**The ISIP Machine Learning Demonstration (IMLD)**

**User’s Guide**

Chart

Description automatically generated

*Prepared By:*

Thao Cap, Aaron Kreitzer, Matthew Miranda, Dakota Vadimsky, and Joseph Picone

The Institute for Signal and Information Processing

College of Engineering ­– ENGR 718

Temple University

1947 North 12th Street

Philadelphia, Pennsylvania 19122

Tel: 215-204-4841; Fax: 215-204-5960

Email Contact: picone@temple.edu

1. **Introduction**

The related fields of machine learning and pattern recognition have enjoyed significant success in recent years due to the impact of deep learning algorithms [1]. Pattern recognition is the automatic recognition of regularities or trends in data. Machine learning, a closely related field that has evolved considerably in the past two decades, refers to the ability of a machine to learn and adapt to data, improving a system’s ability to detect patterns and perform inference. These methods are ubiquitous in engineering today impacting diverse fields including signal and image processing, human language technology, bioinformatics, and finance. The ISIP Machine Learning Demo (IMLD) is a tool used to introduce the basics of machine learning using a highly interactive environment in which users can easily visualize the performance of an algorithm. IMLD was first developed as a Java applet in the late-1990’s when Java applets were envisioned as the future of interactive computing [2] and there was an emphasis on web-based educational tools [3]. The Institute for Signal and Information Processing, then located at Mississippi State University, developed a suite of interactive demos to teach important concepts in signal processing [4].

However, today, due primarily to security issues, Java applets have fallen in disfavor and are no longer being supported. Instead, it makes more sense to deliver such applications in Python, where the bulk of machine learning research is being done. This allows the application to integrate a wider range of algorithms. Hence, a major focus of this work, being conducted at the Neural Engineering Data Consortium at Temple University, is the conversion of this application from Java to Python. However, as we will discuss in this work, delivering a complex interactive application in Python is not as easy as one might think. Major concerns include the stability of the graphical programming languages available and web accessibility.

A typical screenshot of the user interface is shown in Figure 1. IMLD allows users to create unique two-dimensional data sets that can be easily visualized. A set of standard algorithms are available including fully supervised approaches such as Principal Components Analysis [5], unsupervised approaches such as K-MEANS clustering [6], and popular neural network algorithms such as multilayer perceptrons [7]. The number of classes, classification modules, and other key parameters of the data generation are user defined. A dialog box is included that displays step-by-step computations for the algorithm. Users can step through the algorithms or run them in their entirety. Decision surfaces are rendered, and error rates are computed on the training and evaluation sets.

Chart

Description automatically generated

Figure 1. The IMLD main window consists of three main windows: visualizations of the training data (upper left), the eval data (lower left) and a process log (right).

Users can choose from a wide selection of predefined data sets such as multivariate Gaussian data, draw custom data sets, or create a combination of the two. An important new feature of the Python version of this application is that data can be saved to a file or uploaded, allowing users to experiment with their own data sets and use IMLD as a reference implementation. Newly created data can be exported into a CSV file where comment fields are used to hold class names and class color, while each line holds the x and y coordinates. This is a simple format we have used for our machine learning class that makes it easy for novice programmers to interface to the data. It was also a preferred format based on a survey we conducted with the user community. The application also allows CSV files to be imported for further study and modification.

IMLD supports the configuration of data sets and analysis parameters through drop-down menus. Class, Color, and Scale are all tools that allow users to uniquely configure their data. While adding a new class, users are prompted to add a name, and then choose from over the 150 colors options. Once set, users can now draw data points. Users can add a new class at any time by navigating to the Classes menu and adding another class. The Scale of the data can also be managed through the Classes tool while in the Classes menu. Users can at any time choose which of the added classes they would like to delete.

Users can also choose from a selection of prestored, or canned, demos that include classic machine learning datasets such as overlapping Gaussian distributions, toroidally shaped distributions that cannot be classified with a linear classifier, and a yin-yang distribution that requires a nonlinear decision surface. Finally, after creating the data, users can choose the algorithm under the algorithm section and select the ‘run’ or ‘run by step’ options to start classification.

