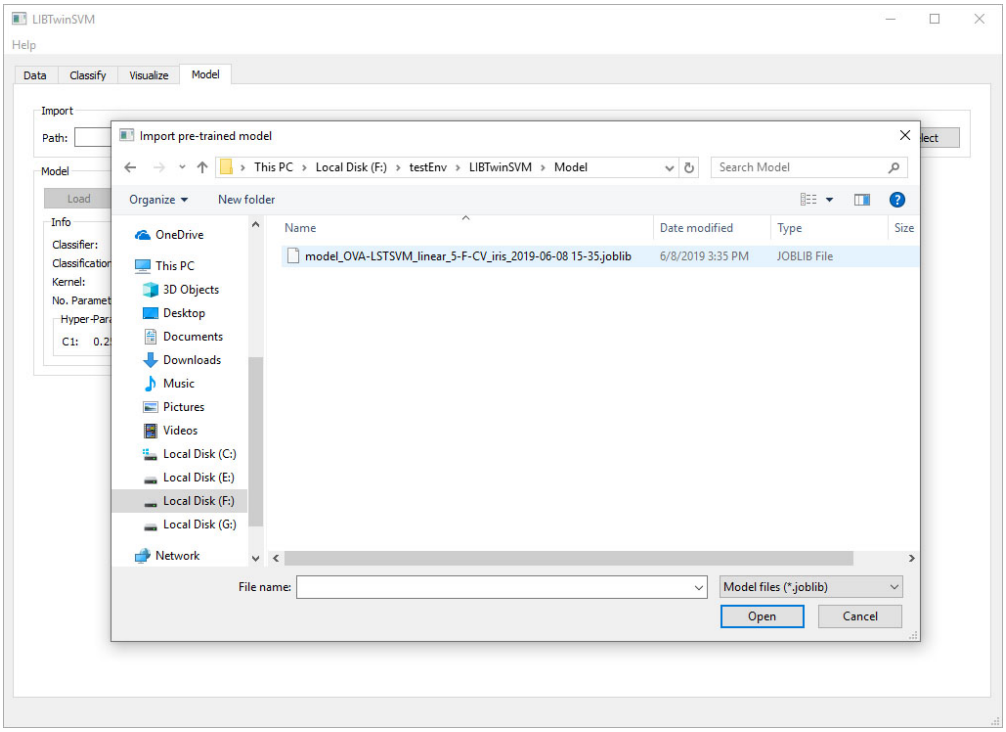
Step 2: Using a Pre-trained Model

Up to now, the dataset should be loaded. Follow the below instructions for evaluating a pre-trained model on test samples.

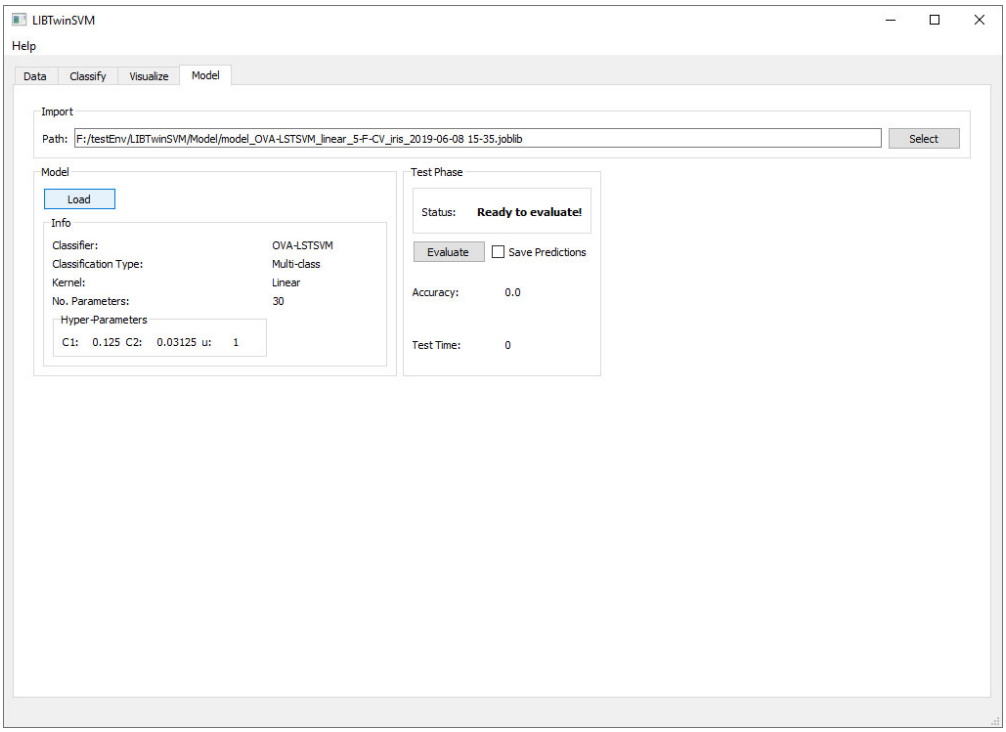
1. First, switch the tabs to **Model** tab.



