

Dealing with Imbalanced Data

A dissertation submitted in partial fulfilment of
the requirements for the degree of
BACHELOR OF *ENGINEERING* in Computer Science
in
The Queen's University of Belfast

by
Rory Cunningham

3rd May 2022

**SCHOOL OF ELECTRONICS, ELECTRICAL ENGINEERING and COMPUTER
SCIENCE**

CSC3002 – COMPUTER SCIENCE PROJECT

Dissertation Cover Sheet

A signed and completed cover sheet must accompany the submission of the Software Engineering dissertation submitted for assessment.

Work submitted without a cover sheet will **NOT** be marked.

Student Name: Rory Cunningham Student Number: 40235348
Project Title: Dealing with Imbalanced Data
Supervisor: Dr Thai Son Mai

Declaration of Academic Integrity

Before submitting your dissertation please check that the submission:

1. Has a full bibliography attached laid out according to the guidelines specified in the Student Project Handbook
2. Contains full acknowledgement of all secondary sources used (paper-based and electronic)
3. Does not exceed the specified page limit
4. Is clearly presented and proof-read
5. Is submitted on, or before, the specified or agreed due date. Late submissions will only be accepted in exceptional circumstances or where a deferment has been granted in advance.

By submitting your dissertation you declare that you have completed the tutorial on plagiarism at <http://www.qub.ac.uk/cite2write/introduction5.html> and are aware that it is an academic offence to plagiarise. You declare that the submission is your own original work. No part of it has been submitted for any other assignment and you have acknowledged all written and electronic sources used.

6. If selected as an exemplar, I agree to allow my dissertation to be used as a sample for future students. (Please delete this if you do not agree.)

Student's signature

Rory Cunningham

Date of submission

3rd May
2022

Acknowledgements

I would like to thank my family and friends for supporting me through this project. I thank Dr Thai Son Mai and Queen's University for giving me the opportunity to complete this dissertation.

Abstract

An explosion has taken place in data generated across the world. With this data comes opportunities to derive meaning and value from it. This has led to the rise in machine learning and data mining, but this rise has come with complex problems. Imbalanced data refers to data that is skewed in classification, and traditional machine learning algorithms fail to perform optimally on such data. This project aims to provide a software suite that aids the comparison and evaluation of methodologies used to alleviate the effects of imbalanced data.

Contents

1.0 Introduction to Problem Area	5
2.0 Solution Description and System Requirements	7
2.1 Initial System Requirements	7
2.2 Solution	7
2.3 Advantages of Solution	8
2.4 Solution Functions	9
3.0 Design	13
3.1 Basic Architectural Design	13
3.1 UI Design	14
3.2 Software System Design	18
3.3 Database Design	20
3.4 Error and Exception handling and Logging	21
4.0 Implementation	22
4.1 Basic Overview	22
4.2 Database implementation	22
4.3 Dataset selection and implementation	22
4.4 Software Libraries used	23
4.5 Key Functions and Algorithms	24
4.5.1 Data Sampling Algorithms	24
4.5.2 Learning Algorithms	26
4.5.3 General Functions	29
5.0 Testing	29
5.1 Testing The GUI	29
5.2 Testing The Database	30
5.3 Testing the rest of the code	31
5.4 Conclusion	33
6.0 System Evaluation	34
6.1 Evaluation of Project	34
6.2 Comparison to Other Products	35
6.3 Future enhancements	35

1.0 Introduction to Problem Area

With the explosion of social media and modern digital technology, unprecedented growth in raw data has come. This data has proved of immense value to both industry and academia, with wide-ranging uses such as simple information processing to government decision-making systems.

However, with this increased volume of data comes issues; when processing large quantities of data, many algorithms used for knowledge discovery and data engineering do not perform as well as expected. This is often caused by what is known as ‘Imbalanced Data’. (What is it?) Most algorithms used in data processing expect that data will be evenly distributed and that each class of data will have an equal or similar amount of examples. Therefore whenever the algorithms are presented with complex and imbalanced datasets, their performance is severely compromised.

When this issue manifests itself in real-world applications, there can be disastrous and wide-ranging effects. Take, for example, the attempt to classify cancerous cells. A dataset of cells to be classified will be skewed heavily, containing many more healthy cells than cancerous cells. As a result, the instances belonging to the minority group, the cancerous cells, are misclassified more often than those belonging to the majority group. This can have serious consequences; the misclassification of cancerous cells as non-cancerous during medical diagnosis can be fatal.

When we are discussing the imbalanced data problem technically any dataset that exhibits any unequal distribution is imbalanced. However, the understanding is that imbalanced data refers to data that shows a significant and often extreme imbalance. Specifically, this form of imbalance is referred to as a between-class imbalance. We often see between-class imbalances in the range of 100:1, 1,000:1, and 10,000:1, wherein in each case, one class severely out represents another [1], [2], [3]. There are also multiclass data in which imbalances exist between the various classes [4], [5], [6]. The types of imbalanced data can be separated into two categories *extrinsic* and *intrinsic*. Extrinsic data imbalances are due to various factors not directly related to the nature of the data space, such as storage of data and time. Intrinsic data imbalances are directly related to the nature of the data space, like the example of cancerous cells as mentioned earlier.

The imbalanced data problem is not as simple as just one class having more examples than the other. To analyse the problem further it can be broken down into five sub-problems as shown by Haixiang et al.[8]:

1. Standard classifiers are only suitable for balanced datasets. When facing imbalanced situations these models provide suboptimal classification results. Usually, this results in good coverage of majority examples, whereas the minority examples are distorted.[9]
2. Many performance metrics like prediction accuracy induce a bias toward the majority class, while the rare episodes remain unknown even if the prediction model has high overall precision. [10]
3. Rare minority examples may be treated as noise by the learning model. On the other hand, noise may also be treated as minority examples, since both of them are rare patterns in the data space. [11]
4. Small disjuncts, a lack of density and a small sample size with high feature dimensionality are challenging to imbalanced learning, which often causes learning models to fail in detecting rare patterns.[12]
5. Minority examples usually overlap with other regions where the prior probabilities of both classes are almost equal[9]

In the machine learning field, many approaches have been developed in the past to deal with imbalanced data. The approaches can be divided into data-level techniques, algorithm-level methods, and hybrid strategies [7]. Data-level techniques are made up of many forms of resampling data. These can include oversampling the minority class, randomly undersampling the majority class, informatively oversampling the minority class, informatively undersampling the majority class, and oversampling the minority class by generating new synthetic data. In addition, we also have combinations of the mentioned techniques.

With such a wide array of techniques developed, it becomes difficult to assess and compare the effectiveness of each. There are currently benchmark platforms that are used to evaluate the effectiveness of machine learning algorithms. However, none are dedicated solely to the imbalanced data problem. Datasets used in existing benchmarks are not suitable for testing imbalanced learning solutions as often the datasets are balanced and therefore require manipulation before use. In addition to this, we know that there are no standard metrics for assessing the effectiveness of models on imbalanced data with traditional options giving misleading results[10]. There is a need for a system to evaluate algorithms specifically for imbalanced learning.

