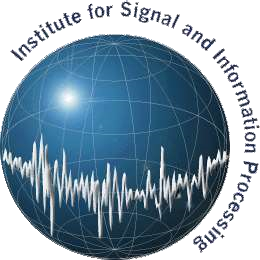
**The ISIP Machine Learning Demonstration (IMLD)**



*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

T

emple University

12

North

1947

th

Street

Philadelphia, Pennsylvania 19122

Tel: 215

-

204

-

484

1

; Fax

: 215

-

204

-

5960

Email

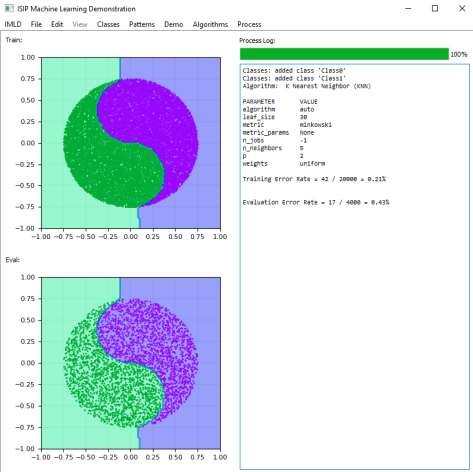
Contact

:

picone

@temple.ed

u



# Abstract

|  |  |
| --- | --- |
| **Team #45** | The ISIP Machine Learning Demo |
| **Team**  **Members** | Thao Cap, Aaron Kreitzer, Matthew Miranda, Dakota Vadimsky |
| **Advisor** | Joseph Picone |
| **Coordinator** | Brian Thomson |
| **Department(s)** | Electrical and Computer Engineering |
| **Project Title** | IMLD: A Python-Based Interactive Machine Learning Demonstration |
| **Abstract** | The ISIP Machine Learning Demo (IMLD) is a tool used to introduce the basics of machine learning (ML) using a highly interactive environment in which users can easily visualize the performance of a number of algorithms. IMLD enhances ML education by providing an environment where students can easily create data and compare algorithm performance across a wide array of algorithm parameters. IMLD is implemented with three different interactions in mind. The first is data generation, where users can import and export comma separated value (.csv) files, draw freehand data, or load preset demonstration datasets. The second is the algorithms and the model API, which allows algorithms to be easily added in with the inclusion of three different functions, namely predict, predict-classification and initialization. Lastly is the graphical user interface (GUI), which communicates with the other two mentioned above and gives the user a streamlined view of how to interact with the application in real-time. |

# Executive Summary

IMLD is an interactive, visual machine learning (ML) and pattern recognition tool developed entirely with open-source libraries in the Python programming language. It can easily be downloaded and run either through and IDE or a command line/terminal in any operating system. Users can create their own classes and draw their own two-dimensional (2D) data points in either single point or Gaussian distributions in both the training and evaluation data input windows. They can also import data in a comma separated value (.csv) format, which is the same format IMLD exports data as when users choose to save their own data. The last option in regards to data creation is the generation of pre-built demonstration datasets that are meant to easily reproduce commonly seen Gaussian distributions. Users can alter the parameters of these distributions to change their location (mean), spread/shape (covariance), and number of points, and then have the distributions plotted to the train or eval windows. Currently users are able to create these distributions with either 2 or 4 classes. Other options include toroidal (donut-shaped) distributions and the Yin-Yang distribution. Once the user has their desired data in the train and eval windows they can choose from a list of algorithms used to classify the data. Algorithms, once selected, will present the user with a list of parameters that they can change to optimize the classification process as they see fit. When they have decided on their algorithms, they may run the classification either all at once or step-by-step if they wish to see what each step of the process entails. These steps will print their results to the process log, which also keeps track of the creation and deletion of classes and the algorithm/parameters chosen by the user. If a step also involves plotting additional objects to the train or eval windows, that will occur as well. The algorithms function differently from one another, but broadly speaking they will all attempt to partition the training window based on class assignment and then calculate the error rate based on how accurate their predictions were. Once the model is created using the training data, it will be applied to the evaluation data and the error rate will be calculated for that dataset as well. Users can then decide to delete the results from each window, delete the data itself, or reset the entire program. By resetting, users can use multiple algorithm parameters to compare error rates and optimize their model on the training or evaluation datasets as desired. The model API has been developed in a way to allow users to easily add their own algorithms once they understand how the program functions, allowing IMLD to grow in its functionality over time.

**Table of Contents**

[Abstract 2](#_Toc31664)

[Executive Summary 3](#_Toc31665)

[1.Problem 6](#_Toc31666)

[1.1. Overall Objectives 6](#_Toc31667)

[1.2. Historical Perspective 6](#_Toc31668)

[1.3. Potential Solutions 8](#_Toc31669)

[1.4. Proposed Solution 9](#_Toc31670)

[1.5. Major Design and Implementation Challenges 10](#_Toc31671)

[1.6. Implications of Project Success 11](#_Toc31672)

[2.Functional Requirements 12](#_Toc31673)

[2.1. Generate Data: 12](#_Toc31674)

[2.2. Provide Demos: 13](#_Toc31675)

[2.3. Apply Algorithms: 13](#_Toc31676)

[2.4. Graphical User Interface (GUI): 13](#_Toc31677)

[2.5. Accurately Classified Data: 14](#_Toc31678)

[2.6. A Note on Testing Metrics: 14](#_Toc31679)

[3.Design Constraints: 15](#_Toc31680)

[3.1. User Interface 15](#_Toc31681)

[3.1.1. User Experience 15](#_Toc31682)

[3.1.2. PyQt 16](#_Toc31683)

[3.2. Programming Language 16](#_Toc31684)

[3.3. Portability 16](#_Toc31685)

[3.3.1. Memory Requirement 16](#_Toc31686)

[3.3.2. Processing Speed 17](#_Toc31687)

[4.Design Approach 17](#_Toc31688)

[4.1. Data 18](#_Toc31689)

[4.1.1. Data I/O 18](#_Toc31690)

[4.1.2. Data Generation 19](#_Toc31691)

[4.2. GUI 24](#_Toc31692)

[4.2.1. Window 24](#_Toc31693)

[4.2.2. Parameters 26](#_Toc31694)

[4.2.3. Events 27](#_Toc31695)

[4.3. Algorithms 27](#_Toc31696)

[4.3.1. Model 27](#_Toc31697)

[4.3.2. PCA 27](#_Toc31698)

[4.3.2.1. Class Independent and Class Dependent 27](#_Toc31699)

[4.3.3. SVM 27](#_Toc31700)

[4.3.4. MLP 28](#_Toc31701)

[4.3.5. LDA 28](#_Toc31702)

[4.3.5.1. Class Independent and Class Dependent 28](#_Toc31703)

[4.3.6. KNN 28](#_Toc31704)

[4.3.7. KMeans 29](#_Toc31705)

[4.3.8. Random Forests 29](#_Toc31706)

[4.4. Python 29](#_Toc31707)

[4.4.1. NumPy 30](#_Toc31708)

[4.4.2. PyQT5 30](#_Toc31709)

[4.4.3. Matplotlib 30](#_Toc31710)

[4.4.4. Pyqtgraph 30](#_Toc31711)

[4.4.5. Scikit-learn/Sklearn 30](#_Toc31712)

[4.4.6. Scipy 30](#_Toc31713)

[4.5. Integration 30](#_Toc31714)

[4.5.1. Testing 30](#_Toc31715)

[4.5.2. Feedback and Surveys 31](#_Toc31716)

[4.5.3. Performance Testing 31](#_Toc31717)

[4.6. Project Management 37](#_Toc31718)

[4.6.1. Jira 37](#_Toc31719)

