University of the Philippines Manila

College of Arts and Sciences

Department of Physical Sciences and Mathematics

PanCan Spectrum: A Pancreatic Cancer Detection Support Tool Using Mass Spectrometry Data and Support Vector Machines

A special problem in partial fulfillment

of the requirements for the degree of

Bachelor of Science in Computer Science

Submitted by:

Emmanuel A. Briones

June 2019

Permission is given for the following people to have access to this SP:

Available to the general public	Yes
Available only after consultation with author/SP adviser	No
Available only to those bound by confidentiality agreement	No

ACCEPTANCE SHEET

The Special Problem entitled "PanCan Spectrum: A Pancreatic Cancer Detection Support Tool Using Mass Spectrometry Data and Support Vector Machines" prepared and submitted by Emmanuel A. Briones in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science has been examined and is recommended for acceptance.

Geoffrey A. Solano, Ph.D. (candidate)
Adviser

EXAMINERS:

	Approved	**Disapproved**
1. Gregorio B. Baes, Ph.D. (candidate)		
2. Avegail D. Carpio, M.S.		
3. Richard Bryann L. Chua, Ph.D. (candidate)		
4. Ma. Sheila A. Magboo, M.S.		
5. Vincent Peter C. Magboo, M.D., M.S.		
6. Marbert John C. Marasigan, M.S. (candidate)		

Accepted and approved as partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science.

Vincent Peter C. Magboo, M.D.
Unit Head
Mathematical and Computing Sciences Unit
Department of Physical Sciences
and Mathematics

Marie Josephine M. De Luna, Ph.D.
Chair
Department of Physical Sciences
and Mathematics

Leonardo R. Estacio, Jr., Ph.D.
Dean
College of Arts and Sciences

Abstract

Pancreatic cancer is one of the most fatal types of cancer due to its difficulty of being diagnosed in the early stages. Presently, multiple screening procedures for this disease are required to determine its presence. In this study, a pancreatic detection support tool implementing machine learning is to be created with support vector machines (SVM) algorithm and mass spectrometry data of pancreatic cancer patients and controls as training and testing datasets. The final output would aid researchers in detecting pancreatic cancer in patients (complementing current and common procedures), and in finding biomarkers of the disease.

Keywords: mass spectrometry, machine learning, pancreatic cancer detection, biomarkers, support vector machines

Contents

List of Figures

List of Tables

I. Introduction

A. Background of the Study

Pancreatic cancer is a type of disease in which malignant cells form in the tissues of the pancreas. It is the 12th most common type of cancer with around 460,000 cases in 2018 [1] and is also one of the deadliest with an estimated survival rate of 9% [2]. There are several pancreatic cancer risk factors. Those that are lifestyle-related include smoking, obesity, being overweight, and workplace exposure to certain chemicals; while the other factors are family history, inherited gene syndromes, diabetes, and age. The risk of developing pancreatic cancer increases as people get older as studies show that almost all patients are older than 45 [3].

In pancreatic cancer diagnosis, there are multiple tests that are administered by oncologists in order to detect the presence of the disease and determine the extent of the condition of a particular patient [4]. Some examples of these tests are Magnetic Resonance Imaging (MRI), Computed Tomography (CT) Scan, Ultrasound, Cholangiopancreatography, blood tests, and Positron Emission Tomography (PET) Scan.

In the present, information technology and computer science play a major role in numerous fields including medicine, biochemistry, and toxicology. Many studies and researches have integrated techniques such as machine learning algorithms, statistical methods, and scientific processes in the identification and classification of diseases, including cancer. Among these techniques is mass spectrometry or mass spectroscopy.

Mass spectrometry (MS), or mass spectroscopy, is an analytical, instrumental technique used for the process of separating electrically charged species in a given biological sample using a mass spectrometer (which then produces a mass spectrum of mass/charge values with corresponding intensities) [5]. This particular technique has a lot of applications in the medical sciences. Some of the common uses of mass spectrometry are the diagnosis of various diseases and identification of the corresponding biomarkers. Furthermore, it has been greatly recognized due to its applications in drug safety evaluation and diagnosis of diseases [6].