2.0 Solution Description and System Requirements

2.1 Initial System Requirements

This dissertation project aims to create a unified, systematic framework to act as a tool to analyse techniques used to alleviate the effects of imbalanced data. This aims to unify and collect available methods for imbalanced data that are in current use and allow them to be analysed and compared under one system. The tool should include the use of a graphical user interface to increase accessibility and make the comparison of techniques a more straightforward and less technical process. The tool should include relevant metrics used for imbalanced learning, such as ROC Curves and Precision-Recall curves so that users can see reflective metrics based on algorithm performance.

2.2 Solution

The purpose of this project is to create a system that allows for the easy and simple analysis and comparison of techniques used to mitigate the effects of imbalanced data in machine learning. It will remove the difficult task of collecting both data and techniques from across a huge range of sources and comparing effectiveness based on unstandardised metrics. The intended audience for this system is academia and industry, or simply anyone who wants to test the efficacy of techniques on learning using imbalanced datasets. The user will be able to quickly and succinctly select the best methods for the problem they are trying to solve.

The system will allow the user to select an imbalanced dataset to perform tests upon. This dataset will be able to be modified by the user by enabling specific feature selection and preprocessing if wanted. The user can then select a sampling technique to be used on this dataset if required. Again the sampling parameters will be able to be customised by the user if needed. Finally, the user will be able to select a learning algorithm and customise it as they see fit. This will allow the user to analyse individual techniques used and how effective they are completely. It will also enable the user to create a model that produces the best results on the dataset that they have selected. It is assumed that the user has prior knowledge of the techniques they are to implement and the parameters they could edit. Once all the possible options have been selected, the user will be able to see the results of the learning performed and obtain the output of multiple assessment metrics.

Functional Requirements:

- Allow for the selection of a dataset from several real-life datasets
- Allow for the customisation of parameters surrounding chosen dataset
- Allow for the appointment of a data sampling technique to be picked from several techniques
- Allow for the customisation of parameters surrounding chosen sampling technique
- Allow for the appointment of a learning algorithm to be picked from several algorithms
- Allow for the customisation of parameters surrounding the desired algorithm
- Allow for a section of assessment metrics to be selected
- Users must be able to view metrics and results, including graphical metrics
- Users must be able to compare results between previous and current runs
- The system must be easy to navigate and simple to use
- Fast and accurate results must be able to be obtained by the system

Non-Functional Requirements:

- Multiple datasets must be available to choose from
- Ten or more sampling methods must be available to choose from
- Fifteen or more learning algorithms must be available to choose from
- The metrics Accuracy, precision, F-measure, recall, ROC curves and Precision-Recall curves must be available to view after a successful run
- Users must be able to preprocess the datasets if required using the following techniques:
Normalization, Binarization, Rescaling, Standardization.
- Users must be able to use Feature selection on the dataset with the following options: PCA, Univariate selection, Recursive Feature Elimination.

2.3 Advantages of Solution

With the system outlined above, there will be many significant advantages to its use. It allows for fast and consistent analysis of many techniques while also providing a selection of assessment metrics to be used so that comparison is relatively simple. The observability offered by the system into how effective a technique means that users can quickly narrow down an effective strategy to create a useful and effective model. A set selection of imbalanced datasets allows for repeatability and reliability that may not be found elsewhere. The additional advantage of being easily able to

combine both sampling and algorithmic techniques with relative ease opens up new avenues of an investigation surrounding novel techniques to address the imbalanced data problem. The fact that the user does not have to write any code or have knowledge of any programming language it is written is a huge advantage. It allows people from a non-technical background to use the system still and obtain results. Importantly it also offers ease of comparison between techniques used, meaning that the shortcomings and advantages of each can be easily viewed. This is further aided by having standard assessment metrics used within the system, meaning that all models that will be created are assessed the same. The metrics ROC Curve and Precision/Recall curve offer good insight into how genuinely effective the model will be as other metrics can be unhelpful by being slightly misleading.

2.4 Solution Functions

Usage	Loading Dataset
Description	Dataset will be split into Train and validation sets
Precondition	N/A
Inputs	Name of datasets, train/validation split
Success Response	Creation of test and validation datasets
Error Response	Default

Usage	Processing Dataset
Description	Dataset will be preprocessed and feature selection applied if applicable
Precondition	Dataset must be loaded and train/validation must be created
Inputs	Preprocessing method, feature selection method
Success Response	Creation of test and validation datasets
Error Response	Default

Usage	Apply Sampling
Description	Train dataset will be sampled according to the method that is selected
Precondition	Dataset must be loaded and train/validation must be created
Inputs	Sampling method, hyper-parameters (variable on method chosen), dataset
Success Response	Successful sampling of train dataset according to the method selected
Error Response	Default

Usage	Modify Sampling Hyper-Parameters
Description	Hyper-Parameters of the sampling technique will be able to be edited
Precondition	Sampling algorithm must be chosen
Inputs	Sampling algorithm
Success Response	Hyper-parameters will be changed
Error Response	Default

Usage	Apply Learning Algorithm
Description	Model will be created based off the chosen learning algorithm
Precondition	Learning algorithm must be chosen and dataset loaded
Inputs	Learning algorithm, dataset
Success Response	Creation of model
Error Response	Default

Usage	Modify Learning Hyper-Parameters
Description	Hyper-Parameters of the learning algorithm will be able to be edited
Precondition	Learning Algorithm must be chosen
Inputs	Learning Algorithm
Success Response	Hyper-Parameters will be changed
Error Response	Default

Usage	Generate Results
Description	The result of the model created against the training dataset will be displayed
Precondition	Learning algorithm must be applied and model created
Inputs	Model, train dataset
Success Response	Creation of results for the model that can be viewed
Error Response	Default

Database Functions

Usage	Create Database
Description	Create a database at the initial run of the system
Precondition	Database management system is installed
Inputs	N/A
Success Response	Database is created
Error Response	N/A

Usage	Create tables for Database
Description	Create a tables for the database when a user account is created
Precondition	Database management system is installed and database exists
Inputs	N/A
Success Response	Database tables are created
Error Response	N/A

Usage	Save User to Database
Description	Save user credentials to the database
Precondition	Dtabase must exist and tables must be created within it
Inputs	User credentials
Success Response	User credentials are saved to the database
Error Response	Invalid User Credentials