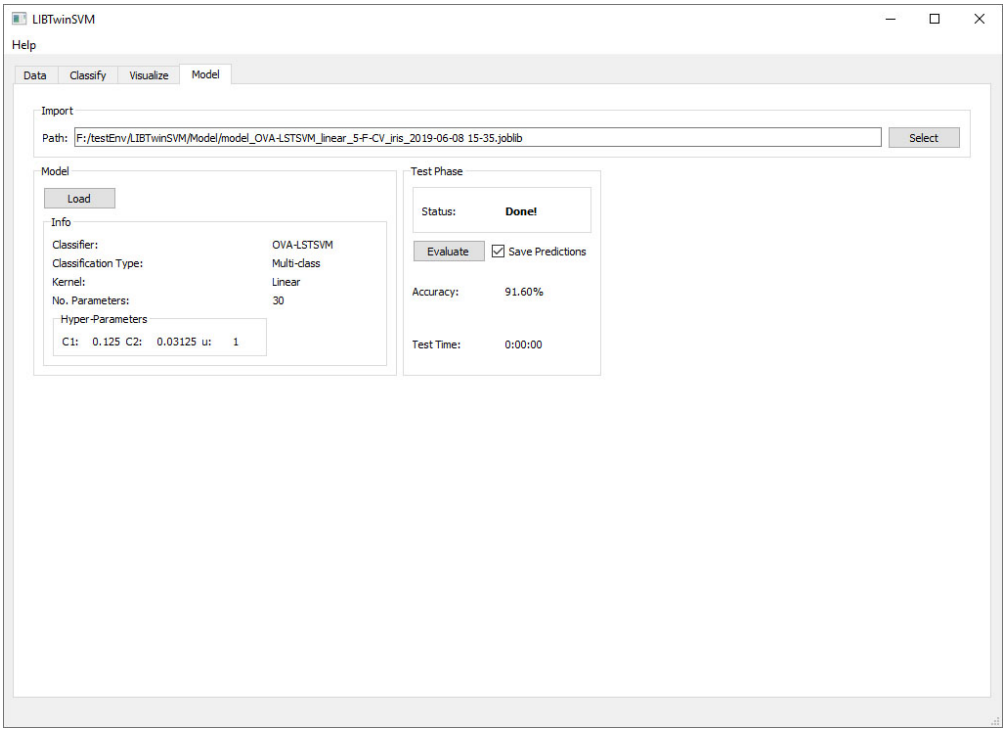
2. Click on the *Select* button in the **Import** box to select your pre-trained model file. You can see a pre-trained model file by LIBTwinSVM in the figure below.



3. After setting the pre-trained model's path, click on the **Load** button to load the model and display the model's characteristics such as its classification type and the used kernel.



4. In the final step, you can evaluate your model by clicking on the **Evaluate** model. Moreover, if you want to save the predicted label of each sample, you can check **Save Predictions** button.



1.1.4 A one-minute example of the GUI usage

This GIF file takes less than a minute to be over!

1.2 API's examples

1.2.1 An example of binary classification using TSVM-based classifiers

Here, we provided an example to help you classify using binary TSVM-based classifiers that are available in the library's API. Comments are also provided in the code example to make API usage clear.

```
from libtsvm.preprocess import DataReader
from libtsvm.estimators import TSVM
from libtsvm.model_selection import Validator

# Step 1: Load your dataset
data_path = '../../dataset/australian.csv'
sep_char = ',' # separtor character of the CSV file
header = True # Whether the dataset has header names.

dataset = DataReader(data_path, sep_char, header)

shuffle_data = True
normalize_data = False

dataset.load_data(shuffle_data, normalize_data)
X, y, file_name = dataset.get_data()
```

(continues on next page)

(continued from previous page)

```
# Step 2: Choose a TSVM-based estimator
kernel = 'linear'
tsvm_clf = TSVM(kernel=kernel)

# Step 3: Evaluate the estimator using train/test split
eval_method = 't_t_split' # Train/Test split
test_set_size = 30 # 30% of samples

val = Validator(X, y, (eval_method, test_set_size), tsvm_clf)
eval_func = val.choose_validator()

# Hyper-parameters of the classifier
h_params =  {'C1': 2**-3, 'C2': 2**-5}

acc, std, full_report = eval_func(h_params)

print("Accuracy: %.2f" % acc)
print(full_report)
```

1.2.2 An example of multi-class classification using OVO-LSTSVM

This example shows how you can use Least Squares TwinSVM classifier with One-vs-One strategy to solve a multi-class classification problem.

```
from libtsvm.preprocess import DataReader
from libtsvm.estimators import LSTSVM
from libtsvm.mc_scheme import OneVsOneClassifier
from libtsvm.model_selection import Validator

# Step 1: Load your dataset
data_path = '../../dataset/iris.csv'
sep_char = ',' # separtor character of the CSV file
header = True # Whether the dataset has header names.

dataset = DataReader(data_path, sep_char, header)

shuffle_data = True
normalize_data = False

dataset.load_data(shuffle_data, normalize_data)
X, y, file_name = dataset.get_data()

# Step 2: Choose a TSVM-based estimator
kernel = 'RBF'
lstsvm_clf = LSTSVM(kernel=kernel)

# Step 3: Select a multi-class approach
ovo_lstsvm = OneVsOneClassifier(lstsvm_clf)

# Step 4: Evaluate the multi-class estimator using train/test split
eval_method = 't_t_split' # Train/Test split
test_set_size = 20 # 20% of samples

val = Validator(X, y, (eval_method, test_set_size), ovo_lstsvm)
eval_func = val.choose_validator()
```

(continues on next page)

(continued from previous page)

```
# Hyper-parameters of the classifier
h_params =  {'C1': 2**-2, 'C2': 2**-2, 'gamma': 2**-7}

acc, std, full_report = eval_func(h_params)

print("Accuracy: %.2f" % acc)
print(full_report)
```

