User manual

ECLiPSE – Enhanced Classification of Localized Pointclouds by Shape Extraction

Current version: version 1.0 (May 9th, 2023)

Reference: Hugelier, S., Kim, H., Gyparaki, M.T., Bond, C., Tang, Q., Santiago-Ruiz, A.N., Porta, S., Lakadamyali, M. ECLiPSE: a versatile classification technique for structural and morphological analysis of super-resolution microscopy data. BioRxiv (2023). DOI: https://doi.org/10.1101/2023.05.10.540077.

Contact: Siewert Hugelier: siewert.hugelier@pennmedicine.upenn.edu & Melike Lakadamyali: melikel@pennmedicine.upenn.edu.

To take full advantage of the ECLiPSE analysis pipeline, please download (a free trial of) the PLS toolbox (www.eigenvector.com).

Introduction

This is the user guide for the automated machine learning analysis pipeline for classification of cellular structures captured using single molecule localization microscopy (SMLM). The method is called *Enhanced Classification of Localized Pointclouds by Shape Extraction* (ECLiPSE) and is implemented in MATLAB (The MathWorks). An overview of the pipeline is shown in Figure 1. ECLiPSE calculates 67 comprehensive shape descriptors encompassing geometric, boundary, skeleton, and other properties. These properties can then be used, before or after the optional variable selection step, to train classification models to accurately predict the type of different cellular structures across various biological contexts.

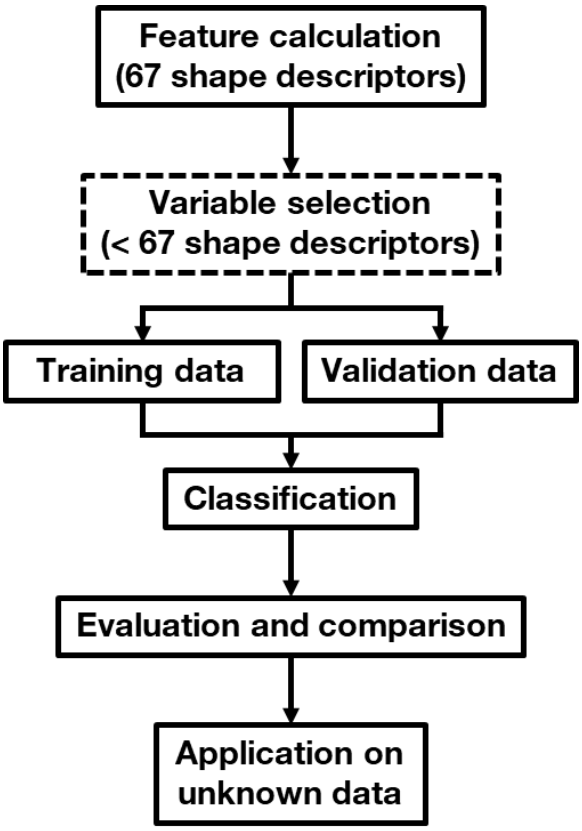

Figure 1 Fully automated pipeline of ECLiPSE. From data acquisition to model application on unknown data.

ECLiPSE workflow

ECLiPSE is to be applied on the already segmented (clustered) localizations, which can be obtained using one of the many available tools (e.g., Voronoi Tessellation, DBScan)[1-3]. Once these clustered localizations are obtained, the full pipeline of ECLiPSE contains the following steps:

Step 1: Descriptor calculation

The descriptors, also called features, are the morphological properties that are calculated from the different clusters. ECLiPSE calculates 67 different morphological descriptors, based on geometric, boundary, skeleton, texture, HuMoment, and fractal properties. Most of these properties are calculated directly from the localizations to accurately characterize the individual structures. A comprehensive list and their implementation can be found in Supplementary Table 1 of the reference listed above.

Step 2: Data exploration using Principal Component Analysis (PCA)

Once the descriptors are calculated, the data obtained for each of these different clusters can be explored using PCA. PCA is a dimensionality reduction technique that linearly transforms the data into a new coordinate system in which (most of) the variation in the data can be explained using fewer variables than in the original data.