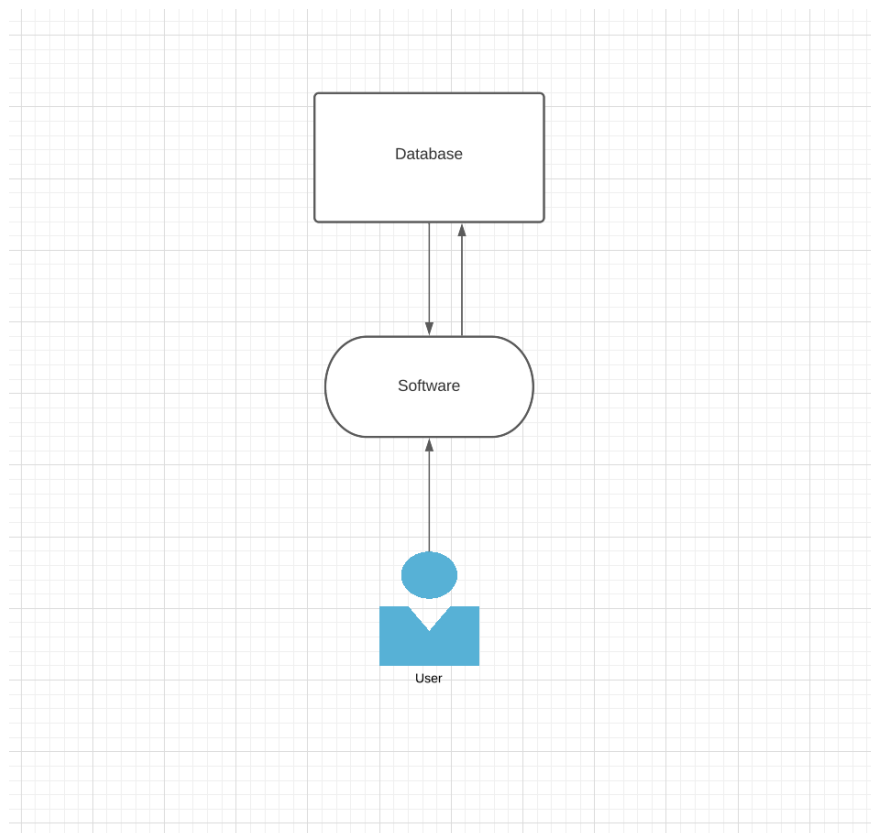
Usage	Save results to Database
Description	Save the results of a model to the database so they can accessed at a later date
Precondition	User credentials must be stored in database
Inputs	Results
Success Response	Results are saved to database
Error Response	No account has been created to save results to

Usage	Fetch results from Database
Description	Results can be fetched from the database based on the user logged in and the name of the results
Precondition	User credentials and previous results must be stored in database
Inputs	Username, results name
Success Response	Results are returned
Error Response	No results found

3.0 Design

3.1 Basic Architectural Design

The system can be divided into three components as displayed in the diagram below. This structure is the database that communicates unilaterally with the software system, both receiving and sending data. Next is the software system that allows user interaction and input; it sends and receives data to the database while also allowing for user input and displaying to the user. Finally is the user, who both inputs and views data to the software system.



3.1 UI Design

Welcome Screen

The Welcome Screen UI consists of a sidebar menu on the left and a main content area. The sidebar menu, titled "Menu", contains five buttons: Home, Data, Sampling, Algorithm, and Results. The main content area has a header with "Login" and "Exit" buttons. Below the header, the text "Welcome!" is centered. Underneath, there are three buttons arranged vertically: Start, Create Account, and Help.

Login Screen

The Login Screen UI consists of a sidebar menu on the left and a main content area. The sidebar menu, titled "Menu", contains five buttons: Home, Data, Sampling, Algorithm, and Results. The main content area has a header with "Login" and "Exit" buttons. Below the header, the text "Login/Create Account" is centered. Underneath, there are two text input fields, each preceded by the label "Text". At the bottom right of the main area is an "Enter" button.



Data Screen

Menu	Login		Exit	
	Choose Dataset	<input type="text"/>		
	Train/Test Split	<input type="text" value="0.0"/>		
	Pre-processing	<input type="text"/>		
	Feature Selection	<input type="text"/>		
		Enter		

Sampling Screen

Menu	Login		Exit	
	Choose Sampling technique	<input type="text"/>		
	Number of Neighbours	<input type="text"/>		
	Number of seeds	<input type="text"/>		
	Feature Selection	<input type="text"/>		
	Threshold	<input type="text"/>		
	Version	<input type="text"/>		
		Enter		

Results Screen

Menu	Login		Exit	
	Results			
	Accuracy		F-measure	
	Recall			
	Precision			
				
Home	ROC		Precision Recall	
Data				
Sampling				
Algorithm				
Results	Enter			

Algorithm Screen

Menu	Login		Exit	
	Choose Algorithm			
	Weights			
	Number of Inputs			
	scale_pos_weight			
	Number of estimators			
Home	Contamination			
Data	Epochs			
Sampling	Enter			
Algorithm				
Results				

Usability and Design Pattern

The UI was designed with some key patterns and components used to make the ideal user experience. The first step is implementing the ‘Lazy Registration’[13] design pattern. This design pattern focuses on allowing the user to access all key functionality without having to undergo formal registration, such as creating an account in the case of this system. Where the system does require an account, such as accessing data that is stored in the database, the creation of an account is made as simple as possible. The design considers the range of potential users that may be using the software; not all are guaranteed to be of a solely technical background, meaning that an easy to use and understand interface is optimal.

Workflow

The user will be greeted with a ‘Welcome Screen’ when the software loads. On this screen, the user has three main options: to begin choosing a dataset, creating an account, or opening a ‘help page’. If the user wishes to log in with a previous account, an option is available in the top right corner and so is an exit button. The buttons down the left side of the screen give the user the ability to jump to different steps in the process of implementing imbalanced learning.

If the user decides to log in or create an account, they will be taken to the “Login” screen. On the ‘Login/Create Account screen, the user can enter previous details or submit the details for a new user. From here, they can navigate back home or start the process of implementing imbalanced learning.

On the “Choose Data” screen, the user can choose their preferred dataset from a drop-down list. The train/test split must next be set using a spin-box to select an appropriate value. From here, further customisation can occur, and the user can choose pre-processing and/or feature selection techniques from two separate drop-down lists. The “Next” button will take the user to the “Choose Sampling” screen.

The “Choose Sampling” screen will allow the selection of data sampling techniques used to tackle imbalanced data. The list of techniques will populate a drop-down list that the user can choose. Once the sampling technique is chosen, the individual hyper-parameters associated with it can be

customised by simply entering in line-edit boxes. The user can now move on to the next step of selecting a learning algorithm by simply clicking a button.

With a design similar to the previous selection screens, the “Choose Algorithm” screen allows the selection of a learning algorithm from a drop-down list. Again the hyper-parameters of this selection can be customised according to the user's needs by entering into line-edit boxes. Finally, the user can click a button to advance to the results screen.

The final screen is the “Results Screen” on this screen, the user will be able to see all the results of their previous choices. The results will be displayed through labels and two graphs showing ROC and Precision/Recall curves. The user can save their results by clicking the “Save Button”. They can also compare their current results to previous results they have generated through the “Compare Results” button. The “Back to Start” button is the last button that returns the user to the “Choose Data” screen.