[4.6.2. Gitlab 38](#_Toc31720)

[5.Testing 40](#_Toc31721)

[5.1. Test Methods and Results 40](#_Toc31722)

[6.Cost and Schedule 42](#_Toc31723)

[7.Summary and Future Work 43](#_Toc31724)

[References 45](#_Toc31725)

# 1. Problem

# 1.1. Overall Objectives

Many students find the prospect of being taught machine learning algorithms too intimidating due to the automated, opaque, and somewhat unintuitive nature of the concept. Given the rise of machine learning technologies in a wide range of industries from banking to manufacturing, to transportation [1], it is important that students be given an efficient method of learning these topics that advances beyond the theoretical. The Institute of Signal and Information Processing (ISIP) Machine Learning Demo (IMLD) aims to make this process easier for students and teachers alike by providing a visual, hands-on application which allows users to be guided through customizable machine learning scenarios step by step. This tool can help students in technologically constrained college classrooms because it is open source, has a low memory cost and displays a visualization at every step. This is important because students have been shown to learn higher complexity concepts more effectively when visualization is involved with learning [2]. There are many other free open-source tools that are similar to our application; however, one feature that makes our application more unique and useful is that it allows users to import or generate their own niche datasets. Compared to alternative online tools, the feature of importing and exporting datasets is not available. It is important for students to be able to classify very specific or niche datasets to experience the widest possible array of uses for different Machine Learning algorithms. Online tools also necessarily require internet access to use, while IMLD, once downloaded, can be used anywhere at any time.

# 1.2. Historical Perspective

As modern computing problems have continued to increase in complexity, increased attention has been given to the field of machine learning. According to IBM, “Machine learning is a branch of [artificial intelligence (AI)](https://www.ibm.com/cloud/learn/what-is-artificial-intelligence) and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy” [3]. As such, these algorithms will, over enough time and with ever expanding pools of data to read from, be capable of classification and analysis with much greater efficiency and accuracy than that of comparable non-ML processes. Machine Learning is used to bring us a greater level of convenience, which makes it important for scientists and engineers to learn to catch up with the future. While this design method is undoubtedly the most efficient at processing and classifying large data sets, the algorithms behind machine learning are notoriously hard to understand. As machine learning increases in popularity and usefulness, it will become crucial to teach it in a way that is visually and practically intuitive. Machine learning roots come from applying statistical methods that are used to find correlation between features in a dataset. This means that to understand the concepts one needs a firm grasp on the mathematical formulas for the various algorithms used. While visualizations are used when teaching machine learning they are restrained to the instructor having made the visualization and the student simply viewing the visual rather than actively interacting with the visual. According to Banerjee, an experiment was conducted where two different groups were constructed. One where students were able to “respond” to a visual and another where students could only “view” the visual. The second group is comparable to how students are currently taught machine learning by simply attending a lecture-based class. The first group was able to ‘respond’ to the visuals provided through an application meant to teach recursion and the students would need to accurately predict the outcome of the application and would repeat until getting the desired result. It was shown that the students in the first group outperformed the second group slightly in medium complexity concepts and significantly outperformed them in highly complex concepts. IMLD aims to supplement students’ learning with visualizations to get similar results [2]. This application began as an applet within the Java framework, dating back to the late 1990’s when Java was considered to be the future of computing [4]. During this time there was also a heavy emphasis placed on web-based educational tools [5]. While the Java version was and still is operational, it became too difficult to keep it up-to-date, and other languages like Python and C++ surpassed Java in terms of machine learning functionality and processing speed. Java has found itself in a position where it is not the best at any particular area in the field of machine learning. Java is faster than Python in most cases, but it is slower than C++ and while Java is not as complex as C++, Python is easier in terms of syntax and better suited for quick prototyping. Python’s strengths have made it popular with other research communities and has consequently produced a wide array of different libraries that can be leveraged for machine learning. The program in Python runs at a higher speed with less memory than Java, and it provides much more easily installed modules than Java. IMLD makes machine learning visualiable to students. They can use IMLD to test different approaches with different algorithms to optimize their model, to generate different dataset with different distributions, and to analyze the data’s behavior by receiving calculated results such as the error percentage after training and evaluating, the mean, and the covariance matrix.

# 1.3. Potential Solutions

In order for IMLD to remain usable on modern computers, improvements on the original Java application were a must. A few potential solutions were discussed, although the most feasible potential solutions were updating the Java applet or rebuilding the applet in a different language like C++ or Python. Updating the IMLD applet was examined first, as we have access to the original source code from 2013. Programming has taken some major strides since the original release of IMLD, thus some major refactoring of code would be one of the first things that needed to be done, for thousands of lines of code. This would be a monumental task, as we would need to spend hours of time reading and comprehending the code in order to understand how to efficiently refactor it. If you haven’t spent time trying to read code someone else has written, it is a very challenging task, often taking much more time than writing code. Further, since the release of the Java IMLD, releases of machine learning libraries have been targeted at languages other than Java. The second potential solution that was considered was rebuilding the applet within C++, a very powerful language that is used by engineers within industry today. Even though C++ is very powerful, there are very few machine learning libraries, as many data scientists stray away from C++ . They mostly choose other programming languages due the rigidness of how elements interact with each other, the lack of library functions, and the difficulty when debugging errors. The final potential solution discussed was rebuilding the IMLD applet with Python. Python is currently the most widely used within the machine learning field, with the most libraries and frameworks are made specifically for Python [6].

Many of the other potential solutions outside the decision of programming language had been decided long before our team was assembled. ISIP had decided early on in their design process that IMLD was to be used as an illustrative, interactive educational tool whose main learning objectives were to teach students through prolonged use the many different applications of data classification through ML, both theoretical and practical, as well as the crucial strengths and weaknesses of different ML algorithms. It is important to our team that, along with clearly illustrating each step of the classification process, we keep IMLD as open-source as possible, and therefore we make our source code as understandable as possible as well. All these factors were considered not only when choosing the final programming language, but also when building the program architecture and at every further step in the ongoing design process.

# 1.4. Proposed Solution

Programming languages are often notoriously difficult to compare to determine which is the better language, engineers have learned to compare the specific goals of the application to determine which language is best for the specific task. Python was designed from the start with machine learning in mind. The simple, easy to understand syntax allows for an easier understanding of written code, and in turn writing code becomes a quicker process. In addition, Python has an ample amount of machine learning libraries that allow for powerful implementations. It can be run on several platforms such as Linux, MacOs, and Windows, as powerful intelligent development environments (IDE) exist for all these languages. The IDE allows for error detection and proposed correction, as well as more powerful tools that allow the user to save time while both writing and debugging code. Even though Python is a simple language to learn, it holds an extensive selection of libraries and frameworks used for machine learning, including Numpy/ Scipy for high data computing performance and Tensorflow/Sklearn for machine learning models. For these reasons we choose to rebuild IMLD and improve upon it within Python, rather than other modern programming languages. Computer science and machine learning have a reputation for how abstract the process can quickly become, which can lead students to not actively learn the concepts and instead memorize them. One way to promote active learning is by increasing engagement levels. Engagement levels increase across six different areas: No Viewing, Viewing, Responding, Changing, Constructing, and Presenting. The three main areas of engagement that our project will focus on are "Viewing", watching visualizations, "Responding", prediction with visualization, and "Changing" which is the manipulation of the visualization and seeing the effect [2]. Topic complexity, in addition

to engagement levels, influence the outcome of active learning. Banerjee states that there was no increase in group performance on simple concepts. However, there was a significant increase in performance on high complexity concepts in the engagement levels for "Viewing", "Changing", and "Responding". When contrasting two groups, one who focused on Viewing and the other who implemented "Responding" and "Changing", there was a significant difference in engagement levels. The "Responding" group was shown to have made more cognitive progress in terms of the rate of problem-solving [2]. Our goal with IMLD is to focus on the engagement levels mentioned above by providing students a visual of an otherwise abstract concept and allowing them to interact with the concept they are learning in step-by-step process with visuals.