IMLD is an educational tool with the ability to walk users through a step-by-step process to visualize various machine learning algorithms. It runs on the three major operating systems available today: Windows, Mac, and Linux. It has been used by a machine learning class we have been teaching since the late 1990’s (*https://www.isip.piconepress.com/courses/temple/ece\_8527/*). It is easily installed on a platform that includes Anaconda v3 and Python v3.7 or later. The source code is available from the course web site at: *https://www.isip.piconepress.com/courses/temple/ece\_8527/resources/imld/*.

1. **Installing IMLD**

IMLD is available at the link below:

<https://isip.piconepress.com/courses/temple/ece_8527/resources/imld/>

Select the compressed .tar file link (e.g., [imld\_v1.7.0.tar.gz](https://isip.piconepress.com/courses/temple/ece_8527/resources/imld/imld_v1.7.0.tar.gz)) corresponding to the most recent version to begin downloading, then unzip the contents of the file to your preferred directory. The distribution will be unpacked into a directory named “/imld” (using a Unix convention):

**nedc027\_[1]: pwd**

**<installation location>/imld/v1.7.0**

**nedc027\_[1]: ls -l**

**total 16**

**-rw-rw-r--@ 1 picone staff 1678 Feb 25 22:22 README.txt**

**drwxrwxr-x@ 3 picone staff 96 Feb 25 22:22 bin/**

**drwxrwxr-x 2 picone staff 64 Feb 25 22:22 DOCS/**

**drwxrwxr-x@ 9 picone staff 288 Feb 25 22:22 files/**

**-rw-rw-r--@ 1 picone staff 112 Feb 25 22:22 requirements.txt**

**drwxrwxr-x@ 7 picone staff 224 Feb 25 22:22 src/**

A readme file, named README.txt, is included in the root node of the distribution that explains the process for installing and running IMLD. This file contains detailed instructions on what versions of various Python libraries are needed to run IMLD. These include:

* Python 3.7.x or higher (we recommend installing Anaconda)
* PyQt5: https://www.riverbankcomputing.com/software/pyqt/download5
* Numpy/SciPy: http://www.numpy.org/
* PyQtGraph: http://www.pyqtgraph.org/ (v0.12.1 or higher)

The automatically generated file named “requirements.txt” describes the specific versions of these tools needed:

**nedc027\_[1]: more requirements.txt**

**matplotlib==3.4.3**

**numpy==1.21.2**

**PyQt5==5.15.4**

**pyqtgraph==0.10.0**

Graphical user interface

Description automatically generated with medium confidence

Figure 2: The IMLD main window consists of three main windows: visualizations of the training data (upper left), the eval data (lower left) and a process log (right).

**scikit-learn==0.24.2**

**scipy==1.7.1**

**sklearn==0.0**

We utilize Python’s SciKit Learn library [8] to implement our algorithms, pulling out features produced by the algorithms to display data and plot necessary components with MatPlotLib [9].

1. **The IMLD User Interface**

The main window of IMLD’s user interface is shown in Figure 2. This is what appears when you open the tool using a sequence of commands something like this:

**cd <installation location>**

**bin/imld**

If you are a frequent user of IMLD, we recommend you put the location of the bin directory in your path.

In this section we will explain the options available under the drop-down menu selections shown at the top of Figure 2.

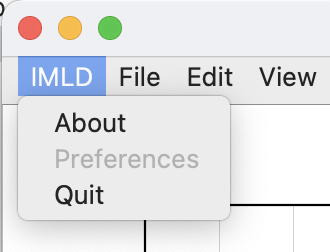


Figure 3: The menu choice IMLD contains three sub-menus: About, Preferences and Quit.

* 1. **Menu Choice: IMLD**

Under the IMLD menu item, as shown in Figure 3, contains three choices: About, Preferences and Quit. The About sub-menu displays a message about the version of the tool. The Preferences sub-menu, as with most tools, allows you to customize some of the basic look and feel of IMLD. The Quit menu is the preferred way to exit the application.