3.2 Software System Design

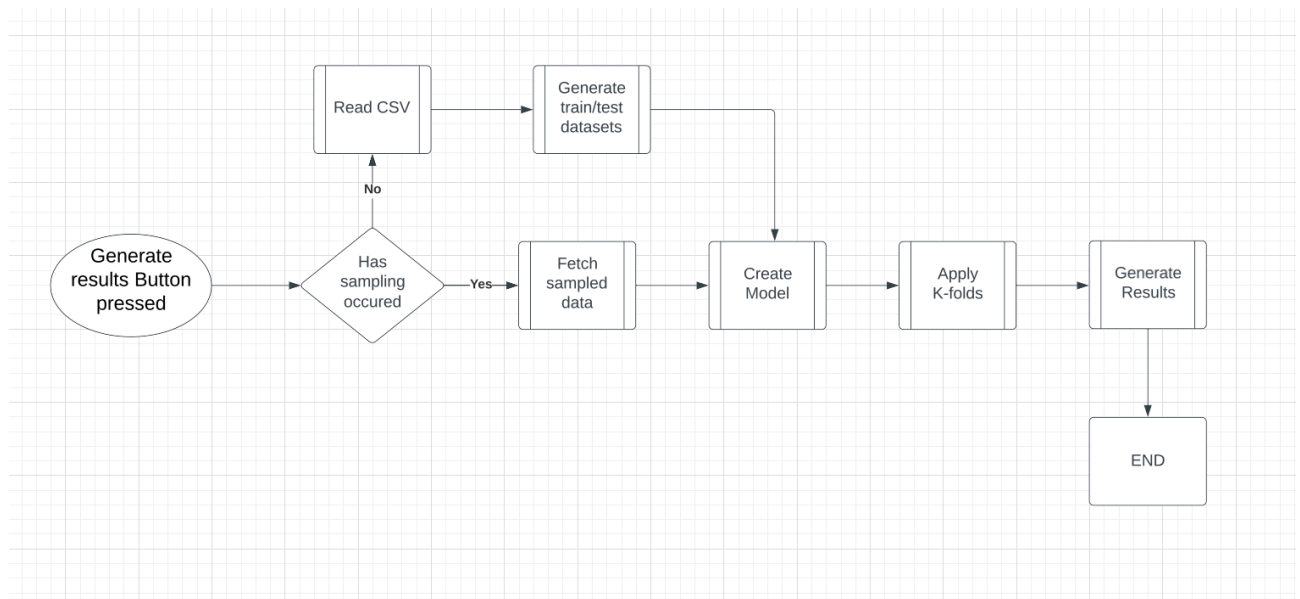
UI Software design

When designing the software for the User Interface, the Model View Control(MCV)[14] design pattern was deemed the most suitable approach to implement. However, PyQt5 allows input events from the user and delegates these to the widgets (Controller) to handle. Widgets also handle the presentation of the current state to the user, meaning they are also part of the “View”. So with PyQt, the View and Controller are merged, creating Model/ViewController architecture. Fortunately, the other principles of MVC are preserved. The “Model” holds the data or a reference to it and returns individual or ranges of records and associated metadata or instructions. ViewControl requests data from the Model and displays what is returned on the widget.

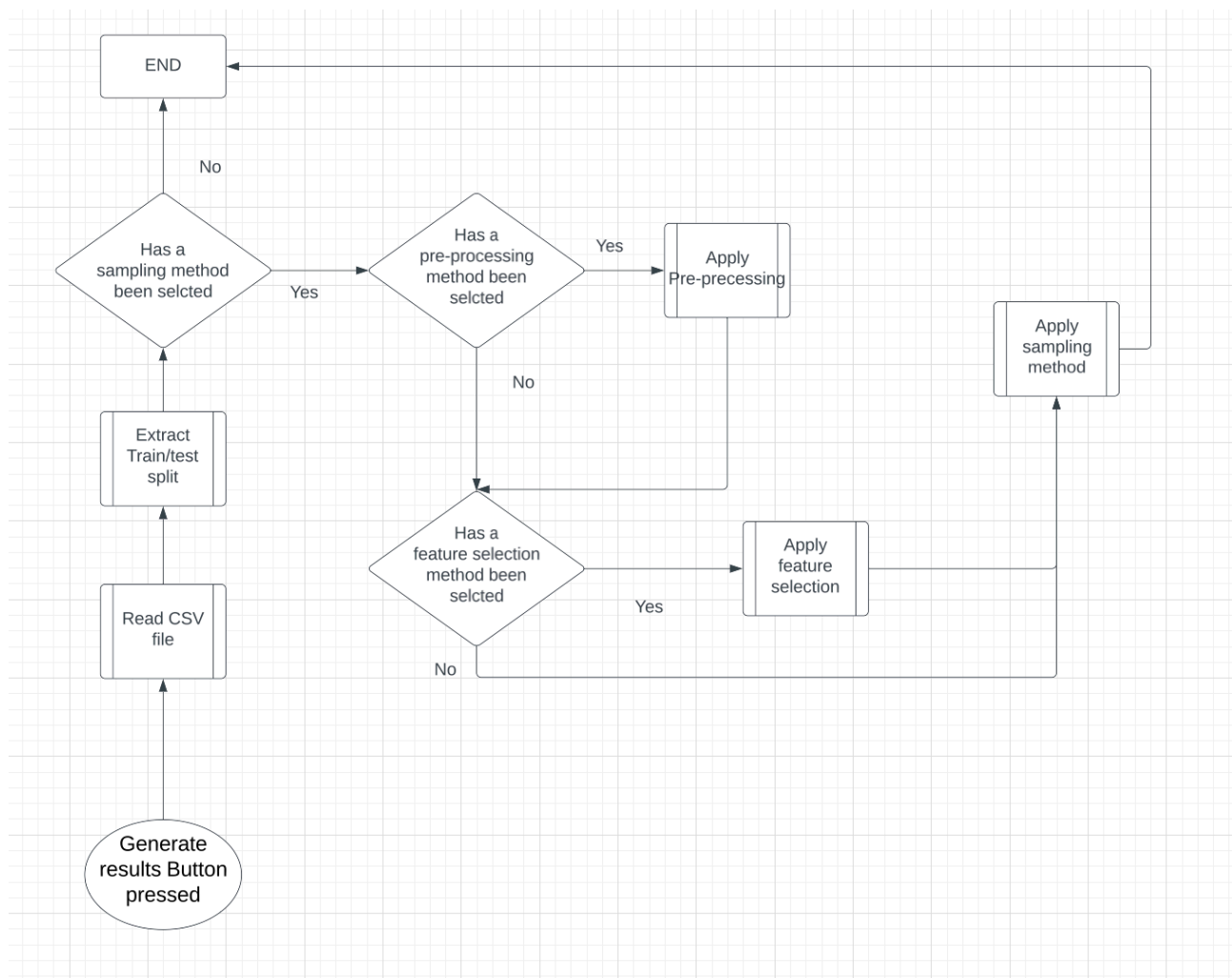
Logic Layer Design

This layer's primary role is to manage communication between the data and the presentation layers (the UI). It handles all the rules, calculations and logic in the application. Our logic layer can be divided into three main areas: Data handling, sampling method application and the application of learning methods. The diagrams below outline the functionality of the logic layer that is called upon by the user.