# 1.5. Major Design and Implementation Challenges

Working on a purely software-based project provides a unique set of challenges for our team. Teams whose projects will culminate in the development of a physical design, tool, or product find themselves working at relatively similar times on relatively similar problems. With software engineering, our team can work remotely at any time available without the need to be in contact with each other. This, however, creates an organizational challenge which our team is solving with online free software such as Github/Gitlab and Jira. With these programs, we can better manage our individual and group workloads, create and observe trouble tickets, and continue development of individual project branches before merging them into testable versions.

As we continue to develop our program and add new features to IMLD, another challenge that must be continually addressed is testing and troubleshooting. IMLD is designed to be user-focused, and so must account for as many user interactions as possible. This means that, at every step along the development process, we must create and provide use cases with the goal of pushing our design to its limits. To this end, we have partnered with local high school volunteers to test our design at regular intervals and provide feedback to us through official reports and screen recordings. Furthermore, IMLD is being integrated into the College of Engineering’s Intro to Machine Learning course. Regular surveys are being provided to students to rate their understanding of the course material before and after introduction to IMLD and to what degree they feel IMLD is assisting in their educational quality. The team will use this feedback, as well as collected assignment grades compared to past semesters to better evaluate the strengths and weaknesses of IMLD and adjust our focus areas accordingly. We apply the ISIP Code Writing Standard when we refactor and harden our code. This includes removing hard codes, building a parameter file used throughout the scripts, cleaning up each line of code to less than 80 characters per line, and adding references and descriptions for each method. The language and framework that we choose must provide a clear and well-documented access and support for both backend and frontend.

# 1.6. Implications of Project Success

When this project succeeds, we will be left with a machine learning visualization tool that would help students learn high complexity concepts regardless of their technological environment. This satisfies the quality education goal of the UN by offering a flexible learning tool that can be used on a wide range of technological platforms. Since this tool can be used outside of the classroom, potentially without the need for a structured teaching environment, its use can aid in the further reduction of the wage and education gap experienced most acutely in third-world countries. As stated in the historical perspective, problems in our societies are increasingly being solved using analysis on large and complex data sets. Thus, being able to teach machine learning to a wider range of people satisfies the innovative need set for by the United Nations Development Programme [7].

Specifically, this program will be used in Temple University’s graduate level machine learning course, where the visualization of student-created datasets is paramount to course completion. Completion of IMLD and successful implementation into the course curriculum will undoubtedly improve comprehension and retention of vital machine learning techniques. Software complexity is a big challenge that we are facing during the development. Because IMLD involves user-interface, the hierarchy of modules are heavy and dependent. To solve this, we have built a very strong structure of modules that each module handles a specific function of the tool. We divide it into 3 separate groups of modules: algorithm, data, and GUI. The algorithm group includes a model that handles algorithms used in IMLD; this model will train, predict, and calculate mean, covariance, and percentage error. Data handles the data generations such as Two- and Four-Gaussians, Ellipse, Toroidal, and Yin-Yang, and data IO is used to read and write user’s data. GUI is used to handle the interactions from the user-interface graphically.

# 2. Functional Requirements

*Table 1: Functional Requirements*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Priority** | **Requirement** | **Metric** | **Target Value** | **Justification** |
| Non-  Negotiable | Load/Save/Draw large amounts of data | # of Possible Data  Points | >= 1 Million Data  Points | Advisor |
| Non-  Negotiable | Provide demos | Complete  Functionality | Pass/Fail | Advisor |
| Non-  Negotiable | Apply Algorithms | test cases | Pass/Fail | Advisor |
| Negotiable | GUI | test cases | Pass/Fail | Advisor |
| Non-  Negotiable | Data calculated correctly | test cases | 0.1% margin error | Self-  Imposed |

# 2.1. Generate Data:

IMLD is an educational tool designed to be used primarily in the classroom. As such, students will be able to draw their own data points within the training and evaluation windows, and/or load previously created datasets directly into the program for analysis and classification. The data that IMLD is designed to handle is two dimensional, consisting of scatter plots for each class in the dataset. Data that is loaded directly into the program will need to be in this format (X and Y coordinates for each point), while data drawn in the input windows will be automatically saved in this format. These datasets will be diverse in their scope and size, and so IMLD must be able to handle potentially very large datasets. The IMLD team chooses 1 million points to be the target value and IMLD should be able to generate and handle up to 1 million data points. This number is chosen because it’s not too large and not too small for a typical student’s use in an educational setting. For example, data that represents different breeds of cat and has two features such as the length and width of the cat could be generated and visualized with IMLD. Users just need to drag their cursor to the window and data points will be generated. The features of the data (length and width) will be corresponding with X, and Y coordinates.

# 2.2. Provide Demos:

As part of the educational aspect of IMLD, multiple demonstration datasets or demos are provided in which parameterized sets of data can be automatically drawn in the data windows and classified by the algorithms. These demos allow users to create a variety of consistent, repeatable patterns of data to be displayed to the training or evaluation windows at the push of a button. While some of these demos represent examples of possible real-world datasets, like the 2- and 4-Gaussian and 2-Ellipse demos that can represent natural clusters of random data, others are included to show the extent of complex data machine-learning algorithms are capable of classifying, such as the Toroidal and Yin-Yang demos. These demos must each be fully functional and user-customizable within the given sets of parameters, including class-specific means and covariances. Complete functionality in this category means that each demo dataset appears in full where the user intends based on their parameters, at up to and including the maximum number of data points specified in the “Generate Data” section.

# 2.3. Apply Algorithms:

IMLD does not rely on a single algorithm, but rather on a multitude of algorithms, each of which functioning uniquely from the others. Currently IMLD includes 7 algorithms, with 2 of them, Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA), supporting Class Independent (CI) and Class Dependent (CD) variations. Since students will be learning about potentially all the algorithms available in the tool, every algorithm will need to be tested to ensure full functionality. In this respect, full functionality refers to the algorithms being properly implemented and successfully handling the data fed into them. It does not refer to the accuracy of their classifications, which is a separate functional requirement. IMLD needs to successfully deliver and apply accurate algorithms’ behavior on the dataset

# 2.4. Graphical User Interface (GUI):

IMLD is designed to be entirely user-directed from data input to analysis completion. Every step must therefore be neatly presented to the user through a GUI. This GUI needs to provide clear and unambiguous directions to the user without being overly invasive. It also needs to catch any errors the user makes and warn the user of said errors before allowing them to continue. While the layout and “usability” of the GUI is discussed further in the design constraints, the functional requirement of the GUI includes these warnings, preventing the user from taking action that would result in inaccurate classifications, loss of data, or program termination (crashing). It is hard to analytically measure that our GUI is displayed neatly, we choose to use the metric of intuitiveness during the second user survey. This feedback allows for honest user experiences while using the GUI for the first time and learning where all the options are within the application, rather than the developers perceiving how the application will be used by end users for the first time.

# 2.5. Accurately Classified Data:

As IMLD is an educational machine-learning tool, its different algorithms will classify data sets at varying degrees of accuracy. This is to be expected, however a minimum accuracy for all algorithms must be reached in order to establish and reinforce the legitimacy of the algorithms and the tool overall. Each algorithm will produce an Error Rate for the data, to test if the algorithm works properly, the Error Rate will be compared with independent code’s result. If the margin of error is less than 0.1, we consider the algorithm works properly.