* 1. **Menu Choice: File**

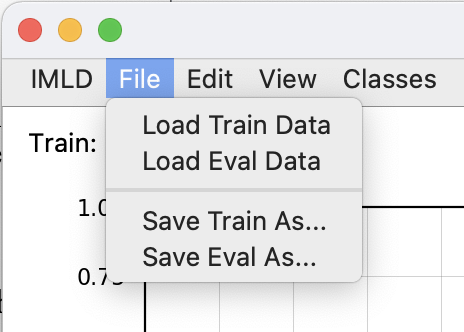


Figure 4: The menu choice File provides a way to load and save data.

The File menu item, as shown in Figure 4, allows users the ability to load their own data sets. This is described in more detail in Section 4, where we explain the spreadsheet file format used which is based on the popular csv format [10]. This is an exciting new feature of IMLD that allows users to generate baseline results for well-known algorithms on their own personal data sets. The application supports two sets of data: train and eval. As the name suggests, algorithms can be trained and evaluated on the training set. Algorithms are only evaluated on the evaluation set using the parameters obtained from training on the train set. The “Save As” choices allow you to save your data to a file.

* 1. **Menu Choice: Edit**

The Edit menu item, shown in Figure 5, allows the user to change three of the key settings affecting the view of IMLD’s train and eval windows. The “Set Ranges” option will allow the user to change the minimum and maximum limits of either axis in either of the data displays. The “Set Gaussian” option is used to change the number of points and variance of drawn Gaussian data. This will be explained more in section 5. The “Set Color” option changes the colormap of the decision surface, to provide more accessibility for color blind users.

Graphical user interface, table

Description automatically generated

Figure 5: The menu choice Edit provides a way to change various view settings.

There are many options to clear one or more windows on IMLD as well. Users can use these options to clear the process log, clear data and/or algorithm results from the train/eval windows or do all three. Choosing “Reset” will perform the above functions, in addition to deleting the data classes that had previously existed.

* 1. **Menu Choice: Classes**

Within this menu choice, users can add and delete classes from their train/eval windows. When adding a class, the user can decide its name and color. If a color is not selected a default color will be chosen. When the “Delete Class” option is chosen, the active class and all its data points will be deleted, and if there is no class to delete and warning message will appear. Once classes are created, they will be shown under this menu choice and users can choose the active class to add points to. This menu can be seen in Figure 6.

Chart

Description automatically generated with medium confidence

Figure 6: The menu choice Classes allows the user to create and delete classes.

* 1. **Menu Choice: Patterns**

This menu choice allows the user to select between drawing points and drawing Gaussian distributions when adding data to the train/eval windows. This will be discussed more in section 5.

Chart

Description automatically generated

Figure 7: The menu choice Patterns has the user choose between drawing methods.

* 1. **Menu Choice: Demo**

The Demos, or Demonstrations, are customizable datasets that are built into IMLD. These allow you to test out different algorithms without needing to import your own unique data and are meant to show you common shapes that data can take. Each demo has its own specific popup window in which you can edit the parameters of the created dataset, such as number of points per class and position on screen. This menu choice is shown in figure 8.

* 1. **Menu Choice: Algorithm**

The Algorithms choice, shown in figure 9, contains algorithms that IMLD supports. These algorithms allow users to customize most of the parameters required to run the algorithms. All these customizable parameters are consistently defaulted by IMLD if the user does not wish to customize the algorithm. These default values are based on the Sklearn recommendation.

A picture containing box and whisker chart

Description automatically generated

Figure 8: The menu choice Demo provides the user with 8 demonstration datasets to create.

* 1. **Process**

In the process section, pictured in figure 10, you can select how you would like to run your selected algorithm. If an algorithm has not been selected, or if there is no data in the training window, the process will not start and instead a warning message will be printed to the process log. If your evaluation data has a different number of classes than your training data, the evaluation data will not be classified, and a warning message will instead be displayed.

Chart

Description automatically generated

Figure 9: The menu choice Algorithms allows the user to select which algorithm they want to use.

It is recommended that once the classification process is complete, that you use the reset or clear commands before attempting to run another algorithm, even if you are using the same algorithm on the same data. This will ensure that no information from your most recent process will affect your new run.