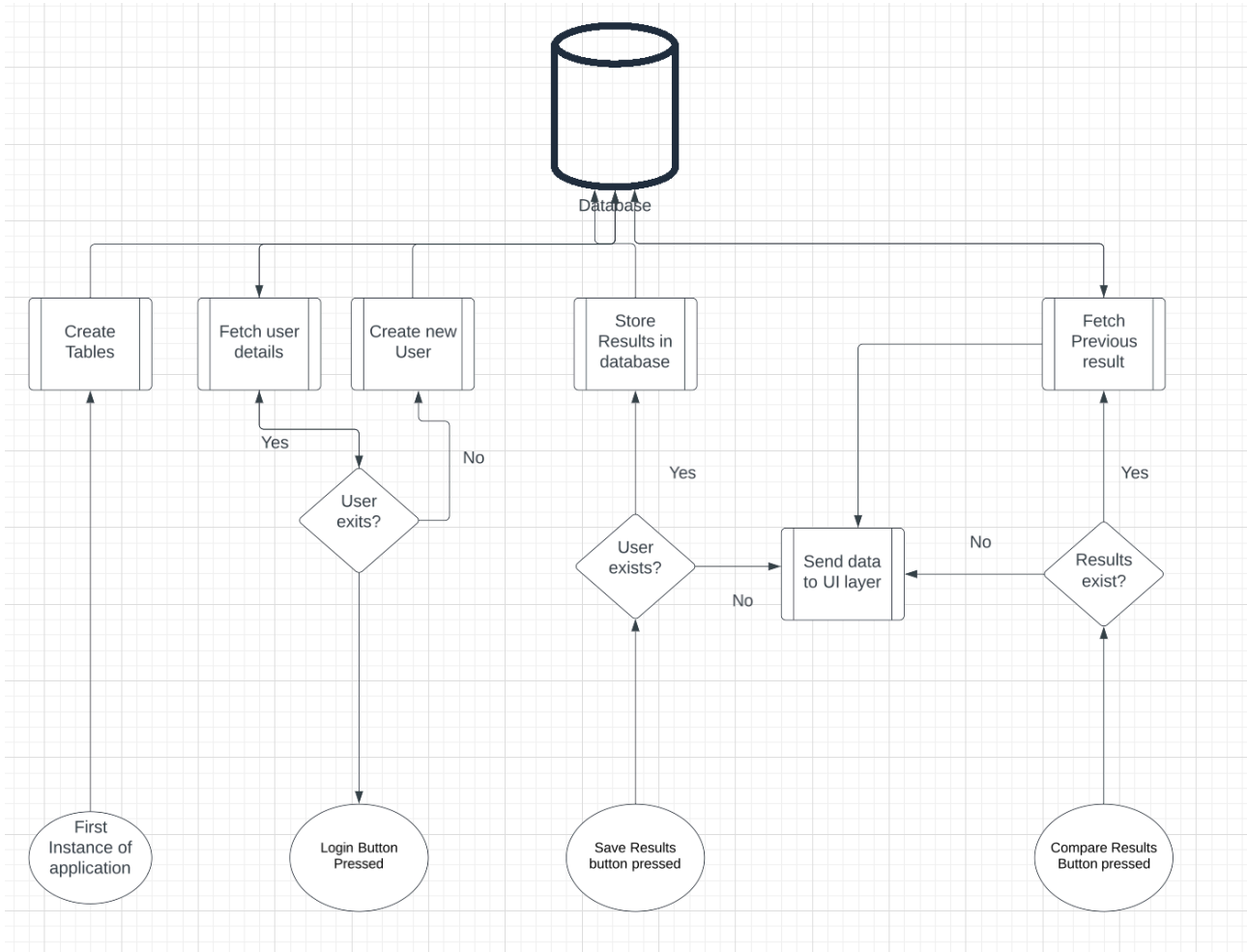
Learning Algorithm Logic Design



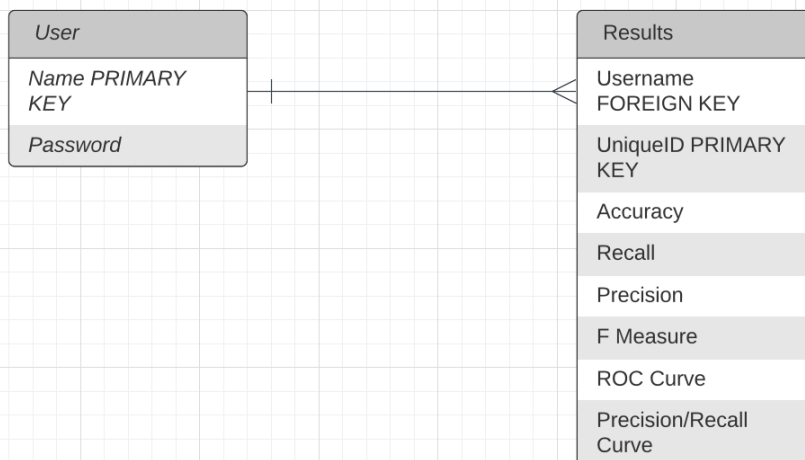
Sampling Logic Design



Data Logic Design



3.3 Database Design



Data Persistence

The data needs to persist to allow for the comparison of results from different models that the user may have generated. This persistence is facilitated by using a relational database, with the information retrievable after the process has ended and a new one has begun.

3.4 Error and Exception handling and Logging

As the system uses a database to store data, there has to be error, and exception handling specifically focused on that area. Often, when users are fetching data from the database, issues such as the data not existing in the database can arise. These types of exceptions must be handled correctly and not allow the system to become faulty, making it more robust. To deal with this, custom exceptions are the standard process. The steps for exception handling of the database are as follows:

- If the user enters results to be found and fetched back to them, but the details do not exist, this will throw an exception. This will be caught and logged; the user will be informed that the credentials they entered do not exist.
- When creating the connection to the database to perform commands on the data, if the database does not exist, this will throw an error, this will be caught, and subsequently, a database will be created.
- If the user cannot save their name and password to the dataset, this will be caught, and the user will be asked to try again with relevant details.

Any errors that the system might encounter will be logged into a file that can later be accessed to remediate the root cause of this error. Other forms of data that could be logged, such as DEBUG and INFO, will be ignored, with only WARNING, ERROR and CRITICAL being logged to the file. Overlogging would clutter the file and may render error investigation more difficult.

4.0 Implementation

4.1 Basic Overview

The system was programmed using the Python programming language. This is the most widely adopted language for machine learning and has many valuable packages and libraries that can be called upon. The version of Python used was Python 3.9. the user interface was chosen to be created using python as well, implementing the PyQt5 package.

4.2 Database implementation

The database management system that was chosen was SQLite. “SQLite is a C library that provides a lightweight disk-based database that doesn’t require a separate server process and allows accessing the database using a nonstandard variant of the SQL query language”[1]. It was chosen as it is lightweight without overheads or dependencies, skipping out the server component. The database required is not overly complex, so a lightweight tool was ideal. With its easy implementation, the database was created using a python script. It is accessed and edited in the same way by being called from a script. The database is relational, with a ‘User’ table that requires a username and a password, the username being a Primary Key. This is then linked to a ‘Results’ table that stores the results saved by that particular user. Retrieval and sending of data are facilitated by setting up a connection to the database in python and passing through the required SQL commands. The saved data is persistent and can be accessed in a later session.

However, as discussed in the design phase, the system does not need a user to have an account to use it. This means that the database is required only for users who wish to save and compare the models' performance.

4.3 Dataset selection and implementation

The datasets required in this system have to be imbalanced. They also have to be of an appropriate size, as too large a dataset would cause the learning process to take far too long. The datasets that were used were as follows:

- Habermann Breast Cancer Dataset[19]
- German Credit Dataset [20]

- Pima Indians [21]

The implementation of these datasets was through a CSV file, as this is the easiest and most common way of storing such data when in use in machine learning. There are no security requirements for such data as they are publically available online, so extra caution was not needed when accessing them. The datasets are ‘real-world’ and not synthetically generated, coming from different areas of industry and research that see the effects of imbalanced data.