B. Statement of the Problem

Pancreatic cancer detection can be difficult since the disease does not usually cause many specific symptoms in the early stages. Symptoms such as loss of appetite, indigestion, jaundice and abdominal pain can also be caused by more common illnesses such as pancreatitis, irritable bowel syndrome, and gallstones [7].

In the present, pancreatic cancer detection is based only on the clinicopathological attributes of a patient. Moreover, patients undergo several procedures such as ultrasound scans, CT scans, biopsies, MRI scans in order to detect the presence of pancreatic cancer. Although there exist many tests for diagnosing pancreatic cancer, no major professional group recommend routine screening for pancreatic cancer in patients who are at average risk because no screening test has been proven to lower the risk of death from pancreatic cancer [8]. This study aims to make the detection process easier and faster but at the same time reliable through the use of mass spectrometry data and machine learning. The results of the final system can also be used in the validation of other detection procedures mentioned beforehand.

Furthermore, the study seeks to answer the following questions:

- What are the common mass-per-charge values with high intensities found in the mass spectra of pancreatic cancer patients?
- Which chemical compounds (based on the mass/charge ratio values in a mass spectrum) are most commonly associated to pancreatic cancer?

- How are the mass-per-charge values associated to pancreatic cancer identified?

C. Objectives of the Study

This study aims to create a pancreatic cancer detection support tool which implements machine learning (SVM) using mass spectrometry data, in order to aid researchers in determining whether a patient has pancreatic cancer based on his or her mass spectrum.

- The researcher should be able to:
  - Input and file parse mass spectrometry data using the tool
  - Pre-process mass spectrometry data and perform the following:
    - Baseline Reduction
    - Smoothing
    - Data Normalization
    - Peak Reduction
    - Peak Alignment
  - View visualization of inputted mass spectrometry data
  - View corresponding numerical values of mass spectrum attributes
  - Identify the most abundant chemical compounds in a patient's sample based on the intensities of the mass/charge values of a spectrum
  - Perform pancreatic cancer detection using mass spectrometry data
  - Download report of detection result in text format
  - View user's manual

- The statistician should be able to:
  - Input and file parse mass spectrometry data using the tool
  - Pre-process mass spectrometry data and perform the following:
    - Baseline Reduction
    - Smoothing
    - Data Normalization
    - Peak Reduction
    - Peak Alignment
  - View visualization of inputted mass spectrometry data
  - View corresponding numerical values of mass spectrum attributes
  - Add inputted data as training and testing dataset to create a classification model
  - View user's manual

D. Significance of the Project

The proposed system would enable oncologists to detect early stages of pancreatic cancer using mass spectrometry data acquired from biological samples extracted from patients. Moreover, biomarker identification for pancreatic cancer based on mass spectra mass-per-charge values with high intensities can be done.

The application of machine learning on mass spectrometry would help researchers in their discovery of new information regarding chemical compounds that are present in patients diagnosed with pancreatic cancer, extending their knowledge with regard to diagnostics of the disease hence giving vital contribution both to the scientific and medical community.

Furthermore, the system to be developed would serve as an aid for those working with mass spectrometry data, seeing that they could utilize it with regard to the establishment of hypotheses regarding the correlation among numerous chemical compounds and pancreatic cancer, based on the system's outputted results. Ultimately, the system would lessen the work of oncologists with regard to running multiple screening tests and it would help them decide with regard to a patient's treatment.

E. Scope and Limitations

The limitations of the system are the following:

- The model would be trained based only on mass spectrometry data of pancreatic cancer-diagnosed patients, and of controls who do not have the disease.
- The output of the system does not indicate the type of pancreatic cancer based on the mass spectrometry data.
- Evaluation of the SVM model is based on accuracy, precision, recall, and f1-score after cross validation.
- The included mass spectrometry data pre-processing steps are to be used based on user's own selection.
- The tables in the chemical compounds database cannot be updated.