Graphical user interface

Description automatically generated

Figure 10: The menu choice Process lets the user run their chosen algorithm step-by-step or all at once

1. **File Formats**

Data in IMLD is saved in comma separated value (.csv) format. You can see an example of this file format, exported from IMLD in Figure 11. Each line in the data designates a single point, with its class label and x- and y-coordinates. These data points are preceded by header lines, which contain identifying information about the dataset contained within the file.

The filename is whatever name you choose to save the file as when you export it and will include the directory path to the folder in which it is saved.

The classes relate to the classes in your data. In the figure we specify three classes “0”, “1”, and “2”, and our data belongs to those same three classes. If you specify a class in this header line that does not have any corresponding points in the data, that class will be discarded, and a warning message will appear. If you have data in your file for which a class is not specified, a class will be created for it.

Text

Description automatically generated

Figure 11: Comma Separated Value (.csv) File Format

The colors header tells IMLD what color to paint the data points of each class. If you have too many colors specified for your number of classes, the excess colors will be discarded. If you do not have enough colors specified for your number of classes, default colors will be selected for the remaining classes.

The limits header specifies the limits that will be set for the window you are importing your data to.

When exporting data from IMLD, these headers will be automatically added. Filename, classes, colors, and limits will use the values that are currently being used in the specified window. Data points will follow these headers, sorted by class.

When importing data into IMLD, however, none of these headers are required. IMLD will construct the classes based on class names it finds for the data, will assign default colors to each class, and will “zoom” out the specified window far enough so that all data is visible. If your file does include headers, they do not need to be in any specific order nor do they have to be at the top of the file. If they are named appropriately, IMLD will extract the necessary information from them. The data does not have to be ordered in any specific way; if it includes the label, x- and y-coordinates, and each part is comma separated, the data will import properly. This has the side benefit of making IMLD a relatively easy option if you wish to sort complex data and assign headers to it; simply import your data then export it.

Graphical user interface, application

Description automatically generated

Figure 12: Overlapping Gaussian Parameters

1. **Data Generation**

Walk users through how to create data, add data to an existing set, save data, load data

To create your own data, you will first need to create classes for that data. Use the “Classes” option as discussed above to create your desired number of classes with unique names and colors. When adding points to these classes, be sure to select the active class by using the “Classes” drop down menu.

The default option when drawing points in the data windows is to draw individual points. You can use the “Patterns” option to change this to instead draw Gaussian distributions. Use the “Set Gaussian” option under “Edit” to change the parameters of this distribution. Clicking the mouse in the data window will draw one point/distribution for the active class, while holding the mouse down will continue to draw points/distributions. If your cursor goes outside the window boundaries, you will cease drawing points. This procedure is the same if you wish to add points to imported datasets.

When importing or exporting datasets, make sure you are exporting from or importing into the right window (train or eval), and ensure that your imported dataset is in .csv format.

1. **Algorithm Overview**
   1. **PCA**

Principal Component Analysis (PCA) aims to reduce the axes along which the data has the most variance. To do this it uses the labeled training data to find the mean and covariance of each class, then uses this new information to calculate the eigenvalues and eigenvectors to perform dimensionality reduction. Once this is complete, for each data point in the training set it calculates the mahanoblis distance to each calculated mean and assigns that point to the class for which this distance is minimized. It then compares these predicted class labels to the known labels to find the error rate. By doing this for all “test points” in the dataspace, it can build a decision surface with a threshold delineating the areas for each class. It can then use this decision surface to predict the classes for the evaluation data.

PCA has two options, Class Independent (CI) and Class Dependent (CD). PCA-CD is what was described above. PCA-CI, on the other hand, combines the training data from each class to calculate the covariance which then affects dimensionality reduction and the rest of the classification process in turn.

* 1. **LDA**

Linear Discriminant Analysis (LDA) is similar to PCA in the sense that it also aims to find the mean and covariance of each class, and then eventually use these values to assign data points to classes based on mahanoblis distance. It also, like PCA, has CI and CD variants. The difference in LDA is in how it achieves dimensionality reduction, opting instead to reduce the dimensions of the data in order to maximize the variance between classes. In other words, LDA aims to separate the classes as much as possible before classification.

