User Manual

FuzzyMind

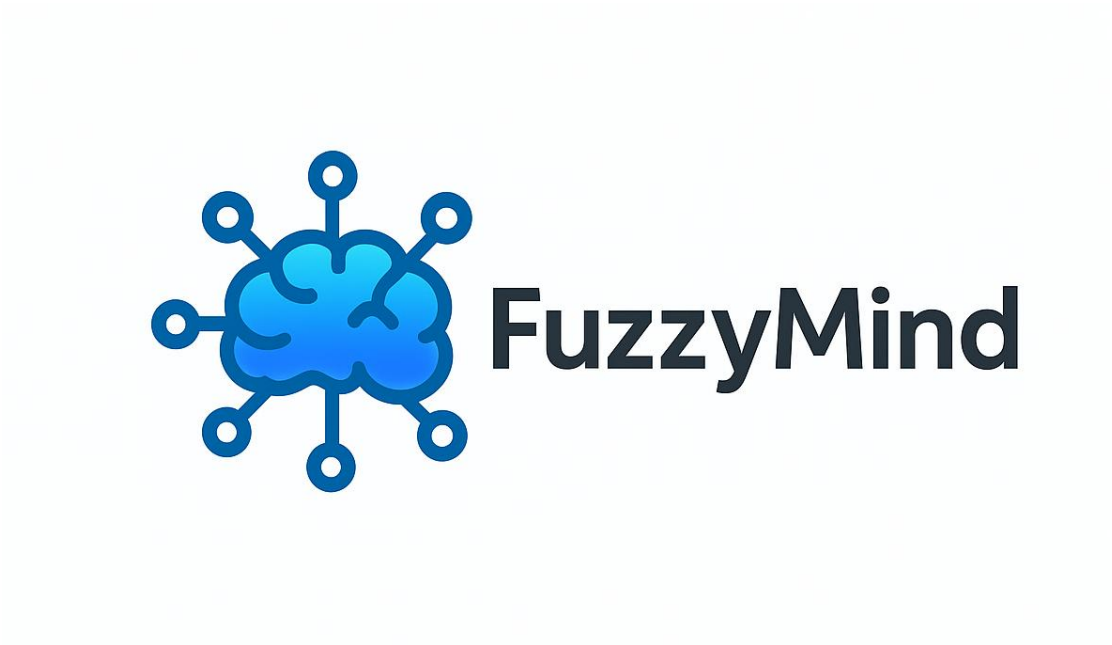


Table of Contents

Introduction

FuzzyMind offers a flexible and intuitive user interface for Fuzzy Cognitive Map (FCM) construction, analysis, visualization, and weight matrix optimization. The software is entirely written in Python, utilizing open-source packages such as Streamlit, TensorFlow, Pandas, Numpy, NetworkX, and Matplotlib.

In the current version, the software offers **the following features**:

1. Manual construction of both numeric and linguistic FCMs
2. Automatic construction of FCMs
3. Knowledge aggregation of linguistic FCMs
4. FCM graph visualization
5. Data preprocessing tools
6. FCM learning for classification tasks (Neural-FCM classifier)
7. FCM learning for regression tasks (Neural-FCM regressor)

Demos for these features are presented** in this manual. For further assistance you can find contact information, as well as FAQs.

The application is structured into four pages. These pages are: 1) **Home**, 2) **FCM Construction and Inference**, 3) **FCM Construction and Inference (Linguistic)** and 4) **FCM Learning**. Use the sidebar to navigate to each page as shown in Figure 1.

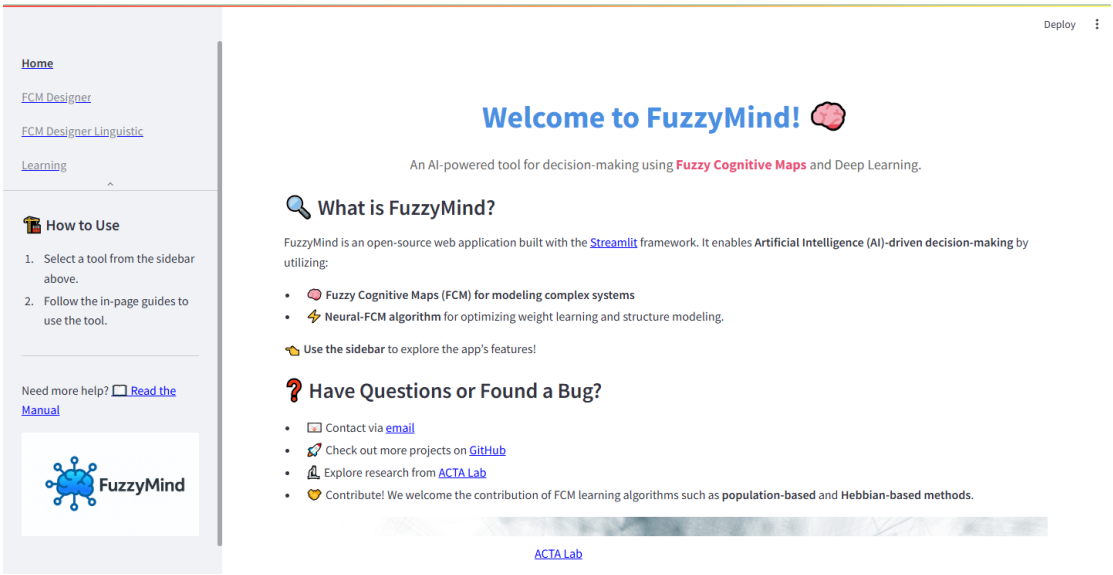


Figure 1 The Home page. All pages are accessible from the sidebar.

***Important note: This manual assumes that the user is familiar with FCM aspects and algorithms. If not, we recommend the following articles:*

1. *Kosko, B. (1986). Fuzzy cognitive maps. International journal of man-machine studies, 24(1), 65-75.*
2. *E. Papageorgiou, J. Salmeron: A review of Fuzzy Cognitive Maps research during the last decade. IEEE Transactions on Fuzzy Systems 21, 2012*

3. *G. Felix, G. Nápoles, R. Falcon, K. Vanhoof, W. Froelich, R. Bello: A review on methods and software for Fuzzy Cognitive Maps. Artificial Intelligence Review, 2017*
4. *E. I. Papageorgiou, "Learning Algorithms for Fuzzy Cognitive Maps—A Review Study," in IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), vol. 42, no. 2, pp. 150-163, March 2012, doi: 10.1109/TSMCC.2011.2138694.*

Installation and running (local version)

Currently, this version is running locally. To run this app locally on your PC you will need to install Python. Although later versions of Python 3.1x.x may run efficiently, it is recommended to use Python 3.10.4 (employed for development) for avoiding any running issues.

Install using virtual environment (Recommended)

Install python libraries in a virtual environment to avoid conflicts with other versions of already installed libraries.

Open a **command prompt window** and navigate to a desired location using the cd command. E.g.:

C:\Users\User>cd Desktop

Then use the following command to create a virtual environment named **App.**

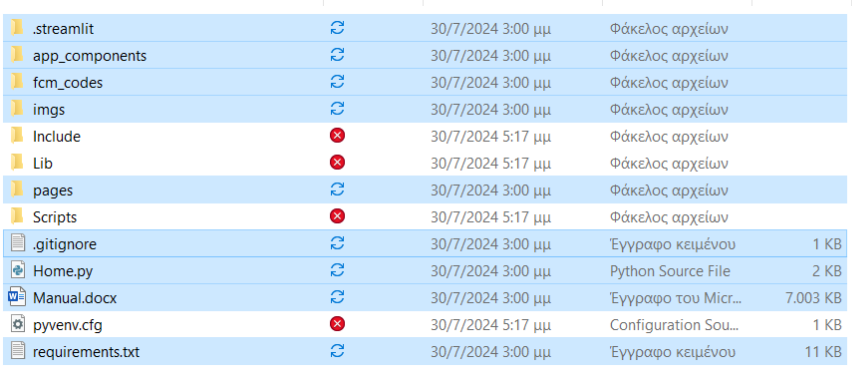
python -m venv App

Activate the virtual environment using the following command:

App\Scripts\activate

Ensure that the virtual environment is activated:

Unzip the fcm-app-master.zip in the App folder:



Subsequently, there is a "requirements.txt" file which points to the python packages that need to be installed locally.

Use the following command to install the python libraries from the requirements.txt file. This may take a while. *If you are facing any issues with package installation please refer to the official documentation of Python here.*

pip install --use-deprecated=legacy-resolver -r App\requirements-gpu.txt

To **run the app** change directory to the App folder using the following command:

cd App

Then use the following command:

streamlit run Home.py

A local Streamlit server will spin up and FuzzyMind will open in a new tab in your default web browser at the address http://localhost:8501/.

FCM Construction and Inference

This page offers widgets for constructing and simulating FCMs. You can construct an FCM **manually** by utilizing the app's widgets, or **automatically** by uploading a .csv file with the weight matrix. These widgets are available through the **FCM Construction and Inference page.**

This page has **two tabs:**

1) **FCM design**
2) **Inference**

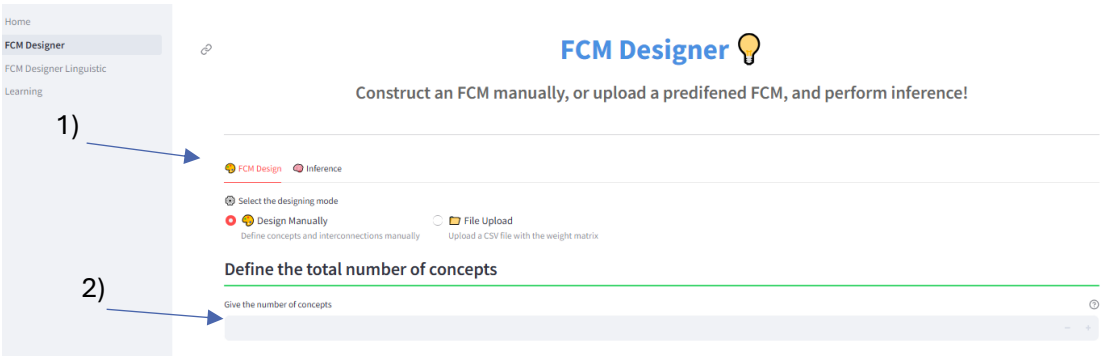
The **inference** tab becomes available after the definition of weighted interconnections in the **FCM design tab**.

The construction and inference process are explained as a stepped procedure with a demonstration of screenshots in the following subsections. The example of [*Montazemi, A. R., & Conrath, D. W. (1986). The use of cognitive mapping for information requirements analysis. MIS quarterly, 45-56.*] is utilized for demonstration purposes.

FCM Designer

a. Design Manually (Manual construction)

1) To construct an FCM manually select the "**Design Manually**" option in the radio widget.
2) Pass the number of concepts your FCM entails, and press enter. The widget accepts any number between 3 and 50.



Once the number of concepts has been defined, two table-widgets appear for naming concepts and defining weighted relations. In more detail:

3) A table widget appears for naming each concept. You can double click at each cell and change the default *"name of concept 1"* value. ***Name all concepts prior to the definition of weighted interconnections***, *as naming a concept triggers a re-run and <u>all weights are re-initialized with zero (0.0) values</u>*.

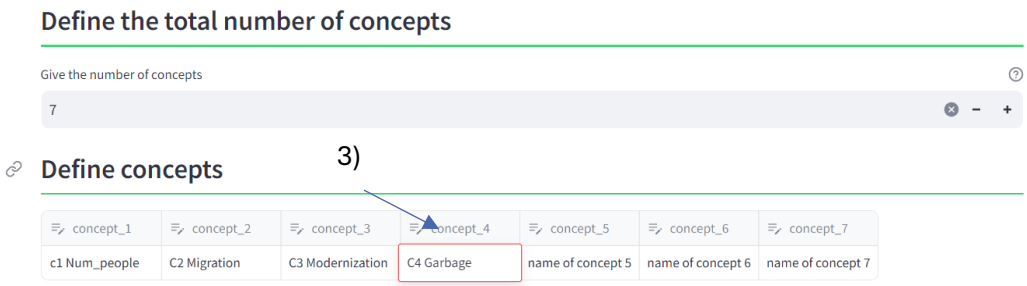


Figure 2 Widget for naming the concepts

4) The weight relation table-widget allows the user to **design the map architecture entirely from scratch**. Each cell accepts only values in [-1, 1] and represents the weighted relation among concepts. The comma "," symbol is used for decimals. In addition, the widget offers an option for downloading the FCM**.

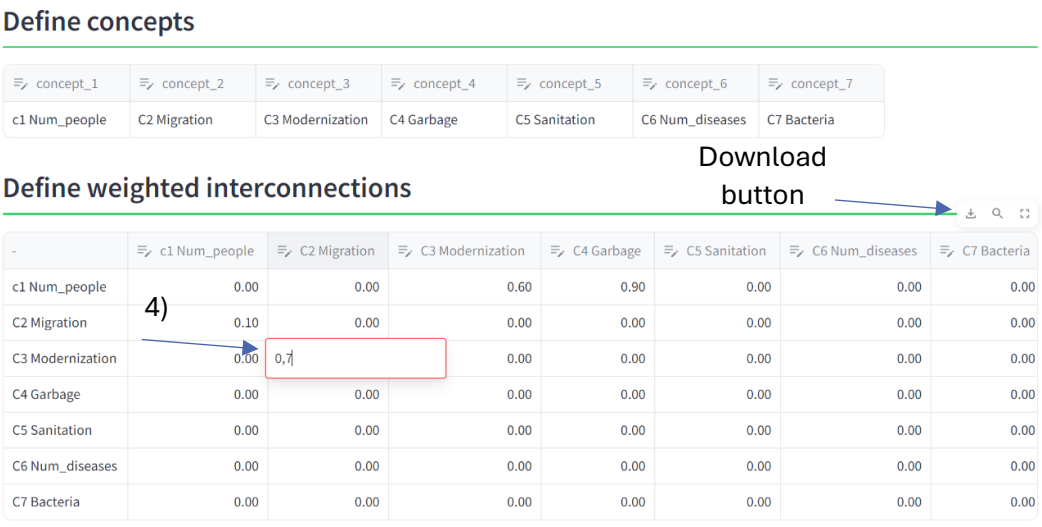


Figure 3 Widget for defining weighted relations and the download button.

Limitation notes: Each re-run (e.g. page reload, page changing)* ***discards all progress*. When working with large FCMs, it is advised to frequently download the weight matrix progress to avoid losing its already defined relations. The next section describes how to upload a .csv file containing an FCM weight matrix.*

b. File Upload (Automatic construction)

Automatic construction refers to using a ".csv" file, containing an FCM weight matrix, for defining an FCM in the **FCM design** tab.

The .csv file is expected to have the **concept names** in the **first row** and the **concept names as index** (first column). The dot "." symbol is used for decimals and comma "," for separating columns (delimiter). The weights values are expected to be in the [-1, 1]. **FCM weight matrices constructed and downloaded within the app (section 1a), already meet these requirements**. An example of such file, opened with a simple text viewer (*Windows 10* notepad), is presented in Figure 4. The user can choose custom .csv parameters with the reading widgets as it is described later in this section.

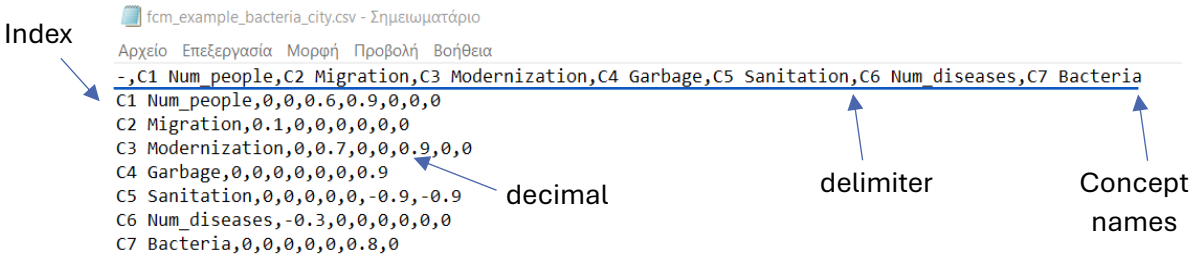
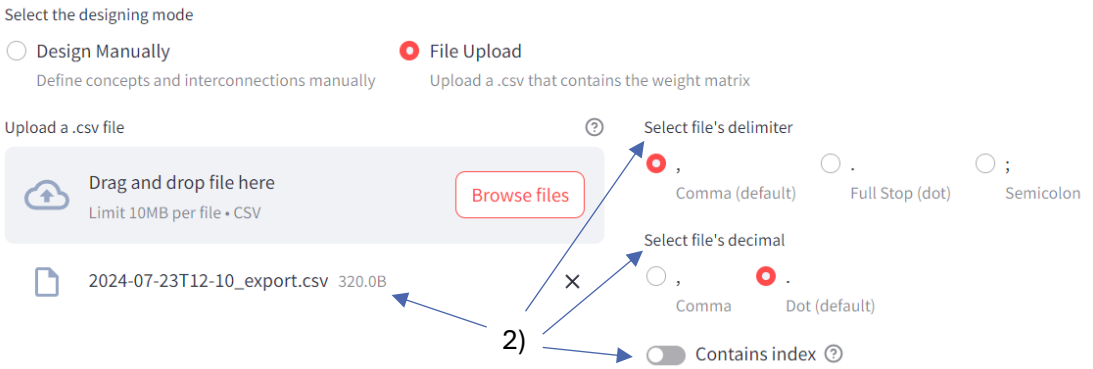


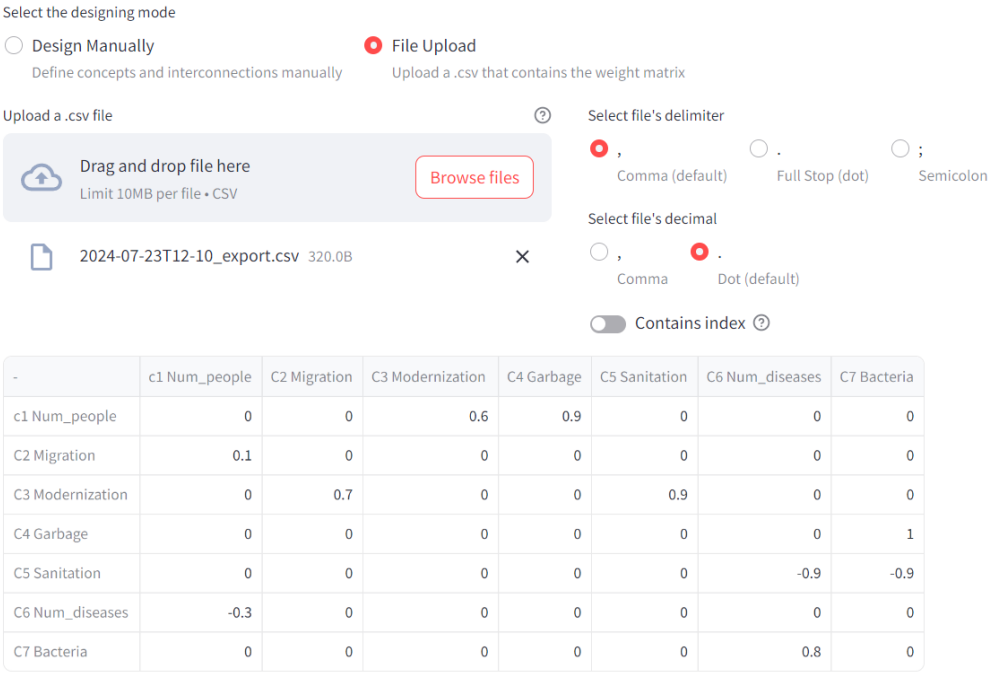
Figure 4 An example of .csv file that is expected for automatic FCM construction.

