**Code Change Guide for the Final Deliverable**

December 03, 2023

**Introduction**

The updated version of ML Vinci Tool by our team features consolidated screens with an improved user interface, a new module for visualizing prediction results, bug fixes in the preset module, and the addition of new models to the application. This document provides a brief overview of the code changes in the final deliverable for this capstone project. We briefly discuss the required Python libraries to run the code, highlight the advantages of the new version, and identify areas for further development.

The codes have been developed and tested using the labeled atom dataset that was shared with us. The relative file location of the data is "Team2\_FinalProject\gui-team2\Input\Additional Datasets and Code (Code Dx)\atom (30 May 2020).xml".

**Module 1:** **UI Consolidation**

1. **Which files have related code changes?**
   * Files with code changes:
     + menu\_ui.py
     + menu.py
2. **Which functions or lines have changes?**
   * **menu\_ui.py:**
     + Changed the layout to accommodate split screen.
     + Changed the font sizes and CSS styles.
     + Added a button to support the visualization functionality.
   * **menu.py:** Integrated the visualization functionality by modifying/creating the following functions:
     + btn\_Datasets\_clicked
     + def btn\_Models\_clicked
     + btn\_Predictions\_clicked
     + btn\_Visualizations\_clicked
     + btn\_Help\_clicked
     + clearRightLayout
3. **How it changes the functionality/UI of the application?**

* The main screen is divided into left and right panes.
* Menu options are listed in the left pane for easy navigation.
* The content of the new window will now be displayed within the right-side pane.
* Introduction of a new button to facilitate visualization of Prediction Results.

1. **What python modules to be installed for the code to run?**

No additional modules are required.

1. **How does the new UI work?**

In the menu\_ui.py, a screen splitter is created with the below code and this helps to rearrange the widgets within the main window.

self.centralwidget = QtWidgets.QWidget(MainWindow)  
self.centralwidget.setObjectName("centralwidget")  
  
  
# Splitter for left and right divisions  
self.splitter = QtWidgets.QSplitter(QtCore.Qt.Horizontal)  
self.splitter.setObjectName("splitter")  
  
# Left division layout (for menu buttons)  
self.leftDivision = QtWidgets.QWidget()  
self.leftLayout = QtWidgets.QVBoxLayout(self.leftDivision)  
self.leftLayout.setObjectName("leftLayout")  
self.splitter.addWidget(self.leftDivision)  
  
# Right division layout (initially empty)  
self.rightDivision = QtWidgets.QWidget()  
self.rightLayout = QtWidgets.QVBoxLayout(self.rightDivision)  
self.rightLayout.setObjectName("rightLayout")  
self.splitter.addWidget(self.rightDivision)  
  
# Adding the splitter as the central widget  
MainWindow.setCentralWidget(self.splitter)

So, when the application is started, the right pane is blank and depending on the button clicked from the menu options, the right pane is loaded with the content of the respective functionality. Whenever a new function is clicked, we clear the existing display by calling the clearRightLayout() and then display the new content.

def clearRightLayout(self):  
 *"""Clears the content of the right layout."""* while self.rightLayout.count():  
 item = self.rightLayout.takeAt(0)  
 widget = item.widget()  
 if widget is not None:  
 widget.deleteLater()

1. **How to enhance this module?**
   * The python UI files can be enhanced to provide uniform layouts for all menu options (Datasets, Modelling, Prediction, and Visualization).
   * One disadvantage is that certain layout changes can be solely made only in the Python file, and the QT Designer restricts such modifications and functionalities to the UI file.

**Module 2: Visualization of Prediction Results**

1. **Which files have related code changes?**
   * Files with code changes:
     + menu\_ui.py
     + menu.py
     + help\_ui.py
   * New files created:
     + visualize.ui
     + visualize\_ui.py
     + visualize.py
2. **Which functions or lines have changes?**
   * **menu\_ui.py:** Added a button to support the visualization functionality.
   * **menu.py:** Integrated the visualization functionality by modifying/creating the following functions:
     + def customEvents(self)
     + def btn\_Visualizations\_clicked(self)
   * **help\_ui.py:** Integrated the visualization functionality by modifying the following function:
     + def retranslateUi(self, HelpWindow)
3. **How it changes the functionality of the application?**

It provides an additional feature to visualize the results from the prediction functionality which is saved in csv file format. The visualization module currently have three graphs and let you save the graph in desired format (.png, .jpeg, .jpg).