1.2.3 An example of model evaluation with cross-validation

This user guide is provided to help you evaluate the model with cross validation.

```
from libtsvm.preprocess import DataReader
from libtsvm.estimators import TSVM
from libtsvm.model_selection import Validator

# Step 1: Load your dataset
data_path = '../../dataset/hepatits.csv'
sep_char = ',' # separtor character of the CSV file
header = True # Whether the dataset has header names.

dataset = DataReader(data_path, sep_char, header)

shuffle_data = True
normalize_data = False

dataset.load_data(shuffle_data, normalize_data)
X, y, file_name = dataset.get_data()

# Step 2: Choose a TSVM-based estimator
kernel = 'linear'
tsvm_clf = TSVM(kernel=kernel)

# Step 3: Evaluate the estimator using cross validation
eval_method = 'CV' # Cross validation
folds = 5

val = Validator(X, y, (eval_method, folds), tsvm_clf)
eval_func = val.choose_validator()

# Hyper-parameters of the classifier
h_params =  {'C1': 2**-2, 'C2': 2**1}

acc, std, full_report = eval_func(h_params)

print("Accuracy: %.2f" % acc)
print(full_report)
```

1.2.4 An example of model selection with grid search and cross-validation

In this example, a code samples is provided to help you find the best model for your classification task. The best model will be found using grid search and cross validation that are available in the library's API. At the end, the classification results is also saved in a speadsheet file for further analysis.

```
from libtsvm.preprocess import DataReader
from libtsvm.estimators import TSVM
from libtsvm.model_selection import Validator, grid_search, save_result

# Step 1: Load your dataset
data_path = './dataset/australian.csv'
sep_char = ',' # separtor character of the CSV file
header = True # Whether the dataset has header names.

dataset = DataReader(data_path, sep_char, header)

shuffle_data = True
normalize_data = False

dataset.load_data(True, False)
X, y, _ = dataset.get_data()

# Step 2: Choose a TSVM-based estimator
tsvm_clf = TSVM(kernel='RBF')

# Step 3: Choose an evaluation method.
val = Validator(X, y, ('CV', 5), tsvm_clf) # 5-fold cross-validation
eval_method = val.choose_validator()

# Step 4: Specify range of each hyper-parameter for a TSVM-based estimator.
params = {'C1': (-2, 2), 'C2': (-2, 2), 'gamma': (-8, 2)}

best_acc, best_acc_std, opt_params, clf_results = grid_search(eval_method, params)

print("Best accuracy: %.2f+-%.2f | Optimal parameters: %s" % (best_acc, best_acc_std,
↪str(opt_params)))

# Step 5: Save the classification results
clf_type = 'binary' # Type of classification problem
save_result(val, clf_type, clf_results, 'TSVM-RBF-Australian')
```

CHAPTER 2

API Reference

This page contains the list of the project's modules

estimators	In this module, Standard TwinSVM and Least Squares TwinSVM estimators are defined.
mc_scheme	In this module, multi-class schemes such as One-vs-One and One-vs-All are implemented.
model	This modules models data, user input in classes and functions
model_selection	This module contains functions and classes for model evaluation and selection.
preprocess	In this module, functions for reading and processing datasets are defined.
model_eval	This module contains code for saving, loading, and evaluating pre-trained models.

2.1 estimators

In this module, Standard TwinSVM and Least Squares TwinSVM estimators are defined.

Functions

rbf_kernel(x, y, u)	It transforms samples into higher dimension using Gaussian (RBF) kernel.

Classes

BaseTSVM(kernel, rect_kernel, C1, C2, gamma)	Base class for TSVM-based estimators
LSTSVM([kernel, rect_kernel, C1, C2, gamma, ...])	Least Squares Twin Support Vector Machine (LSTSVM) for binary classification.
TSVM([kernel, rect_kernel, C1, C2, gamma])	Standard Twin Support Vector Machine for binary classification.

class estimators.**BaseTSVM**(*kernel*, *rect_kernel*, *C1*, *C2*, *gamma*)

Bases: sklearn.base.BaseEstimator

Base class for TSVM-based estimators

Parameters **kernel** : str

Type of the kernel function which is either 'linear' or 'RBF'.

rect_kernel : float

Percentage of training samples for Rectangular kernel.

C1 : float

Penalty parameter of first optimization problem.

C2 : float

Penalty parameter of second optimization problem.

gamma : float

Parameter of the RBF kernel function.

Attributes

mat_C_t	(array-like, shape = [n_samples, n_samples]) A matrix that contains kernel values.
cls_name	(str) Name of the classifier.
w1	(array-like, shape=[n_features]) Weight vector of class +1's hyperplane.
b1	(float) Bias of class +1's hyperplane.
w2	(array-like, shape=[n_features]) Weight vector of class -1's hyperplane.
b2	(float) Bias of class -1's hyperplane.

Methods

check_clf_params()	Checks whether the estimator's input parameters are valid.
decision_function(X)	Computes distance of test samples from both non-parallel hyperplanes
fit(X, y)	It fits a TSVM-based estimator.
get_params([deep])	Get parameters for this estimator.
get_params_names()	For retrieving the names of hyper-parameters of the TSVM-based estimator.
predict(X)	Performs classification on samples in X using the TSVM-based model.
set_params(**params)	Set the parameters of this estimator.

check_clf_params()
Checks whether the estimator's input parameters are valid.

get_params_names()
For retrieving the names of hyper-parameters of the TSVM-based estimator.

Returns **parameters** : list of str, {['C1', 'C2'], ['C1', 'C2', 'gamma']}

Returns the names of the hyperparameters which are same as the class' attributes.

fit(*X*, *y*)
It fits a TSVM-based estimator. THIS METHOD SHOULD BE IMPLEMENTED IN CHILD CLASS.