1) Select the **File Upload** option in the radio widget inside the FCM design tab. Use the **upload button widget** to **browse** local folders and to upload a weight matrix in .csv format. Currently, you can upload a **.csv file < 10 MB**.

2) Once a .csv file has been selected, csv **parameter widgets** become available for reading any csv file. In detail these are:
   a. The **delimiter** symbol
   b. The **decimal** symbol
   c. Whether the first column contains **numeric index instead of the concept names** (usually matrices initialized with other apps such as Microsoft Excel contain a numeric index column).



3) If the proper parameters have been selected the weight matrix appears below the widgets (Figure 5). Otherwise, an **error** is displayed (Figure 6).



-	c1 Num_people	C2 Migration	C3 Modernization	C4 Garbage	C5 Sanitation	C6 Num_diseases	C7 Bacteria
c1 Num_people	0	0	0.6	0.9	0	0	0
C2 Migration	0.1	0	0	0	0	0	0
C3 Modernization	0	0.7	0	0	0.9	0	0
C4 Garbage	0	0	0	0	0	0	1
C5 Sanitation	0	0	0	0	0	-0.9	-0.9
C6 Num_diseases	-0.3	0	0	0	0	0	0
C7 Bacteria	0	0	0	0	0	0.8	0

Figure 5 Properly reading a .csv file containing the weight matrix

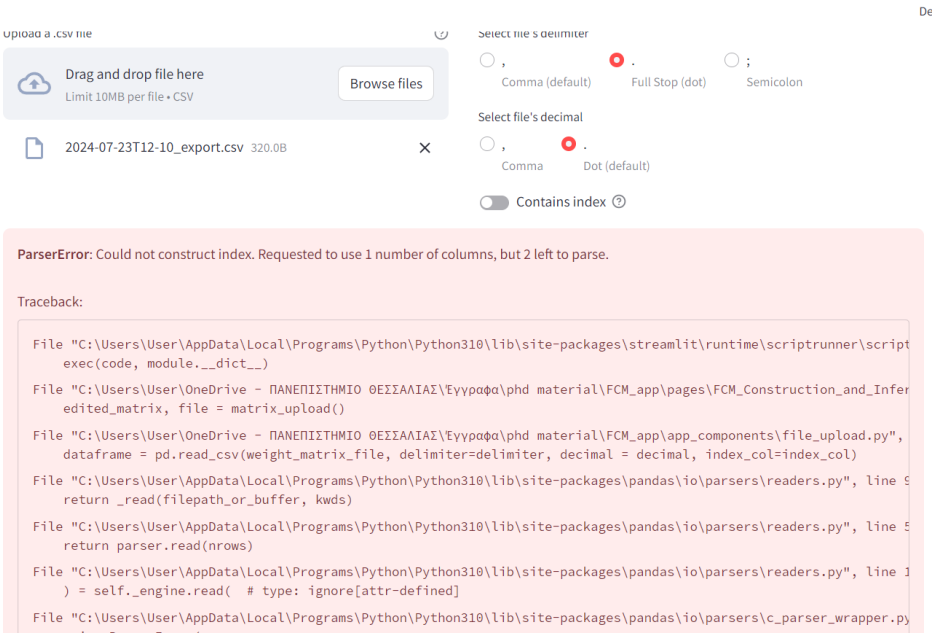


Figure 6 An example of an error in reading the .csv file containing the weight matrix.

4) Activate the toggle widget if you want to modify the weights in the uploaded weight matrix. This will provide an additional table-widget where the cells are editable** and accept values in [-1,1].

-	c1 Num_people	C2 Migration	C3 Modernization	C4 Garbage	C5 Sanitation	C6 Num_diseases	C7 Bacteria
c1 Num_people	0.00	0.00	0.60	0.90	0.00	0.00	0.00
C2 Migration	0,2	0.00	0.00	0.00	0.00	0.00	0.00
C3 Modernization	0.00	0.70	0.00	0.00	0.90	0.00	0.00
C4 Garbage	0.00	0.00	0.00	0.00	0.00	0.00	1.00
C5 Sanitation	0.00	0.00	0.00	0.00	0.00	-0.90	-0.90
C6 Num_diseases	0.50	0.00	0.00	0.00	0.00	0.00	0.00
C7 Bacteria	0.00	0.00	0.00	0.00	0.00	0.80	0.00

Figure 7 The widget for modifying weight values (cells).

***Limitation notes: Concept names and index cannot be modified. Please ensure that your csv file contains the right concept names in the first row and the index column.*

FCM visualization

Either in manually or in automatic mode, you can use the app visualization tools to render a plot of the defined FCM **by activating the toggle widget** "Generate FCM graph". Once it is activated, a map is generated that uses a **diverging blue/red colormap** to **represent the negative and positive relations**. The user can further customize the map by selecting:

1) Different **position layouts** in the concepts:
    a. **Circular** (Position nodes on a circle),
    b. **Random** (Random position of nodes),

c. **Shell** (Position nodes in concentric circles).

2) **Figure size**. This affects the actual size of the figure and correspondingly the node and text size.
3) **Node size**. It affects the size of the nodes, as well as the size of the text that is proportional to the node size.

The map can be directly downloaded (predefined DPI = 500) with the "Download figure" button.

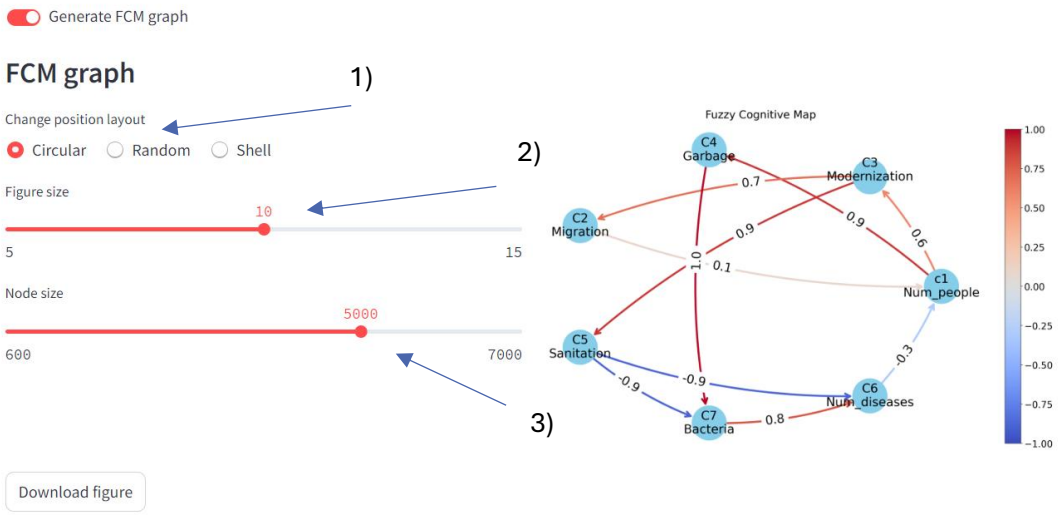


Figure 8 The widgets for custom visualization of the FCM

Inference Tab

Once an FCM has been defined (sections 1a, 1b), the inference widgets become available. To access the **inference widgets**, navigate to the **Inference tab** inside the FCM Construction and Inference page.

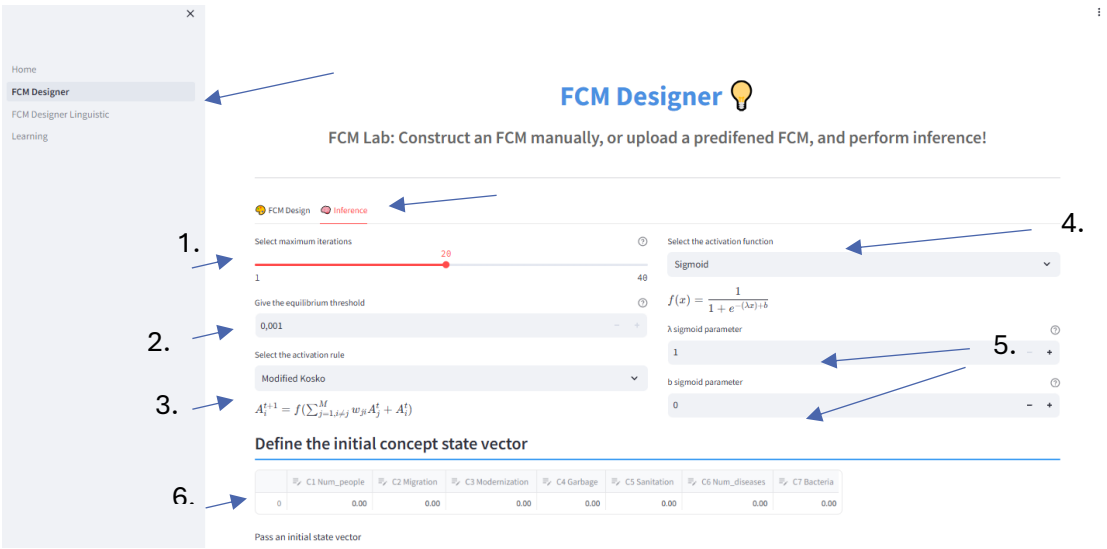


*Figure 9 The FCM inference widgets inside the **Inference tab**.*

These widgets allow to run custom experiments/simulations with different parameters and formulas derived from the FCM theory. More explicitly, the app offers experimentation with the following (Figure 9):

1. **Maximum number of iterations**: Slider widget that defines the upper limit of iterations to be executed before terminating the process, **even if equilibrium has not been reached**.
2. **Equilibrium threshold**: A predefined float (*h*) that is used as an inference stopping criterion. After each iteration the algorithm compares the concepts values (concepts state) $\boldsymbol{a}^{t}$ at the current iteration *t* with the values $\boldsymbol{a}^{t-1}$ of the previous iteration. This termination criterion can be formally defined as: $a_i^t - a_i^{t-1} \leq h, \forall\, concept\ i$.
3. **Activation (inference) rule**: The formula for updating the concepts' values. The available rules are the following**:
    - **Kosko**
    - **Modified Kosko**
    - **Rescaled**
4. **Activation function**: A non-linear function that is applied to the updated concepts' state and affect the convergence of the FCM. The user can select one of**:
    - **Sigmoid**
    - **Bivalent**
    - **Trivalent**
    - **Tanh (Hyperbolic)**
5. **λ, b sigmoid parameters**: These are parameters that are only visible when the Sigmoid activation function is selected. They affect the steepness and the shifting of the sigmoid curve respectively. λ widget accepts integer values in [1, 10], whereas b widget accepts integer values in [-10, 10].
6. **Initial concept state vector**: This table-widget allows the user to provide the initial state $\boldsymbol{a}^{t=0}$ by double clicking at each concept and typing a value. For consistency, the accepted value range are based on the activation functions, meaning that:
    - Values in [-1,1] are accepted for **Tanh,** with accuracy of two decimals (0.01 step).
    - Values in [0, 1] are accepted for **Sigmoid,** with accuracy of two decimals (0.01 step).
    - {0, 1} values can be passed with **Bivalent.**
    - {-1, 0, 1} values can be passed with **Trivalent**,

***Note: For practicality, these formulas are exhibited below the widgets as a text that is dynamically updated based on user's selection*.

Once the **initial concept state $\boldsymbol{a}^{t=0}$ has been defined** (at least one concept has value ≠ 0)**,** a **button** "Inference" becomes visible that initializes the inference process when pressed. In Figure 10, an inference scenario (simulation) is considered where C1, C2, C3 have initial values of 1. We employ Kosko rule and Bivalent activation function, and we press the "Inference" button to initialize the simulation.

When inference process finishes, a table and a graph appear that entail the inference progress as it is presented in Figure 11. Both the table and the graph can be downloaded locally. It is observed that in this scenario, the FCM converged after 3 iterations. Press the "Clear" button or change the values in the inference widgets, to discard the current results and perform new simulations.

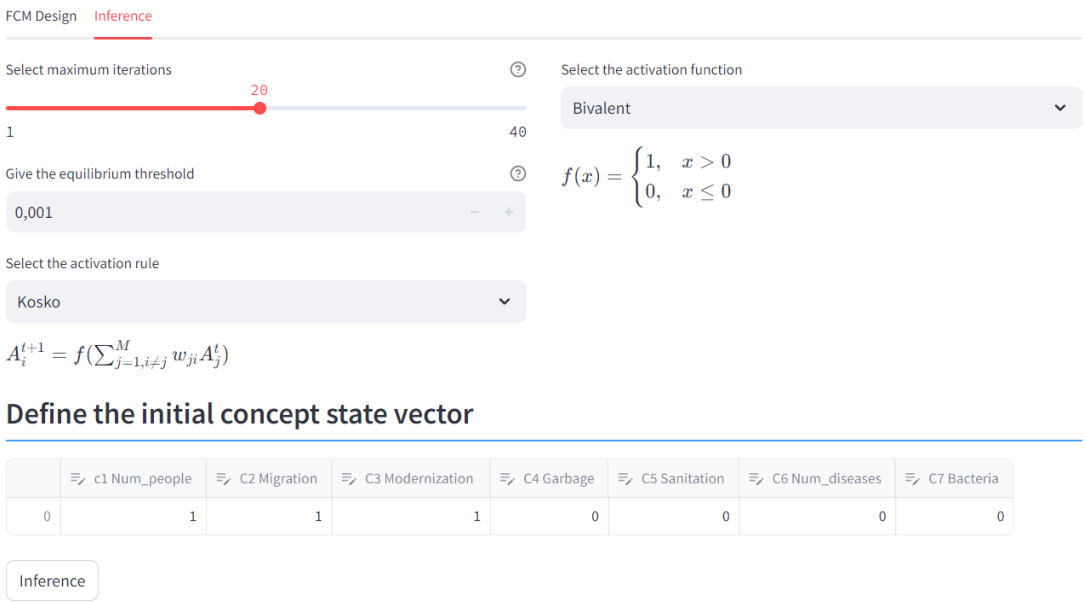


Figure 10 The parameters of an FCM simulation

Figure 11 The inference results of the FCM simulation

FCM Designer Linguistic

Another key feature of FuzzyMind is the ability **to construct FCMs with linguistic relations and fuzziness**. This key feature leverages expertise for designing powerful FCMs for decision-making. Regarding the **terminology** that Is used in this section, we refer to **linguistic variables,** as the names of the membership functions given to concepts interconnections, that define the degree of causality (fuzzy variable), in the FCM.

This page has two main tabs: 1) FCM Design and 2) Inference. The FCM design offers **three methods for constructing an FCM**:

a) **constructing manually** by defining the fuzzy interconnections from scratch,
b) **constructing automatically** by uploading a **pair of csv and json files** that contain the weight matrix and membership function parameters respectively,
c) **knowledge aggregation**, where multiple pairs of csv and json files are uploaded to construct an FCM based on multiple experts/stakeholders.

These are further explained in the following subsections.

FCM Design Tab

a. Design Manually (Manual construction)

1) To construct an FCM with linguistic variables manually select the **"Design Manually"** option in "FCM Design" tab as it presented in Figure 12.

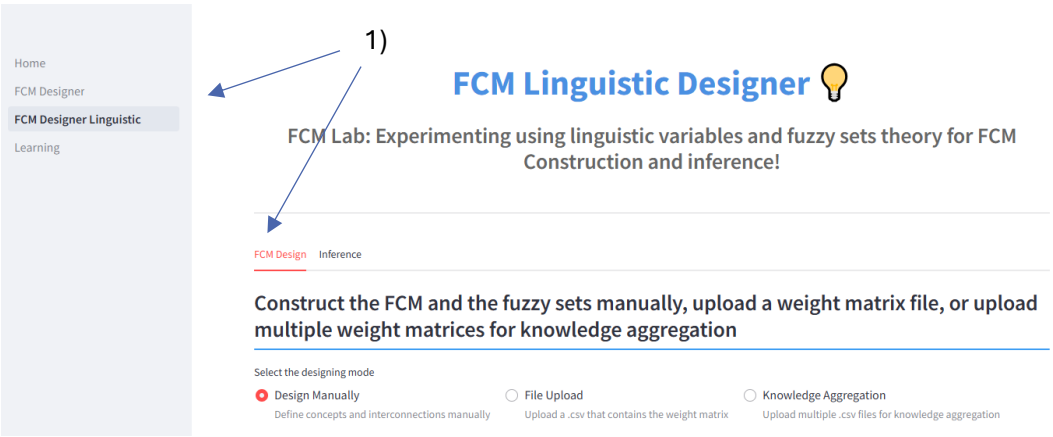


Figure 12 The "Design Manually" option

2) Once "Design Manually" has been selected, a widget for selecting **the number of linguistic variables,** as well as an **expander for modifying them** becomes available. You can currently use **5, 7,** or **11 linguistic variables** with **predefined names** as it is presented in Figure 13.

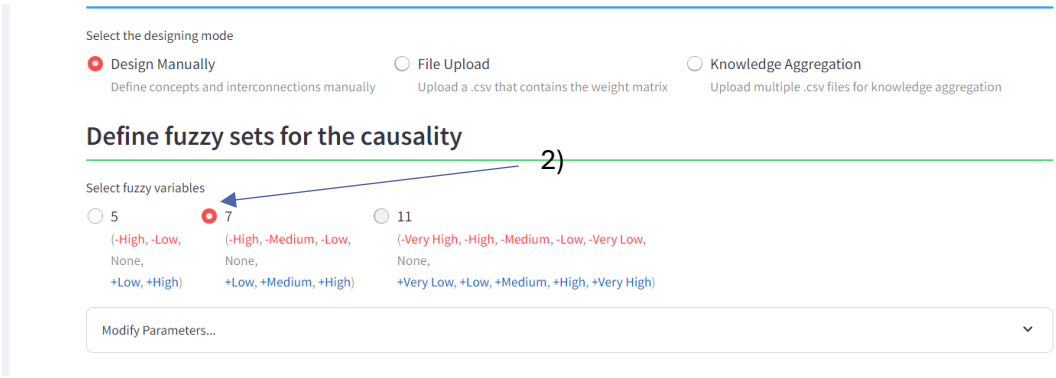


Figure 13 UI widgets for fuzzy variables.

Using the expander, you can **modify the type of MF and choose one of the following: Triangular, Trapezoidal or Gaussian**. Note that the Universe of Discourse [-1, 1] cannot be modified. However, you can modify the parameters of each MF. In Figure 14 we present the widgets for modifying Triangular MFs. There are **two columns of slider widgets** and a plot that is dynamically updated and depicts the **MFs parameters.** Each row of the slider widgets corresponds to one MF. Considering the sliders on the first column, they have **two points** that can be adjusted and correspond to **the start** and **the end values of the triangles**. The sliders on the second column are dynamically updated based on the range provided by the sliders in the first column and correspond to the value of the **top corner of the triangle**. Finally, **a "Download JSON" button** is located at the bottom right of the menu with which the user can download the defined fuzzy parameters. This JSON can be used within the app for automatic construction and is further explained in the next section.