F. Assumptions

The assumptions in the study are the following:

- The datasets used for creating the machine learning model (training and testing) are acquired from the Global Natural Products Social Molecular Networking (GNPS)

MassIVE Database and the PRoteomics IDEntifications (PRIDE) Archive, all of which are provided by researches and professionals working on studies related to mass spectrometry and other analytical techniques that involve natural products.

- The mass spectrometry datasets vary with regard to what type of mass spectrometer was used for acquisition.
- The data including the identification of chemical compounds are acquired from the Chemical Entities of Biological Interest (ChEBI) database, which is also contributed to by research professionals.
- The mass spectrometry data to be used for the system are in *mzml* format, which is the standard format for mass spectra (mass spectrometer output).
- The types of pancreatic cancer in the mass spectrometry dataset used for creating the machine learning model are not considered.
- The mass spectrometry data to be used are pre-processed using the tool before proceeding to the disease detection functionality.

II. Review of Related Literature

Some methodologies, researches, and concepts that relate to the proposed system would be discussed in this section. In brief, the following contains review of literature concerning mass spectrometry, machine learning, and classification algorithms.

Mass spectrometry is an analytical technique for the characterization of biological molecules and is increasingly used due to its targeted, nontargeted, and high throughput abilities. Machine learning has been applied to mass spectrometry data in the past from different biological disciplines, particularly for various cancers. The objectives of such investigations have been to identify biomarkers and to aid in detection, prognosis, and treatment of particular diseases [9].

In 2013, Swan et al. discussed the use of machine learning applied to proteomics data for identification of biomarkers. It was shown that there are several algorithms suitable for classification of samples and identification of biomarkers (i.e. Support Vector Machines, Decision Trees, Bayesian classifiers, Artificial Neural Networks, etc.), hence a study requiring the optimal method based on the objective and available resources involves the consideration of several important matters. These include the quantity of data to be used for training, the type of data may it be mass spectral peak data or identified proteins, and whether biomarker identification is needed besides unknown sample classification [9].

A study performed by Rajapakse et al. in 2005 shows that there are several basic steps involved before machine learning could be applied to mass spectrometry data with regard to cancer classification and biomarker discovery. First involves the collection and analysis of mass spectrometry data, then followed by baseline subtraction and noise reduction, which are necessary before peak extraction and alignment could be performed. Once the datasets have been prepared, partitioning into training and test sets is done. Afterwards, the peak selection on training sets and the classification and validation is finally executed [10].

A typical data set used in a clinical application of mass spectrometry contains tens or hundreds of spectra; each spectrum contains thousands of intensity measurements representing an unknown number of protein peaks. Any attempt to make sense of this volume of data requires extensive low-level processing in order to identify the locations of peaks and to quantify their sizes accurately. Inadequate or incorrect preprocessing methods, however, can result in data sets that exhibit substantial biases and make it difficult to reach meaningful biological conclusions [11].

In 2006, Smith et al. introduced a tool called XCMS (various forms (X) Chromatography Mass Spectrometry) which incorporates the needed steps in MS data pre-processing mentioned. The three highlighted aspects of XCMS are its availability (being open-source), design, and flexibility. Moreover, it includes easily integrated tools for MS data pre-processing, analysis, and can be customized and tweaked for varying MS data analysis purposes or optimized for a specific application. Lastly, even if XCMS has been designed for LC/MS data, it could be modified to be able to use other data types [12].

One of mass spectrometry's strengths is its potential for biomarker discovery, which is one of the reasons why it has been used in medical and biochemical research for a long time [13]. Despite its potential, it is recognized that the acquisition of significant proteomic features from mass spectrometry data requires precise assessment. Consequently, a good feature selection method is needed.