Data points that are close together in this new coordinate system are closely related data points, and in the context of this application, this roughly means that they look morphologically very similar. Data points can also be color coded in the PCA implementation of ECLiPSE if information about distinct groups is known, to reveal their degree of separation.

Step 3: Variable selection

Not all shape descriptors will be informative or relevant in every biological application, and some features may even obscure the information present in the data. To circumvent these issues, and maximize model performance, reduce model overfitting and remove correlations, variable selection methods can be used. These methods are (supervised) algorithms that determine the usefulness and relevance of single variables to attain the analysis goal. For ECLiPSE, this goal is to maximize classification prediction accuracy. It is worth noting that some classification algorithms (see Step 4) have some intrinsic variable selection capabilities (in the form of data compression), but this is not the case for all classification algorithms. It was therefore explicitly implemented in the ECLiPSE pipeline.

A collection of ten different variable selection methods is implemented in ECLiPSE, which can be used separately or in a combined fashion. Six methods are either provided as built-in MATLAB functions or as stand-alone functions: backwards interval PLS, recursive PLS, Chi square test, Minimum Redundancy Maximum Relevance, Boruta. The remaining four methods are available through the PLS Toolbox (Eigenvector Inc.): backwards interval PLS, recursive PLS, interval PLS, genetic algorithm. Note that two methods have multiple implementations which may lead to somewhat different results related to their slightly different implementations and selection criteria. A comparison of these methods is found in Supplementary Note 1 of the reference listed above.

Once this optional variable selection step is completed, the variable-selected data can be explored using PCA as well, and compared to the original, non-variable-selected, data.

Step 4: Classification

The classification step in ECLiPSE contains both unsupervised and supervised classification methods. Unsupervised classification aims at grouping similar samples without relying on any prior knowledge of class membership within the data, and therefore identifies natural clusters

inherent in the data, whereas supervised classification leverages known class information to construct its models. Two different unsupervised methods (k-means and hierarchical clustering) and seven supervised classification methods (binary/multiclass adaptive boosting, k-nearest neighbors, adaptive logistic regression, random forest, random undersampling boosting, logistic regression, and partial least squares classification) are implemented in the ECLiPSE pipeline.
Once the data has been separated into training and validation data (i.e., a subset of the total data), the different available methods can all be applied onto the same data (this can be selected within the pipeline). This allows for a direct comparison of the performance of the classification models and their prediction accuracy to automatically select the best performing one(s). The performance of the models can also be visualized in a so-called confusion matrix (which shows the true positive, false positive, true negative, and false negative rates for each modelled class).
A commonly used classification strategy is to perform multiple iterations of the classification step, each time selecting a different training/validation data subset. This process removes randomness from the classification performance which could occur using a single iteration (i.e., extreme performances, good or bad, by coincidentally selecting well-separated or poorly separated data points), and provides an accuracy on the performance (mean ± std). Moreover, it also allows model agglomeration, which can lead to better prediction capabilities when predicting class association of unknown samples.

Step 5: Predicting unknown samples

The last step in the ECLiPSE analysis pipeline is to use the best model(s) obtained from the previous step and apply them on samples of which the class association is not known (i.e., data that was not used during training and validation of the models). This step is also called the prediction step, and can be used to make interpretations of the data (e.g., if new data points are available that were not used before).

MATLAB example script

An example script of the ECLiPSE analysis pipeline is provided, together with an example data set, which was taken from the validation data set used in the paper (see reference above). This validation data set contains clusters from five distinct cellular structures: Tau protein aggregates, Nuclear Pore Complexes (NPCs), Microtubules, Lysosomes, and Mitochondria. Each of these examples contains 500 individual clusters, which is a subset of the entire data used in the paper. Because this is a subset of the data, and the example script is not run entirely in the entire same manner as was done in the paper, the results slightly deviate from the ones presented in the paper. This was done to save time, as some of the algorithms may run for a long time. The total run time of this example script on an i7-12700H 2.30GHz Windows 10 system (32GB Ram Memory) was ~ 50 minutes.