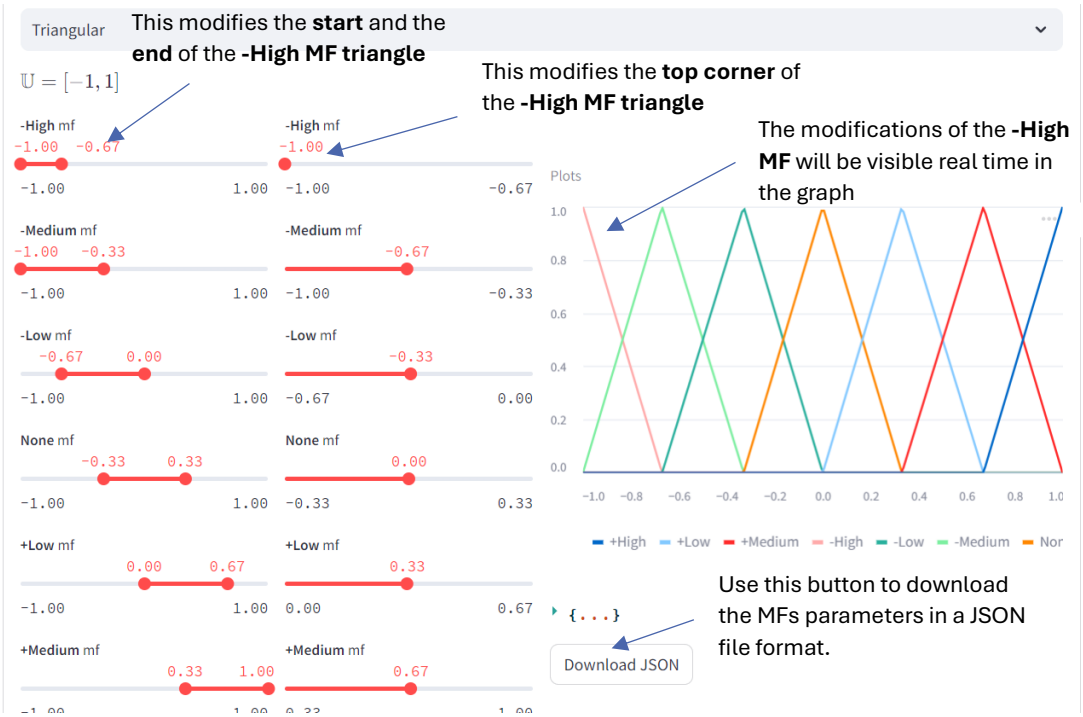


Figure 14 The UI widgets for modifying the MFs parameters.

3) Pass the number of concepts your FCM entails, and press enter. The widget accepts any number between 3 and 50.

Once the number of concepts has been defined, two table-widgets appear for naming concepts and defining the linguistic relations (Figure 15). In more detail:

4) A table widget appears for naming each concept. You can double click at each cell and change the default *"name of concept 1"* value. ***Name all concepts prior to the definition of linguistic relations,*** *as naming a concept triggers a re-run and all weights are re-initialized with "None" values*

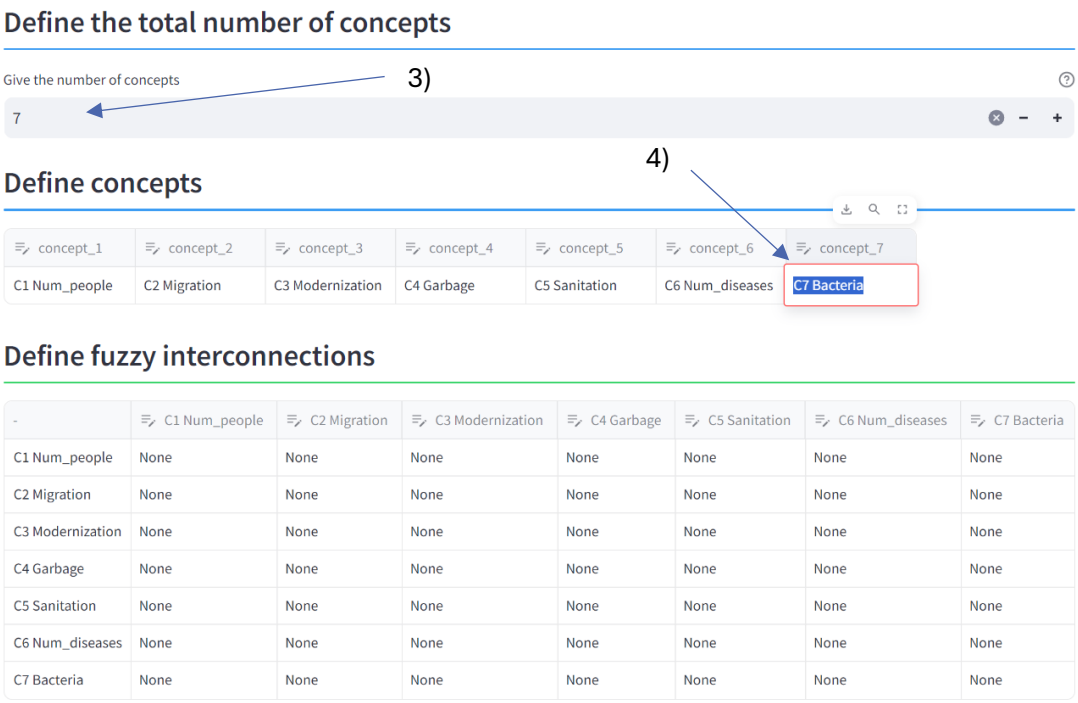


Figure 15 The UI widgets for defining the number of concepts, naming them and the initialization of a matrix with fuzzy interconnections

5) The fuzzy relation table-widget allows the user to **design the map architecture entirely from scratch**. Each cell accepts only the defined linguistic variables providing a dropdown selection menu for selection.

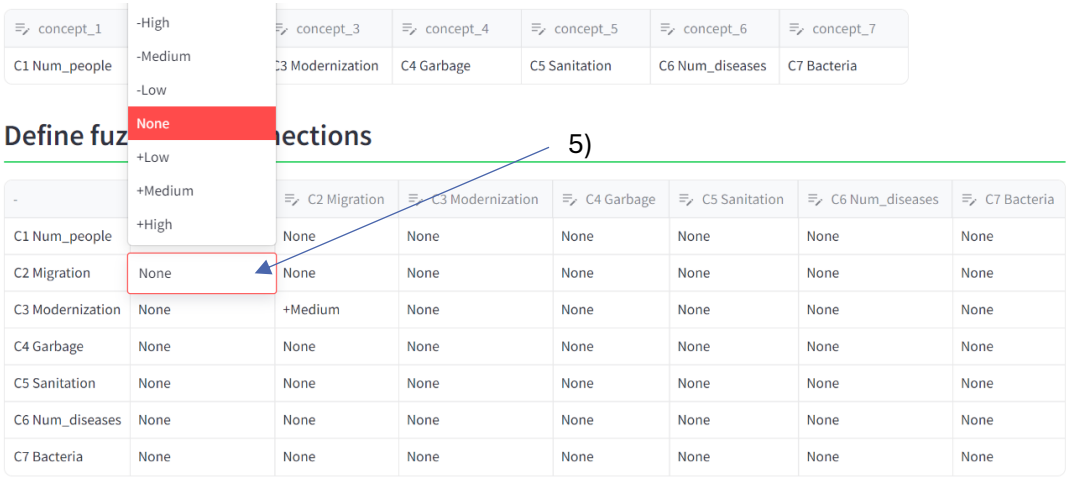


Figure 16 The widget for defining fuzzy relations

6) After the provision of **fuzzy relations**, a selection widget for **defining the defuzzification method** becomes available and the **weighted interconnections are automatically calculated** and presented (Figure 17). The user can select one of the **Centroid**, **Bisector** and **Mean of Maximum (MoM) defuzzification methods**.

	C1 Num_people	C2 Migration	C3 Modernization	C4 Garbage	C5 Sanitation	C6 Num_diseases	C7 Bacteria
C1 Num_people	0	0	0.67	0.89	0	0	0
C2 Migration	0.33	0	0	0	0	0	0
C3 Modernization	0	0.67	0	0.89	0	0	0
C4 Garbage	0	0	0	0	0	0	0.89
C5 Sanitation	0	0	0	0	0	-0.89	-0.89
C6 Num_diseases	-0.67	0	0	0	0	0	0
C7 Bacteria	0	0	0	0	0	0.67	0

Figure 17 The defuzzification step and the result using the centroid method

b. File Upload (Automatic construction)

Use this option to upload files for automatic construction of FCMs. For maximum compatibility, we recommend users to **use this app to construct FCMs with linguistic interconnections**, with the UI widgets that described in the previous section.

1. To construct an FCM directly from files you need to provide a pair of csv/json files. The .csv file is expected to have the **concept names** in the **first row** and the **concept names as index** (first column). The values are expected to be in linguistic form (text)**. The .json file provides the information to define the membership functions attributes of the fuzzy relation. Examples of proper .csv and .json files, opened with Windows 10 notepad, are presented in the following figures.

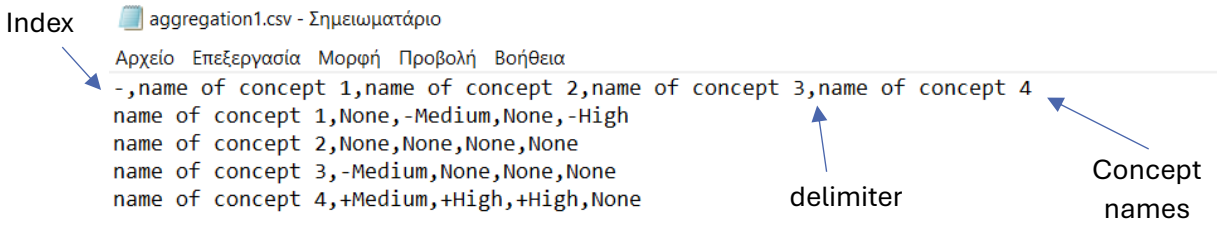


Figure 18 An example of a .csv file containing the relations matrix

fuzzy_info (2).json - Σημειωματάριο

Αρχείο Επεξεργασία Μορφή Προβολή Βοήθεια


```
{
"method": "Triangular",
"range": [-1.0, 1.0],
"step": 0.01,
"memberships": {
        "-High": [-1.0, -1.0, -0.67],
        "-Medium": [-1.0, -0.67, -0.33],
        "-Low": [-0.67, -0.33, 0.0],
        "None": [-0.33, 0.0, 0.33],
        "+Low": [0.0, 0.33, 0.67],
        "+Medium": [0.33, 0.67, 1.0],
        "+High": [0.67, 1.0, 1.0]
        }
}
```


Figure 19 An example of a .json file containing information regarding MFs

2. The UI provides two different upload buttons inside an expander container for uploading csv/json files separately as it is presented in Figure 20. When a .csv is selected, widgets appear for selecting the delimiter symbol and the index parameters.

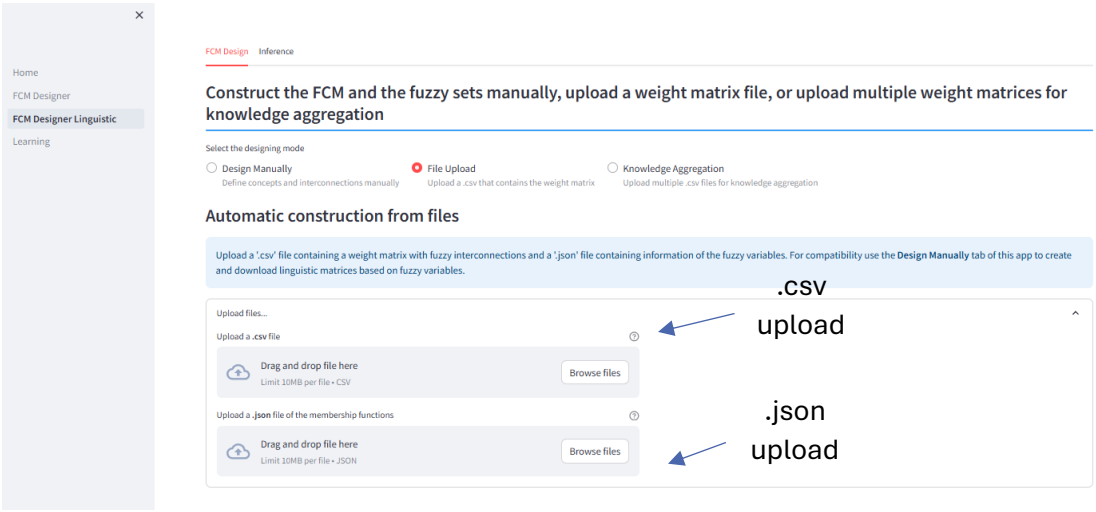
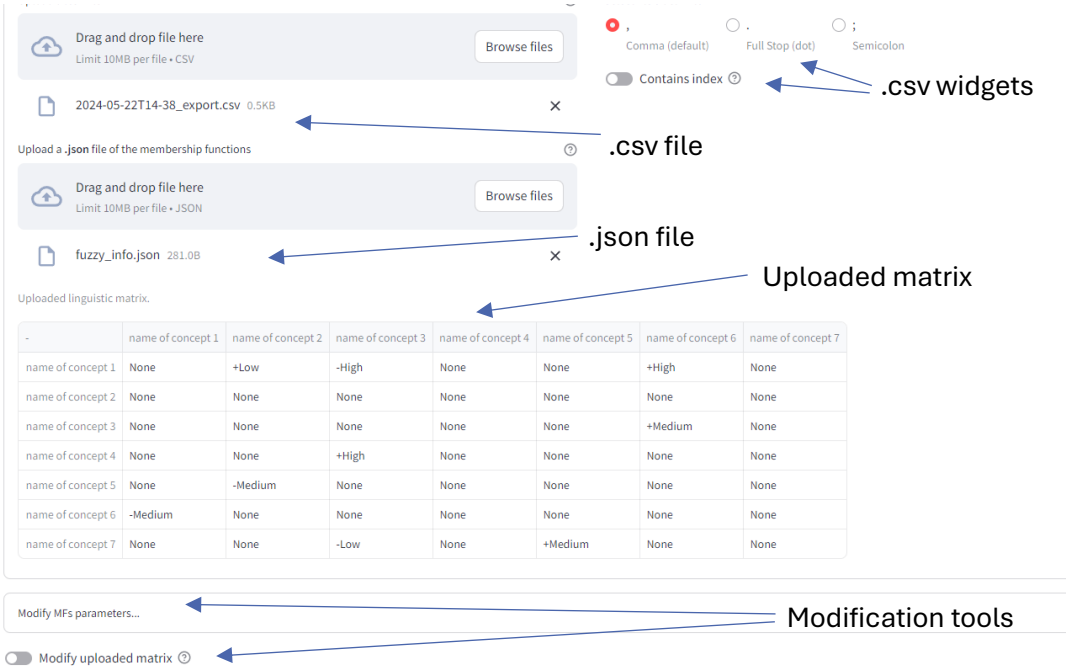


Figure 20 The UI widgets for uploading the necessary files

3. If both the csv and the json have been uploaded correctly, **the relation matrix**, a **"Modify uploaded matrix" option** and an **expander** for modifying MFs will appear in the UI (Figure 21). An error is displayed otherwise. The **"Modify uploaded matrix" option** allows the user to modify the defined relations among concepts. Both the matrix and the MFs modification, as well as the next step of **defuzzification** have been described in the previous section.



-	name of concept 1	name of concept 2	name of concept 3	name of concept 4	name of concept 5	name of concept 6	name of concept 7
name of concept 1	None	+Low	-High	None	None	+High	None
name of concept 2	None	None	None	None	None	None	None
name of concept 3	None	None	None	None	None	+Medium	None
name of concept 4	None	None	+High	None	None	None	None
name of concept 5	None	-Medium	None	None	None	None	None
name of concept 6	-Medium	None	None	None	None	None	None
name of concept 7	None	None	-Low	None	+Medium	None	None

Figure 21 An example of a correct upload procedure and the UI

**Limitation notes: Currently, compatibility is asserted with predefined linguistic variables (names) that have been presented within the previous section.

c. Knowledge Aggregation

The "knowledge Aggregation" option allows **multiple pairs of .csv/.json to be uploaded**, thus, **combining knowledge from multiple experts**. For maximum compatibility, It is recommended to use the "Design Manually" option to obtain such pairs.

1. The process, the file characteristics and the UI widgets (Figure 22) for uploading files is similar to what has been already described in the previous section. However, it is further required that pairs of .csv/.json files **have the same name** e.g. file1.csv/file1.json. This allows the backend to understand and combine the information of both files.
2. In the following figures we present an example in which three pairs of csv/json files have been utilized. Moreover, when pairs are uploaded, the UI renders new widgets, inside an "Aggregation info" expander container, such as:
    a. A table that presents the **total concepts and their names** (Figure 24).
    b. The **aggregation matrix** that exhibits what linguistic variables are passed at each interconnection (cell) (Figure 24).
    c. The defined MFs that were found in jsons (Figure 25).
    d. And two dropdown widgets to select a **specific interconnection for visualizing the MFs aggregation**. For aggregation we employ the max method.