A research done by Datta in 2013 involved the use of several algorithms including Random Forests (RF), Support Vector Machine (SVM), Partial Least Squares (PLS), Linear and Quadratic Discriminant Analysis (LDA and QDA, respectively). A performance comparison for the five algorithms was provided and the results show that PLS has the highest overall accuracy and sensitivity, while SVM has the highest specificity or the ability to identify negative results (True Negative). Furthermore, it was stated that SVM has the advantage over other classifiers when it comes to flexibility, and has low classification error rates and robustness to several types of features [13].

Another study that involved the use and comparison of several machine learning techniques for feature selection was done by Ahmed et al. in 2013. The mainly proposed approach is the use of genetic programming (GP) for both feature selection and classification of mass spectrometry data. In this approach, two feature selection metrics are utilized: Information Gain (IG), which dictates the total amount of information gained with regard to a class when a particular feature existent or not; and Relief-F (REFS-F), which locates the two closest neighbors for a specific example: one from the same class (hit) and the other from a different class (miss); and then computes the value of the feature. Subsequently, the performance of the proposed method was analyzed and contrasted with IG and REFS-F on five mass spectrometry datasets with varying number of instances and features. Naïve-Bayes (NB), Support Vector Machines (SVM), and J48 Decision Trees (J48) are used to calculate the selected features' classification accuracy. It was concluded in the study that Genetic Programming as a feature selection method can pick out lesser number of features with better classification accuracy than IG and REFS-F using Naïve-Bayes, Support Vector Machines, and J48 DT. Moreover, GP surpasses the performance of NB and J48 as a classification method, and has minorly better performance than SVMs on the data sets used [14].

A study research done by Kim et al. in 2014 involved the use of various classification methods such as Random Forests, Bagging, Lasso, and Classification and Regression Tree (CART). Using mass spectrometry data of blood samples taken from patients with diabetes and pancreatic cancer, they were able to identify biomarkers linked to the diseases. Their methodology included the typical MS data pre-processing, use of various classification algorithms, and cross validations [15].

In 2015, Nguyen et al. proposed a hybrid feature extraction method for cancer diagnosis using mass spectrometry data. After using Haar wavelets as features to convert mass spectrometry data into wavelet coefficients, genetic algorithm (GA) is applied to acquire feature sets from the most prominent wavelets. Using a variety of performance metrics such as accuracy, F-measure, and area under the curve (AUC), the Wavelet-GA methodology was shown to hold a significant advantage compared to other feature extraction methods such as Wilcoxon Test, t-test, principal component analysis (PCA), sequential search, etc. with regard to robustness [16]. Furthermore, a cross-validation was applied to three benchmark mass spectrometry datasets in the study, validating the conclusions that were formulated from the study. In conclusion, the Wavelet-GA can be implemented to create classification models that are to be used by researchers and medical experts for decision support systems in the field of oncology.

Another study that includes several feature selection techniques was done by Wong et al. in 2008. The researchers used Student t test, Wilcoxon rank sum test, and genetic algorithm to reduce the high dimensionality of pancreatic cancer MS data. With the features selected from each method, the performances of decision-tree based classifier ensembles were compared with that of a single decision-tree algorithm. The results show that classifier ensembles have better accuracies compared to single decision-tree [17].

Aside from the accuracy rate of feature selection and classification algorithms, another factor to be considered is the robustness to noise and outliers in a specific mass spectrometry dataset. In 2006, Zhang et al. developed a recursive support vector machine (R-SVM) algorithm to be used for the selection of biomarkers that are significant for classification of noisy data. The performance of the algorithm was compared to SVM Recursive Feature Elimination (SVM-RFE), focusing on both the algorithms' ability to mark truly informational biomarkers, and the robustness to the mass spectrometry dataset's outliers. After conducting the research, it was found out that R-SVM had 5%-20% better performance than SVM-RFE with regard to the features aforementioned. The R-SVM algorithm was subsequently implemented to two proteomics datasets: one regarding a breast cancer study and another from a research on rat liver cirrhosis. After the biomarkers have been found and validated, it was stated that the R-SVM algorithm is appropriate for evaluating both proteomics and microarray data which contains a relatively high amount of noise [18]. Furthermore, the proposed method performs better than SVM-RFE when it comes to gathering informative feature in the data and robustness to data noise.