4.4 Software Libraries used

Package Used	Why was it used?
Pandas	Pandas was used to read the data in csv files and prepare it propely for use in the machine learning process. The csv’s were read and converted in a Pandas dataframe to be used in the program.
PyQt5	PyQt5 is the package used to build the GUI of the program. The front end is composed of PyQt widgets that offer a range of functionality.
Sklearn	Sklearn is a library focused on machine learning, it offers algorithms, pre-processing and metrics to score your model by.
Sklearn.neighbours, Sklearn.linear_model, Sklearn.svm, Sklearn.tree, Sklearn.ensemble	This collection of subpackages all are learning algorithms that are used within the machine learning community. Within the program they ar used to create models off of the chsoen dataset
Sklearn.metrics	The metrics subpackage allows for the calculation of metrics associated with machine learning models in order to asess how effective they are.
Sqlite3	Sqlit3 package allows the creation and subsequent use of a sqlite3 databse through running of python commands.

Package Used	Why was it used?
Imblearn	This is one of the most important packages used in the program. It allows access to specific machine learning algorithms and sampling methods associated with imbalanced data learning.
Imblearn.under_sampling, Imblearn.over_sampling	These allow us access to both under sampling and over sampling methods to be used on imbalanced datasets.
Xgboost	This specific package allows the use of the XGBoost, a gradient boosting algorithm that can be used in imbalanced learning.
Logging	The logging package allows the logging of information from our system. This can include simple INFO messages to CRITICAL issues.

4.5 Key Functions and Algorithms

4.5.1 Data Sampling Algorithms

Data sampling involves techniques that will change and transform the nature of the dataset. This is done to try and balance the classes in the dataset, usually either by removing samples from the majority class or increasing samples from the minority class. From this, a learning algorithm can then be trained on the newly balanced dataset. However, simply changing the composition of the dataset does not always lead to increased performance of learning algorithms. Several factors such as class rarity and accidentally creating ambiguous new examples can lead to a reduction in the expected performance boost.

Random Undersampling - *SamplingMethods.randomUndersampling*

Undersampling methods delete or select a subset of examples from the majority class. This is the simplest undersampling method, and as the name suggests, it randomly deletes samples from the majority class in the training dataset. The X and y subsets of the training data are passed in with the method returning the altered X and y.

Condensed Nearest Neighbour Rule - *SamplingMethods.condensedNN*

This algorithm works by constructing a subset of examples which can correctly classify the original data set using a 1-NN algorithm. It is applied by reducing the number of samples in the majority class after all samples in the minority class have been added to the subset. This algorithm was implemented using the Imblearn package for Python by supplying X and y train data and fitting a model appropriately. This model can then return altered X and y data at the end of the function.

Edited Nearest Neighbour Rule - *SamplingMethods.ENN*

Edited nearest neighbour (ENN) uses $k=3$ nearest neighbours to find misclassified examples in the dataset and subsequently deletes them. This algorithm can be run several times on the data, repeatedly reducing the examples in the majority class. This process of running multiple times is known as Repeated Edited Nearest Neighbours. The number of neighbours can be edited in the implementation used. It is for the user's discretion to decide this value. The number is passed in as an argument of the function.

Near Miss Undersampling - *SamplingMethods.nearMiss*

Near miss can be subdivided into three distinct methods: Near-Miss 1, Near-Miss 2 and Near-Miss 3. This collection of methods uses K-nearest neighbour to pick out examples from the majority class. Near-Miss 1 works by selecting examples from the majority class with the lowest average distance from the three nearest examples of the majority class. With this algorithm's implementation, the user chooses the version by entering it in the GUI, with the subsequent algorithm being run on the dataset.

Tomek Links Undersampling - *SamplingMethods.tomekLinks*

A tomek link is a pair of examples from different classes who are their own nearest neighbour. They are often found along the class boundary, being misclassified examples. The examples in the majority class are deleted when used in undersampling.

One-Sided Selection - *SamplingMethods.OSS*

This is a combination of Tomek links and Condensed nearest neighbour. Both these methods target different parts of the dataset. Tomek Links undersampling removes examples at the edge of the dataset, whereas CNN is used to remove examples at the interior of the dataset grouping.

Neighbourhood Cleaning Rule - *SamplingMethods.NCR*

This is a combination of Condensed nearest neighbour and Edited nearest neighbour. The CNN removes examples from the interior of the dataset, and ENN removes the noisy outliers in the dataset.

Random Oversampling - *SamplingMethods.randomOversampling*

Random Oversampling works by randomly duplicating examples from the minority class in the training set. Altered X and y values are returned on the completion of the function.

Synthetic Minority Oversampling Technique (SMOTE) - *SamplingMethods.Smote*

Smote is one of the most widely known and commonly used sampling techniques for imbalanced learning. SMOTE works by generating synthetic examples of the minority class. The advantage here is that we are not duplicating examples but creating new ones. The Algorithm works by “selecting examples that are close in the feature space, drawing a line between the examples in the feature space and drawing a new sample as a point along that line”[15].

BorderLineSMOTE - *SamplingMethods.borderlineSmote*

This variant of SMOTE detects borderline samples and is then used to generate new samples. Therefore it is only generating samples that are difficult to classify. The way that borderline samples are selected is through a k-nearest neighbour classification model.

ADASYN - *SamplingMethods.adasyn*

ADASYN is a variation of SMOTE that generates synthetic samples inversely proportional to the density of the examples in the minority class. The reasoning for this is that it aims to create synthetic samples where the density of minority class samples is low and avoid the creation of examples where there is a high density.

4.5.2 Learning Algorithms

A collection of learning algorithms were implemented, not all associated with imbalanced learning. This is so that users can get an accurate and unbiased view of how sampling techniques on their own and in combination with learning algorithms affect results.

Non Imbalanced learning algorithms:

- Logistic Regression
- K-nearest neighbour
- SVM
- Linear Discriminant Analysis
- Naive Bayes

Weighted Logistic Regression - *CostModels.WeightedLR*

Standard Logistic Regression does not take into account imbalanced data. So to modify the algorithm to take this into account, a class weighting configuration is specified. This weighting configuration is used to change how much the logistic regression coefficients are updated.

Weighted Decision Tree - *CostModels.WeightedTree*

At the split points of decision trees, often in an imbalanced dataset, the minority class is ignored. So to overcome this problem, the importance of the minority class is more heavily weighted. This means that the model will know to value the minority class higher than that of the majority class at the split-point in a decision tree.

Cost-Sensitive SVM - *CostModels.WeightedSVM*

The difference between normal SVM and cost-sensitive SVM is that the weighting of the class is changed, meaning that the SVM weighs the margin proportional to the class importance. The margin is the distance between the classification boundary and the closest training set point[16]

Cost-Sensitive XGBoost - *CostModels.weightedXGBoost*

XGBoost is a decision-tree based ensemble algorithm that uses gradient boosting. XGBoost offers a wide range of hyperparameters. One of these parameters is “scale_pos_weight”, which weighs the balance of minority examples to majority examples when boosting the decision trees. [17]

Bagging Classifier - *CostModels.baggingClassifier*

Bagging, also known as bootstrap aggregation, involves selecting random samples from the training dataset. In bagging, a random sample of data in a training set is selected with replacement, meaning

that the individual data points can be chosen more than once. After this has been done several times, these weak models are trained, and the majority of those predictions yield a more accurate estimate.