1. **What python modules to be installed for the code to run?**
   * PyQt5
   * Matplotlib
   * PIL
2. **What is the need of the 3 new files?**
   * **visualize.ui:** This is the .ui file created using QT Designer application. It provides a overall layout for the visualization window.
   * **visualize\_ui.py:** This is the python version of the UI file. The function used to split the screen and embed the graph within the same window is not available in the QT Designer and hence it is manually coded.
   * **visualize.py:** This file has the code the connect the front and back end of the application. The custom events are coded here. Some of the import used here are repetitive and needs to be removed.
3. **How to troubleshoot if the graphs are not working?**

Check the function code for "loadDataframe(self, filename)." Ideally, when the dataset file is processed in the Labeller, the labeled data should have the column name "Status," which represents the actual/labelled values. In the prediction model phase, the resulting file includes four additional columns. These columns contain the predicted values generated by the model, and they are used to calculate the accuracy of the model. Feel free to experiment with different columns for further customization needs when creating other graphs.

def loadDataframe(self, filename):  
 *"""Loads a dataframe from filepath  
  
 Args:  
 filename (str): File path to dataframe.  
 """* self.df\_dataset = pd.read\_csv(filename)  
 self.btn\_Vis\_ConfMatrix.setEnabled(True)  
 self.btn\_Vis\_ROC.setEnabled(True)  
 self.btn\_Vis\_PRC.setEnabled(True)  
  
 # Assuming your CSV file has columns for 'true\_labels' and 'predicted\_labels'  
 self.true\_labels = self.df\_dataset['Status']  
 self.predicted\_labels = self.df\_dataset['Escalated\_predicted']  
 # Predicted probabilities for positive class  
 self.predicted\_probabilities = self.df\_dataset['Escalated\_values\_predicted']  
 # Encode categorical labels to numeric format  
 self.true\_labels\_mapped = self.df\_dataset['Status'].map(self.status\_mapping)

1. **How to enhance this module?**
   * Plots can be added for various use-cases.
   * The UI file can be enhanced to provide uniform layouts for all menu options (Datasets, Modelling, Prediction, and Visualization).
   * The module can be modified to accept two or more input files and generate comparative graphs to understand how different models perform on the same dataset.

**Module 3: GitHub Integration**

1. **Which files have related code changes?**

* Datasets.py
* Risk.py
* Datasets\_labeler.py
* Model.py

**2. Which functions or lines have changes?**

* **Models.py:** In these module we integrate the functionality with “ btn\_SaveModel\_clicked”.
  + - Line No: 448 – 455
* **Datasets.py:** In these modules we integrate the functionality with “saveDatasetFile”.
  + - Line No: 194 - 202
* **Risk.py:** In these modules we integrate the functionality with “btn\_SaveSchema\_clicked”.
  + - Line No: 423 - 432
* **Datasets\_labeler.py:** In these modules we integrate the functionality with “btn\_SaveDatasetLabeling\_clicked”.
  + - Line No: 554 – 562

**3.** **What python modules to be installed for the code to run?**

We have installed the following:

**from github import Auth, Github**

Note: “For integrating each module, the above commands must be used."

**4**. **How it changes the functionality of the application?**

Integrating GitHub into app development really changes things. It keeps track of all code changes, making it easy to fix mistakes by going back to older versions. It also lets multiple people work on the project more smoothly, as everyone can contribute without conflicts. Plus, GitHub links with other tools to automate tasks like testing. It's great for remote work and makes the whole development process more organized and collaborative.

These is an actual GitHub integration part. It’s a simple process. Please follow the following steps.

1. Create a new repository on GitHub:

- Go to the 'Repositories' section and click the 'New' button.

- Enter a name for your repository (choose any name you like). If you want a private repository, select the 'Private' option.

- Optionally, add a README file by selecting the corresponding option.

**A screenshot of a computer

Description automatically generated**

2. Create a new folder on your computer (in any location you prefer).

3. Open the newly created folder and type "cmd" in the navigation bar to open the Command Prompt.

In the Command Prompt, type the `git clone` command. Then, go to GitHub, click on "Code", copy the HTTPS address, and paste it into the Command Prompt. In the following fig, I mentioned how to get the HTTP’s Commands.

**For example: git clone https://github.com/BASH-EPIC/test.git**

**A screenshot of a computer

Description automatically generated**

5. Save all the files you want to upload into the new folder you just created on your computer.

6. Enter the following commands in the Command Prompt:

- git add . (git space add space .)