The example data contains five different variables with the raw (already clustered) localizations for each cluster, and five different variables with the class associations (i.e., a vector with the same length as the number of clusters each cellular structures has – 500 in this case – of 1s, 2s, 3s, 4s, or 5s, respectively). The input variables are listed below (alphabetically):

- ClassLysosomes (1 x 500 double; all elements: 4)
- ClassMicrotubules (1 x 500 double; all elements: 3)
- ClassMitochondria (1 x 500 double; all elements: 5)
- ClassNPCs (1 x 500 double; all elements: 2)
- ClassTau (1 x 500 double; all elements: 1)
- Lysosomes (500 x 1 cell; each cell contains 2 columns: [x y])
- Microtubules (500 x 1 cell; each cell contains 2 columns: [x y])
- Mitochondria (500 x 1 cell; each cell contains 2 columns: [x y])
- NPCs (500 x 1 cell; each cell contains 2 columns: [x y])
- Tau (500 x 1 cell; each cell contains 2 columns: [x y])

The example script can be run using the following options:

- Option 1: add the ECLiPSE folder and subfolder to the path of MATLAB
  Type ‘run ExampleScript.m’ in the command window
- Option 2: Open the ExampleScript.m script in MATLAB
  Press ‘Run’ in the Editor tab
- Option 3: Open the ExampleScript.m script in MATLAB
  Run each section independently by placing your cursor in the appropriate section and press ‘Run Section’ in the Editor tab (shortcut: ctrl + enter).

To run ECLiPSE on your own data, you can recreate the structure of the example data and change the ExampleScript.m accordingly, so it recognizes the different variables. The first lines of some of the sections (Sections 4, 5, 7, and 8) also contain some input that can be changed accordingly.

Please ensure that the PLS Toolbox is installed before you run the ExampleScript.m file in its current form. If not, it will throw some errors during the variable selection and classification sections (sections 4, 5, 7 and 8).

Section 1 – Data preparation

This section makes all the necessary preparations to run the script. It adds the folders and subfolders to the MATLAB path, and then loads the ExampleData.mat file.

Section 2 – Descriptor calculation
This section calculates all the morphological descriptors for the individual clusters of each type of cellular structure. It then removes the clusters for which NaN or Inf values are obtained (in this example, there are none). **If there are less than 500 samples for each type after this step, something has gone wrong.**

ECLiPSE functions used (see next section):
- CalcDescriptors.m
- RemoveNaNInf.m

Section 3 – Data exploration with Principal Component Analysis (PCA)
This section explores the data using PCA. Both the 2D and 3D plots are created (and saved as .png and .fig). The data is autoscaled before performing the PCA to ensure that all variables have an equal contribution to the PCA.

ECLiPSE functions used (see next section):
- PCAPlots.m

Section 4 – Variable selection
This section runs the variable selection on the data. It runs 50 iterations on a subset of the data (using a random selection of 50% of the minimum number of class samples at each time; 100 per class in this example script) of the 'rPLS', 'ReliefF', and 'rPLS_PLSToolbox' variable selection methods. These are the 3 input parameters that can be changed in this section of the script. **Please make sure to remove the 'rPLS_PLSToolbox' method if the PLS toolbox is not installed.**

ECLiPSE functions used (see next section):
- CreateTrainData.m
- VariableSelection.m

Section 5 – Plot variable selection results
This section plots the results obtained by the variable selection algorithms. It plots the results obtained for the three individual variable selection methods, and then also plots the agglomerated results. It is recommended to use more than one variable selection algorithm to determine the usefulness of the variables as this increases the robustness of the selections.

ECLiPSE functions used (see next section):
- VariableSelectionPlots.m

Section 6 – Data exploration (after variable selection) with PCA
This section explores the variable selected data using PCA. Both the 2D and 3D plots are once again created (and saved as .png and .fig). The data is again autoscaled before doing the PCA to ensure that all variables have an equal contribution to the PCA.