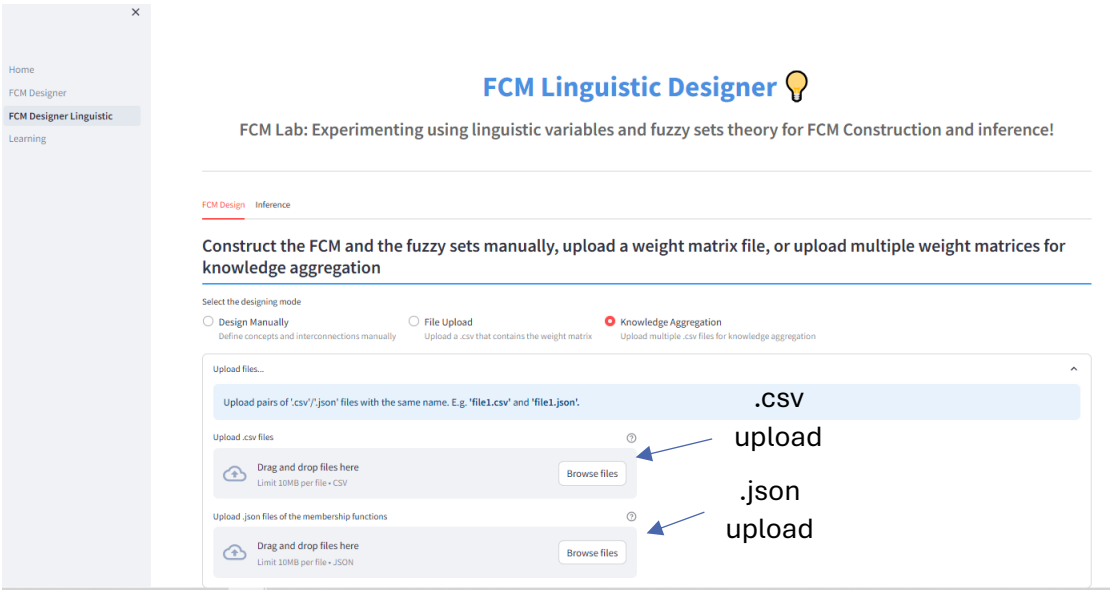


Figure 22 The UI of Knowledge Aggregation option and the upload widgets

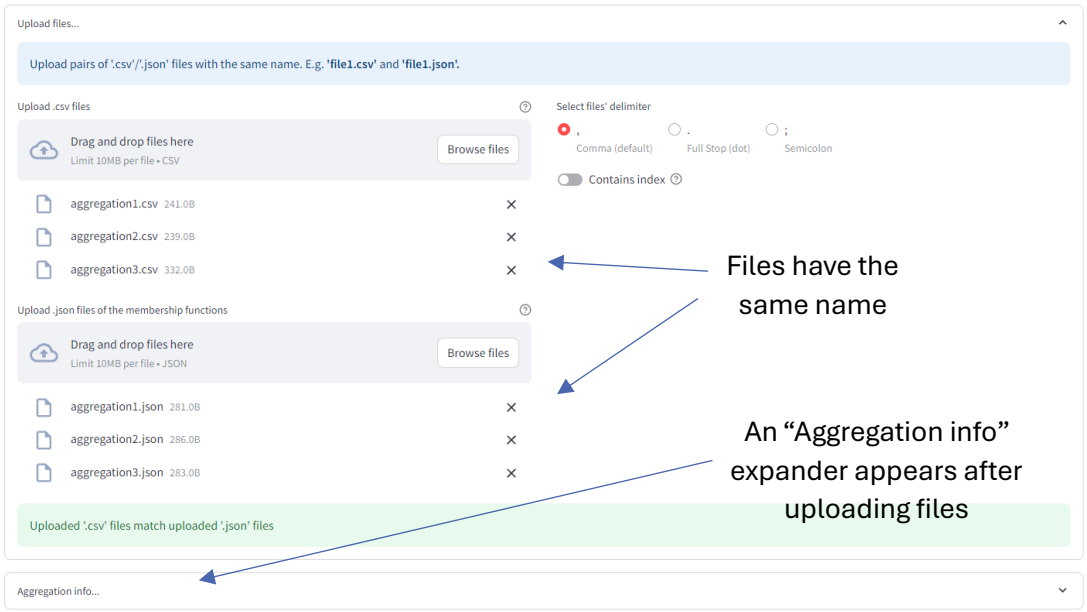


Figure 23 An example of uploading three pairs of csv/json files

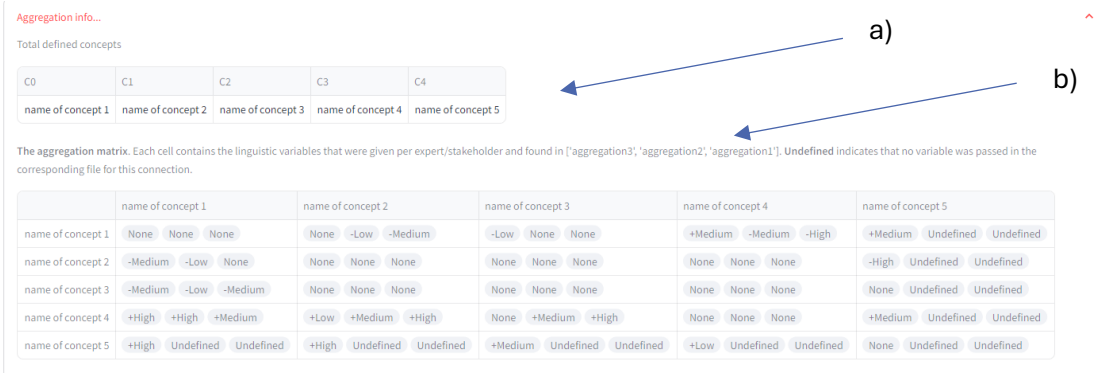


Figure 24 The widgets first widgets a,b, inside the "Aggregation info" expander

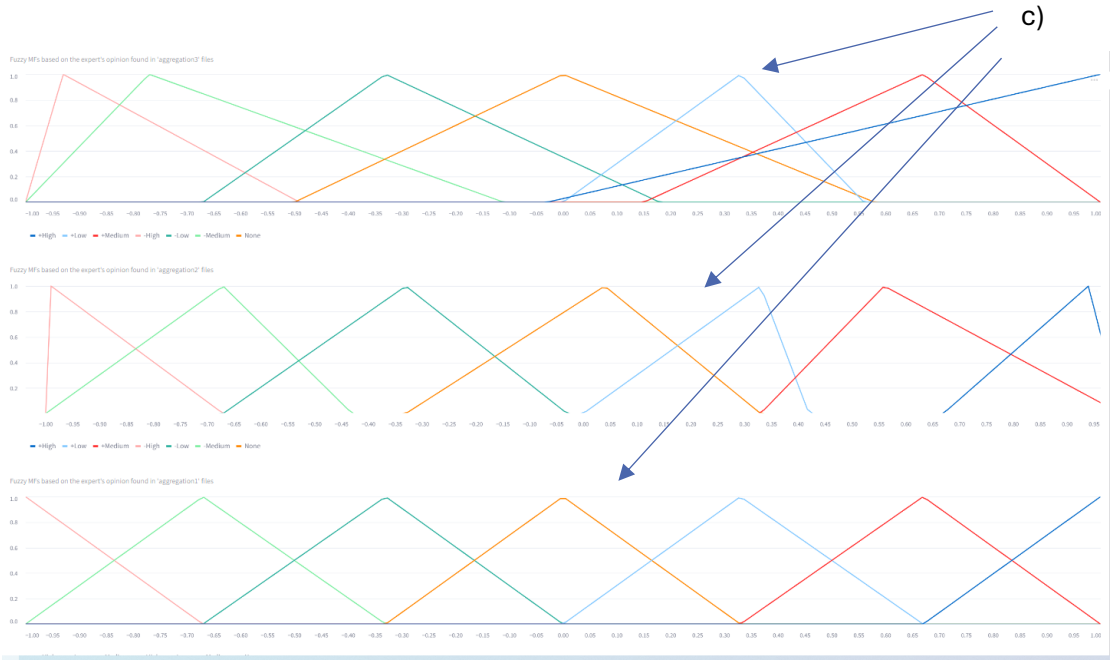


Figure 25 The plots of the MFs (c) that have been found in jsons

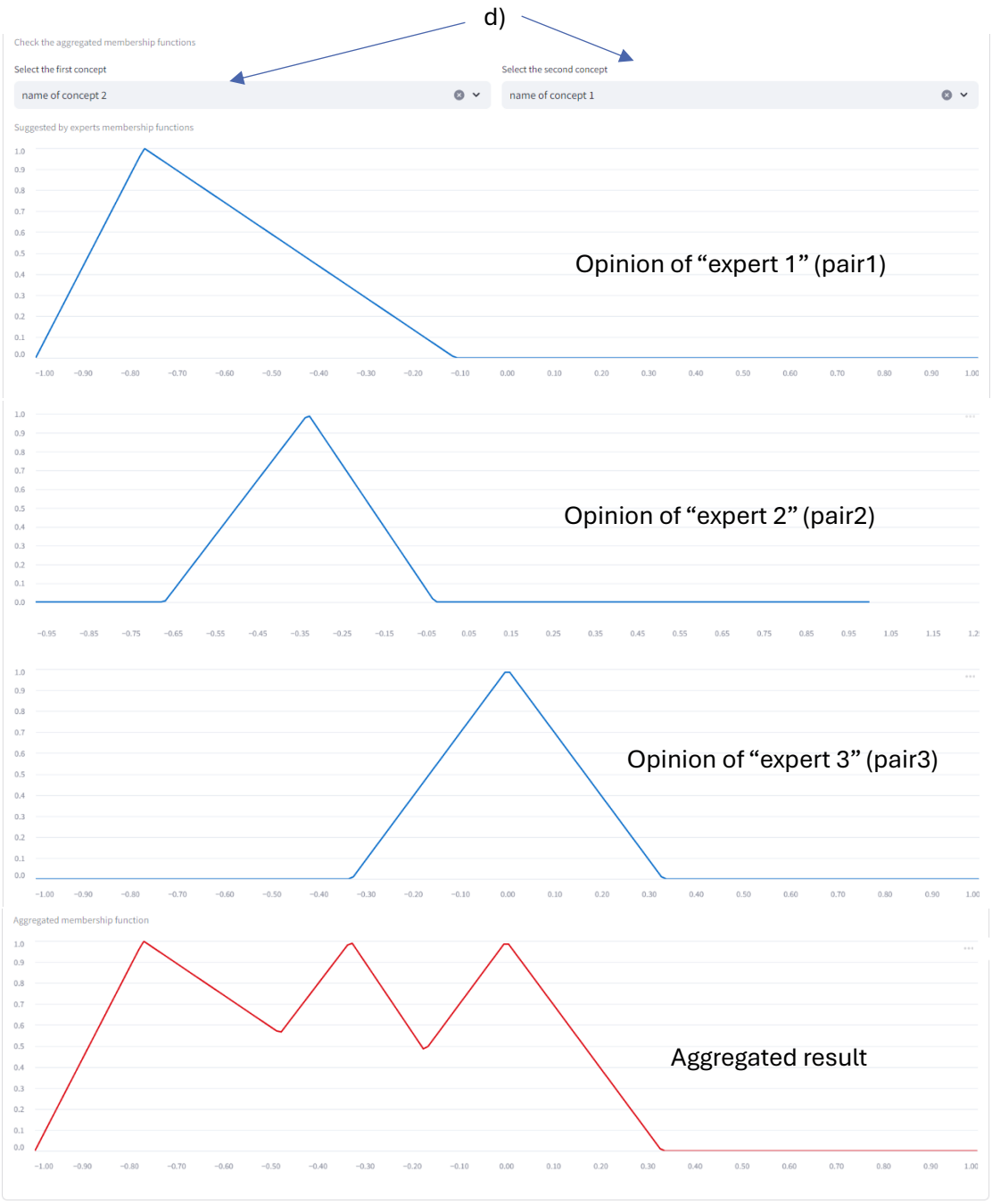
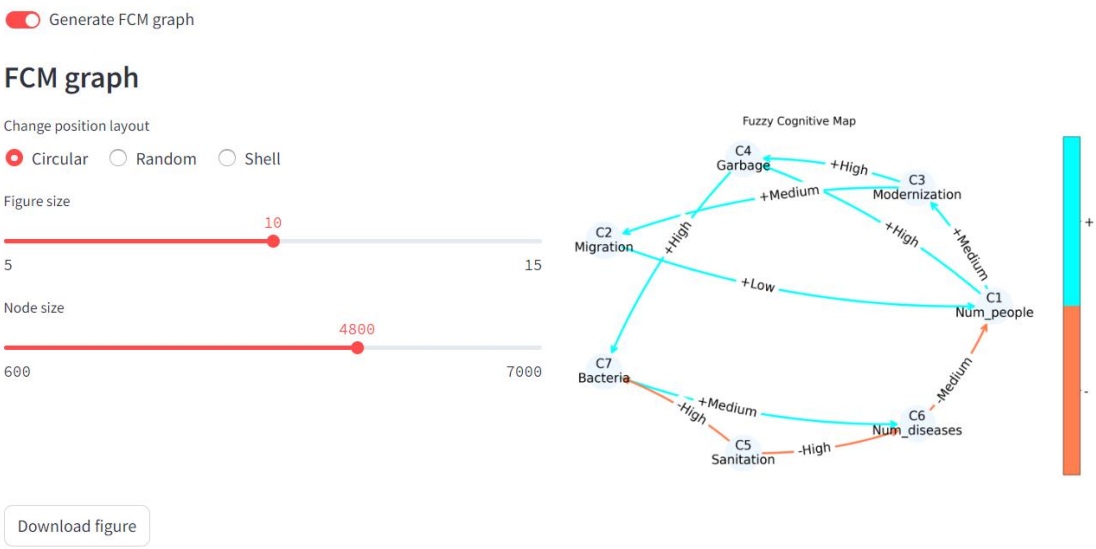


Figure 26 The aggregated MFs regarding the degree of causality of concept 2 towards concept 1.

Finally, the user can select defuzzification methods to acquire a numeric matrix and to proceed to FCM simulations. The defuzzification procedure and UI have been described in a previous section.

d. FCM Visualization

This page provides visualization tools for both the **fuzzy interconnections** and the **weighted interconnections** (after the defuzzification**)**. The latter tool has been already described in a previous section and we prompt the reader to visit this section to understand visualization widgets' behavior. The former tool utilizes the colors of *cyan* and *purple* to depict the **positive** and **negative** relations respectively.



Inference Tab

After the initialization and the defuzzification of the weight matrix the inference process becomes accessible. The reader is prompt to use the Section 3 of the arithmetic FCM Construction, as in the current app version simulation accepts only numerical initial state values and not linguistic. This may change in future versions.

FCM Learning

Another key feature of the FCM-APP is the provision of learning algorithms for extracting knowledge and constructing FCM from data. Currently, this version supports the novel Neural-FCM algorithm (To be published) for supervised learning. Future versions will include population-based algorithms such as the Particle Swarm Optimization (PSO), the Real Coded Genetic Algorithm (RCGA) etc.

We begin by providing general information regarding the learning UI and its abilities. Finally, this section includes a demonstration regarding the learning application in a classification task.

User Interface

The FCM learning page utilizes **four tabs** and the **sidebar**, as it is presented in Figure 27. These tabs are:

1. Data upload
2. Data visualization
3. Data preprocessing
4. Learning

This structure aims to assist non-experienced users as it mimics the data-driven methodology that in general is composed of the following four steps 1) **Data gathering**, 2) **Data understanding**, 3) **Data preparation** and 4) **model selection and learning**. More information is provided in the following sections.

The sidebar is used for information purposes regarding the uploaded dataset and for selecting **the learning task** option, **classification** or **regression**, as it is observed with Figure 27 and Figure 29.

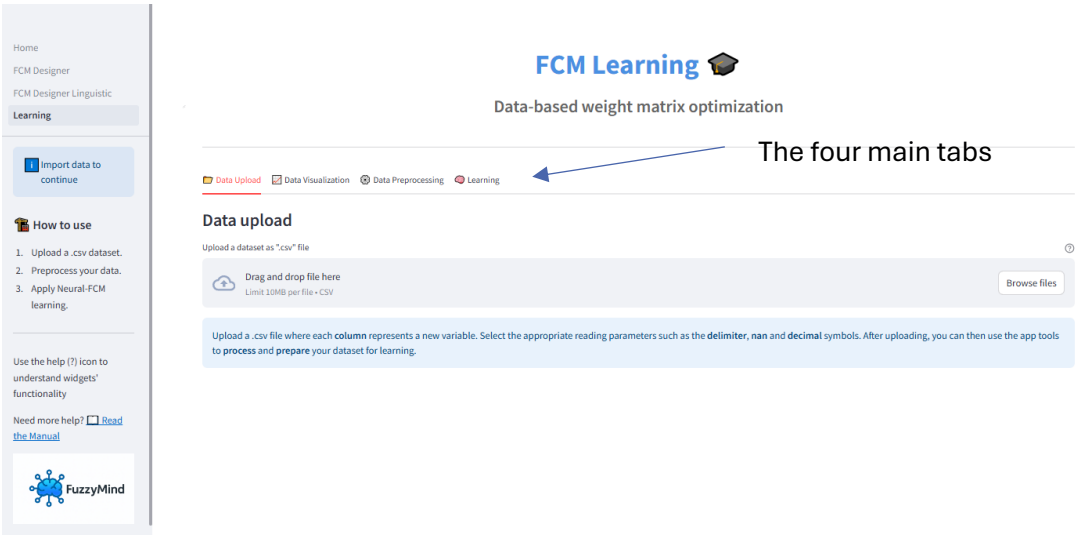


Figure 27 The FCM learning page

Data upload

This tab provides the tools for uploading and reading custom datasets. The UI widgets provide a variety of options for reading .csv files. However, the following is expected:

1. The dataset is a **two-dimensional table**.
2. Variables in the dataset **must be column-wise** and not row-wise, meaning that the horizontal axis of the table contains the total variables and the vertical axis the total observations.
3. The first row in .csv files is expected to have the **names of the variables**. The following rows must contain the data observations.
4. Both **input** and **output** variables (columns) must **be provided within the same file**.
5. A **single continuous dataset** is expected which can be later divided within the app into **train/validation/test sub datasets**.
6. The maximum size of the file must not exceed the size of 10 MB.

Upload widgets

As it is presented in Figure 27, initially, only an **upload button "Browse Files" is available** for selecting and uploading .csv files to the app. When a **file is selected a variety of other widgets are dynamically become available,** as it is presented in Figure 28. These are:

- The **delimiter symbol**: The symbol used to separate the columns (variables) in the csv. The predefined value is the *comma* ",". Other options are available as well.
- The **decimal symbol**: The symbol used to indicate float (decimal) numbers. The predefined value is the *dot* ".". Comma "," is available as well.
- The **NaN values**: Symbols for defining missing or NaN values. Several options are available**.

- The **Advanced options**: This dropdown allows to select multiple columns for datetime index. This is usually beneficial when working with timeseries data and the dataset splits the datetime information into multiple columns.
- The **dataset table**: This widget dynamically exhibits how the dataset will be read inside the app with the provided parameters. If the app cannot read the table an error is presented.
- The **"show dataset info" button**: This button renders two new tabs that provide information regarding the datasets, the statistics and the data type of variables. This information is particularly useful as it indicates columns that:
    - contain non-numeric data (object, boolean, datetime) and must be transformed,
    - contain missing values which need to be processed prior to learning.
- The **"Import data" button**: That imports the dataset in the runtime memory.