- git commit -m "<your comment>" ( put what changes u did on Ur comments)

- git push

7. If your laptop isn't connected to GitHub, run the following commands to configure Git with your GitHub account (use the email ID associated with your GitHub account):

- git config --global user.email "your\_email@example.com"

- git config --global user.name "your\_github\_username"

**A screenshot of a computer

Description automatically generated**

**Note:**

* While running the code, we encountered issues with the JSON part crashing frequently, and at times, JSON was not functioning correctly with GitHub files. Due to time constraints, we couldn't fully investigate the cause of these errors. Please be aware that there were problems with the JSON file, but all files were saved successfully.
* Another point to consider is when creating an access token. Be careful to specify the permissions you want to grant. Also, in the code file, you must use your own token to ensure all your files are saved correctly.

**A black background with white text

Description automatically generated**

* Please refer the below link on how to create a personal access token:

[**https://docs.github.com/en/enterprise-server@3.9/authentication/keeping-your-account-and-data-secure/managing-your-personal-access-tokens**](https://docs.github.com/en/enterprise-server@3.9/authentication/keeping-your-account-and-data-secure/managing-your-personal-access-tokens)

* From the following article we got the idea of Integration:[**https://pygithub.readthedocs.io/en/latest/introduction.html**](https://pygithub.readthedocs.io/en/latest/introduction.html)

**Module 4: Preset Upgradation**

Regarding the presets section, we conducted comprehensive testing of the preset root functions across all available choices. No additional libraries to be installed for the code to execute.

**Which files and functions have related code changes?**

* **vinci\_utils.py:** Modifications were made to various preset files within vinci\_utils.py, including sonarqube\_preset\_xml, pytest\_preset\_xml, pylint\_presest\_json, pylint\_preset\_xml, jslint\_presest\_xml, eslint\_preset\_xml, and eslint\_preset\_json.
* **dataset.py:** Within the dataset.py file, we imported these modified presets from vinci\_utils.py and updated the loadsDatasetFile function accordingly. Additionally, adjustments were implemented in the cBox\_PresetcurrentTextChanged function to enable the execution of existing preset choices.
* **help\_ui.py:** To enhance the dataset window's help content, we made edits based on the condition where the HelpWindow.type is identified as 'Dataset.'

**Module 5: User Interface for Multiple Modelling Module**

**1. Which files have related code changes?**

* + Files with code changes:
    - models\_ui.py
    - models.py
    - models\_workers.py

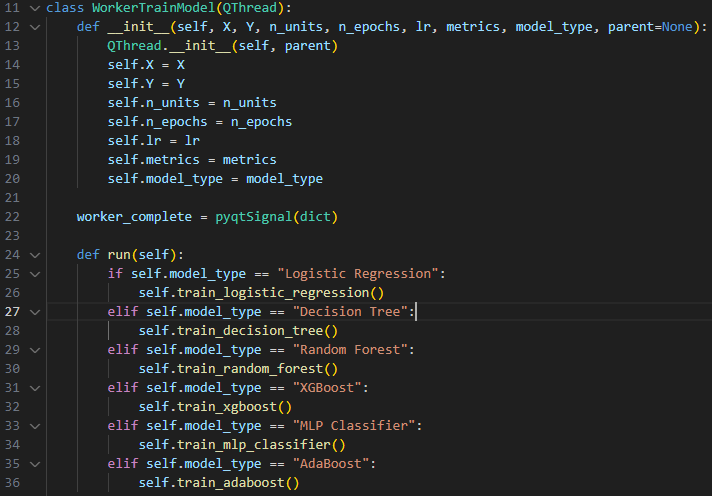
1. **Which functions or lines have changes?**
   * **models\_ui.py:** A dropdown menu for "Model Type" has been added using a QComboBoxA screen shot of a computer program

     Description automatically generated.