Also emphasizing an algorithm's robustness to noise is a study by Pham et al. in 2011. The proposal involves another feature extraction method that considers the noise in a particular mass spectrometry data. The method used in the study integrates stationary wavelet transformation (SWT) and bivariate shrinkage estimator for denoising and feature extraction from MS data. Subsequently, two types of statistical feature testing, including Kolmogorov–Smirnov (KS) test and Mann–Whitney U (MW) test, are implemented on denoised wavelet coefficients in order to correctly pick the significant features that would then be made use of for identification of biomarkers. For method performance evaluation with regard to cancer classification, the researchers used a double-cross validation SVM classifier, accentuated to have high generalizability; and a Modest AdaBoost classifier which has a significantly better runtime. After application of both methods on Matrix-assisted laser desorption/ionization—time-of-flight (MALDI-TOF) datasets, the results of the research show that although the proposed method has better runtime than SVM, the latter still outperforms the former when it comes to sensitivity (True Positive) and specificity (True Negative) rates [19].

III. Theoretical Framework

A. Definition of Terms

1. **Mass Spectrum** - an intensity vs. mass-to-charge (m/z) plot of a chemical sample, acquired using a mass spectrometer [20].

2. **Ionization** – the process in which a molecule or an atom gains a positive or negative charge through losing or gaining electrons to form ions, usually happening with other chemical changes [21].

3. **Biomarker** – measurable substance from a patient which shows indications of a particular medical state, may it be a disease, infection, abnormal condition, etc. [22].

B. Pancreatic Cancer

Pancreatic cancer is a type of disease in which cancer cells form in the tissues of the pancreas [4]. It begins in the tissues of the pancreas — an organ the human abdomen that lies horizontally behind the lower part the stomach. The pancreas releases enzymes that aid digestion and hormones that help blood sugar levels [23].

Pancreatic cancer typically spreads rapidly to nearby organs. It is seldom detected in its early stages, but for people with pancreatic cysts or a family history of pancreatic cancer, some screening steps might help detect a problem early [23]. This type of disease is also considered to be a silent one since there are not many noticeable symptoms in its early stages. As the cancer develops, symptoms may include [24]:

- Yellow skin and eyes, darkening of the urine, itching, and clay-colored stool, which are signs of jaundice caused by a blockage of the bile ducts

- Pain in upper abdomen or upper back

- Burning feeling in stomach or other gastrointestinal discomforts
- Loss of appetite
- Nausea and vomiting
- Unexplained weight loss

Currently, there are no proven biomarkers that could make the early detection of pancreatic cancer more efficient. Nevertheless, researchers are striving to discover biomarkers that could indicate whether a person may have undiagnosed pancreatic cancer [25]. Many of the projects aimed at finding biomarkers to detect pancreatic cancer earlier focus on blood samples, comparing blood from patients who have the disease to healthy individuals.

There are multiple tests that are administered to patients in order to detect pancreatic cancer. These include [26]:

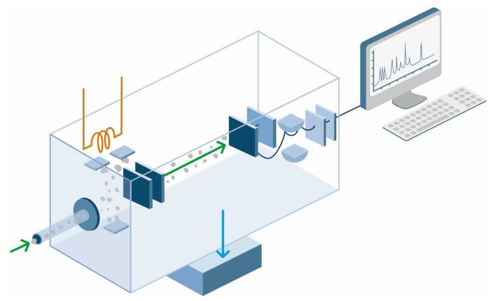
- Imaging tests such as Computerized Tomography (CT) scans, Magnetic Resonance Imaging (MRI) and Positron Emission Tomography (PET) scans
- Endoscopic Ultrasound (EUS)
- Removing a tissue sample for testing (biopsy)
- Blood tests

After a patient has been diagnosed with pancreatic cancer, he/she may undergo the following treatments [27]:

- Chemotherapy or Radiation Therapy
- Whipple procedure (pancreaticoduodenectomy)
- Distal pancreatectomy
- Total pancreatectomy

C. Mass Spectrometry

Mass spectrometry (MS), or mass spectroscopy, is an analytical type of instrumental technique used for the process of separating electrically charged species in a particular sample, using a mass spectrometer [28].