Random Forest - *CostModels.randomForest*

Random forest is an extension of the bagging classifier, using both bagging and feature randomness to create an uncorrelated forest of decision trees. To tune the random forest for imbalanced data, we alter the “class_weight” argument. This weight has an impact when calculating the impurity score at a split point.

EasyEnsemble - *CostModels.easyEnsemble*

[Exploratory Undersampling for Class-Imbalance Learning, 2008.]

The classifier is an ensemble of ‘AdaBoost’ learners trained on different balanced bootstrap samples. The balancing is achieved by random under-sampling. This means that samples that are “difficult to classify receive increasingly larger weights until the algorithm identifies a model that correctly classifies these samples”.^[16]

One-Class SVM - *CostModels.One_ClassSVM*

One-Class classification involves fitting a model on data and then predicting whether any new data is normal or an anomaly. With a One-Class SVM When modelling one class, the algorithm captures the density of the majority class and classifies examples on the extremes of the density function as outliers.

Isolation Forest - *CostModels.iForest*

Isolation Forests are similar to Random Forests as they are both built on the bedrock of decision trees. They are an ensemble of binary decision trees. Randomly sub-sampled data is processed in a tree structure in an Isolation Forest based on randomly selected features. The samples that travel deeper into the tree are less likely to be anomalies as they require more cuts to isolate them.

Minimum Covariance Determinant - *CostModels.MCD*

This algorithm is usually applied to Gaussian-distributed data. It is unlikely to have data distributed so power laws are applied to the dataset that the algorithm will be working on. This approach can be

generalised by defining an ellipsoid covering the normal data. Any data that falls outside of this is considered to be an outlier.

Local Outlier Factor - *CostModels.LOF*

Local outlier Factor tries to identify outliers by using the nearest neighbour principle. Based on the size of its local neighbourhood, each example is assigned a score. The larger the score, the more likely that it is an outlier.

4.5.3 General Functions

Pre-process Data - *processData*

Several pre-processing methods are available to choose from: Rescale, Standardize, Normalize, and Binarize. With rescaling, all attributes of the dataset are rescaled to have the same scale.

Standardisation is useful when transforming attributes with a Gaussian distribution and differing means and standard deviations to a normal Gaussian distribution with a mean of 0 and a standard deviation of 1. If Normalize is selected, it means that each observation is rescaled to have a length of 1. Finally, with Binarize, the data is transformed with a binary threshold; values above the threshold are given a value of 1; if below it, they are given a value of 0.

5.0 Testing

5.1 Testing The GUI

Testing of the GUI was facilitated with the use of PyQt5's QTest unified with the standard Python unit test. The tests that were developed were to test the functionality of the GUI and that all widgets implemented worked correctly. The testing can be ran through the GUI testing suite in the code, below is a general outline of tests covered:

Name	Open Login Page
Steps	Click 'Create User' Button / click 'Login' Button
Outcome	User is taken to Login Page

Name	Open Open Help Pop-up
------	-----------------------

Steps	Click 'Help' Button
Outcome	User sees pop up containing helpful information

Name	Exit application
Steps	Click 'Exit' button
Outcome	Application closes

Name	Open Data Page
Steps	Click "Let's Go" Button / Click "Data Button"
Outcome	User is taken to Data screen

Name	Open Sampling Page
Steps	Click "Next" Button on Data screen / Click "Sampling" Button
Outcome	User is taken to Sampling Page

Name	Open Algorithm Page
Steps	Click "Next" Button on Sampling screen / Click "Algorithm" Button
Outcome	User is taken to Algorithm screen

Name	Combo boxes are populated with appropriate data
Steps	Click dropdown and see populated values
Outcome	User can see values in dropdown

Name	Results and Graphs are displayed correctly
Steps	Click "Enter" Button on Algorithm Screen
Outcome	User is taken to results screen Labels are populated with results Both graphs are visible

5.2 Testing The Database

Unit tests were developed to test the functions surrounding the database. These unit tests can be run in the Database testing suite in the code. These tests will ensure that the methods in *DatabaseCommand.DatabaseConnector* work appropriately. Below is a brief outline:

Function	create_connection
Expected Result	Connection to database is established
Pass/Fail	Pass

Function	create_table
Expected Result	Tables are created for user and their results
Pass/Fail	Pass

Function	create_new_user
Expected Result	Details of a new user are passed to the database
Pass/Fail	Pass

Function	find_user
Expected Result	Users details are returned from the database
Pass/Fail	Pass

Function	find_results
Expected Result	Users results from previous runs are returned according to what name was entered
Pass/Fail	Pass

5.3 Testing the rest of the code

As with the previous testing of the GUI and the calls to the database, unit testing was adopted. Unit testing was adopted as it was the simplest to implement and tested that each individual part was working as expected. For the creation of the model and the creation of results, integration testing was used to connect the multiple parts that would have to be called. These tests can be viewed by running the testing suite `Testing/user_test`. A brief outline of tests applied is below:

Goal	Apply Sampling to Dataset
Expected Result	Dataset is altered by the sampling that is performed

Pass/Fail	Pass
-----------	------

Goal	Apply Algorithm to Dataset
Expected Result	Model is created from the learning algorithm trained on the dataset supplied
Pass/Fail	Pass

Goal	Read Dataset from CSV
Expected Result	Dataset is converted into a pandas dataframe
Pass/Fail	Pass

Goal	Generate Results
Expected Result	Results are generated based on the model supplied and the train dataset
Pass/Fail	Pass

Goal	Apply Pre-processing to Dataset
Expected Result	Pre-processing is applied to the dataset and it is altered
Pass/Fail	Pass

Goal	Apply feature-selection to Dataset
Expected Result	Feature selection is applied to the dataset
Pass/Fail	Pass

Goal	Change Hyper-parameters of Sampling
Expected Result	Hyper-parameters are altered
Pass/Fail	Pass

Goal	Change Hyper-parameters of Algorithm
Expected Result	Hyper-parameters are altered
Pass/Fail	Pass

5.4 Conclusion

Multiple datasets must be available to choose from	Requirement Met
Ten or more sampling methods must be available to choose from	Requirement Met
Fifteen or more learning algorithms must be available to choose from	Requirement Met
The metrics Accuracy, precision, F-measure, recall, ROC curves and Precision-Recall curves must be available to view after a successful run	Requirement Met
Users must be able to preprocess the datasets if required using the following techniques: Normalization, Binarization, Rescaling, Standardization.	Requirement Met
Users must be able to use Feature selection on the dataset with the following options: PCA, Univariate selection, Recursive Feature Elimination.	Requirement Met

Looking back to the requirements set in 2.2 and 2.4 of this document, we can see that the testing has covered these key requirements and functions that have been set out. The tests that have been created make sure that all functionality that is required of the system is met.