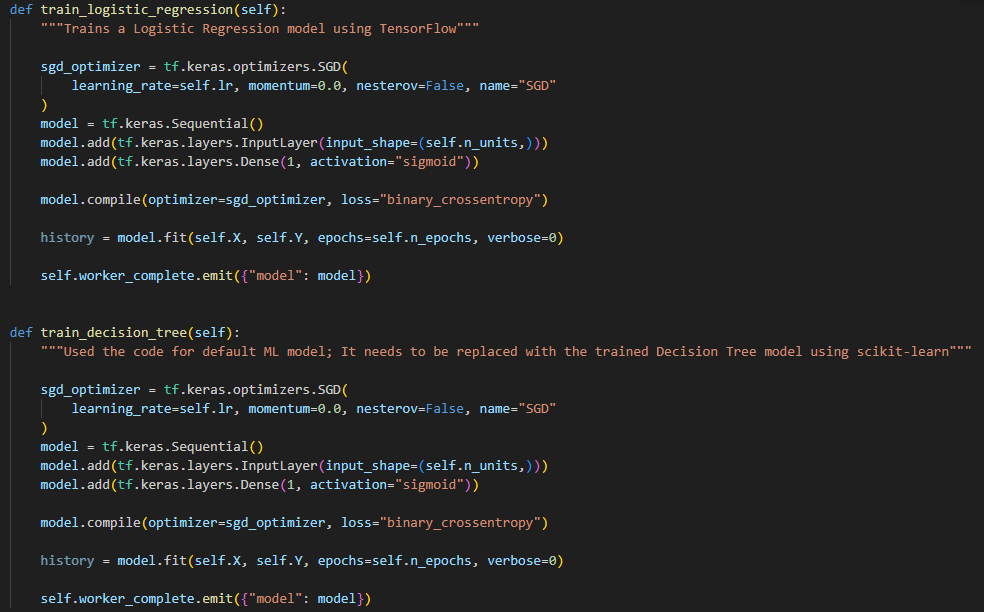
**Explanation**: A QLabel (label\_ModelType) is added to display the text "Model Type:". A QComboBox (comboBox\_ModelType) is added to provide a dropdown menu for selecting the model type. Model types such as "Logistic Regression," "Decision Tree," etc., are added as items to the QComboBox.

* **models.py:** The selected model type is retrieved from the combo box (comboBox\_ModelType) and printed to the console.

**Explanation**: This code connects the currentTextChanged signal of the comboBox\_ModelType to the comboBox\_ModelType\_currentTextChanged method. When the user selects a different model type from the combo box in the GUI, this method is called, and it prints the selected model type to the console.

* **Models\_workers.py:** We have defined a WorkerTrainModel class, which is responsible for training various machine learning models based on the specified model type.

**Explanation**: The supported model types include Logistic Regression, Decision Tree, Random Forest, XGBoost, MLP Classifier, and AdaBoost.



**Explanation**: The placeholder methods (train\_logistic\_regression, train\_decision\_tree, etc.) need to be replaced with the actual code for training the respective machine learning models using scikit-learn, TensorFlow or other relevant libraries.

1. **How it changes the functionality/UI of the application?**

* The model type dropdown in the user interface (UI) will help users to select the type of machine learning model they want to train.

1. **What python modules to be installed for the code to run?**

We need to ensure the corresponding libraries for different machine learning models are installed.

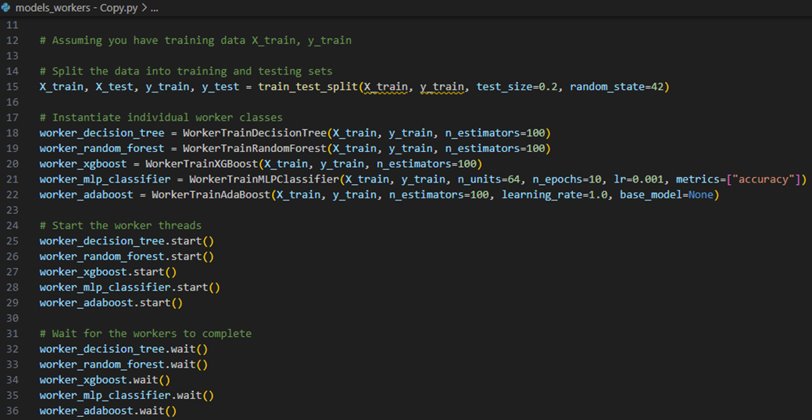
1. **How to enhance this module?**

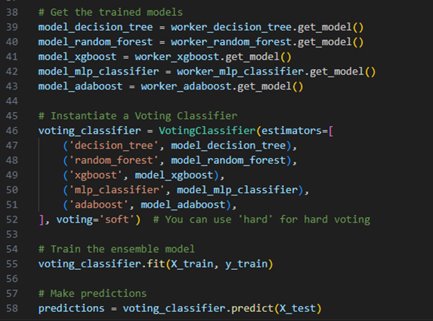
* Expand the dropdown menu to include a broader selection of machine learning models, covering a variety of algorithms.
* Implement dynamic adjustment of UI elements based on the selected model. For instance, display relevant hyperparameter input fields or configuration options specific to the chosen model.
* Incorporate a mechanism to fetch and display default hyperparameter values for each model, assisting users in setting up models more effectively.
* Extend the module to support model comparison features, allowing users to train and evaluate multiple models simultaneously.
* Include functionality for displaying and comparing evaluation metrics for different models on the same dataset.
* Implement robust error handling mechanisms to gracefully handle unexpected issues during model training.
* Alternatively, we can utilize the Voting Classifier algorithm for using multiple ML models. The concept is that by leveraging the strengths of various models, the ensemble can often outperform individual models. Multiple classifiers are trained independently, and their predictions are aggregated through a voting mechanism to arrive at the final decision. There are two main types of voting in a Voting Classifier:

Hard Voting: In hard voting, each model in the ensemble "votes" for a class, and the class with the majority of votes is chosen as the final prediction. This is effective when each individual model has decent predictive accuracy.

Soft Voting: In soft voting, each model provides a probability estimate for each class, and the average probabilities for each class are calculated. The class with the highest average probability is chosen as the final prediction. This is effective when the models in the ensemble can output probabilities. Below is the test code for the same.

You can find the sample code in **“test\_VotingClassifer.py”.**





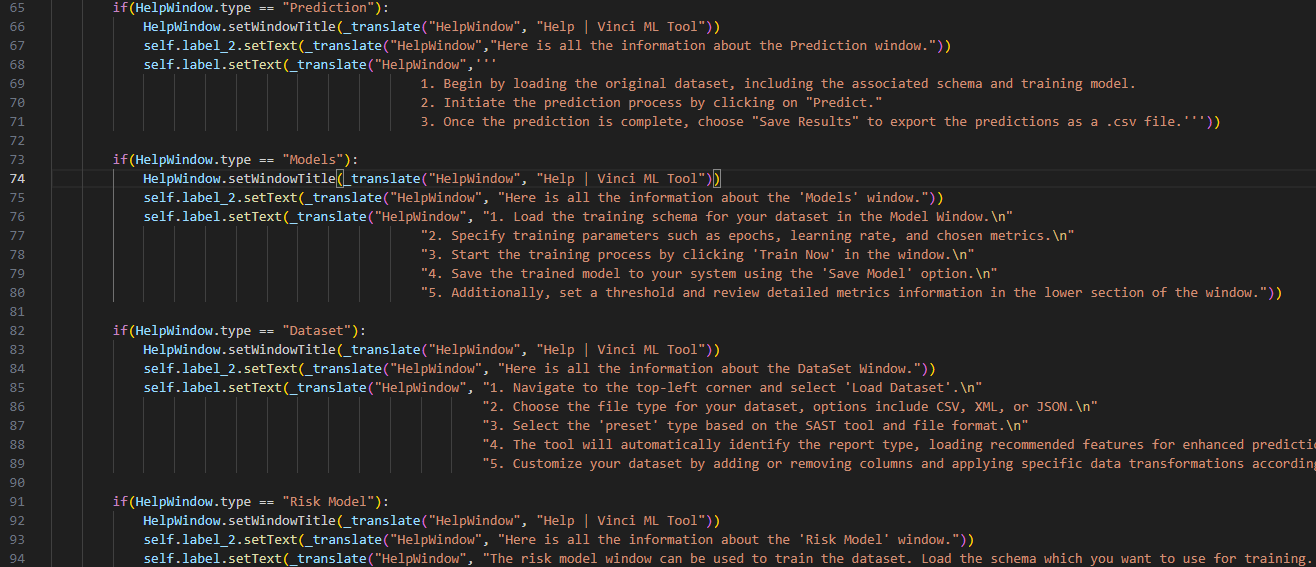
**Module 6: Help content improvement in User Interface Module**

**1. Which files have related code changes?**

* + Files with code changes:
    - help\_ui.py

**2. Which functions or lines have changes?**

* + **help\_ui.py:** The text content in the widget of the HelpWindow is modified to make it more user friendly and actionable.



**Explanation**: These changes enhance the visual appearance of the HelpWindow and provide informative content tailored to different sections of the application.

**3. How it changes the functionality/UI of the application?**

* The modifications include step-by-step instructions, tips, and suggestions on how users can effectively perform tasks within the application.

**4. What python modules to be installed for the code to run?**

* No additional modules are required.

**5. How to enhance this module?**

* Include images, diagrams, or videos that visually explain complex processes or functionalities.
* Implement a search functionality within the HelpWindow, allowing users to quickly find information related to specific keywords or topics.

**Contact:**

Please feel free to contact us if you have any questions or concerns. Although each module is developed by individual team members, we've actively learned and shared knowledge among ourselves, ensuring we can address all your inquiries.