The mass spectrometry process can be sectioned into three main parts: the ionization, the analyzation, and the detection [5]. A particular sample is put into the ionization source of the mass spectrometer. Subsequently, once in the ionisation source, the molecules of the sample are ionized. This is done because ions are relatively easier to manipulate compared to neutral (neither positively or negatively charged) molecules. Afterwards, the ions are extracted into the analyzer part of the mass spectrometer in which the process of separation according to mass-to-charge (m/z) ratio is done. Lastly, the categorized ions are then detected and each signal (peak) is sent to a data system such that the m/z ratio data are inputted along with their corresponding relative abundance, all in a plot format called a mass spectrum.

A mass spectrum can be viewed as a histogram which provides data on the quantity of ions at varying values of mass-to-charge ratio (m/z) [28]. The detected ions indicate the presence of molecules or other chemical species developed over the process of ionization. Hence, mass spectrometry allows the identification of molecules based on the mass-to-charge ratio and its corresponding intensity, as well as the fragmentation patterns.

Mass spectrometry has a lot of applications in the fields of medicinal science. In medicine, some of the common uses of mass spectrometry are the diagnosis of many types of diseases and identification of the corresponding biomarkers. The utilization of mass spectrometry, together with multiple types of chromatography, is also being continuously developed due to its applications in drug safety evaluation and diagnosis of diseases [6]. Furthermore, the technique is used in a wide scope of applications, ranging from cancer diagnostics to forensic toxicology [29].

D. Machine Learning

Machine learning (ML) is a core subdiscipline of artificial intelligence (AI) which focuses on algorithms that allow programs to be capable of learning and modifying their structure through learning based on a set of inputted data [30].

A machine learning algorithm has three components: Representation, which describes the way knowledge is represented (e.g. Support Vector Machines, Decision Trees, Neural Networks, Model Ensembles, etc.); Evaluation, concerning how programs are evaluated (e.g. accuracy, specificity, sensitivity, squared error, etc.); and Optimization. [31] Machine learning algorithms differ vastly from each other based on the way programs are represented and depending on the method of learning through a particular scope of programs [32].

Being a field between computer science and statistics, and at the core of data science and AI, machine learning has been a topic of great activity and interest in the field of biomedicine since they provide the possibility of improving the accuracy of disease diagnosis and early detection, whilst improving the process of decision-making with regard to a patient's status [30]. Machine learning provides methods for analyzing various types of datasets and extracting significant relationships and key characteristics in the data by developing models that best describes the given datasets [33].

E. Supervised Learning

Supervised learning, in the context of artificial intelligence (AI) and machine learning, is a type of machine learning in which both the input and desired output data are included in the training dataset [34]. The objectives in supervised learning are to create a model characterizing the class labels in terms of features found in a particular dataset [35]; and to approximate a mapping function such that when new input data is acquired, predicting the corresponding output data for it can be done [36]. Supervised learning may be categorized into regression or classification.

The defining property of supervised learning is the presence of labelled training datasets. Different types of algorithms under supervised learning generate learning models from these training datasets and the models generated are used to classify unlabeled data from new input datasets [37].

F. Classification in Machine Learning

In machine learning, classification is the process of identifying to which class a new input data belongs, based on a model created through learning from a corresponding training dataset of which the class of the included data is known [38]. Classes are sometimes referred to as categories, targets or labels [39].

A classification model aims to draw some generalization from input data values. Machine learning algorithms used in classification either predict categorical classes or classify data based on the training set and the values (class labels) in classifying attributes and uses it in classifying new data. [36] The resulting classifier model from training is used to assign class labels to the testing datasets of which the values of the predictor features are known, while the class label values are unknown [37].