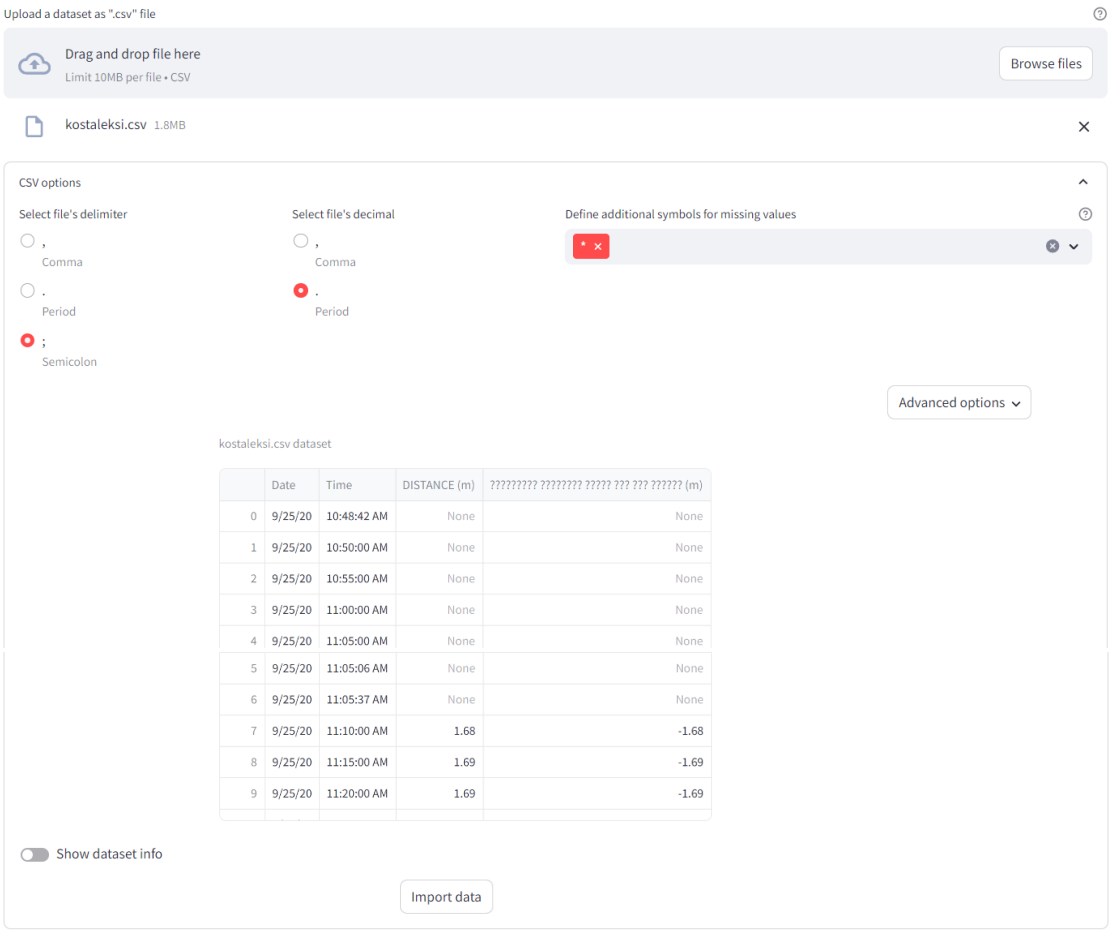


Figure 28 The upload widgets

Once the dataset is imported, three other widgets are rendered that can be utilized for modifying the existing dataset. These widgets are:

- **Change column names popover**: It provides a two-column table-widget, where the user can rename columns. Double click to the cells of the second column to provide a new (text) value and press enter.

- **Change index popover**: This popover provides two options. 1) The user can specify **a column to be used for indexing observations**. 2) The user can convert the current index into a datetime datatype. *Datetime index is used in timeseries data*.
- **Delete column popover:** The user can select a column to be deleted.

Use the **"Submit" button** inside the popovers to apply changes. After a modification is applied, a "**restore changes**" becomes visible in the UI of the page for discarding all changes and preprocessing methods that were applied in the dataset.

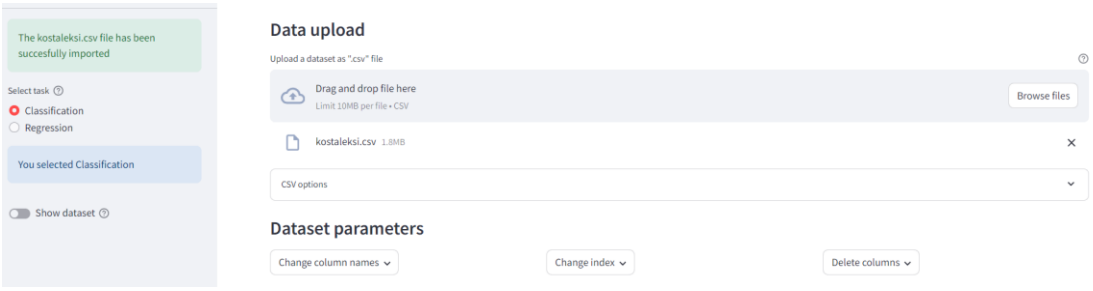


Figure 29 Widgets for dataset modifications

Data visualization

The app provides **four types of graphs** for data visualization. These are:

- **Line**: A graph that uses lines to connect individual data points. A single column is used for visualization each time.
- **Area**: Similar to the line graph, it uses lines to connect individual data points and highlights the area below (for positive values) or above (for negative values) the line. A single column is used for visualization each time.
- **Bar**: Graphical display of data using bars of different heights. A single column is used for visualization each time.
- **Boxplot:** Demonstrates graphically the locality, spread and skewness groups of numerical data. Multiple columns can be used in a single graph.
- **Histogram**: A visual representation of the distribution of quantitative data. A single column is used for visualization each time. The user can customize the plot by defining the number of bins and the size of the graph (height, width).

Use the first selection box **"Select chart type"** to choose the graph type. Subsequently, use the column(s) to be plotted and a toggle option **"Generate graph"** appears. **Line, Area and Bar graphs are interactive**, meaning that you can zoom and shift the graph. All graphs can be expanded for a full screen view and downloaded. An example of area chart is displayed in Figure 30.

Figure 30 An example of an area chart

Data Preprocessing

Data preprocessing is an essential step in any data-driven method. The app provides several tools for **preparing the data prior to FCM learning**. These tools are grouped into the following tabs (Figure 31):

- **Data Cleansing**: Is used for handling missing data** and for processing outlier data**.**
- **Data Transformation**: Is used for transforming columns. It can be used to either transform text variables** to numerical values, or for one-hot encoding.
- **Data Normalization**: Changes the range of the values in the dataset.
- **Data split**: Split the dataset into train/validation/test sub datasets and define inputs/output variables**.

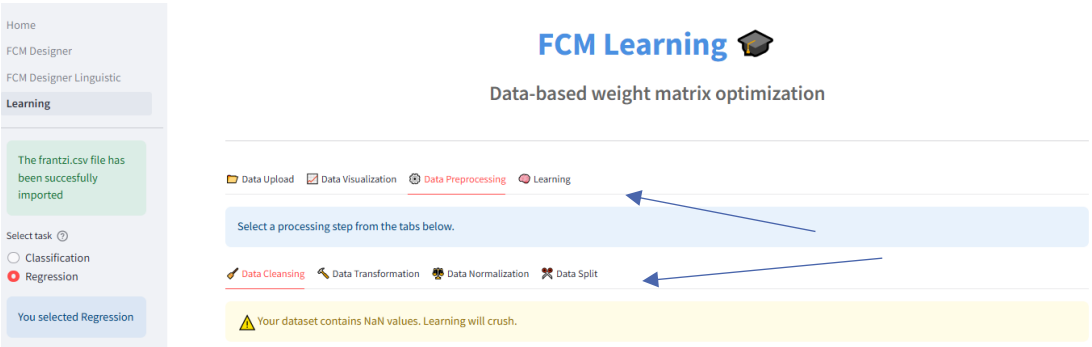


Figure 31 The preprocessing structure

***Important notes**: These steps are essential, as missing data and non-numerical columns will crash the learning procedure. Warnings are displayed in the UI when such issues exist in the working dataset (*Figure 32*). The "Learning" tab will be available after defining input/output variables.*

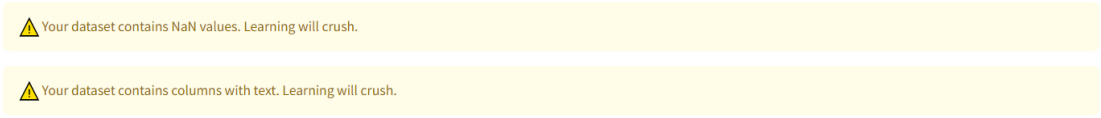


Figure 32 UI warnings for dataset incompatibility

Data Cleansing Widgets

Use these widgets to **handle missing data** and to process **outliers**. The UI offers widgets (Figure 33) for:

- Imputation of values in missing data
- Deletion of columns or rows containing missing data
- Outlier processing
- Manual processing

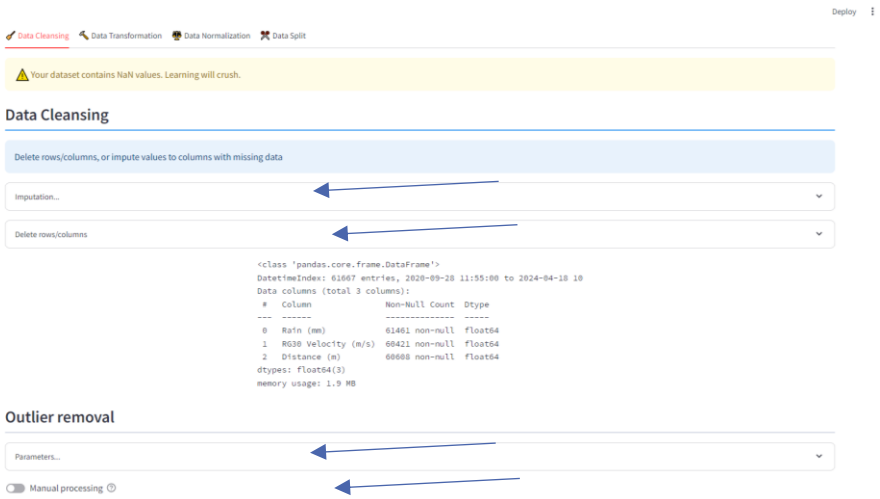


Figure 33 The Data Cleansing UI

Handling missing data

1. If missing values do not exist in the dataset, an information message and the dataset information appears on the UI, for more user-friendly experience.

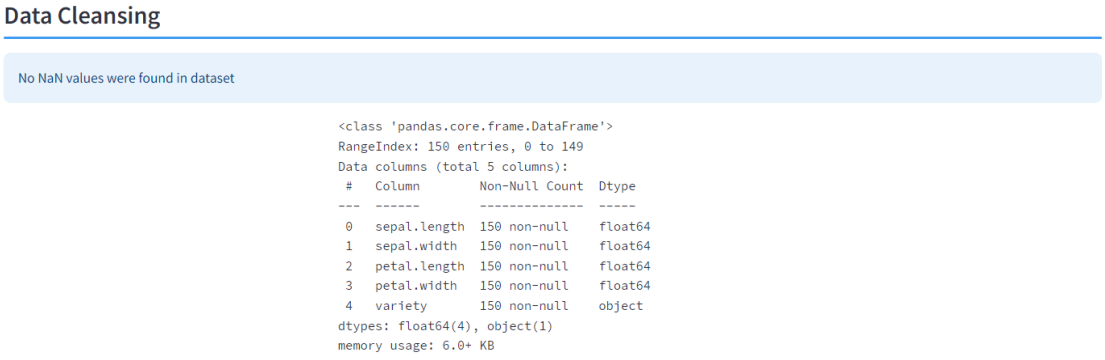


Figure 34 UI when no missing data

2. Two widgets are rendered, an **"Imputation"** and a **"Deletion"** expander, in case missing data exist, as is presented in Figure 33.
3. Imputation expander contains:
    a. a widget for selecting column for processing

b. a widget for selecting the imputation method. The user can accept one of the following methods:
   i. **Interpolation** (Linear)
   ii. **Ffil** (Forward fill): Fill values by propagating the last valid observation to next valid.
   iii. **Bfil** (Backward fill): Fill values by using the next valid observation to fill the gap.
   iv. **Value**: Provision of a specific value by the user.
   v. **Statistics**: This will render widgets for selecting a statistic value (mean, median, min, max).

c. Once a column and a method are selected, the UI renders a graph that visualizes the actual values and the imputed values, as it is depicted in Figure 35. Press "Submit" to apply changes.

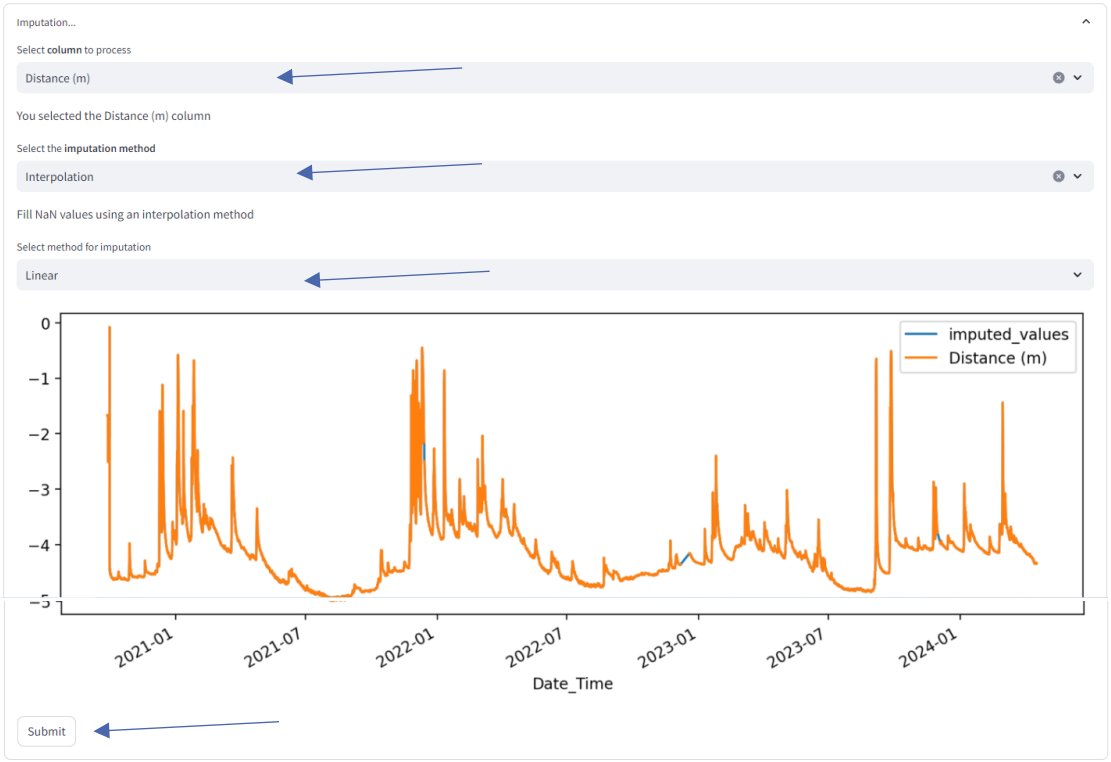


Figure 35 An example of filling missing values using the interpolation method.

4. The "Delete rows/column with NaN values" expander entails two widgets (Figure 36):
   a. A widget for selecting a whole column for deletion.
   b. A button for discarding all rows, where at least one column contains a missing value.

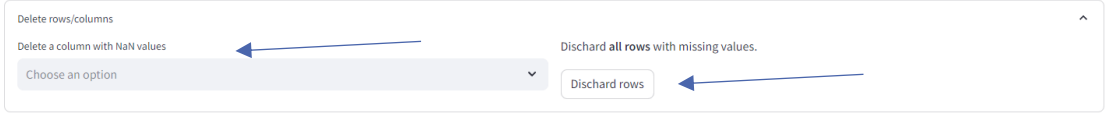


Figure 36 Widgets for deleting missing rows/columns with missing data

Outlier Removal

It offers widgets for processing outliers based on the **standard score (z-score) method**. Future versions may provide more methods based on clustering algorithms. This score helps to understand if a data value is greater or smaller than mean and how far away it is from the mean. More specifically, Z score tells how many standard deviations away a data point is from the mean. Z score = (x -mean) / std. deviation.

As it is presented in Figure 37, the user must provide the column** and the z-score threshold.

A graph then appears showing which outliers are detected. Subsequently, two tabs are rendered, and the user is prompt to select a) a replacing method or b) the deletion of these rows (the whole row in the dataset is going to be deleted).

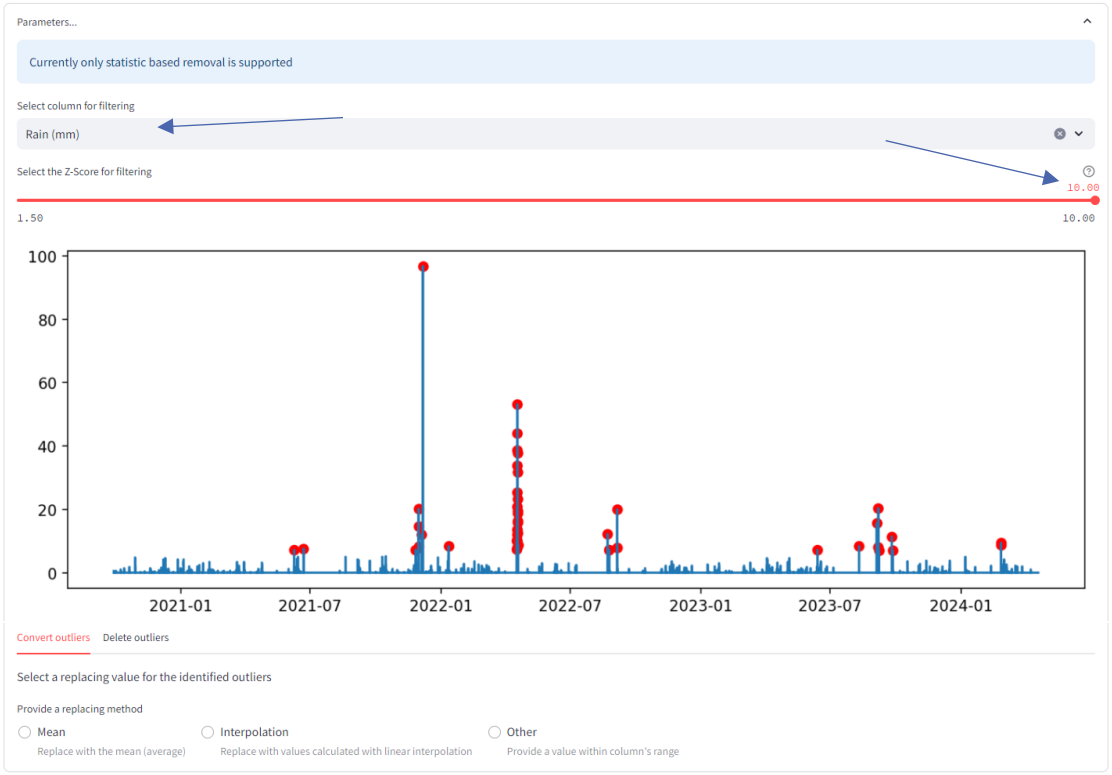
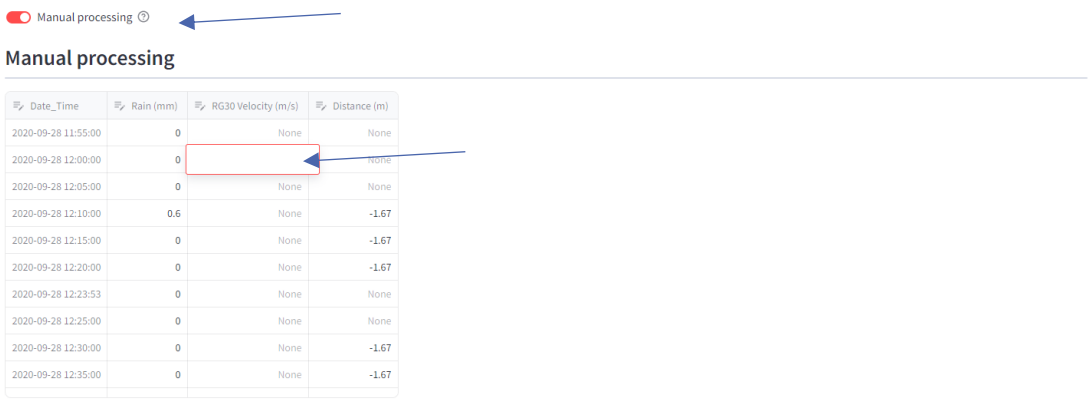


Figure 37 The UI of the z-score outlier removal method.