Parameters **X** : array-like, shape (n_samples, n_features)

Training feature vectors, where n_samples is the number of samples and n_features is the number of features.

y : array-like, shape(n_samples,)

Target values or class labels.

predict(*X*)
Performs classification on samples in X using the TSVM-based model.

Parameters **X** : array-like, shape (n_samples, n_features)

Feature vectors of test data.

Returns array, shape (n_samples,)

Predicted class lables of test data.

decision_function(*X*)
Computes distance of test samples from both non-parallel hyperplanes

Parameters **X** : array-like, shape (n_samples, n_features)

Returns array-like, shape(n_samples, 2)

distance from both hyperplanes.

class estimators.**TSVM**(*kernel='linear'*, *rect_kernel=1*, *C1=1*, *C2=1*, *gamma=1*)
Bases: *estimators.BaseTSVM*

Standard Twin Support Vector Machine for binary classification. It inherits attributes of *BaseTSVM*.

Parameters **kernel** : str, optional (default='linear')

Type of the kernel function which is either 'linear' or 'RBF'.

rect_kernel : float, optional (default=1.0)

Percentage of training samples for Rectangular kernel.

C1 : float, optional (default=1.0)

Penalty parameter of first optimization problem.

C2 : float, optional (default=1.0)

Penalty parameter of second optimization problem.

gamma : float, optional (default=1.0)

Parameter of the RBF kernel function.

Methods

check_clf_params()	Checks whether the estimator's input parameters are valid.
decision_function(X)	Computes distance of test samples from both non-parallel hyperplanes
fit(X_train, y_train)	It fits the binary TwinSVM model according to the given training data.
get_params([deep])	Get parameters for this estimator.
get_params_names()	For retrieving the names of hyper-parameters of the TSVM-based estimator.
predict(X)	Performs classification on samples in X using the TSVM-based model.
set_params(**params)	Set the parameters of this estimator.

fit (*X_train*, *y_train*)

It fits the binary TwinSVM model according to the given training data.

Parameters **X_train** : array-like, shape (n_samples, n_features)

Training feature vectors, where n_samples is the number of samples and n_features is the number of features.

y_train : array-like, shape(n_samples,)

Target values or class labels.

class estimators.**LSTSVM**(*kernel='linear'*, *rect_kernel=1*, *C1=1*, *C2=1*, *gamma=1*, *mem_optimize=False*)

Bases: *estimators.BaseTSVM*

Least Squares Twin Support Vector Machine (LSTSVM) for binary classification. It inherits attributes of *BaseTSVM*.

Parameters **kernel** : str, optional (default='linear')

Type of the kernel function which is either 'linear' or 'RBF'.

rect_kernel : float, optional (default=1.0)

Percentage of training samples for Rectangular kernel.

C1 : float, optional (default=1.0)

Penalty parameter of first optimization problem.

C2 : float, optional (default=1.0)

Penalty parameter of second optimization problem.

gamma : float, optional (default=1.0)

Parameter of the RBF kernel function.

mem_optimize : boolean, optional (default=False)

If it's True, it optimizes the memory consumption siginificantly. However, the memory optimization increases the CPU time.

Methods

check_clf_params()	Checks whether the estimator's input parameters are valid.
decision_function(X)	Computes distance of test samples from both non-parallel hyperplanes
fit(X, y)	It fits the binary Least Squares TwinSVM model according to the given training data.
get_params([deep])	Get parameters for this estimator.
get_params_names()	For retrieving the names of hyper-parameters of the TSVM-based estimator.
predict(X)	Performs classification on samples in X using the TSVM-based model.
set_params(**params)	Set the parameters of this estimator.

fit (*X*, *y*)

It fits the binary Least Squares TwinSVM model according to the given training data.

Parameters **X** : array-like, shape (n_samples, n_features)

Training feature vectors, where n_samples is the number of samples and n_features is the number of features.

y : array-like, shape(n_samples,)

Target values or class labels.

estimators.**rbf_kernel** (*x*, *y*, *u*)

It transforms samples into higher dimension using Gaussian (RBF) kernel.

Parameters **x, y** : array-like, shape (n_features,)

A feature vector or sample.

u : float

Parameter of the RBF kernel function.

Returns float

Value of kernel matrix for feature vector x and y.

2.2 mc_scheme

In this module, multi-class schemes such as One-vs-One and One-vs-All are implemented.

Functions

mc_clf_no_params(bin_clfs)	It calculates number of parameters for a multi-class model.

Classes

OneVsAllClassifier(estimator)	Multi-class classification using One-vs-One scheme
OneVsOneClassifier(estimator)	Multi-class classification using One-vs-One scheme

class mc_scheme.**OneVsOneClassifier**(*estimator*)

Bases: sklearn.base.BaseEstimator, sklearn.base.ClassifierMixin

Multi-class classification using One-vs-One scheme

The *OneVsOneClassifier* is scikit-learn compatible, which means scikit-learn tools such as cross_val_score and GridSearchCV can be used for an instance of *OneVsOneClassifier*

Parameters **estimator** : estimator object

An estimator object implementing *fit* and *predict*.

Attributes

clf_name	(str) Name of the classifier.
bin_clf_	(list) Stores intances of each binary TSVM classifier.

Methods

fit(X, y)	It fits the OVO-classfier model according to the given training data.
get_params([deep])	Get parameters for this estimator.
predict(X)	Performs classification on samples in X using the OVO-classifier model.
score(X, y[, sample_weight])	Return the mean accuracy on the given test data and labels.
set_params(**params)	Set the parameters of this estimator.

fit(*X*, *y*)

It fits the OVO-classfier model according to the given training data.