# 2.6. A Note on Testing Metrics:

The testing metrics for many of the functional requirements are labeled as “test cases”. During the development process of IMLD, the team is constantly creating and updating a comprehensive list of possible use test cases. These test cases will be closely followed by the testers, who will record and log their test sessions and report to the team any unsatisfactory test values. This feedback will then be used to update IMLD, and this iterative process will continue until IMLD reaches its target values for all functional requirements. Depending on the update of the release, testers have different sets of test cases. Mainly, the test cases are targeting the codes that are implemented for the releases. For example, in v1.6.0, IMLD is updated with a more flexible CSV format and with PCA and LDA algorithms. Test cases for that release will involve loading different CSV files with different formats, applying PCA and LDA. IMLD should be able to handle tested CSV files and produces the Error of Prediction that has 0.1 margin error compared to other sources.

Now that IMLD has also been released for use in the classroom, we have adopted many of the course’s homework assignments as additional test cases, and have worked diligently with students to fix any issues that may hinder them or prevent completion of said assignments.

# 3. Design Constraints:

*Table 2: Design Constraints*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Priority** | **Requirement** | **Metric** | **Target Value** | **Justification** |
| Negotiable | User Interface | UI Usability, ease of access, learning curve | Average >= 3.5 (1-5 scale) | Advisor |
| Non-  Negotiable | User Interface | Dependencies | Version | Advisor |
| Non-  Negotiable | Programming  Language | Python | Pass/Fail | Advisor |
| Negotiable | Portability | Memory Requirement | < 15MB | Self-Imposed |
| Negotiable | Portability | Processor Requirement | < 6 GB | Self-Imposed |

# 3.1. User Interface

## 3.1.1. User Experience

IMLD’s release and subsequent introduction to Temple’s Intro to Machine Learning class has allowed us to gather more qualitative information in the form of survey data. At the beginning of the semester we had students in the class fill out a survey detailing their prior knowledge of various topics listed in the course syllabus, as well as their comfortability and experience with Python. Throughout the course of the semester, we are administering followup surveys asking similar questions, with the desired goal to see a general increase in understanding among students and an indication that IMLD is chiefly responsible for it. All surveys are on a 1-5 scale and will include sections for students to leave comments and suggestions to help us improve IMLD. Our goal is for the averages of these survey question scores to increase; for those questions whose scores were previously under 3.5 we are looking to get them above this threshold, for all other questions we are hoping for an increase of at least 0.5.

## 3.1.2. PyQt

PyQt library is often being updated. IMLD needs to be up to date with the most current version of PyQt and other Python libraries. The GUI has to be stable and able to run on the newest PyQt without significant changes. Stability is defined on whether the app has any bugs meaning that IMLD should be able to achieve its intended outcome without program termination.

# 3.2. Programming Language

IMLD is written entirely in the Python programming language. While Python is a relatively simple language to read and understand, a complex program such as ours can be difficult for new users to parse out. To better accommodate this, the team has been adhering to general ISIP coding practices. This includes creating files for default values so as to avoid “hard-coding”, limiting all code to 80 characters per line, and reformatting code at periodic intervals to maximize readability. This last requirement is judged by the advisor during weekly or bi-weekly code review sessions.

# 3.3. Portability

## 3.3.1. Memory Requirement

IMLD needs to be able to run on the widest range of student systems, preferably without the students needing to specifically make dedicated space for it on their hard drives. To this end, we have consciously tried to limit the size of IMLD and its supporting files, including sample datasets that we release with each updated version. Our goal is to keep the total size of IMLD below 15MB, ensuring that it can be easily downloaded by any student who should need it. This value is somewhat arbitrary, but it provides us a good benchmark for ensuring that our code is not bloated, and motivates us to more strictly adhere to the ISIP coding practices detailed above.

## 3.3.2. Processing Speed

IMLD, like all machine learning programs, provides more accurate classifications when provided with larger datasets. However, we must keep in mind that when classifying large datasets, we can run the risk of overflowing a computer’s memory, especially when the program is running on student computers with small amounts of available memory. Whenever the team releases large updates for IMLD, we test the program’s memory usage on larger and larger datasets to analyze exactly how much memory each action uses. Our goal is to keep this under 6GB, as we know that most students will have computers with at least 8GB of RAM and popular operating systems like Windows and MacOS use at most 2GB [8]. Currently, our latest version of IMLD can load in a dataset consisting of 10 million twodimensional (2D) data points and complete the classification process while using roughly 5.5GB of RAM. We do not expect students to be regularly classifying datasets with more than 1 million data points, let alone 10 million, so we are satisfied with these results.

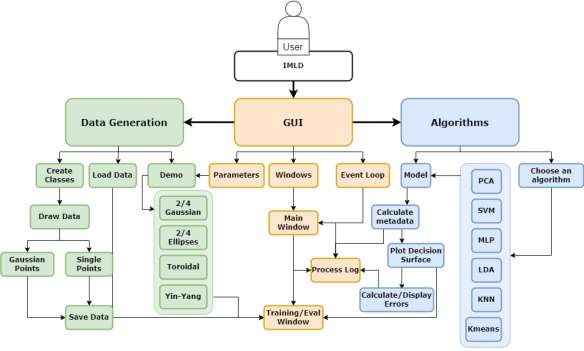
# 4. Design Approach

IMLD operates within the Python environment. Using existing libraries as a supporting framework, we built up separate modules to execute each step of the machine learning process, following the chart pictured below:

*Figure*

*1*

*Architecture Flowchart*

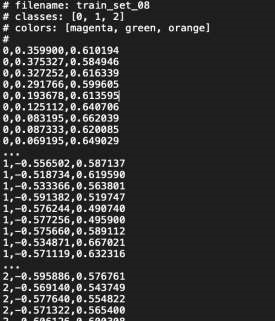


# 4.1. Data

IMLD supports three methods of data creation: loading data from comma separated value (.csv) files, generating data via demos, and manually drawing data points within the training or evaluations windows. The modules within the Data branch of the applet handle the first two methods.

## 4.1.1. Data I/O

The Data I/O class contains within its methods to load .csv files from local storage and read the data contained therein, or write data to a file and save it to local storage. IMLD formats the written data into a .csv file for future reading according to the following template:

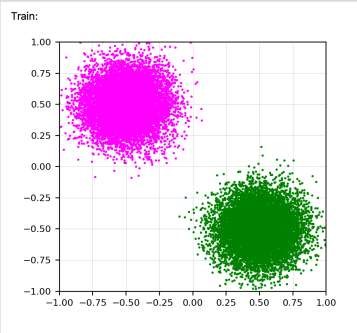


#### Figure 2 Example of .csv file formatting

Allowing users to import their own data was a necessary component of IMLD, as it gave users the ability to witness how different machine-learning algorithms would evaluate real-world data.

## 4.1.2. Data Generation

IMLD supports the generation of robust data sets according to a variety of preprogrammed demos. These demos include datasets likely to be encountered in the real world, like two- or four-Gaussian plots, overlapping Gaussian plots, two- and four-ellipses plots, as well as toroidal (donut-shaped) and yin-yang plots, which although unrealistic nevertheless illustrate the machine-learning process for complex datasets.