ECLiPSE functions used (see next section):
- PCAPlots.m
- HistogramPlots.m

Section 7 – Classification
This section performs the classification of the data on both the data without and with variable selection. It runs 200 iterations (using a random selection of 50% of the minimum number of class

samples at each time; 100 per class in this example script) for the ‘knn’, ‘RandomForest’, and ‘LReg-DA’ classification methods. The models for ‘knn’ and ‘LReg-DA’ are saved at each time (the models for ‘RandomForest’ are too large). The ‘Kmeans’ unsupervised classification is ran on the entire data set. These are also the 3 input parameters that can be changed by the user. **Please make sure to remove the ‘LReg-DA’ method if the PLS toolbox is not installed.**

ECLiPSE functions used (see next section):

- CreateTrainData.m
- ClassificationTrain.m
- ClassificationPrediction.m

Section 8 – Plot classification results

This section plots the results obtained by the different classification methods. It calculates and shows the confusion matrices for the supervised methods (and additionally for the 10 best ones each time), and saves the images of the results obtained by the unsupervised method. Please note that the ‘Kmeans’ results are inferior to the ‘HCA’ results, but ‘Kmeans’ was included in the example script due to it not being dependent on the PLS Toolbox.

ECLiPSE functions used (see next section):

- ConfusionMatPlots.m
- ClusterPlots.m

MATLAB functions

There are eleven main functions in the ECLiPSE pipeline. These are the only functions that should be communicated with when using ECLiPSE, and running the analysis on your own data. The remainder of the functions that is provided in the available code are support functions. Every function is well documented and additional help on how to use it can be accessed by typing 'help function' in the MATLAB command window.

If you notice any bug in one of the available codes, please do not hesitate to contact siewert.hugelier@pennmedicine.upenn.edu! We appreciate you testing out the software and will respond in a timely manner to get these bugs fixed.

The available main functions are alphabetically listed below, with a short description and the required input, optional input (if any) and their output (if any).

CalcDescriptors.m

This function calculates all the morphological features of each of the pointcloud clusters (67 total). An overview of the descriptors in found in Supplementary Table 1 of the reference listed above.

Input: **Data** – a cell (n x 1) of m clusters. Each cell contains a list of coordinates (2 columns of x and y coordinates).

Output: **Descriptorstable** – a table (n x 67) containing the calculated descriptors. Each row will be representative of a cluster.

ClassificationPrediction.m

This function uses a classification model to predict class associations of unknown samples. The data of the unknown samples must be provided in the same way as the model was made, or compatibility issues will arise.

Input: **Data** – a matrix (m x n) of data containing each unknown sample in the rows (m samples), and the descriptor variables in the columns (n descriptors).
Model – a trained classification model (should be of the same type as the 'Method' provided).
Method – the classification method used.
Choices: 'knn' – 'RandomForest' – 'PLS-DA' – 'LReg-DA' – 'AdaboostM1' (binary) – 'AdaBoostM2' (multiclass) – 'RUSBoost' – 'LogitBoost' – 'HCA' – 'Kmeans'.

Optional Input: **Options** – the options for either 'PLS-DA' or 'LReg-DA'. Only available with the PLS toolbox.
DoPCA – perform Principal Component Analysis before the classification. Only available for the 'Kmeans' method.
nComp – the number of components retained for the PCA analysis. Only available for the 'Kmeans' method.

Output: **Prediction** – a vector (m x 1) containing the predictions of the unknown samples.

ClassificationTrain.m

This function uses training data and ground truth class association (supervised methods) or the number of classes (unsupervised methods) to train a classification model.

Input: **Data** – a matrix (m x n) of data containing each unknown sample in the rows (m samples), and the descriptor variables in the columns (n descriptors).
Class – a vector (m x 1) containing the class associations for each sample (supervised methods) or a scalar representing the number of classes in the data (unsupervised methods)
Method – the classification method used.
Choices: 'knn' – 'RandomForest' – 'PLS-DA' – 'LReg-DA' – 'AdaboostM1' (binary) – 'AdaBoostM2' (multiclass) – 'RUSBoost' – 'LogitBoost' – 'HCA' – 'Kmeans'.