Parameters **X** : array-like, shape (n_samples, n_features)

Training feature vectors, where n_samples is the number of samples and n_features is the number of features.

y : array-like, shape(n_samples,)

Target values or class labels.

Returns **self** : object

predict(*X*)

Performs classification on samples in X using the OVO-classifier model.

Parameters **X** : array-like, shape (n_samples, n_features)

Feature vectors of test data.

Returns **y_pred** : array, shape (n_samples,)

Predicted class lables of test data.

class mc_scheme.**OneVsAllClassifier**(*estimator*)

Bases: sklearn.base.BaseEstimator, sklearn.base.ClassifierMixin

Multi-class classification using One-vs-One scheme

Parameters **estimator** : estimator object

An estimator object implementing *fit* and *predict*.

Attributes

clf_name	(str) Name of the classifier.
bin_clf_	(list) Stores intances of each binary TSVM classifier.

Methods

fit(X, y)	**Parameters**
get_params([deep])	Get parameters for this estimator.
predict(X)	Performs classification on samples in X using the OVO-classifier model.
score(X, y[, sample_weight])	Return the mean accuracy on the given test data and labels.
set_params(**params)	Set the parameters of this estimator.

fit(*X*, *y*)

Parameters **X** : array-like, shape (n_samples, n_features)

Training feature vectors, where n_samples is the number of samples and n_features is the number of features.

y : array-like, shape(n_samples,)

Target values or class labels.

Returns **self** : object

predict(*X*)

Performs classification on samples in X using the OVO-classifier model.

Parameters **X** : array-like, shape (n_samples, n_features)

Feature vectors of test data.

Returns **test_labels** : array, shape (n_samples,)

Predicted class lables of test data.

mc_scheme.**mc_clf_no_params**(*bin_clfs*)

It calculates number of parameters for a multi-class model.

Parameters **bin_clfs** : list

Instances of binary TSVM-based estimators.

Returns int

Number of parameters of a multi-class model.

2.3 model

This modules models data, user input in classes and functions

Classes

DataInfo(no_samples, no_features, no_class, ...)	It stores dataset characteristics such as no.
UserInput()	It encapsulates a user's input.

class model.**DataInfo**(*no_samples*, *no_features*, *no_class*, *class_labels*, *header_names*)

Bases: object

It stores dataset characteristics such as no. samples, no. features and etc.

Parameters **no_samples** : int

Number of samples in dataset.

no_features : init

Number of features in dataset.

no_class : int

Number of classes in dataset.

class_labels: array-like

Unique class labels.

header_names: list

Name of every feature in dataset.

class model.**UserInput**

Bases: object

It encapsulates a user's input.

Attributes

X_train	(array-like, shape (n_samples, n_features)) Training feature vectors, where n_samples is the number of samples and n_features is the number of features.
y_train	(array-like, shape(n_samples,)) Target values or class labels.
data_filename	(str) The filename of a user's dataset.
clf_type	(str, {'tsvm', 'lstsvm'}) Type of the classifier.
class_type	(str, {'binary', 'multiclass'}) Type of classification problem.
mc_scheme	(str, {'ova', 'ovo'}) The multi-class strategy
result_path	(str) Path for saving classification results.
save_clf_results	(boolean (default=True)) Whether to save the classification results or not.
save_best_model	(boolean (default=False)) Whether to save the best fitted model or not.
log_file	(boolean) Whether to create a log file or not.
kernel_type	(str, {'linear', 'RBF'}) Type of the kernel function
rect_kernel	(float (default=1.0)) Percentage of training samples for Rectangular kernel.
test_method_tuple	(tuple) A two-element tuple which contains type of evaluation method and its parameter.
step_size	(float) Step size for generating search elements.
C1_range	(tuple) Lower and upper bound for C1 penalty parameter. example: (-4, 5), first element is lower bound and second element is upper bound
C2_range	(tuple) Lower and upper bound for C2 penalty parameter.
u_range	(tuple) Lower and upper bound for gamma parameter.
C1	(float) The penalty parameter.
C2	(float) The penalty parameter.
u	(float) The parameter of the RBF kernel function.
input_complete	(boolean) Whether all the required inputs are set.
linear_db	(boolean) Whether to plot decision boundary or not.
fig_save	(boolean) Whether to save the figure or not.
fig_dpi	(int) DPI of the figure. It determines the quality of the output image.
fig_save_path	(str) The path at which a figure will be saved.
pre_trained_model	(object) A pre-trained TSVM-based classifer.
save_pred	(boolean) Whether to save predicted labels of test samples in a file or not.
save_pred_path	(str) The path at which the file of predicted labels will be saved.

Methods

get_clf_params()	It returns hyper-parameters of the classifier in a dictionary.
get_current_selection()	It returns a user's current selection for confirmation
get_fig_name()	Returns the figure's name based on the user's selection for saving a file.
get_selected_clf()	It returns the classifier that is selected by user.
validate_step_size()	Checks whether step size for generating search elements are valid or not.

get_current_selection()
It returns a user's current selection for confirmation

get_selected_clf()

It returns the classifier that is selected by user.

Returns **clf_obj** : object

An estimator object.

get_clf_params()
It returns hyper-parameters of the classifier in a dictionary.

Returns dict

Hyper-parameters of the classifier.

get_fig_name()
Returns the figure's name based on the user's selection for saving a file.

validate_step_size()
Checks whether step size for generating search elements are valid or not.

Returns boolean

Whether step size is valid or not.

2.4 model_selection

This module contains functions and classes for model evaluation and selection.