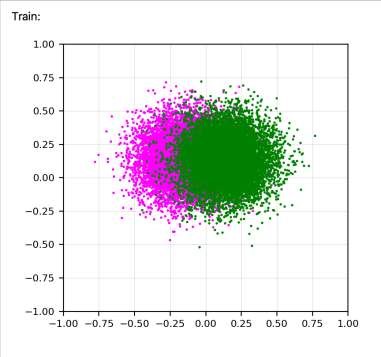
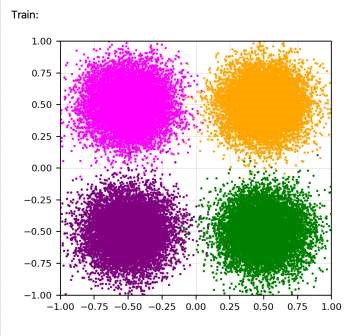


*Figure 3 Two Gaussian plots*

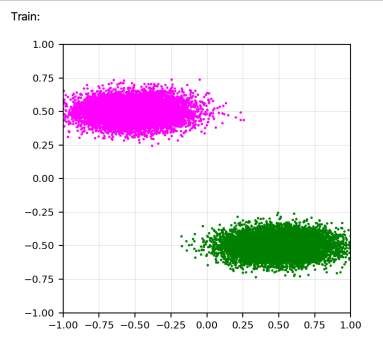
*Figure*

*4*

*Four Gaussian*



#### Figure 5 Overlapping Gaussian

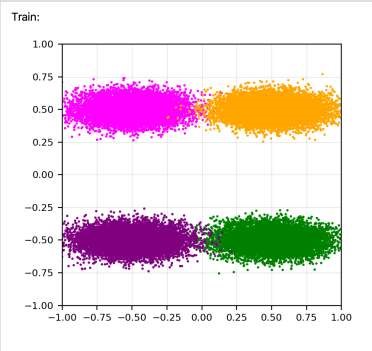


#### Figure 6 Two Ellipse

*Figure*

*7*

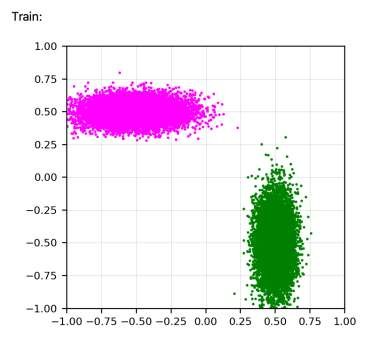
*Four Ellipse*



*Figure*

*8*

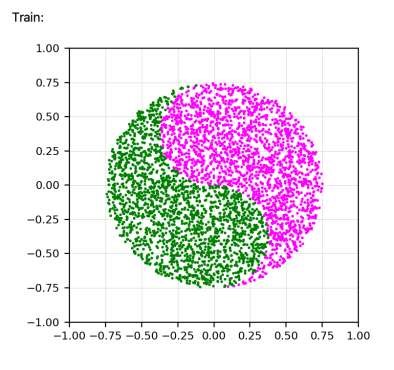
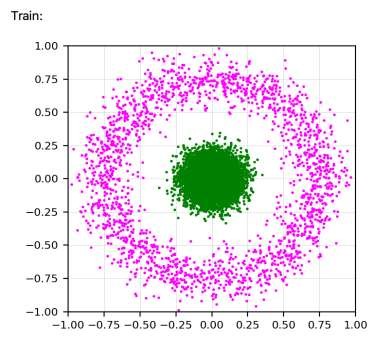
*Rotated Ellipse*



*Figure*

*9*

*Toroidal*



#### Figure 10 Ying-Yang

# 4.2. GUI

The GUI module contains all aspects of the IMLD graphical user interface (gui). Whenever the user interacts with the applet, or whenever the applet visually displays information to the user, the methods within GUI are being utilized.

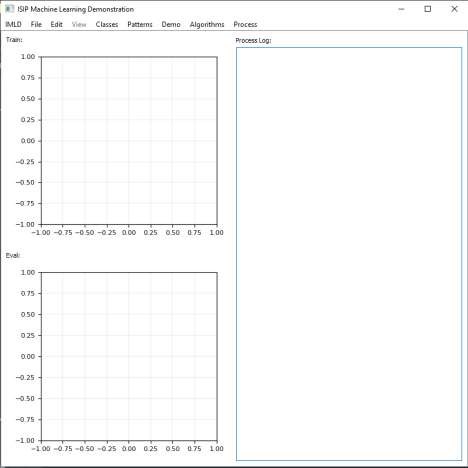
## 4.2.1. Window

The Window module contains the classes for each window displayed when the IMLD applet is opened. These are the main window, training and evaluation windows, and the process log, all pictured below:

*Figure*

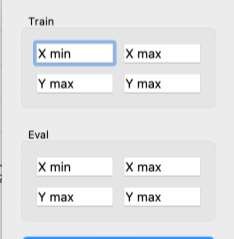
*11*

*IMLD window at startup*



The main window class, as well as holding the widgets for the other three classes, also supports the functionality of the main window ribbon and all the buttons found therein. It is from this ribbon that users will be able to load and save data, add, and delete classes, draw points in the training/evaluation windows, choose different demo datasets to generate, select which machine-learning algorithm to use, and finally run the classification process.

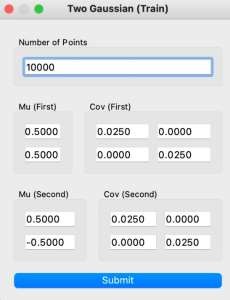
The training and evaluation windows hold all training and evaluation data, respectively, that the user wishes to have classified. When loading in datasets or generating demo datasets, the user can choose to automatically set the limits of these windows to better visualize their data. They can also manually change the ranges of the windows individually using the Edit option on the main window ribbon:



#### Figure 12 Set Ranges Popup Window

## 4.2.2. Parameters

The parameter class holds different widgets that receive the user's parameter input and apply those inputs to the Data Generation class. Parameter widgets allow the user to configure values such as the mean and covariance for Gaussian and Ellipse demos. After the user submits the configured parameters, the Events class will apply them to Data Generation to produce corresponding datasets.



#### Figure 13 Example Parameter Box for Two Gaussian

## 4.2.3. Events

The event class is a backbone of our program. The class controls asynchronous tasks and callbacks between user interface and other module handlers. The event class is responsible for all the methods that comes from the input of the GUI. This ranges from adding/deleting classes, opening the correct parameter box for data generation, calling the algorithm that user selects and running the model driver either in one run or step by step depending on the user input.

# 4.3. Algorithms

## 4.3.1. Model

The model class is a driver program that acts as the communicator between the GUI and Algorithms classes. The model accepts the user inputted data and the user selected algorithm. In addition, the model class can also accept new algorithms submitted by the user granted that they follow the same structure as the provided algorithms.

## 4.3.2. PCA

Principal Component Analysis (PCA) is a dimensionality reduction technique that is used to find patterns in a dataset that will then be transformed into a more digestible data set. The PCA technique starts by standardizing the data so that all of the information falls within a controlled range. The covariance matrix is then calculated to account for any bias or repeated information in the data. Lastly the eigenvectors and eigenvalues are sorted in descending order where the eigenvectors with the largest eigenvalues are chosen as the principal components.

### 4.3.2.1. Class Independent and Class Dependent

The PCA feature offers two available modes, class dependent and class independent. The difference between the two is that class dependent generate multiple covariances for each data set while class independent has only one.

## 4.3.3. SVM

Support Vector Machine (SVM) is a supervised machine learning method that acts as a discriminative classifier where a hyperplane separates the data into different classes.

## 4.3.4. MLP

Multilayer Perceptron (MLP) is a deep learning algorithm that builds off the perceptron algorithm. The perceptron is a single neuron whose purpose is to classify different classes by creating a hyperplane based on the weights and biases found in the data. This is only for one layer, MLP builds off this for multiple layers. There are three different types of layers consisting of the input layer, hidden layer(s), and the output layer. There can be multiple hidden layers depending on the complexity of the model. There are three main steps to train the model, the first is forward passing which is how the outputs of each layer communicate with the inputs of the next layer. This is done by multiplying the weights and adding bias at each layer. The next step is calculating the loss or error found between the predicted output and the expected output. After we calculate the loss, the model then backpropagates using gradient flow and updates the weights of the model.