Manual Processing

Finally, the user has the option to manually modify values by enabling the "Manual processing" which renders a table-widget. By double-clicking the cell in the table the user can provide any value and press enter.



Data Transformation Widgets

Data transformation widgets can be employed for categorical data that may be either text or numeric variables. “Integer Encoding” and “One Hot Encoding” methods are offered, as it is presented in Figure 38.

- When **One Hot Encoding**** is selected, the user is prompted to specify a column for encoding. The algorithm identifies the unique values within this column and generates a new table. In this new table, the selected column is removed and replaced by multiple new columns, corresponding to the number of unique values found. Each new column contains binary values (0 or 1), where 1 indicates the presence of the original value in that specific column. Both the current dataset and the encoded dataset are rendered in the UI to assist users understand the transformation (Figure 39). Press “Submit changes” to apply changes.
- When **Integer Encoding** (Ordinal Encoding) is selected the user is prompt to provide a column **containing text values** which are mapped and transformed to **integer values**. The UI provides a table that shows the transformation (Figure 40). Press “Replace” to apply changes.

Important note: Transform the output variable with One Hot Encoding when working with categorical (classification) data for efficient FCM learning.

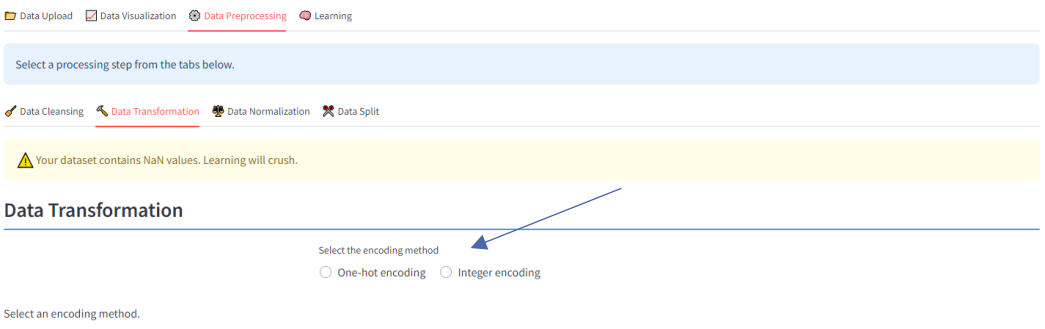


Figure 38 Data transformation options.

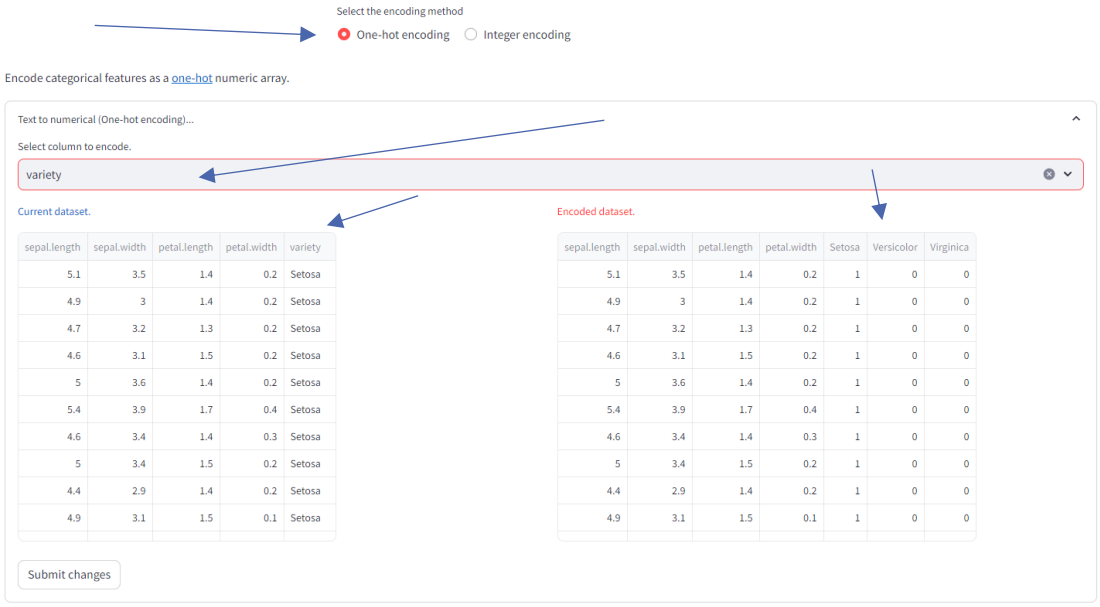


Figure 39 One Hot Encoding. The values of "variety" column {Setosa, Versicolor, Virginica} were transformed into new columns with {0,1} values. (Iris dataset Anderson, 1936; Fisher, 1936)

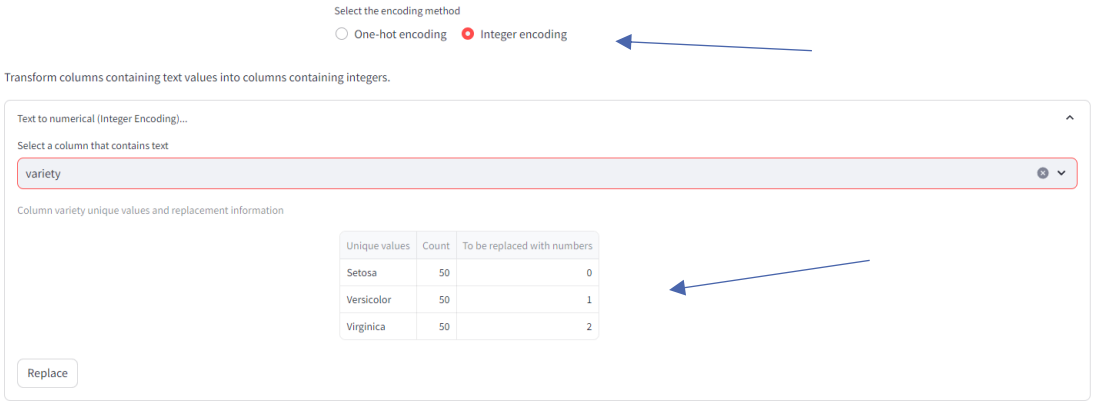


Figure 40 Integer Encoding. The values of "variety" column {Setosa, Versicolor, Virginica} were transformed into integers values {0,1,2}. (Iris dataset Anderson, 1936; Fisher, 1936)

Data Normalization Widgets

Data normalization is available only when all dataset columns contain numerical values. If this condition is not met, a warning message appears in the Data Normalization tab UI, as in Figure 41. Otherwise, the UI renders a selection box to choose the normalization method and a button to apply it (Figure 42). Currently, only the Min-Max method is available to comply with the FCM theory where FCM concept values are in [0,1].

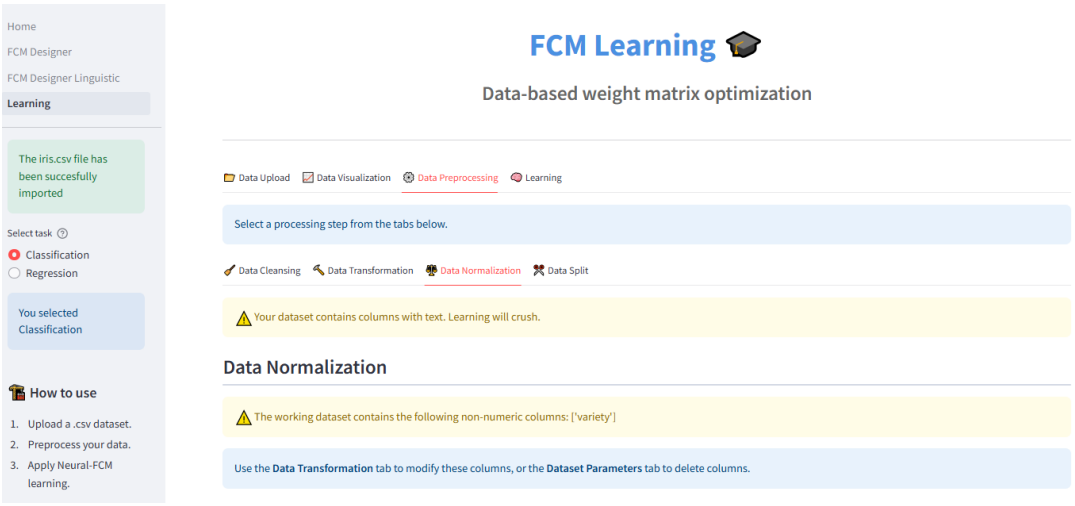


Figure 41 The Data Normalization UI does not permit normalization if a dataset contains non-numerical values to avoid crushing. A warning message appears instead.

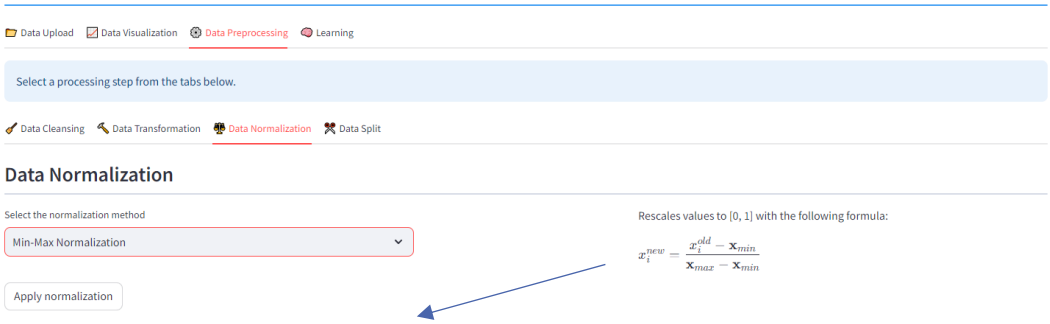


Figure 42 The normalization UI

Data Split Widgets

Data split widgets are employed to assist splitting your dataset into *train/validation/test sub-datasets* and to define the input-output variables for supervised learning.

Train/Validation/Test Split

Initially, the user is prompted to select the splitting method. The available splitting options are:

- **KFold**: Split dataset into 5 or 10 consecutive folds (Figure 43).
- **Standard**: Uses a splitting ratio in [0.6, 0.9] where for instance 0.8 indicates that 80% of the total observations are going to be used for training and the remaining for testing (Figure 44).

Additionally, the user is prompted to select the proportion of the validation dataset [0.1, 0.3] which is a sample of data held back from training your model that is used to provide an unbiased evaluation of a model fit on the training data set. Shuffling option is also available.

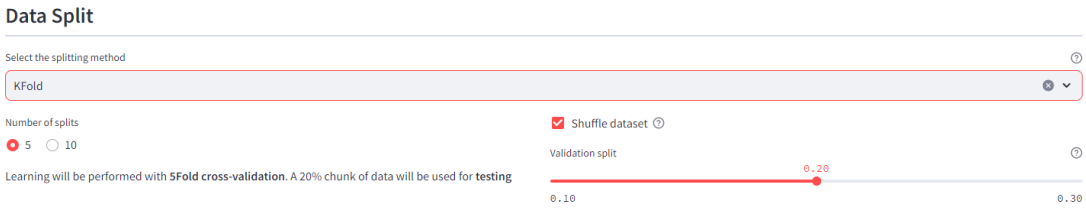


Figure 43 The "KFold" option in Data Split UI

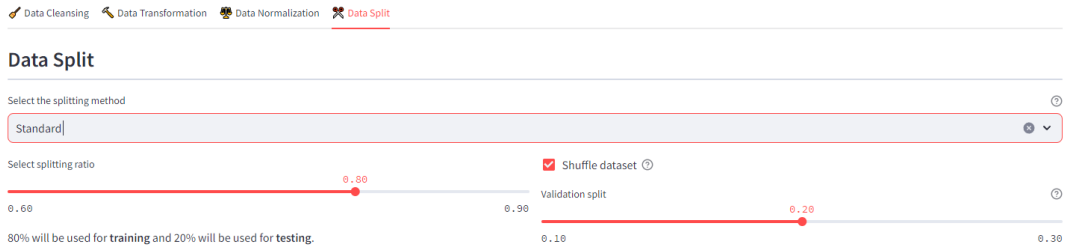


Figure 44 The" Standard" option in Data Split UI

Input/output definition

FCM-APP requires the definition of input/output variables to render the learning UI.

The user is allowed to select multiple output columns. After selecting the output column(s), new UI widgets are rendered with some minor differences between the Classification and Regression tasks, based on the FCM theory. In more detail, for a dataset with $x \in \mathbb{N}$ number of columns, and $y \in \mathbb{N}$ number of rows, and when $n \in \{1, x\text{-}1\}$ output columns (target) are defined, then the UI during:

- **classification**, prompts the user to define a (dummy) value for the *n* number of columns (Figure 45). To perform FCM inference, an initial (dummy) value must be provided to the output concepts as the FCM initialization requires an input vector with length *x* (as the total FCM concepts). After inference, the activation state of the output *n* concepts is utilized for decision making. For classification, it is recommended to define multiple output columns with One Hot Encoding. The UI allows to pass dummy values of {0, 0.5, 1}.
- **Regression** prompts the user to select if **Standard** input/output split (the way defined previously) or if **Timestep** split will be considered.
    - The **Standard** split works similarly to the classification, as it utilizes an *x*-length input vector containing *n* dummy values. However, the output activation concept state is utilized to predict a continues value.
    - The **Timestep** split introduces the concept of time and is used for timeseries forecasting. The $\boldsymbol{x}^{t-i}$ with $t \in \{1, y\text{-}i\}$, and with *i* a small integer indicating the forecasting window, is used as an input vector (without dummy values). $\boldsymbol{x}^{t}$ is used as an output vector. After *i* FCM inference iterations, x_n^{t-i} is compared with x_n^t for decision making. The UI allows to pass an integer *i* in range [1, 8]. Timestep split cannot be combined with shuffling and Kfold.

  For regression, it is recommended to utilize a single output column and to deactivate shuffling.

Press the "Submit" button to apply changes and to proceed with the Learning tab UI.

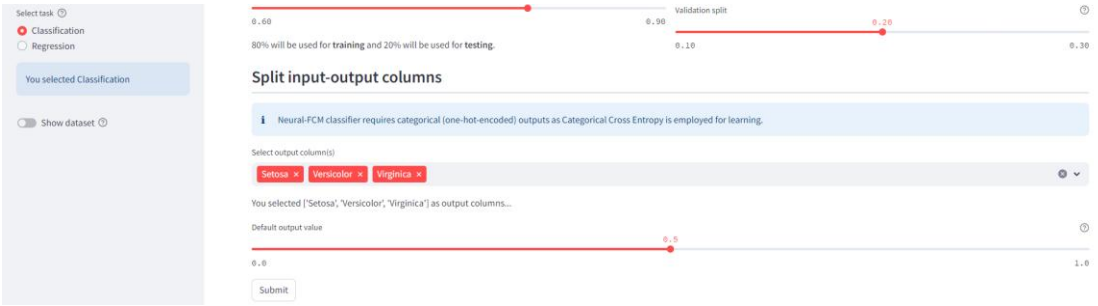


Figure 45 The input/output split UI when classification task has been selected from the sidebar

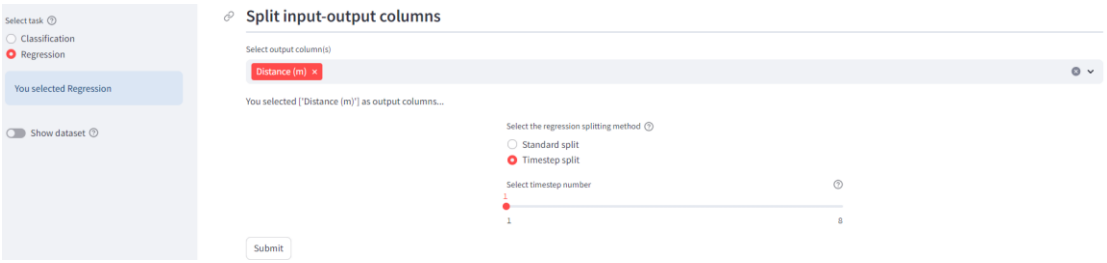


Figure 46 The input/output split UI when regression task has been selected from the sidebar

Learning

This tab renders widgets that allow you to (Figure 47):

- **Select the learning algorithm**. (Currently only Neural-FCM is provided, more options will be added in future versions).
- **Modify fitting and algorithm parameters.**
- **View learning results. After learning** the UI renders the **results widgets** which include tables, metrics and visualization graphs such as training/validation loss, confusion matrices, actual/predicted graphs.

For a deeper insight into this tab's UI, navigate to the Classification demo.

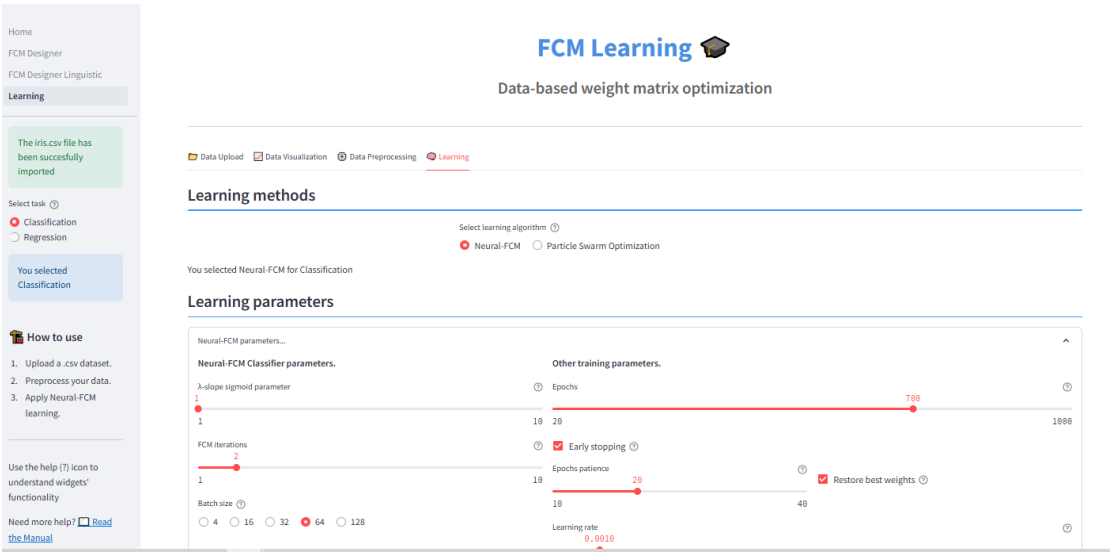


Figure 47 The Learning tab UI

Demo (Classification Learning)

In this demo the Iris dataset will be used (Anderson, 1936; Fisher, 1936). The dataset can be downloaded in a csv format from this link. This is a popular dataset for assessing classification algorithms and consists of 150 observations of three different types of irises (Setosa, Versicolour, and Virginica), with four input variables (sepal length, sepal width, petal length, petal width).

Step 1.