Functions

cm_element(y_true, y_pred)	It computes the elements of a confusion matrix.
eval_metrics(y_true, y_pred)	It computes common evaluation metrics such as Accuracy, Recall, Precision, F1-measure, and elements of the confusion matrix.
get_results_filename(file_name, clf_name, ...)	It returns the filename of the results based on user's input.
grid_search(func_eval, params_range[, log_file])	It does grid search for a TSVM-based estimator.
performance_eval(tp, tn, fp, fn)	It computes common evaluation metrics based on the elements of a confusion matrix.
save_result(validator_obj, problem_type, ...)	It saves the detailed classification results in a spreadsheet file (Excel).
search_space(kernel_type, search_type, ...)	It generates all combination of search elements based on the given range of hyperparameters.

Classes

ThreadGS(usr_input)	It runs the Grid Search in a separate thread.
Validator(X_train, y_train, validator_type, ...)	It evaluates a TSVM-based estimator based on the specified evaluation method.

model_selection.**cm_element**(*y_true*, *y_pred*)
It computes the elements of a confusion matrix.

Parameters **y_true** : array-like

Target values of samples.

y_pred : array-like

Predicted class lables.

Returns **tp** : int

True positive.

tn : int

True negative.

fp : int

False positive.

fn : int

False negative.

model_selection.**performance_eval**(*tp*, *tn*, *fp*, *fn*)

It computes common evaluation metrics based on the elements of a confusion matrix.

Parameters **tp** : int

True positive.

tn : int

True negative.

fp : int

False positive.

fn : int

False negative.

Returns **accuracy** : float

Overall accuracy of the model.

recall_p : float

Recall of positive class.

precision_p : float

Precision of positive class.

f1_p : float

F1-measure of positive class.

recall_n : float

Recall of negative class.

precision_n : float

Precision of negative class.

f1_n : float

F1-measure of negative class.

model_selection.**eval_metrics**(*y_true*, *y_pred*)

It computes common evaluation metrics such as Accuracy, Recall, Precision, F1-measure, and elements of the confusion matrix.

Parameters **y_true** : array-like

Target values of samples.

y_pred : array-like

Predicted class lables.

Returns **tp** : int

True positive.

tn : int

True negative.

fp : int

False positive.

fn : int

False negative.

accuracy : float

Overall accuracy of the model.

recall_p : float

Recall of positive class.

precision_p : float

Precision of positive class.

f1_p : float

F1-measure of positive class.

recall_n : float

Recall of negative class.

precision_n : float

Precision of negative class.

f1_n : float

F1-measure of negative class.

class model_selection.**Validator**(*X_train*, *y_train*, *validator_type*, *estimator*)

Bases: object

It evaluates a TSVM-based estimator based on the specified evaluation method.

Parameters **X_train** : array-like, shape (n_samples, n_features)

Training feature vectors, where n_samples is the number of samples and n_features is the number of features.

y_train : array-like, shape (n_samples,)

Target values or class labels.

validator_type : tuple

A two-element tuple which contains type of evaluation method and its parameter. Example: ('CV', 5) -> 5-fold cross-validation, ('t_t_split', 30) -> 30% of samples for test set.

estimator : estimator object

A TSVM-based estimator which inherits from the `BaseTSVM`.

Methods

choose_validator()	It selects an appropriate evaluation method based on the input paramters.
cv_validator(dict_param)	It evaluates a TSVM-based estimator using the cross-validation method.
cv_validator_mc(dict_param)	It evaluates a multi-class TSVM-based estimator using the cross-validation.
tt_validator(dict_param)	It evaluates a TSVM-based estimator using the train/test split method.
tt_validator_mc(dict_param)	It evaluates a multi-class TSVM-based estimator using the train/test split method.

cv_validator(*dict_param*)

It evaluates a TSVM-based estimator using the cross-validation method.

Parameters **dict_param** : dict

Values of hyper-parameters for a TSVM-based estimator

Returns float

Mean accuracy of the model.

float

Standard deviation of accuracy.

dict

Evaluation metrics such as Recall, Percision and F1-measure for both classes as well as elements of the confusion matrix.

tt_validator(*dict_param*)

It evaluates a TSVM-based estimator using the train/test split method.

Parameters **dict_param** : dict

Values of hyper-parameters for a TSVM-based estimator

Returns float

Accuracy of the model.

float

Zero standard deviation.

dict

Evaluation metrics such as Recall, Percision and F1-measure for both classes as well as elements of the confusion matrix.

cv_validator_mc(*dict_param*)
It evaluates a multi-class TSVM-based estimator using the cross-validation.

Parameters **dict_param** : dict

Values of hyper-parameters for a multi-class TSVM-based estimator.

Returns float

Accuracy of the model.

float

Zero standard deviation.

dict

Evaluation metrics such as Recall, Percision and F1-measure.

tt_validator_mc(*dict_param*)
It evaluates a multi-class TSVM-based estimator using the train/test split method.

Parameters **dict_param** : dict

Values of hyper-parameters for a TSVM-based estimator

Returns float

Accuracy of the model.

float

Zero standard deviation.

dict

Evaluation metrics such as Recall, Percision and F1-measure.

choose_validator()
It selects an appropriate evaluation method based on the input paramters.

Returns object

An evaluation method for assesing a TSVM-based estimator's performance.

model_selection.**search_space**(*kernel_type*, *search_type*, *C1_range*, *C2_range*, *u_range*, *step=1*)
It generates all combination of search elements based on the given range of hyperparameters.

Parameters **kernel_type** : str, {'linear', 'RBF'}

Type of the kernel function which is either 'linear' or 'RBF'.

search_type : str, {'full', 'partial'}

Type of search space

C1_range : tuple

Lower and upper bound for C1 penalty parameter.