## 4.3.5. LDA

Linear Discriminant Analysis (LDA) is a dimensionality reduction technique and a multiclass classification model. The goal of this method is to find linear combinations of input variables to find both the maximum distance for samples between class means and the minimum distance between the samples for each class.

It’s similar to PCA in that both should standardize the data and follow up with matrix factorization.

### 4.3.5.1. Class Independent and Class Dependent

LDA, like PCA, offers two different modes, class dependent and independent. The difference between the two is that the global mean is calculated for each data set while class independent takes only one global mean for all datasets.

## 4.3.6. KNN

K-Nearest Neighbors is a supervised classification learning algorithm that determines the class of a datapoint based on the proximity and classification of the neighbors around it. There is no model for KNN since it relies on the whole dataset when classifying. The first step of KNN is to rescale the data and store all of the classifications of the training dataset. Then the Euclidean distance for each new input sample is calculated for all of the training set samples. Lastly, the distances are sorted in descending order where only the first K distances are used for classifying the test sample.

## 4.3.7. KMeans

KMeans is an unsupervised clustering algorithm which categorizes samples based on similarity. The K symbolizes the predetermined number of clusters used for the model. The algorithm is divided into four different steps, the first is randomly creating the means of the clusters that will be used. The Euclidean distance for each point to the random centroid centers are computed and each data point is allocated to the nearest cluster. After, update the mean of the cluster to the center of the sample points for each cluster. This process repeats until the clusters stop moving.

## 4.3.8. Random Forests

Random Forests is a supervised learning algorithm constructed from multiple decision trees. Decision trees are simple models that divide the data into separate categories based on simple constraints and continue separating the data until it reaches a result. The problem with this model is that it is prone to overfitting and bias. Overfitting can be lower through an ensemble technique called bagging, where multiple models are run with randomly sampled data. The performance is averaged for best results. Both methods combine and result in random forests where feature randomness is leveraged, resulting in low correlation among decision trees. Users will be able to choose the number of trees and the depth of the trees.

# 4.4. Python

As discussed in the Design Constraints, we are using Python as the platform for IMLD. We are using Python in large part because of its plethora of libraries that can make up the foundation of our applet.

## 4.4.1. NumPy

Numpy is used for performing various mathematical operations on arrays. We mainly use Numpy to store our X and Y coordinates of a dataset because of its powerful matrix data structure.

## 4.4.2. PyQT5

PyQT5 provides the Qt GUI framework which is used to develop the user interface.

## 4.4.3. Matplotlib

Matplotlib is used for visualizing data and graphical plotting. We use Matplotlib to create a canvas for users to create datasets. The module provides similar alternatives with MATLAB

## 4.4.4. Pyqtgraph

Pyqtgraph serves a similar purpose as PyQT5. It is used for developing the user interface.

## 4.4.5. Scikit-learn/Sklearn

Scikit-learn/Sklearn is a robust library used for machine learning. We import different algorithms using this module

## 4.4.6. Scipy

Scipy is used for scientific computing and technical computing. For example, we use this module to calculate a distance between points, or to generate the multivariate normal gaussian distribution.

# 4.5. Integration

## 4.5.1. Testing

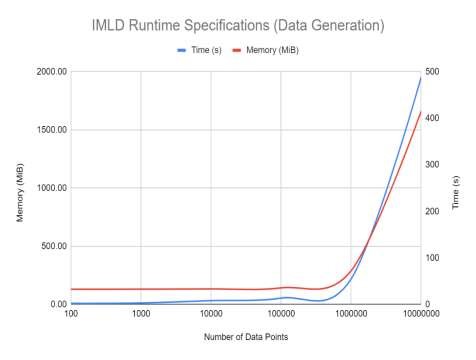
While members of the IMLD team perform individual testing during each sprint and as part of each merge request, large-scale testing of the entire applet is left up to a separate team of testers. These testers are given updated versions of the applet on a set schedule and are asked to perform rigorous testing based on a script of use cases provided to them by the IMLD team. During early testing periods these use cases were basic, meant to acclimate the testers to the applet’s various functions and its GUI. As newer versions were released these use cases became more sophisticated, encouraging the testers to attempt to break the applet at every step of the process.

## 4.5.2. Feedback and Surveys

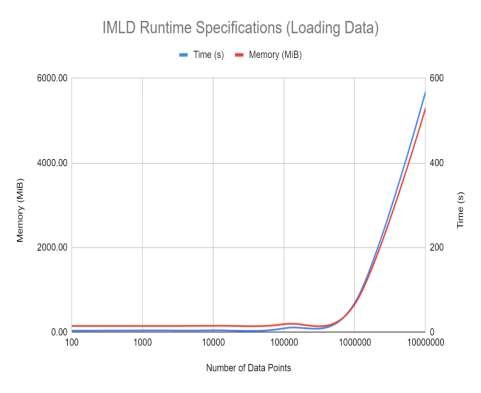
Included with the use cases scripts is a feedback form where testers can record any bugs or crashes, they encounter, noting down their exact actions and attaching screen recordings for recreation purposes. They are also asked to answer a questionnaire, rating the applet in various areas, and providing suggestions on how to improve those areas, or add on functions that they feel would make the user experience better.

## 4.5.3. Performance Testing

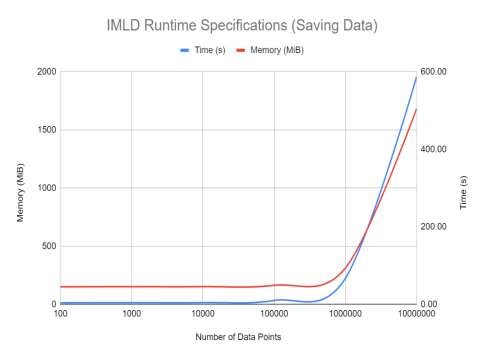
Part of the integration process involves testing the applet at various stages of development to evaluate its performance in terms of required time and memory usage. To this end, we recreated typical IMLD user actions while running a memory profiler with datasets at factors of 10 until the applet failed. We then repeated these tests ten times for each number of data points and recorded the averages. The results for three common activities are shown below:



#### Figure 14 IMLD Memory Usage and Duration for Data Generation and Classification



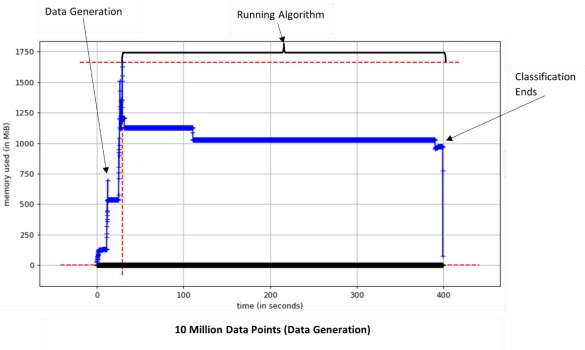
#### Figure 15 IMLD Memory Usage and Duration Data for Data Loading and Classification



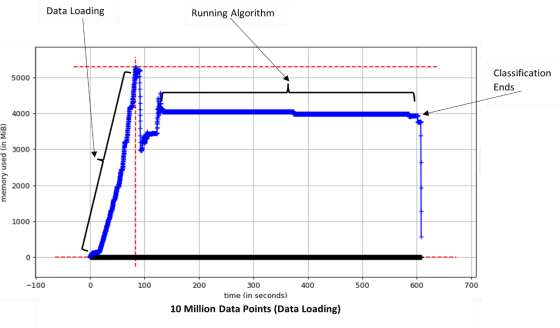
*Figure 16 IMLD Memory Usage and Duration for Data Generation, Saving and Classification* The relationship between memory usage and number of data points is linear, as is shown in the charts above. The same is true for the activity duration, and we can see that these two relationships are very closely related, which is expected. We also noticed during testing that loading data into the applet takes significantly more memory than the other two operations, while the time difference is relatively small.