To begin with, we navigate to the Data Upload tab, in **FCM Learning** page, to upload our dataset(Figure 48, Figure 49). The default reading parameters of delimiter (comma) and decimal (dot) are matching the csv file properties. In addition, Iris dataset does not involve any missing data, so there is nothing else to define during uploading. Therefore, we are ready to import the dataset into runtime memory with the button “Import data” (Figure 50).

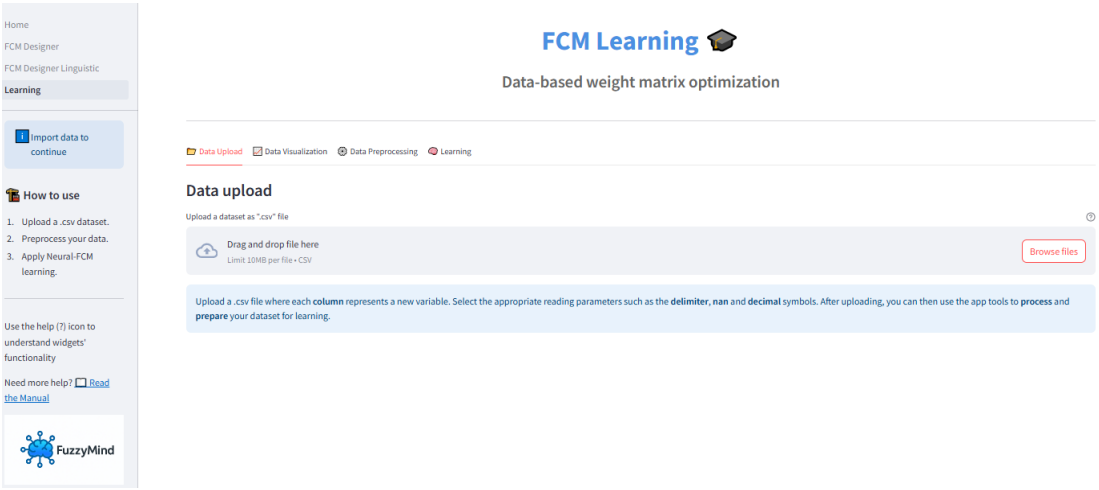


Figure 48 Data upload UI

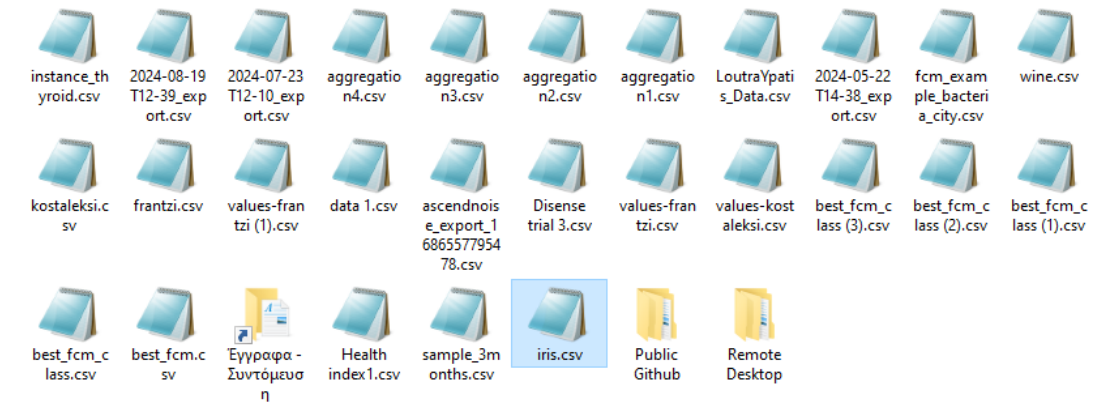


Figure 49 Loading Iris dataset from a local folder

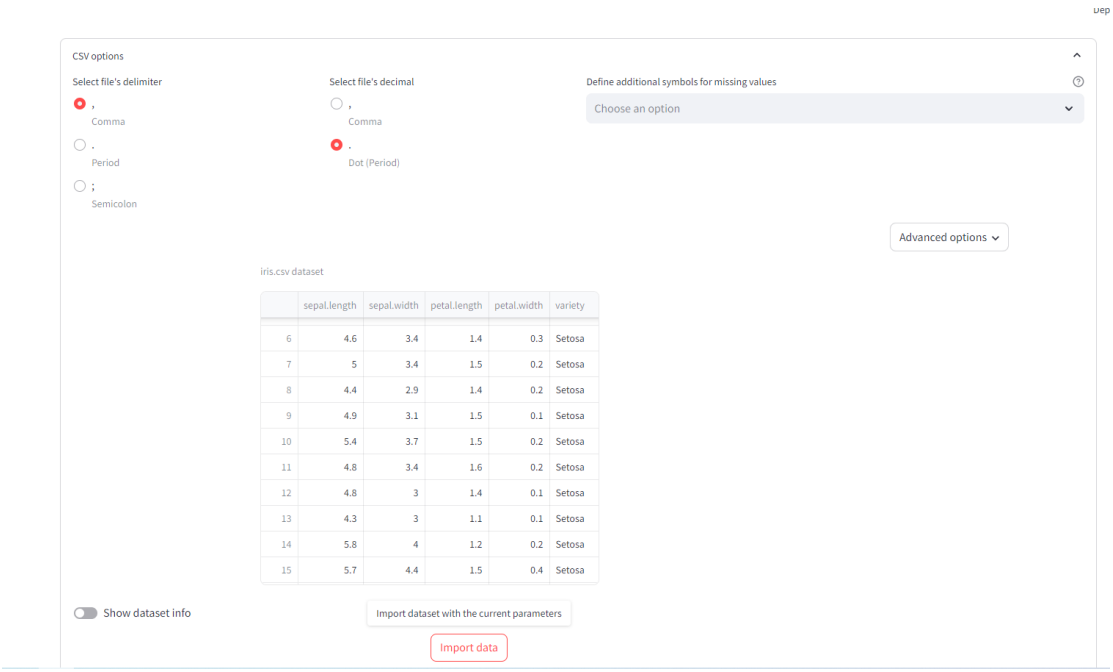


Figure 50 Reading widgets

After importing, the UI utilizes the **Sidebar** to inform us that the dataset has been imported, and to prompt us to select the learning task from the options “Classification” and “Regression” (Figure 51). We choose classification (default). In addition, the UI of the main page renders new widgets for modifying the imported dataset. You can find more about these widgets in the Data Upload section.

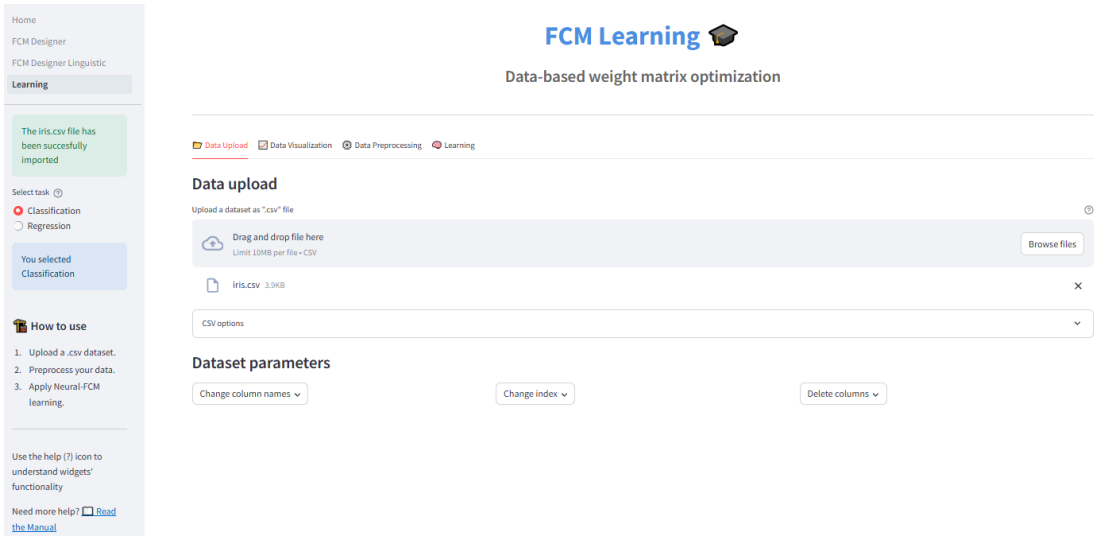


Figure 51 The UI after importing the Iris dataset.

Step 2.

This dataset contains text in the column "variety", which is the class column. Text needs to be transformed into numeric values. Hence, we will navigate to the **Data Preprocessing** tab and then to the **Data Transformation** tab (Figure 52). As it is a classification scheme, we will choose One-Hot-Encoding transformation and we are going to select the "variety" column. One-Hot-Encoding** transformation will split the "variety" into three new columns, one for each class. The UI renders the current and the transformed dataset, to point out the differences. We press the "Submit changes" button to apply changes.

***In FCM theory, One-Hot-Encoding practically means that we will have three output concepts, one for each class. Moreover, classification learning with Neural-FCM requires this multi-output concepts architecture as it utilizes the categorical cross entropy loss function.*

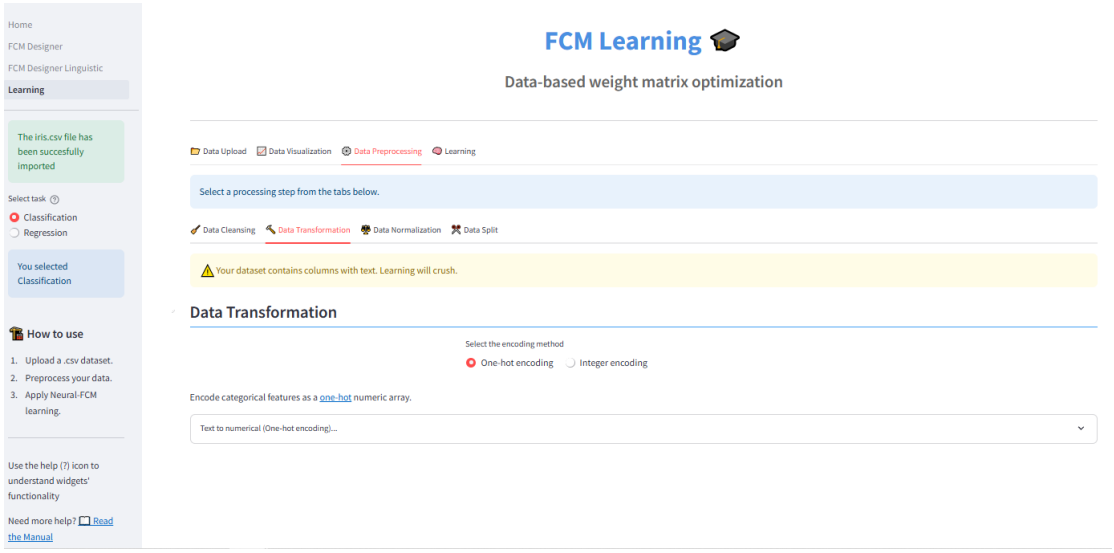


*Figure 52 **Data Transformation** tab inside the **Data Preprocessing** Tab*

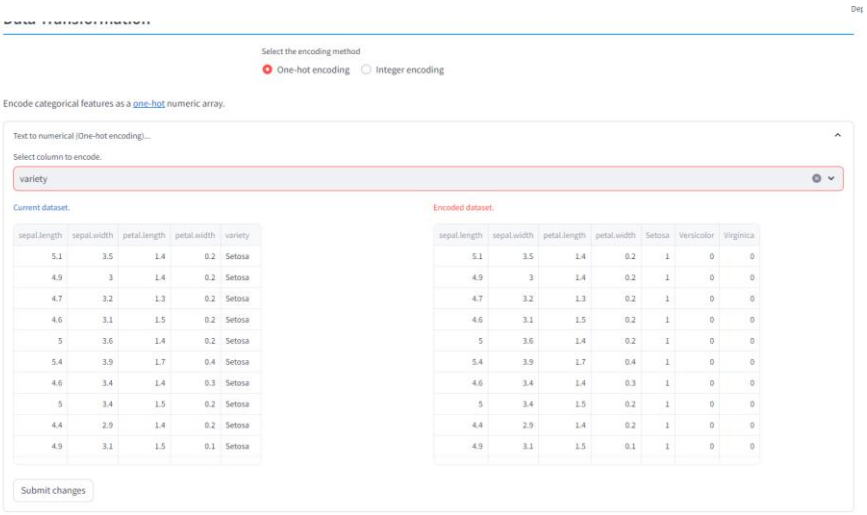


Figure 53 One-Hot-Encoding of the output "variety" column

Step 3.

In this step we will normalize the dataset using the min-max normalization method. In this way the values at each column will be scaled to [0,1]. Normalization is reported to assist in the learning convergence and enhances model performance and accuracy. Navigate to the "Data Normalization" tab where the normalization widgets are offered (Figure 54). Before pressing the "Apply normalization" button, enable the "Show dataset" in the sidebar to observe the before (Figure 55) and after effect (Figure 56). This sidebar option is available throughout the whole procedure and offers three tabs to see the 1) dataset, 2) dataset statistics and 3) dataset generic info.

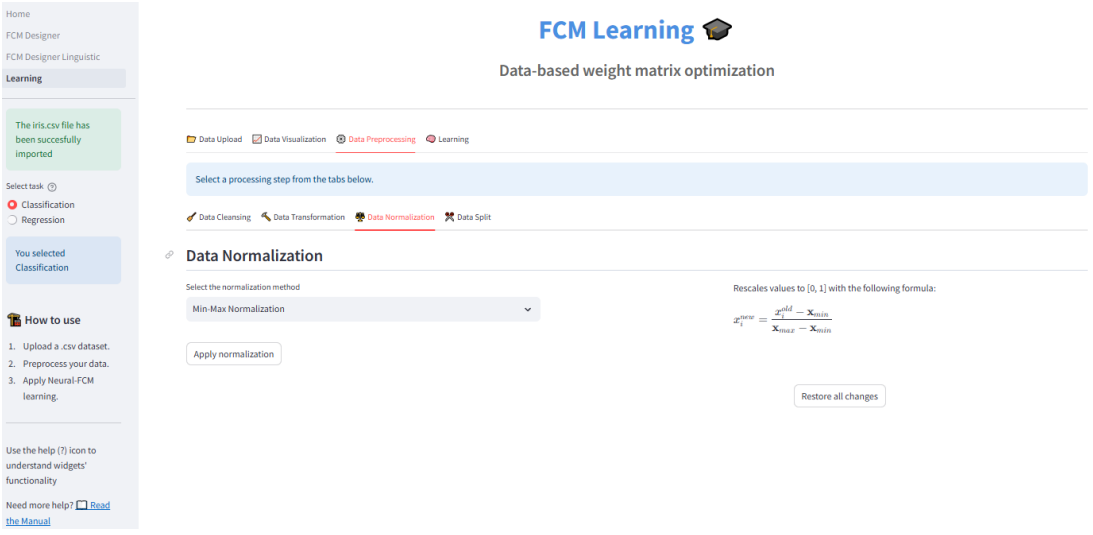


Figure 54 The normalization UI

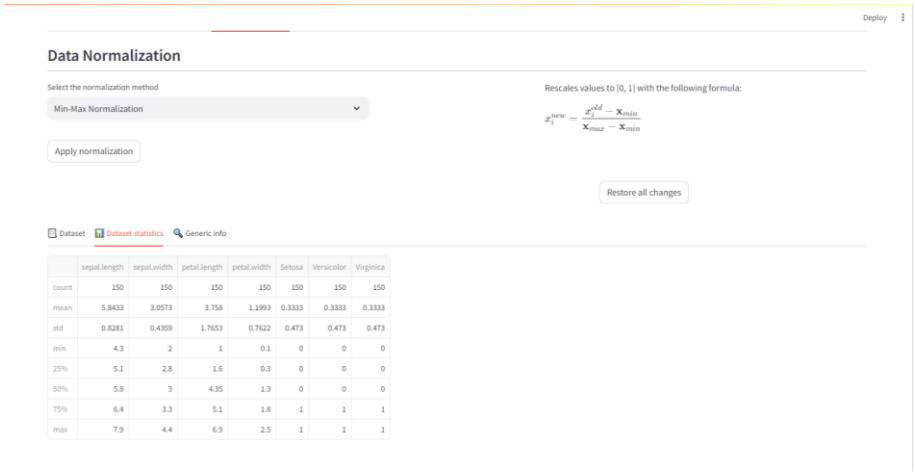


Figure 55 The dataset statistics before normalization

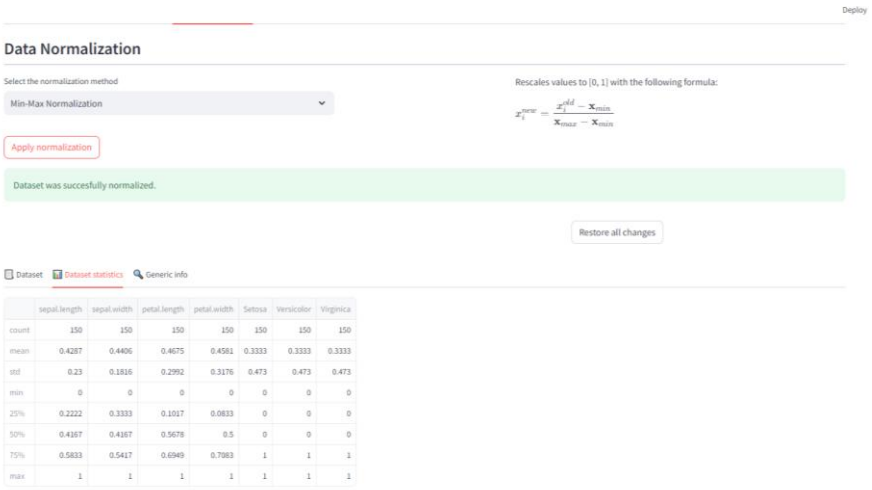


Figure 56 The dataset statistics after normalization

Step 4.

This is the final preprocessing step and will define the way we will split the dataset into train/validation/test sub-datasets, as well as the input/output variables.

We navigate to the “Data Split” tab. KFold method allows us to obtain a more reliable model evaluation. Here we will initially choose a 5-Fold scheme. Moreover, we want to split the dataset into five chunks (equally and with random order) and using four chunks for training and the remaining for testing each time. In addition, we will keep 20% random instances of the training chunks for validating training. To define these, we provide the following to the UI widgets (Figure 57).

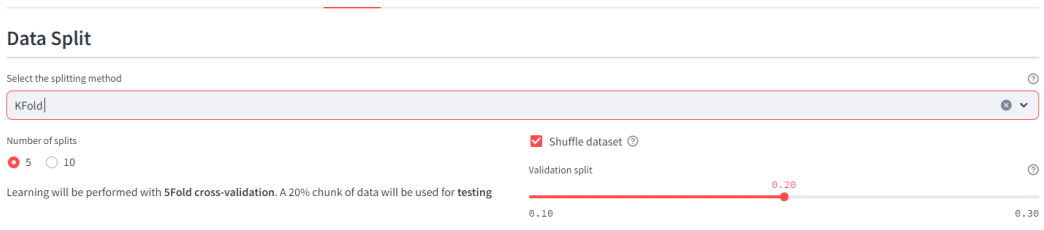


Figure 57 The 5Fold splitting method

Finally, the UI prompts us to define the input and output variables, or in FCM words to define the FCM structure (input and output concepts). This is mandatory to activate the widgets in the Learning tab UI. We select the output columns in the respective widget, and we select 0.5 as (dummy) value, and we press submit (Figure 58). The purpose of the dummy value has been explained here. The UI informs us of the success and renders the input and the output data in the main page.