C2_range : tuple

Lower and upper bound for C2 penalty parameter.

u_range : tuple

Lower and upper bound for gamma parameter.

step : int, optinal (default=1)

Step size to increase power of 2.

Returns list

Search elements.

model_selection.**get_results_filename**(*file_name*, *clf_name*, *kernel_name*, *test_method*)

It returns the filename of the results based on user's input.

Parameters **file_name** : str

Name of the dataset file.

clf_name : str

Name of the classifier.

kernel_name : str

Name of kernel function.

test_method : tuple

A two-element tuple which contains type of evaluation method and its

parameter.

Returns **output** : str

Filename of the results.

model_selection.**save_result**(*validator_obj*, *problem_type*, *gs_result*, *output_file*)

It saves the detailed classification results in a spreadsheet file (Excel).

Parameters **problem_type** : str, {'binary', 'multiclass'}

Type of the classification problem.

validator_obj : object

The evaluation method that was used for the assesment of the TwinSVM classifier.

gs_result : list

Classification results of the TwinSVM classifier using different set of hyperparameters.

output_file : str

The full path and filename of the classification results. ex. C:UsersMirfile.xlsx

Returns str

Path to the saved spreadsheet (Excel) file.

model_selection.**grid_search**(*func_eval*, *params_range*, *log_file=None*)

It does grid search for a TSVM-based estimator. Note that this function is defined for API usage.

Parameters **func_eval** : object

An evaluation method for assesing a TSVM-based estimator's performance.

params_range : dict

Range of each hyper-parameter.

log_file : object (default=None)

An opened file for logging best classification accuracy.

Returns max_acc

Best accuracy obtained after the grid search.

max_acc_std

Standard deviation of the best accuracy.

dict

Optimal hyper-parameters.

list

Classification results for every hyper-parameters.

class model_selection.**ThreadGS**(*usr_input*)

Bases: PyQt5.QtCore.QObject

It runs the Grid Search in a separate thread.

Parameters **usr_input** : object

An instance of UserInput class which holds the user input.

Methods

blockSignals(self, b)	
childEvent(self, a0)	
children(self)	
connectNotify(self, signal)	
customEvent(self, a0)	
deleteLater(self)	
destroyed	destroyed(self, object: typing.Optional[QObject] = None) [signal]
disconnect(a0)	
disconnectNotify(self, signal)	
dumpObjectInfo(self)	
dumpObjectTree(self)	
dynamicPropertyNames(self)	
event(self, a0)	
eventFilter(self, a0, a1)	
findChild(self, type, name, options, ...)	findChild(self, types: Tuple, name: str = '', options: Union[Qt.FindChildOptions, Qt.FindChildOption] = Qt.FindChildrenRecursively) -> QObject

Continued on next page

Table 16 – continued from previous page

findChildren(self, type, name, options, ...)	findChildren(self, types: Tuple, name: str = '', options: Union[Qt.FindChildOptions, Qt.FindChildOption] = Qt.FindChildrenRecursively) -> List[QObject] findChildren(self, type: type, regExp: QRegExp, options: Union[Qt.FindChildOptions, Qt.FindChildOption] = Qt.FindChildrenRecursively) -> List[QObject] findChildren(self, types: Tuple, regExp: QRegExp, options: Union[Qt.FindChildOptions, Qt.FindChildOption] = Qt.FindChildrenRecursively) -> List[QObject] findChildren(self, type: type, re: QRegularExpression, options: Union[Qt.FindChildOptions, Qt.FindChildOption] = Qt.FindChildrenRecursively) -> List[QObject] findChildren(self, types: Tuple, re: QRegularExpression, options: Union[Qt.FindChildOptions, Qt.FindChildOption] = Qt.FindChildrenRecursively) -> List[QObject]
inherits(self, classname)	
initialize()	It passes a user's input to the functions and classes for solving a classification task.
installEventFilter(self, a0)	
isSignalConnected(self, signal)	
isWidgetType(self)	
isWindowType(self)	
killTimer(self, id)	
metaObject(self)	
moveToThread(self, thread)	
objectName(self)	
objectNameChanged	objectNameChanged(self, objectName: str) [signal]
parent(self)	
property(self, name)	
pyqtConfigure(...)	Each keyword argument is either the name of a Qt property or a Qt signal.
receivers(self, signal)	
removeEventFilter(self, a0)	
run_gs(func_eval, search_space)	Runs grid search for the selected classifier on specified hyper-parameters.
sender(self)	
senderSignalIndex(self)	
setObjectName(self, name)	
setParent(self, a0)	
setProperty(self, name, value)	
sig_finished	
sig_gs_info_set	
sig_pbar_set	
signalsBlocked(self)	
startTimer(self, interval, timerType)	
stop()	Stops the thread of the grid search.

Continued on next page

Table 16 – continued from previous page

thread(self)	
timerEvent(self, a0)	
tr(self, sourceText, disambiguation, n)	

run_gs (*func_eval*, *search_space*)
Runs grid search for the selected classifier on specified hyper-parameters.

Parameters **func_eval** : object

An evaluation method for assesing a TSVM-based estimator's performance.

search_space : list

Search elements.

Returns list

Classification results for every hyper-parameters.

initialize()
It passes a user's input to the functions and classes for solving a classification task. The steps that this function performs can be summarized as follows:

1. Specifies a TwinSVM classifier based on the user's input.
2. Chooses an evaluation method for assessment of the classifier.
3. Computes all the combination of search elements.

#. Computes the evaluation metrics for all the search element using grid search. #. Saves the detailed classification results in a spreadsheet file (Excel).