We have attributed this anomaly to the computer’s difference in efficiency between its read and write commands.

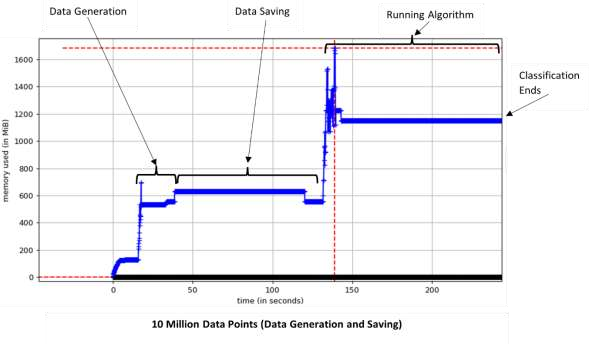
Specifically for datasets of 10 million data points, we can see below the memory usage and duration of specific operations within the classification processes. The specific operation runtimes were captured by running a stopwatch between the time when an operation started (when its button was pressed in the program), and when the operation ended (its results were printed to the GUI).



*Figure 17 Operation Specific Memory Usage and Duration for Data Generation*



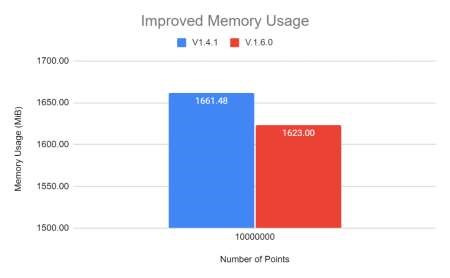
#### Figure 18 Operation Specific Memory Usage and Duration for Data Loading



*Figure 19 Operation Specific Memory Usage and Duration for Data Generation and Saving* Data generation for 10 million data points takes roughly 25 seconds, while the algorithm duration hovers around 400 seconds. The “data generation and saving” graph is shortened to its first 250 seconds in order to better show the detail in the data generation/saving portion, but as we can see in the previous section, this process still takes 600 seconds in total. We can also see that the loading operation is slightly faster than the saving operation. This reinforces our theory regarding writing and reading operations in the CPU.

Through out iterative testing process the IMLD team has been able to reduce both the memory usage and process time of our more complicated operations. In the charts below, you can see that we have reduced our memory usage by nearly 2%, while also halving the process duration for two of our most complex normal tasks.

We see similar results for a wide range of operations and dataset sizes as well.



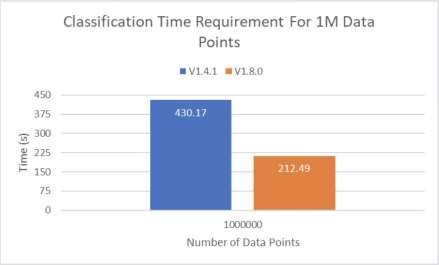
#### Figure 20: Reduced Memory Usage Between v1.4.0 and v1.6.0 for 10M Data Point Classification

*Figure*

*21*

*:*

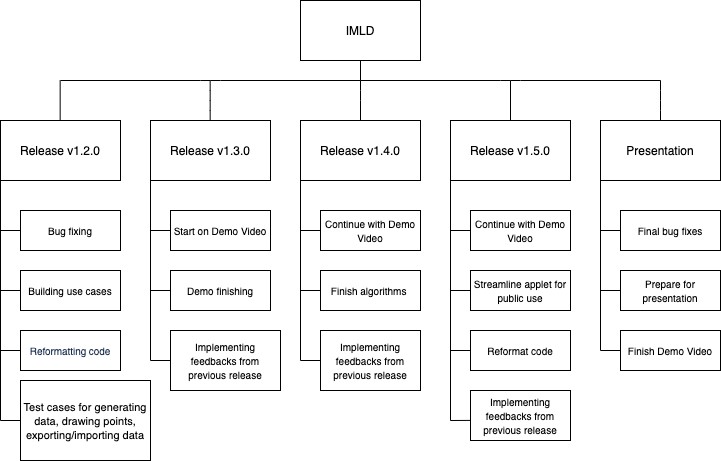
*Reduced Time Requirement*



# 4.6. Project Management

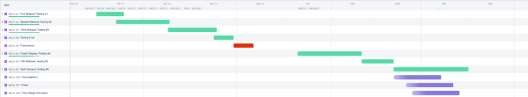
## 4.6.1. Jira

The IMLD team has, during the design process, been using the organizational online platform Jira to assign and track working tickets. Each week is associated with a corresponding sprint within Jira. At the beginning of each sprint tickets are assigned to each team member to be completed during that sprint. Team members can view all tickets on the board, which displays the current sprint. A roadmap is formed by grouping tickets within different epics. These epics contain details about the overall goals of the desired period of time. Tickets are created either from feedback forms gathered from testers, or from the roadmap of test versions displayed below:



#### Figure 22 IMLD Test Release Version Roadmap

As tickets are worked on can be placed into categories based on their current status (To do, in progress, completed, or blocked). Time spent working on the ticket can be logged to the ticket. In addition comments can be posted within tickets to add to the detail of work completed. If necessary tickets that were not completed can be migrated from past sprints to present or future sprints. A roadmap of sprints for the duration of the IMLD project can be seen below, with the 7 green bars and 1 red bar representing release versions and the time it took to complete them:



#### Figure 23: Jira Roadmap Release Schedule

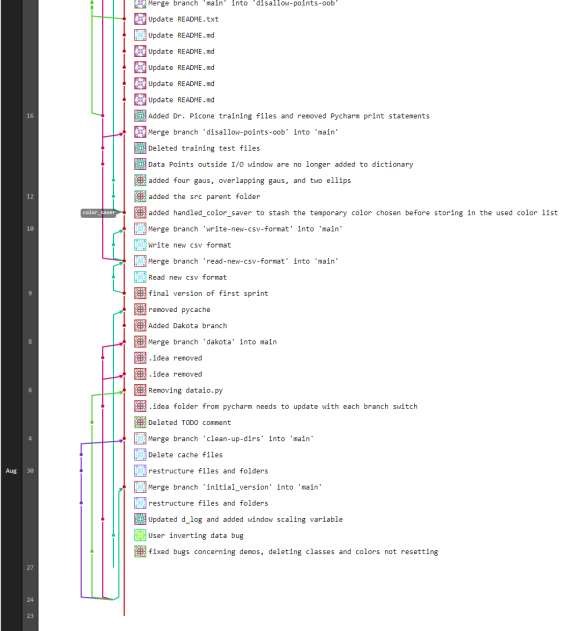
## 4.6.2. Gitlab

All IMLD files are stored in a shared, protected repository on Gitlab, from which team members create branches during each sprint to work on their tickets. When a ticket is completed, it is pushed from that member’s local repository back to Gitlab where another team member can test it before it is merged into main. Tickets are created in such a way so as to avoid two team members performing work on the same modules or similar cross-module methods. This is done to prevent one member’s work during any particular sprint superseding or making obsolete another member’s work in that same sprint. Once a finished ticket’s branch integrity is verified by two team members, it is merged with the main branch and tested again. When these tests are concluded, other members pull the main branch onto their working branches in order to remain up-to-date. This is especially helpful when particularly complicated tickets require multiple sprints to complete. Pictured below is a section from the IMLD Gitlab graph, showing this design process over a period of time:

*Figure*

*24*

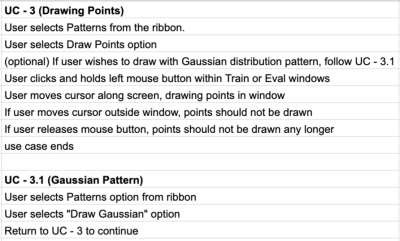
*Gitlab Branches Graph*



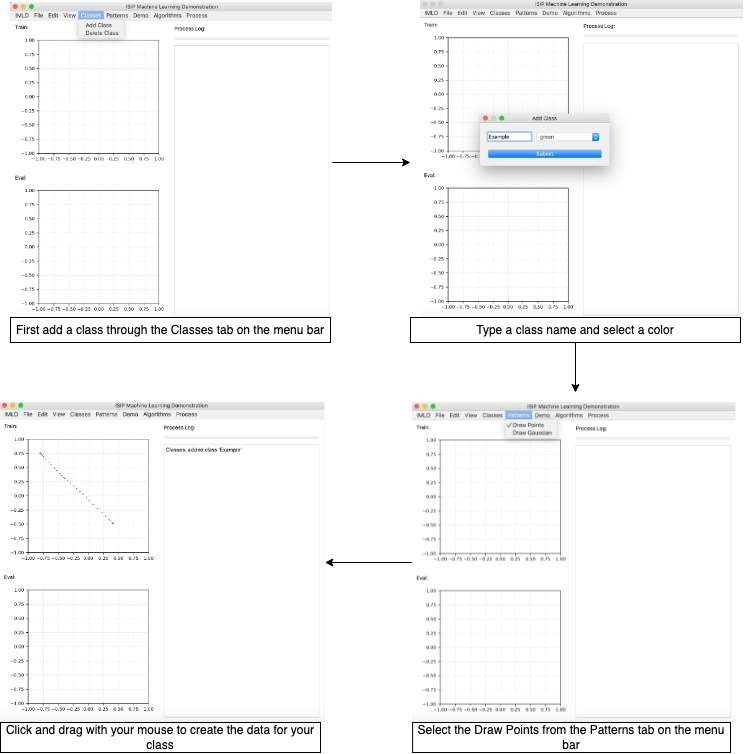
# 5. Testing

# 5.1. Test Methods and Results

To test many our “pass/fail” functional requirements, namely “provide demos”, “apply algorithms”, and “GUI”, we instituted an increasing and increasingly complex number of test cases to find where IMLD would break. One of these test cases and the flowchart for its completion is shown below:



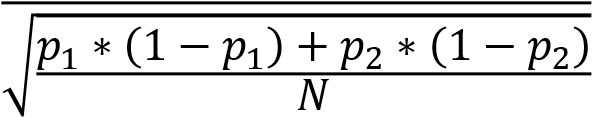
#### Figure 25: Example Test Case



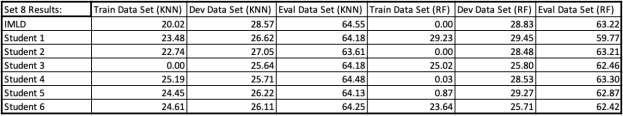
*Figure 26: Example Test Case Completion Flowchart* When determining the accuracy of our algorithms, we wished to reach a consistency between the results IMLD would produce and those produced by students manually implementing these algorithms on the same datasets. We can see the results of this testing in the table below. We can see that IMLD’s default parameters optimize these algorithms on the training set data, while some students optimize on the evaluation set data. We can determine if IMLD is providing accurate results if the difference between its results and students’ results is not statistically significant up to a certain confidence level, which we have decided to be 99.9%. We can determine these results based on equation 1, shown below:

#### Equation 1: Z-Score Equation

𝑝1−𝑝2



#### Table 3: Accuracy Testing Results



By using equation 1 on the evaluation data in the above results, we can see that for many comparisons between IMLD and student results, they are within our prescribed confidence level. Since we assume before using this test that most students will be using the default parameters for their models and knowing that these algorithms have some level of inherent randomness, we can confidently say that IMLD produces accurate results.

We can see in the design approach section the results of testing on data drawing and performance. The rest of our design constraints do not require specific testing, as we keep python and the dependent libraries up to date with each release version.

# 6. Cost and Schedule

The schedule for IMLD has been detailed above in the design process, as our organizational ability was one of the core pillars of our design approach. We released new versions of the program every month according to our Release and Jira roadmaps.

IMLD is built entirely from open source, free to use software, and so our budget has been $0.

The only cost associated with the production of IMLD has been our countless man-hours.

# 7. Summary and Future Work

As machine learning increases in popularity and usefulness, it will become crucial to teach it in a way that is visually and practically intuitive. IMLD is an educational tool with the ability to walk users through a step-by-step process to visualize various machine learning algorithm. This visual tool can help students to develop a comprehensive understanding of the basic concepts of machine learning, and the fundamentals of supervised and unsupervised learning. There are many other free open-source tools that are similar to our application; however, IMLD has some unique features that make the tool stand out. IMLD allows users to import or generate user’s own niche datasets, compared to alternative online tools, the feature of importing and exporting datasets is not available. Moreover, IMLD provides a complete control to users over configuring parameters of selected model. The tool includes 8 demonstration datasets that usually represent some typical shapes, as well as uncommon shapes to serve the efficacy of certain algorithms over others. IMLD also supports 7 different algorithms, with most of them allow users to change their model parameters.

Throughout 8 different versions that the team has contributed, the increase of performance of IMLD becomes more significant. We have successfully managed to reduce the overall memory usage of IMLD’s most demanding activities by roughly 2%. We have been able to drastically reduce the duration of some of the more complex operations by almost 50%.

IMLD has been used by a machine learning course that we have been teaching since the late 1990’s. It is easily installed on a platform that includes Anaconda v3 and Python v3.7. The application is available at www.isip.piconepress.com/courses/temple/ece\_8527/resources/imld. A detailed user manual demonstration use of the tool and instructional videos are also available.

The Institute of Signal and Information Processing will take over and work on maintaining the application.

# References

1. P. Dialani, "Machine Learning Adoption Will Influence These Five Industries," *Analytics Insight,* 16 March 2021.
2. G. Banerjee, S. Murthy and S. Iyer, "Effect of Active Learning Using Program Visualization in Technology-Constrained College Classrooms," 31 July 2015. [Online]. Available: https://telrp.springeropen.com/articles/10.1186/s41039-015-0014-0. [Accessed 01 September 2021].
3. IBM Cloud Education, "What is Machine Learning?," IBM, [Online]. Available: https://www.ibm.com/cloud/learn/machine-learning.. [Accessed 14 October 2021].
4. J. Picone, R. Duncan and J. Hamaker, "Internet-Accessible Speech Recognition Technology," in *O'Reilly Open Source Convention*, 2001.
5. 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 5 July 2021].

1. "Top Programming Languages 2021," Institute of Electrical and Electronics Engineers Spectrum, [Online]. Available: https://spectrum.ieee.org/top-programming-languages-2021. [Accessed 10 October 2021].
2. United Nations, "Sustainable Development Goals," United Nations Development Programme, [Online]. Available: https://www.undp.org/sustainable-development-goals. [Accessed 01 September 2021].
3. "Computer Memory Usage When Idle: How Much?," Tech With Tech, [Online]. Available: https://techwithtech.com/how-much-memory-computer-idle/. [Accessed 16 Feb 2022].
4. 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\_rec ognition/current/. [Accessed 05 July 2021].