Optional input: **knn_Neighbours** – the number of neighbors considered in the knn model (see 'help fitcknn' for more information).
knn_Distance – the distance metric considered in the knn model (see 'help fitcknn' for more information).
RF_Trees – the number of trees considered in the Random Forest model (see 'help treebagger' for more information).
Options – the options for either 'PLS-DA' or 'LReg-DA' classification. Only available with the PLS toolbox.
Learner – the learner type for the 'AdaBoostM1', 'AdaBoostM2', 'RUSBoost', and 'LogitBoost' classification methods. See 'help fitcensemble' for more information.
NumLearningCycles – the number of cycles used in the learning process for 'AdaBoostM1', 'AdaBoostM2', 'RUSBoost', and 'LogitBoost' classification methods. See 'help fitcensemble' for more information.
RatioToSmallest – the sampling portion for the 'RUSBoost' classification method. See 'help fitcensemble' for more information.

Output: **ClassificationModel** – the model of the classification. The structure is different depending on the classification method used.

ClusterPlots.m
This function plots the localizations of the individual clusters.

Input: **Clusters** – a cell (n x 1) containing the localizations of each cluster individually.

Optional input: **SaveAs** – the path to save the images in. If this is empty, the images will not be saved.
PlotConfig – the number of subplots within the figure. This is provided as a [m n] vector (m: number of rows; n: number of columns).
PixelSize – the size of a single pixel in nm.
PlotSize – the size of each individual plot, provided in pixels (axis will be: [-PlotSize PlotSize -PlotSize PlotSize]).
FullScreen – whether the figure should be plotted as fullscreen (1) or not (0).
PlotScalebar – the size of the scalebar in nm. Empty if none.

ConfusionMatPlots.m
This function plots the confusion matrices of the classification results. A confusion matrix shows the true positive, false positive, true negative, and false negative rates for each modelled class,

and is a measure for how well the prediction is (usually evaluated on validation data that was not included during the training process).

Input: **Predicted** – the predicted class associations, specified as a matrix (m x n; m: number of samples, n: number of models).
GroundTruth – the known ground truth for each sample, specified as a matrix (m x n; m: number of samples, n: number of models).

Optional input: **SaveAs** – the path to save the images in. If this is empty, the confusion matrices will not be saved.
Plot – whether the confusion matrices should be plotted (1) or not (0).
FullScreen – whether the figure should be plotted as fullscreen (1) or not (0).
Title – the title of the confusion matrices.
Labels – the labels for each of the individual classes.
BestNumModels – the number of best models that should be retained (based on the average prediction accuracy).
BestNumModelsIdx – the indices of the best models (or any model) that should be used for the calculations.
OnlyBestModels – only plot the best models (1) or not (0).

Output: **Confusionmat** – the confusion matrices for each of the individual models.
SumDiags – the average prediction accuracy for each of the individual models.
BestModelIndex – the indices of the best performing model (based on the average prediction accuracy).

CreateTrainData.m
This function automatically separates the data into a training data set and test data set or validation data set, if the latter is needed.

Input: **Data** – the data containing the descriptors (m x n; m: number of samples, n: number of descriptors).
Class – the known class associations that are associated to each data point in the Data matrix (m x 1; m: number of samples).

Optional input: **ClassSize** – the number of the samples of each class that should be used to create the training data (provided as fraction or absolute numbers, either as a scalar or a vector).
NeedTestData – whether a test data set must be created as well (1) or not (0).

Output: **TrainData** – the training data obtained from randomly selecting from the input data (m x n; m: number of samples, n: number of descriptors).
TrainClass – the known class associations of each sample in the TrainData (m x 1; m: number of samples).
TestData – the test data obtained from randomly selecting from the input data (m x n; m: number of samples, n: number of descriptors).
TestClass – the known class associations of each sample in the TestData (m x 1; m: number of samples).

HistogramPlots.m
This function plots the histograms of the distributions of each individual variable for each class separately.

Input: **Data** – the data containing the descriptors (m x n; m: number of samples, n: number of descriptors).
Class – the known class associations that are associated to each data point in the Data matrix (m x 1; m: number of samples).