* 1. **KNN**

K Nearest Neighbors uses labeled training data to find a user-designated number of closest neighbors to each point in the dataspace. It does this by populating the dataspace with “test points”, then using the training data to assign each point to a class based on the majority vote of its K closest neighbors. This process creates a decision surface for each class. IMLD then compares each point in the training data to the class decision surface it falls within to calculate the error rate. The same decision surface is used when calculating the error rate of evaluation data. To prevent ties in majority voting, it is recommended that you use an odd number of neighbors in this algorithm.

* 1. **K-Means**

K-Means is a clustering algorithm that IMLD uses to assign clusters of training data points to their predicted classes. It accomplishes this by first designating K random points as the centroids of K clusters, the number K decided by the user. The clusters then expand to include all the points closest to them until all data points are assigned to a cluster. Then, using the points in each cluster, the centroids are recalculated to be the mean of each cluster. The process is then repeated until these centroids do not change between two iterations. Next, each cluster is assigned to a class whose mean (calculated separately) it is closest to. Each point in that cluster then has this predicted class compared with its actual class to determine the error rate. These clusters and their class assignments are used to calculate the error rate of the evaluation data as well. It is important to note that the number of clusters does not have to correlate to the number of data classes, the two values are independent of one another.

* 1. **SVM**

Support Vector Machines (SVM) create a line or hyperplane which separates the data into separate classes. The algorithm's goal is to have all classes separated by this line of discrimination. It accomplishes this through the repetition of two separate steps. The first is training the support vectors by focusing only on the data close to the line of discrimination. The decision region is then calculated based on the current support vector, giving more weight to past support vectors and data points. The algorithm repeats these two steps until there is a minimal error. This algorithm effectively uses high-dimensional data, which can be leveraged thanks to the kernel trick. The kernel trick is a technique used to map lower-dimensional data into higher dimensional data giving more information about the location of a potential line of discrimination.

* 1. **RF**

Random Forests (RF) are based on decision trees built on different samples and rely on a majority vote for classification problems. This algorithm combines both decision trees for decision making and ensemble techniques that combine these decision trees. The ‘forest’ is generated through bagging, a technique used to build models using a sample subset and then combine the predictions to minimize variance. This technique allows the algorithm to nullify the decision tree’s weakness of overfitting and instead supplement it with different variations.

* 1. **MLP**

Multilayer Perceptron (MLP) is a feedforward neural network connecting multiple layers in a single direction, starting with an input layer made from numerous weights. The input layer then feeds into the hidden layers where each node has a non-linear activation function. A single output layer is created after traversing through the hidden layers, and backpropagation is then used to train the network. After each piece of data is processed, the weights of the previous nodes are changed to reduce error and repeated until a satisfactory result is reached.

1. **Examples**

Go through three examples end to end that demonstrate the application. Include screenshots and discuss the meaning of the results.

* 1. **Simple PCA using two overlapping Gaussians**

To begin, we need to create the demonstration dataset. Go to “Demo” -> “Overlapping Gaussian” -> “Train” to open the parameter window shown in Figure 12.

The number of points for this and all other demos is per class, meaning in this instance that we will have a total of 20,000 points. The Mu (or mean) for each class represents the center point for that Gaussian distribution, while the Cov (or covariance) represents how “spread out” the distribution will be. For this example, will keep these parameters unchanged from their default values. Hit Submit to plot this demo to the Train window as shown in Figure 13.

Next, select PCA algorithm by going to “Algorithms” -> “PCA” -> “Class Independent”. As discussed in the Algorithms section, Class Independent (CI) PCA works by combining classes and calculating the mean and covariance for the entire dataset as a whole, whereas Class Dependent (CD) calculates these values for each individual class. For this example, we will use CI-PCA. Once the algorithm is selected, you may select either “Run” or “Step” under the “Process” menu choice to run the entire classification process at once or step-by-step.