6.0 System Evaluation

6.1 Evaluation of Project

Reflecting on the initially set out requirements, the system performs moderately well. It offers users a chance to explore and experiment with many ways of creating machine learning models from imbalanced data. The user can build their chosen model and quickly see results for it, which also have some meaning to how effective it is (ROC Curves and Precision/Recall Curves). There is a wide range of Sampling methods and Learning algorithms available to the user, showing them various possible techniques. Comparing different models' results is very useful, allowing the user to see the benefits and costs of using specific techniques.

The UI developed was reasonably easy to use. There is no barrier to someone being able to pick up and run the software from scratch. The interface allows users from all backgrounds, whether technical or not, to create and evaluate machine learning models based on imbalanced data. With the UI, the display of results was made accessible and easy to see in an understandable way. The many customisation options offered to the user throughout the model building and result generating process meant that users could, for example, tune algorithms with relative ease.

However, there are some fundamental limitations and drawbacks to the system developed. The first is the granularity offered. Although the user has plenty of options available with the system, there is no match compared to if the user implemented the code themselves. Many hyperparameters can be tinkered with and changed and the system could not accommodate all of these. An especially technical user may find the system quite limiting, not offering them the flexibility and customisation they may want. Another limiting factor is the ability only to choose those sampling and learning methods that are built into the system. If the user wished to add an algorithm and test it, it would not be possible. Some users may also find that specific algorithms they want to try are not available, like those that are more computationally heavy, such as Deep Learning algorithms. Finally, many of the issues surrounding imbalanced data are found in massive datasets or "Big Data" as it is known. The system does not have the capacity to test the algorithms on such large quantities of data, omitting a large area of imbalanced learning. The reason for this is simply the timeframe involved in training models on such large amounts of data. It would not be feasible to create such a system and expect it to be as lightweight as the one in this project.

6.2 Comparison to Other Products

Several common datasets are used to benchmark imbalanced learning, such as the Pima Indians dataset, but no suite has been created as of yet to act as a comparison to this project. However, there are other programs used for machine learning that are used to benchmark that we can compare to.

Penn Machine Learning Benchmark (PMLB) Comparison

PMLB [18] is a benchmark resource to “facilitate the identification of the strengths and weaknesses of different machine learning methodologies”. Compared with this benchmarking tool, there is a stark contrast to the one created with this project. There are three types of data used to benchmark “Real-world” data, “Synthetic data”, and “Toy data”, whereas only “Real-world” data is used in this dissertations project. The suite has 165 datasets that are all standardised and also provides interfaces for bringing datasets from the web. This is a vast and extensive amount of data to compare algorithms, highlighting the difficulty in gaining accurate insights from the small number of datasets used in this project. The PMLB does not have a front end GUI but does display graphical results. It requires a much more technical user and offers a far more extensive suite attracting those in research and industry with the skills to implement it. In comparison, the system created here for imbalanced data does not come close to the evaluation offered to users of PMLB. However, it remains accessible to those of a less technical background with the GUI interface.

6.3 Future enhancements

The developed system has some reasonably straightforward enhancements that could be made. Adding deep learning algorithms to the collection of learning algorithms would be the first step. Allowing a user to customise a deep learning algorithm fully may prove difficult but is definitely feasible. For the more technical user, the code should be refactored to work as a software framework that can be imported into python and accessed with the command line without using the GUI. With this framework, there could be an added level of granularity offered to the user. The number of datasets could be expanded to yield more reliable comparisons between models. If it was possible to leverage cloud computing, we could speed up processing times for creating models, thus processing larger datasets.

References

<https://gitlab2.eecs.qub.ac.uk/40235348/tsm01---dissertation-project.git>

- [1] H. He and X. Shen, "A Ranked Subspace Learning Method for Gene Expression Data Classification," *Proc. Int'l Conf. Artificial Intelligence*, pp. 358-364, 2007
- [2] M. Kubat, R.C. Holte, and S. Matwin, "Machine Learning for the Detection of Oil Spills in Satellite Radar Images," *Machine Learning*, vol. 30, nos. 2/3, pp. 195-215, 1998.
- [3] R. Pearson, G. Goney, and J. Shwaber, "Imbalanced Clustering for Microarray Time-Series," *Proc. Int'l Conf. Machine Learning, Workshop Learning from Imbalanced Data Sets II*, 2003.
- [4] Y. Sun, M.S. Kamel, and Y. Wang, "Boosting for Learning Multiple Classes with Imbalanced Class Distribution," *Proc. Int'l Conf. Data Mining*, pp. 592-602, 2006.
- [5] N. Abe, B. Zadrozny, and J. Langford, "An Iterative Method for Multi-Class Cost-Sensitive Learning," *Proc. ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining*, pp. 3-11, 2004.
- [6] K. Chen, B.L. Lu, and J. Kwok, "Efficient Classification of Multi-Label and Imbalanced Data Using Min-Max Modular Classifiers," *Proc. World Congress on Computation Intelligence—Int'l Joint Conf. Neural Networks*, pp. 1770-1775, 2006
- [7] Z.H. Zhou and X.Y. Liu, "On Multi-Class Cost-Sensitive Learning," *Proc. Nat'l Conf. Artificial Intelligence*, pp. 567-572, 2006.
- [8] Guo Haixiang, Li Yijing, Jennifer Shang, Gu Mingyun, Huang Yuanyue, Gong Bing, *Learning from class-imbalanced data: Review of methods and applications, Expert Systems with Applications*, Volume 73, 2017, Pages 220-239, ISSN 0957-4174
- [9] On developing robust models for favourability analysis: Model choice, feature sets and imbalanced data, P.C. Lane, D. Clarke and P. Hender, *Decision Support Systems*, 53 (4) (2012), pp. 712-718
- [10] Classifying imbalanced data sets using similarity based hierarchical decomposition, C. Beyan and R. Fisher, *Pattern Recognition*, 48 (5) (2015), pp. 1653-1672
- [11] G.M. Weiss, H. Hirsh Learning to predict extremely rare events AAAI workshop on learning from imbalanced data sets (2000)
- [12] J.F. Díez-Pastor, J.J. Rodríguez, C.I. García-Osorio, L.I. Kuncheva Diversity techniques improve the performance of the best imbalance learning ensembles *Information Sciences*, 325 (2015), pp. 98-117

- [13] A. Toxboe and A. Toxboe, "Lazy Registration design pattern", *Ui-patterns.com*, 2022. [Online]. Available: <https://ui-patterns.com/patterns/LazyRegistration>.
- [14] Gamma, E.; Helm, R.; Johnson, R. & Vlissides, J. M. (1994), Design Patterns: Elements of Reusable Object-Oriented Software , Addison-Wesley Professional
- [15] J.Brownlee (2021) Imbalanced Classification in Python [Online] Available: <https://machinelearningmastery.com>
- [16] Kuhn, M. and Johnson, K., 2013. *Applied predictive modeling* (Vol. 26, p. 13). New York: Springer.
- [17] "XGBoost Documentation — xgboost 1.6.0 documentation", *Xgboost.readthedocs.io*, 2022. [Online]. Available: <https://xgboost.readthedocs.io/en/stable/>.
- [18] Olson, R.S., La Cava, W., Orzechowski, P. *et al.* PMLB: a large benchmark suite for machine learning evaluation and comparison. *BioData Mining* **10**, 36 (2017).