Optional input: **SaveAs** – the path to save the images in. If this is empty, the images will not be saved.
AutoScale – whether the data must be autoscaled before calculating and plotting the histograms.
VariableNames – the name of the variables of which the distributions are being plotted.
Bins – the number of bins each histogram has.
Range – the percentage of the data that should be excluded (to ensure that outliers are removed).
PlotConfig – the number of subplots within the figure. This is provided as a [m n] vector (m: number of rows; n: number of columns).
FullScreen – whether the figure should be plotted as fullscreen (1) or not (0).
Legend – the legend of each of the classes.

PCAPlots.m
This function shows the PCA plots of the data. IT calculates the PCA, and then plots the results in 2D or 3D plots. The data points can be pseudo-colored according to their class associations.

Input: **Data** – the data containing the descriptors (m x n; m: number of samples, n: number of descriptors).
Class – the known class associations that are associated to each data point in the Data matrix (m x 1; m: number of samples).

D2orD3 – whether the plots have to be visualized in 2D or 3D.

Optional input: **AutoScale** – whether the data must be autoscaled before calculating and plotting the PCA plots.
SaveAs – the path to save the images in. If this is empty, the images will not be saved.
FullScreen – whether the figure should be plotted as fullscreen (1) or not (0).
Title – the title of the confusion matrices.
Legend – the legend of each of the classes.
MarkerSize – the size of each of the data points in the PCA plots.
AxisView – the axis scale for the view. This is to ensure that outliers do not heavily influence the visual representation of the plot
Ax1LabelPos – the position of the first axis label (useful for 3D plots).
Ax2LabelPos – the position of the second axis label (useful for 3D plots).

RemoveNaNInf.m
This function removes any row (or column) that contains an NaN or Inf value from the data set to avoid issues in subsequent steps of the analysis.

Input: **Data** – the data containing the descriptors (m x n; m: number of samples, n: number of descriptors).

Optional input: **Class** – the known class associations that are associated to each data point in the Data matrix (m x 1; m: number of samples).
Col – input that determines whether rows or columns should be removed from the data.

Output: **Data** – the data containing the descriptors (m x n; m: number of samples, n: number of descriptors) in which any row (column) that contained an NaN or Inf value is removed.
Class – the known class associations that are associated to each data point in the output Data matrix.
NaNIdx – the indices of the rows (columns) containing NaN values.
InfIdx – the indices of the rows (columns) containing NaN values.

VariableSelection.m
This function performs the variable selection, in which the important variables are selected for the desired analysis goal (i.e., best classification prediction).

Input: **Data** – the data containing the descriptors (m x n; m: number of samples, n: number of descriptors).
Class – the known class associations that are associated to each data point in the Data matrix (m x 1; m: number of samples).
Method – the variable selection method used.
Choices: 'biPLS' – 'rPLS' – 'biPLS_PLSToolbox' – 'rPLS_PLSToolbox' – 'iPLS_PLSToolbox' – 'GA_PLSToolbox' – 'ChiSquare' – 'MRMR' – 'ReliefF' – 'Boruta'.

Output: **VarSelModel** – a structure containing the model of the variable selection and/or the different variables that were selected by the method.

VariableSelectionPlots.m
This function plots a bar plot of the results of the automated variable selection.

Input: **Model** – a cell containing all variable selection models.

Optional input: **Threshold** – the minimum proportion of times a variable has to be selected to be considered important in the model.
VariableNames – a cell containing all the names of the descriptors.
SaveAs – the path to save the images in. If this is empty, the images will not be saved.

Output: **VariablesSelected** – the selected variables after combining all the iterations for the variable selection method.

References

1. Andronov, L., Orlov, I., Lutz, Y., Vonesch, J.-L. & Klaholz, B. P. ClusterViSu, a method for clustering of protein complexes by Voronoi tessellation in super-resolution microscopy. *Scientific Reports* **6**, 24084, doi:10.1038/srep24084 (2016).
2. Lagache, T. et al. Mapping molecular assemblies with fluorescence microscopy and object-based spatial statistics. *Nat Commun.* 9, 698, doi:10.1038/s41467-018-03053-x (2018).
3. Levet, F. et al. SR-Tesseler: a method to segment and quantify localization-based super-resolution microscopy data. *Nat. Methods* 12, 1065-1071, doi:10.1038/nmeth.3579 (2015).