Chart, scatter chart, bubble chart

Description automatically generated

Figure 13: Overlapping Gaussian Plot

The decision surface for this completed classification can be seen in Figure 14, while the process log is shown in Figure 15.

* 1. **Data Generation, Saving to a file, loading and evaluating**

For this example, we will start by creating 3 classes, named “0”, “1”, and “2”. We will allow IMLD to assign these classes default colors. Once we create each class, we will Select “Patterns” -> “Draw Gaussian”, then draw points to the Train Window. This results in a plot like the one in Figure 16.

Chart, bubble chart

Description automatically generated

Figure 14: Decision Surface Plot

Once we have our dataset, we can export it to our “files” folder using the default name “imld\_train”. Select “File” -> “Save Train As”, then navigate to the “files” folder and save. These folder can be seen in Figure 17, while the beginning of the file is in Figure 18.

We can then import this data back into IMLD for evaluation. Select “Edit” -> “Reset” to delete the plotted data and classes. Next, select “File” -> “Load Train Data”, then navigate to the “files” folder and select our previously exported “imld\_train” file. This imported data will again look like Figure 16.

Text

Description automatically generated

Figure 15: Completed Classification Process Log

Once imported, we can evaluate the data with any of our algorithms.

* 1. **Yin-Yang comparing CD-PCA, SVM and MLP**

For this example, we will use a Yin-Yang demonstration set to compare three different algorithms. To begin, select “Demo” -> “Yin-Yang” -> “Train” to open the parameter window for the demo. You will be asked to pick a number of points for the two halves of the figure, for now keep them at their default values and hit Submit to plot the Yin-Yang, as shown in Figure 19.

Now we will evaluate this Yin-Yang will CD-PCA, SVM, and MLP. For each, you can select the algorithm from the “Algorithm” menu choice. For now, keep the default parameters for each algorithm, you can feel free to change them at your leisure later.

The results for each of these algorithms can be seen in Figures 20 (CD-PCA), 21 (SVM), and 22 (MLP).

While it is not the purpose of this user guide to explain all the differences between these algorithms and their results, you can see in the figures how IMLD handles each algorithm and at least how they differ visually.

Original text here

* 1. **Open**

You can use an IDE, such as PyCharm, to open, edit and run IMLD (though editing IMLD is not recommended). Simply open your desired IDE, click File -> Open, and navigate to the folder you unzipped IMLD to.

Chart, scatter chart

Description automatically generated

Figure 16: Drawn Training Data

Once IMLD is open in your IDE, find the file named “imld.py” in the src folder and run it. This will open the program and allow you to continue.

Alternatively, you can run the imld.py file directly from the command line / terminal. This process varies slightly depending on the system you are using.

* + 1. **Windows**

For windows, open the command prompt by typing “cmd” into your search bar.

Graphical user interface, text, application

Description automatically generated

Figure 17: Exporting Data to Files Folder

Next, use your file explorer to navigate to the directory where your files are stored. Find and enter the folder marked “src”, then copy the file path to your clipboard.

In your command prompt, type the following:

**cd <your copied path pasted here>**

Once this is done, use the following command to open IMLD:

Text, letter

Description automatically generated

Figure 18: IMLD\_Train .csv File

**Python imld.py**

* + 1. **Mac**
    2. **Linux**

Chart, scatter chart

Description automatically generated

Figure 19: Example Yin-Yang Plot

1. **SUMMARY**

Python allows for Machine Learning to be the focal point within our code due to the simple and consistent syntax of Python. In addition, the flexibility of Python allows for users to apply custom scripts for unique processes that doesn’t require recompiling source code. The libraries and framework offered within Python allow for developers to implement well-tested Machine Learning algorithms. This allows for algorithms to be implemented without a full understanding of every algorithm. With Python being the top language for Machine Learning there is an ample amount of resources online.

Chart

Description automatically generated with medium confidence

Figure 20: CD-PCA Classification (8.67% error)

The most recent version of this application is available from the ISIP web site at this URL: *www.isip.piconepress.com/projects/speech/software/demonstrations/applets/util/pattern\_recognition/current/*. Any feedback or questions should be directed to [help@nedcdata.org](mailto:help@nedcdata.org) or the authors of this document.