Returns object

The evalution method.

dict

Grids of search elements.

stop()
Stops the thread of the grid search.

2.5 preprocess

In this module, functions for reading and processing datasets are defined.

Functions

read_libsvm(filename)	It reads LIBSVM data files for doing classification using the TwinSVM model.

Classes

DataReader(file_path, sep, header)	It handels data-related tasks like reading, etc.

class preprocess.**DataReader**(*file_path*, *sep*, *header*)

Bases: object

It handels data-related tasks like reading, etc.

Parameters **file_path** : str

Path to the dataset file.

sep : str

Separator character

header : boolean

whether the dataset has header names or not.

Attributes

X_train	(array-like, shape (n_samples, n_features)) Training samples in NumPy array.
y_train	(array-like, shape(n_samples,)) Class labels of training samples.
hdr_names	(list) Header names of datasets.
filename	(str) dataset's filename

Methods

get_data()	It returns processed dataset.
get_data_info()	It returns data characteristics from dataset.
load_data(shuffle, normalize)	It reads a CSV file into pandas DataFrame.

load_data(*shuffle*, *normalize*)

It reads a CSV file into pandas DataFrame.

Parameters **shuffle** : boolean

Whether to shuffle the dataset or not.

normalize : boolean

Whether to normalize the dataset or not.

get_data()

It returns processed dataset.

Returns array-like

Training samples in NumPy array.

array-like

Class labels of training samples.

str

The dataset's filename

get_data_info()

It returns data characteristics from dataset.

object data characteristics

preprocess.**read_libsvm**(*filename*)
It reads LIBSVM data files for doing classification using the TwinSVM model.

Parameters **filename** : str

Path to the LIBSVM data file.

Returns array-like

Training samples.

array-like

Class labels of training samples.

str

Dataset's filename

2.6 model_eval

This module contains code for saving, loading, and evaluating pre-trained models.

Functions

load_model(model_path)	It loads a pre-trained TSVM-based estimator.
save_model(validator, params, output_file)	It saves an estimator with specified hyper-parameters and a evaluation method.

Classes

ModelThread(usr_in)	Evaluates a pre-trained model in a thread.

model_eval.**save_model**(*validator*, *params*, *output_file*)
It saves an estimator with specified hyper-parameters and a evaluation method.

Parameters **validator** : object

An evaluation method.

params : dict

Hyper-parameters of the estimator.

output_file : str

The full path and filename of the saved model.

model_eval.**load_model**(*model_path*)
It loads a pre-trained TSVM-based estimator.

Parameters **model_path** : str

The path at which the model is stored.

Returns object

A pre-trained estimator.

dict

Model information.

class model_eval.**ModelThread**(*usr_in*)

Bases: PyQt5.QtCore.QObject

Evaluates a pre-trained model in a thread.

Parameters **usr_input** : object

An instance of UserInput class which holds the user input.

Methods

blockSignals(self, b)	
childEvent(self, a0)	
children(self)	
connectNotify(self, signal)	
customEvent(self, a0)	
deleteLater(self)	
destroyed	destroyed(self, object: typing.Optional[QObject] = None) [signal]
disconnect(a0)	
disconnectNotify(self, signal)	
dumpObjectInfo(self)	
dumpObjectTree(self)	
dynamicPropertyNames(self)	
eval_model()	It evaluates a pre-trained model on test samples.
event(self, a0)	
eventFilter(self, a0, a1)	
findChild(self, type, name, options, ...)	findChild(self, types: Tuple, name: str = '', options: Union[Qt.FindChildOptions, Qt.FindChildOption] = Qt.FindChildrenRecursively) -> QObject

Continued on next page

Table 22 – continued from previous page

findChildren(self, type, name, options, ...)	findChildren(self, types: Tuple, name: str = '', options: Union[Qt.FindChildOptions, Qt.FindChildOption] = Qt.FindChildrenRecursively) -> List[QObject] findChildren(self, type: type, regExp: QRegExp, options: Union[Qt.FindChildOptions, Qt.FindChildOption] = Qt.FindChildrenRecursively) -> List[QObject] findChildren(self, types: Tuple, regExp: QRegExp, options: Union[Qt.FindChildOptions, Qt.FindChildOption] = Qt.FindChildrenRecursively) -> List[QObject] findChildren(self, type: type, re: QRegularExpression, options: Union[Qt.FindChildOptions, Qt.FindChildOption] = Qt.FindChildrenRecursively) -> List[QObject] findChildren(self, types: Tuple, re: QRegularExpression, options: Union[Qt.FindChildOptions, Qt.FindChildOption] = Qt.FindChildrenRecursively) -> List[QObject]
inherits(self, classname)	
installEventFilter(self, a0)	
isSignalConnected(self, signal)	
isWidgetType(self)	
isWindowType(self)	
killTimer(self, id)	
metaObject(self)	
moveToThread(self, thread)	
objectName(self)	
objectNameChanged	objectNameChanged(self, objectName: str) [signal]
parent(self)	
property(self, name)	
pyqtConfigure(...)	Each keyword argument is either the name of a Qt property or a Qt signal.
receivers(self, signal)	
removeEventFilter(self, a0)	
sender(self)	
senderSignalIndex(self)	
setObjectName(self, name)	
setParent(self, a0)	
setProperty(self, name, value)	
sig_update_model_eval	
signalsBlocked(self)	
startTimer(self, interval, timerType)	
thread(self)	
timerEvent(self, a0)	
tr(self, sourceText, disambiguation, n)	

eval_model()

It evaluates a pre-trained model on test samples.

Python Module Index

Index

R

S

T

U

V