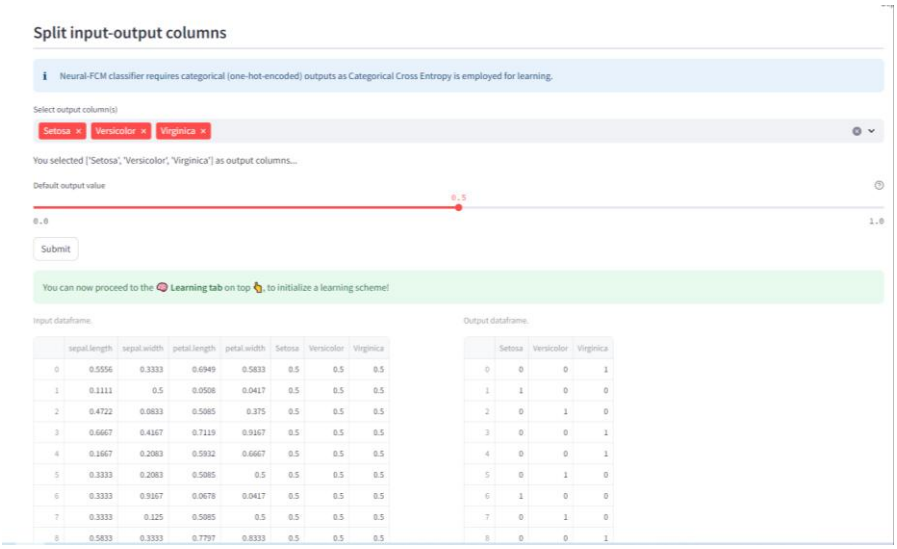


Figure 58 The input/output data definition

Step 5.

In this step we will:

1. select the learning algorithm (Neural-FCM),
2. define the algorithm and fitting parameters,
3. initialize learning

Initially we navigate to the **Learning** tab, and we select the Neural-FCM algorithm (Figure 59). Subsequently we will choose the appropriate parameters. We will choose a 2 FCM inference iterations. This practically means that we want to acquire an FCM matrix that activates the output class concept (output concept with the highest value) after 2 FCM iterations (with the modified kosko inference rule). We will use the Sigmoid function with slope of 1 (λ parameter) in the FCM inference.

Neural-FCM will utilize a neural network that will learn to output appropriate weight matrices. This neural network will be trained for 700 epochs (at maximum), with batch size of 64, learning rate of 0.002 (in Adam optimizer), and the training will be terminated if the validation loss will fail to improve for 20 consecutive epochs (patience). These are the parameters that are passed to the learning widgets (Figure 60). To initialize learning we will press the "Fit on data" button.

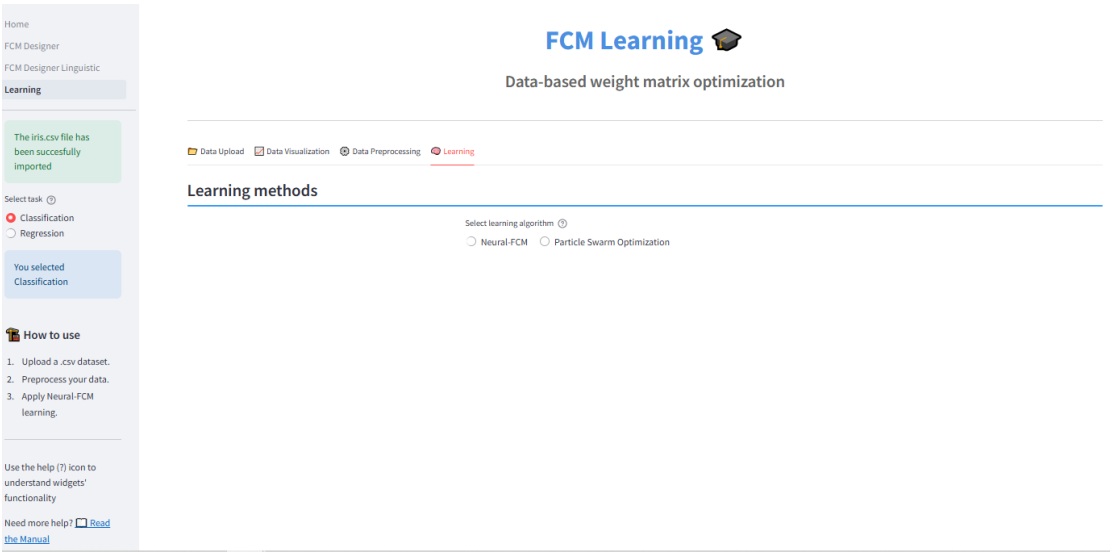


Figure 59 Learning Tab.

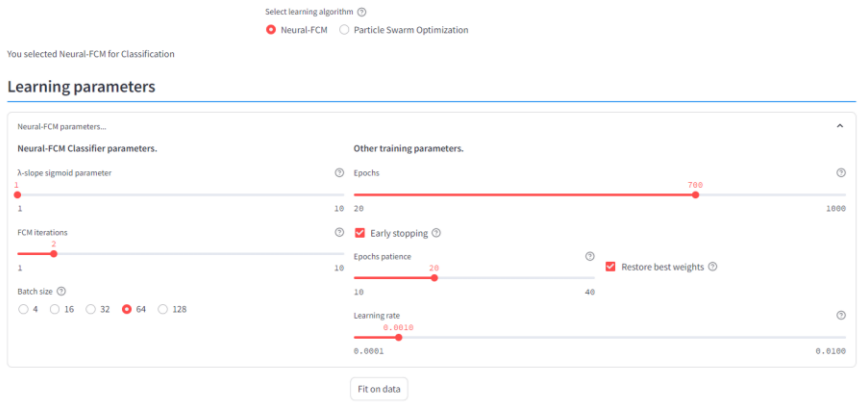


Figure 60 The parameters of Neural-FCM

Step 6.

In this step we assess the learning procedure and the learned weight matrices. Moreover, in contrast to other learning algorithms where one matrix is acquired for the whole dataset, Neural-FCM outputs a weight matrix for each instance.

As soon as the learning is finished**, the results are rendered on the bottom of the **Learning tab** page. When KFold evaluation is used, average results from all folds are rendered first. Two tabs are provided. The first tab exhibits the average learning metrics, whereas the second tab exhibits the average testing results. The table widgets can be downloaded as csv files.

***Important: Do not modify widget values outside of the results as this will trigger a re-run and the current results will be lost. Additionally, do not navigate into different pages as all progress will be lost (uploaded data, preprocessing, learning, results).*

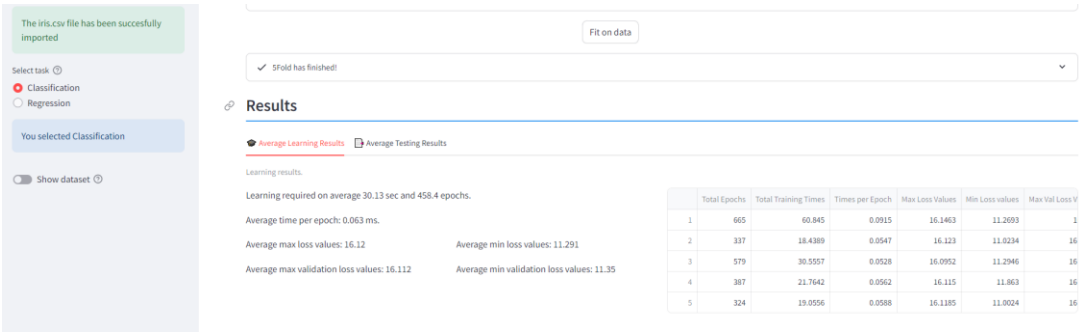


Figure 61 Average learning results. Contains information of the learning performance at each fold.

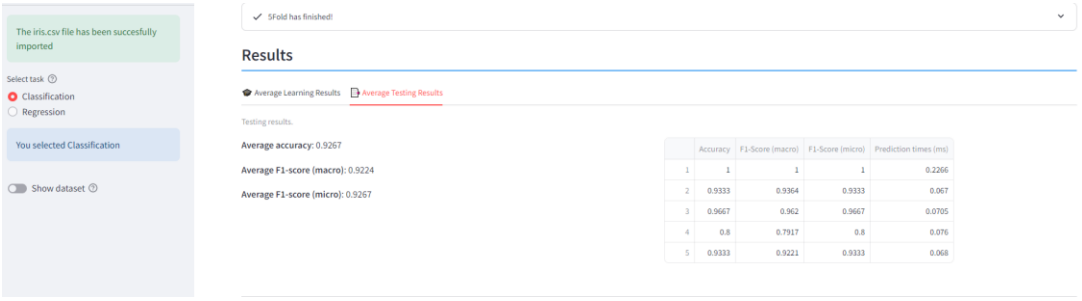


Figure 62 Average testing results. Contains information of the testing performance

After the average results, the user has the option to examine results of a specific fold. This will render static text of the achieved metrics, as well as a variety of graphs such as the:

- learning/validation loss,
- confusion matrix,
- weight matrix for each testing instance
- the FCM visualization of the weight matrix

These results are grouped into three tabs: 1) Learning results (Figure 63), 2) Testing Results (Figure 64) and 3) Weight Matrix Results (Figure 65).

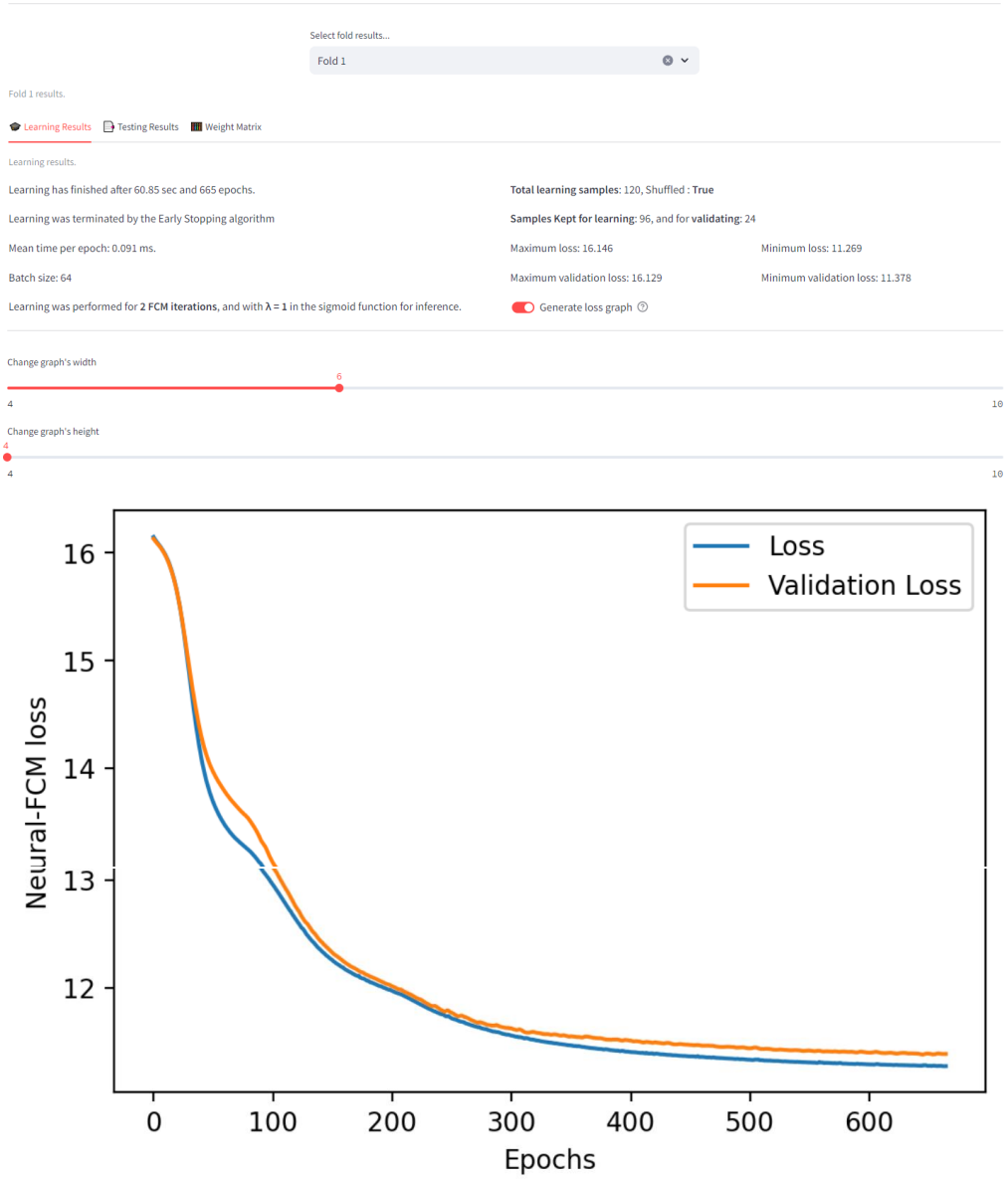


Figure 63 Results for learning

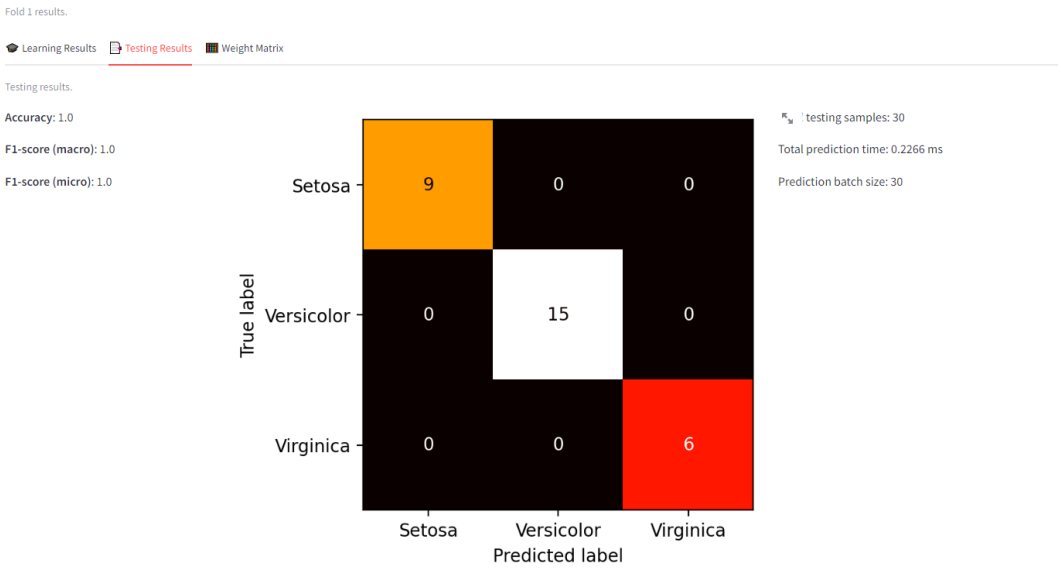


Figure 64 Results for Testing

Figure 65 Results for the FCM matrix

FAQs

- **Q: What are the app's advantages?**
  A: The app offers tools for FCM construction, visualization, data processing and learning with a straightforward UI. Therefore, it is a complete and beginner friendly tool for utilizing FCMs in custom tasks.
- **Q: Who should use this app?**
  A: The app is designed to provide an easy stepped procedure to construct FCMs manually or from data. Therefore, it can be used both by students for educational purposes, as well as by researchers who want to entail FCM aspects in their research.
- **Q: I don't have experience with Fuzzy Cognitive Maps, can I use this app?**

A: It is strongly recommended that the user should understand basic FCM theory. This manual contains several references that the user can exploit before using this app. However, anyone can replicate the experiment of the demo that is presented in this manual.

- **Q: What are the app's limitations?**
  A: Currently, the app will lose any progress when the user triggers a re-run by reloading or navigating through different pages. Additionally, limited learning algorithms are offered in this version.
- **Q: Why I lost all my progress?**
  A: User interaction with widgets triggers a re-run in the app. Several methods have been employed to provide an uninterrupted and seamlessly interaction with the app widgets and without losing any progress. However, changing pages or re-loading pages clears all session memory.
- **Q: Can I save my work/progress?**
  A: You can store locally, to your pc, your progress by downloading tables, graphs, figures and more. Currently, you cannot store your progress online.
- **Q: What dataset should be used for learning?**
  A: Table datasets in .csv files are expected. Datasets should contain the dataset as whole. You can use the app's tools to split and preprocess your dataset. You can find more information regarding the appropriate form in this manual.
- **Q: Can I use custom data?**
  A: Of course. The UI allows you to upload and process any table-like dataset in the .csv format. Check the Data Upload section to find out more, about the expected datasets.
- **Why .csv files?**
  A: .csv files are widely used for storing table-like data. Other applications, such as Microsoft Excell offer .csv exports. Hence, .csv ensures compatibility.
- **Q: I'm facing an issue that has not been identified in the manual or in the FAQs, what should I do?**
  A: This app is under continue testing and integration. User feedback, suggestion, and contribution is always welcome. Please consider contacting us for more.
- **Q: How can I know what each widget is used for?**
  A: There is a help icon next to most of widgets that reveals further information (as static text) when you mouse over them.
- **Q: I uploaded data but cannot use the learning tab, why this happens?**
  A: We try to provide a straightforward stepped data-driven procedure that would be beginner-friendly. If the backend identifies data issues such as non-numeric values, missing data, no definition of input/output variables, then the Learning tab is going to be disabled to avoid crashing. You can read this manual and the demo to understand how to exploit the app's tools in your custom data.
- **Q: Learning provides poor metrics, what can I do?**
  A: This is a common issue in data-driven methods. FCM learning may be less efficacious in some tasks and more in others. You can always compare the achieved metrics by the FCM learning to other machine learning methods. You should also verify that you provide appropriate, pre-processed, clean and normalized data to the learning algorithm. You can always contact us for more.
- **Q: Can I use other package versions?**

A: The app was developed with TensorFlow version 2.9.1 which supports CUDA acceleration on Windows, and Streamlit 1.33.0. These 2 libraries have an unmatched dependency, the protobuf library. Using --use-deprecated=legacy-resolver in *pip install* command currently solves this issue. If you are running a different OS, or you don't have a CUDA-GPU, consider installing the latest versions.

Contact Information

This app was developed by PhD student **Theodoros Tziolas** and under the supervision of Professor **Elpiniki Papageorgiou**. If you have any questions, suggestions, facing any issue or need any further information you can find us in the following

Tziolas Theodoros,

M.Sc Electrical Engineer, PhD Student

ttziolas@uth.gr

Dr. Elpiniki Papageorgiou

Professor in Artificial Intelligence

elpinikipapageorgiou@uth.gr

GitHub

https://github.com/theotziol

https://github.com/ACTA-Lab-University-of-Thessaly

https://acta.energy.uth.gr/