Chart, scatter chart

Description automatically generated

Figure 21: SVM Classification (6.53% error)

…Section Break …

Chart, scatter chart

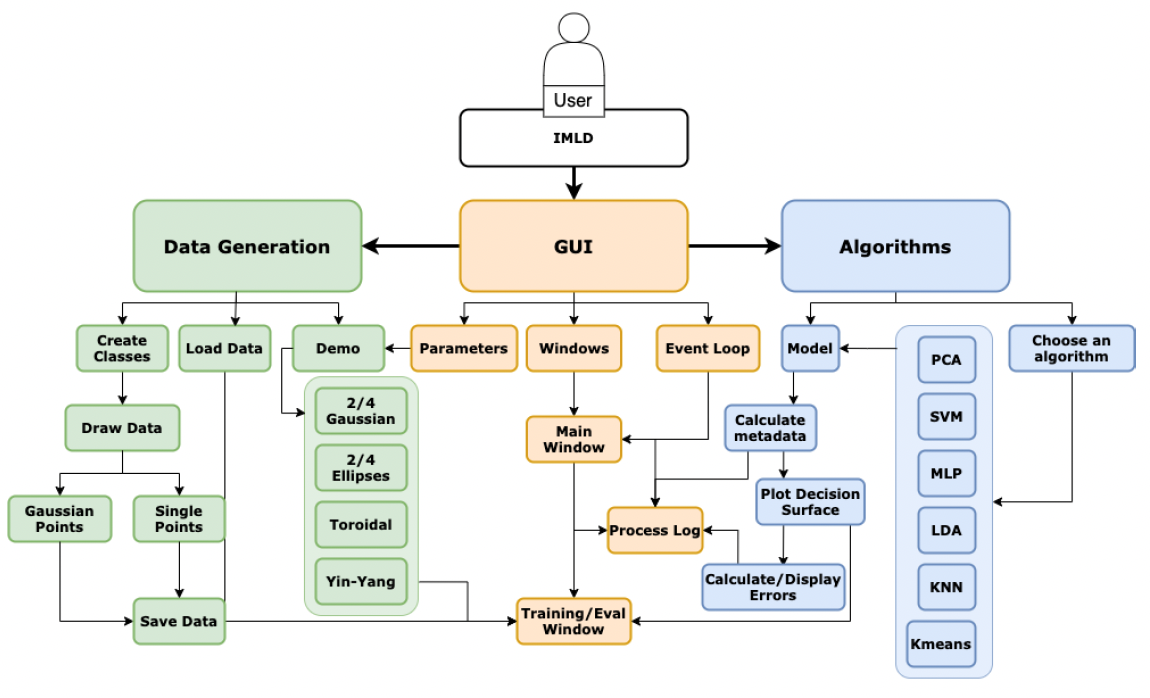
Description automatically generated

Figure 22: RF Classification (0.00% error)

Currently, IMLD does not allow users to insert their own algorithms. In the future, we plan to implement a method by which users can do this. This feature would significantly enhance the educational capacity of IMLD, as users will not be restricted to the algorithms offered only by the tool.

An overview of IMLD’s software architecture is shown in Figure 2. IMLD is dependent on a variety of third-party libraries. Many third-party libraries were considered based on the wide variety of components required to generate the applet. The following libraries were decided based on the specific needs of IMLD. PyQT5 is the framework used to create the front-end design of the application, chosen for its ability to handle large quantities of data. The backend uses NumPy for calculations, Sklearn/Scipy for implementing algorithms, and Matplotlib for interactive graphs. The software is organized into three major components: the graphical user interface (GUI), data handler, and machine learning algorithms.

The first part of the architecture is the GUI, where three key modules handle the window design, events, and parameters. The window module manages all the front-end design for the application. The module includes the menu bar, input/output graphical displays, and output log. The menu bar holds all the functionality for adding/deleting classes, choosing an algorithm, and giving users the option to either import, export, or create test data. The two graphical displays allow users to click and drag their mouse to create either point-like data or Gaussian-like data. The last section for this module is the output log which records all interactions between users and the application, i.e., adding a class or choosing an algorithm. The event module handles execution for the buttons displayed on the GUI. Lastly, the parameter module handles all secondary user information and allows users to configure the data generators.



**Figure 14.** The IMLD software architecture

Another part of the application is the data handler. One of the features that the Python version of IMLD enhances from the Pattern Recognition Applet written in Java is the software that allows users to generate data. The tool provides two underlying mechanisms of the generation of the data: generation by drawing points or generation using a functional form such as a Gaussian distribution. The training and evaluation sets are generated independently in separate windows so that a wide range of machine learning scenarios (e.g., generalization) can be evaluated. A dictionary stores the data where the key is the name of the class added, and the value is an array that holds the class color and an array of x and y coordinates.

Acknowledgments

Research reported in this publication was most recently supported by the National Science Foundation’s ﻿Industrial Innovation and Partnerships (IIP) Research Experience for Undergraduates award number ﻿1827565. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the official views of any of these organizations. Open source libraries that were used to develop the IMLD are: NumPy v1.15.4, PyQT5 v5.13.1, SkLearn v0.20.0, Scipy v1.1.0, and Matplotlib v3.0.2.

References

1. W. Samek, G. Montavon, S. Lapuschkin, C. J. Anders, and K.-R. Müller, “Explaining Deep Neural Networks and Beyond: A Review of Methods and Applications,” *Proceedings of the IEEE*, vol. 109, no. 3, pp. 247–278, 2021. ﻿*https://ieeexplore.ieee.org/document/9369420*.
2. D. May and J. Picone, “The ISIP Pattern Recognition Applet,” Institute for Signal and Information Processing, College of Engineering, Mississippi State University, 2002. [Online]. Available: *https://www.isip.piconepress.com/projects/speech/software/demonstrations/applets/util/pattern\_recognition/current/*. [Accessed: 05-Jul-2021].
3. J. Picone, R. Duncan, and J. Hamaker, “Internet-Accessible Speech Recognition Technology,” in O’Reilly Open Source Convention, 2001. ﻿*http://www.isip.piconepress.com/publications/confer ence\_presentations/2001/oscon/software/*.
4. J. Picone, “Internet-Accessible Technology Demonstrations,” *Institute for Signal and Information Processing, College of Engineering, Mississippi State University*, 2000. [Online]. Available: *https://www.isip.piconepress.com/projects/speech/software/demonstrations/*. [Accessed: 05-Jul-2021].
5. R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*. New York City, New York, USA: John Wiley & Sons, Inc, 2001. *https://www.wiley.com/en-us/Pattern+Classification%2C+2nd+ Edition-p-9780471056690*.
6. K. Mallapragada, R. Jin, and A. Jain, “Non-parametric Mixture Models for Clustering BT - Structural, Syntactic, and Statistical Pattern Recognition,” E. R. Hancock, R. C. Wilson, T. Windeatt, I. Ulusoy, and F. Escolano, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 334–343. ﻿*https://link.springer.com/chapter/10.1007/978-3-642-14980-1\_32*.
7. C. Bishop, *Pattern Recognition and Machine Learning*, 2nd ed. New York, New York, USA: Springer, 2011. ﻿*https://www.springer.com/us/book/9780387310732*.
8. F. Pedregosa *et al.*, “Scikit-learn: Machine Learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011. *https://dl.acm.org/doi/10.5555/ 1953048.2078195*. (Also see *https://scikit-learn.org/*.)
9. J. D. Hunter, “Matplotlib: A 2D graphics environment,” *Computing in Science and Engineering*, vol. 9, no. 3, pp. 90–95, 2007. (Also see *https://matplotlib.org/*.)
10. Csv file format
11. Matplotlib
12. [1]

**Acknowledgements**

The development of the Python version of this Java applet was funded entirely from internal funds. The software is available as unencumbered open source software.

**REFERENCES**

Shaffer, J., Hamaker, J., & Picone, J. (1998). Visualization of signal processing concepts. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, *3*, 1853–1856. *https://doi.org/10.1109/ICASSP.1998.681824*.