G. MS Data Pre-processing

Mass spectrometry data, in the form of mass spectra, contain huge quantities of m/z and intensity measurements which represent a great number of peaks. In order to efficiently analyze MS data, some data pre-processing steps are required [11][40]. The steps and algorithms used for each are discussed in the subsequent sections.

1. Noise Reduction

Noise Reduction is defined as the removal of mass spectrometry data noise which may be chemical or electronic in origin. This step may be executed through smoothing or baseline reduction.

1..1 Smoothing

Smoothing is a pre-processing step in which data points are averaged with their neighbors in a series of data. The main reason for applying smoothing is to increase signal to noise ratio of a spectrum [11]. One of the most commonly used smoothing techniques is the **Savitzky-Golay Algorithm**. This algorithm involves performing a least squares fit of a small set of consecutive data points to a polynomial and take the calculated central point of the fitted polynomial curve as the new smoothed data point. A set of integers $(A_{-n}, A_{-(n-1)}, \ldots, A_{n-1}, A_n)$ could be derived and used as weighting coefficients to carry out the smoothing operation. The use of these weighting coefficients, known as convolution integers, turns out to be exactly equivalent to fitting the data to a polynomial, as just described and it is computationally more effective and much faster. Therefore, the smoothed data point $(y_k)_s$ by the Savitzky-Golay algorithm is given by the following equation:

$$(y_k)_s = \frac{\sum_{i=-n}^{i=n} A_i(y_{k+i})}{\sum_{i=-n}^{i=n} A_i}$$

1..2 Baseline Reduction

Baseline reduction involves the removal of the baseline slope and offset from a mass spectrum in order to flatten its base profile [11]. Moreover, this pre-processing step is considered to be a linear operation [41].

The baseline reduction step includes the following:

- Set the first and last m/z values of the spectrum as initial reference points
- Subtract the distance of the reference points from the baseline (offset value) from the other intensity values
- Select the troughs of the spectrum as new reference points
- Calculate the average offset value of the troughs and subtract from other intensity values except the initial reference points
- Repeat steps (except first) until the troughs are adjacent to the baseline

2. Data Normalization

Data normalization enables the comparison of different samples since the absolute peak values of different fraction of spectrum could be incomparable. Moreover, it removes sources of systematic variations between spectra due to varying amounts of sample or degradation over time in the sample or even variation in the instrument detector sensitivity [11]. Some techniques for normalization are:

- **Simple Feature Scaling**

  Let x_0 be an initial value, x_f a normalized value, and A_{max} the maximum value of an attribute. Normalization of x_0 to x_f is done using the following equation:

  $$x_f = \frac{x_0}{A_{max}}$$

  Each value is divided by the maximum among all values of a particular feature in a dataset. This makes the values range between zero and one.

- **Min-Max Normalization**

  Min-max normalization involves linear alteration on chosen initial data where the values are normalized within a given range [42]. Let x_0 be an initial value, x_f a normalized value, A_{min} the minimum value of an attribute, and A_{max} the maximum value of an attribute. For mapping a value x_0 of an attribute A from range A_{min}-A_{max} to a new range A_{newmin}-A_{newmax}, the computation for the transformation is given by:

  $$x_f = \left(\frac{x_0 - A_{min}}{A_{max} - A_{min}}\right) * (A_{newmax} - A_{new\_min}) + A_{new\_min}$$

- **Z-Score Normalization**

  Z-score normalization is one of the most commonly used data normalization techniques [43]. Let x_0 be an initial value, x_f a normalized value, μ_A the mean, and σ_A the standard deviation of the values of an attribute, respectively. The value x_f is obtained using the following equation:

  $$x_f = \left(\frac{x_0 - \mu_A}{\sigma_A}\right)$$

3. Data Reduction

This step is crucial for preserving raw data information while performing a dimensional reduction for subsequent processing [11]. The most common technique is **Binning**, which performs data dimensionality reduction by grouping measured data into ”bins” in order to combine a set of continuous values into a single value. The execution of this process involves the following [40]:

- Split mass spectrum into intervals (bins) of m/z values
- Calculate for each interval of m/z values(bin):
  - an aggregate intensity (e.g. the sum of the intensities in the bin)
  - a representative m/z value (e.g. the median of the one with maximum intensity)
- Replace each bin with the calculated representative m/z value

4. Peak Alignment

The peak alignment step is used to find out which peaks among different spectra refer to the same peak, since some mass spectra in a particular MS dataset could correspond to the same chemical compound.

After correcting the retention time shifts in every spectrum, the matching methodology done is as follows [44]:

- Select a reference sample s_R from sample set $S(s_1, s_2, s_3, ..., s_N)$ and denoting the remaining samples as $S' = s_i(i = 1, 2, 3, ..., N - 1)$
- The corresponding peak list of the reference sample is defined as the reference table (*RefTbl*), and the corresponding peak lists of the remaining samples are designated as search tables (*SchTbl*) numbered from $SchTbl_1$ to $SchTbl_{n-1}$. A variable m is initialized $m = 1$.
- Select the m^{th} search table $SchTbl_m$ from the search tables and matching each landmark peak in $SchTbl_m$ to the landmark peaks in $RefTbl$.

H. Support Vector Machines

Support Vector Machines (SVM) is a machine learning algorithm under supervised learning which is applicable for creating models for solving both classification and regression problems [45]. The objective of the SVM algorithm is based on finding the line decision boundary (hyperplane) that provides the largest distance to the closest data points known as support vectors [46]. There are many possible hyperplanes to choose from when separating data points into classes. Consequently, maximizing the margin distance results to a better model since future data values can be classified with higher confidence [47].

Compared to other classification algorithms, Support Vector Machines is less prone to data values which are considered to be outliers since it only regards the values that are nearest to the support vectors or decision boundary [48]. Moreover, SVM is eminently preferred by many due to its capability of producing significant accuracy using less computational power. Even though it can be used for both regression and classification tasks, it is usually used for the latter [47].

Given a labeled training dataset:

$$(x_1, y_1), ..., (x_n, y_n), x_i \in \mathbb{R}^d \text{ and } y_i \in (-1, +1)$$

where x_i is a feature vector representation and y_i is the class label of a training compound i. The hyperplane can be defined as follows:

$$\vec{w}\vec{x}^T + b = 0$$

where w is the weight vector, x is the input feature vector, and b is the bias. The w and b would satisfy the following inequalities for all elements of the training dataset:

$$\vec{w}\vec{x_i}^T + b \geq +1 \text{ if } y_i = 1$$

$$\vec{w}\vec{x_i}^T + b \leq -1 \text{ if } y_i = -1$$

The objective of training an SVM model is to find the $\vec{w}$ and b so that the hyperplane separates the data and maximizes the margin $\frac{1}{||\vec{w}||^2}$.

In the case that the dataset to be used for training is not linearly separable, the "kernel trick" may be used. The "kernel trick" enables the mapping of data to a higher dimensional space where it becomes linearly separable [49].

One the most prominently used kernels is the Gaussian Radial Basis Function (RBF) Kernel. An RBF is a general model used to build a mapping from $\mathbb{R}^n$ to $\mathbb{R}^m$ [50]. When using an RBF, one needs to choose:

- The training function, $\phi(r)$ that will be used. (Some choices: $\phi(r) = e^{\frac{-r^2}{\sigma^2}}$, $\phi(r) = r^3$)
- The model parameters for an RBF
  - The centers c_i, which are also in $\mathbb{R}^n$
  - The number of centers, k (independent of n or m)
  - The weights (or coefficients) $\omega_i, i = 1, 2, ..., k$.

After choosing the centers, the coefficients are obtained using a least squares solution:

- Given p data points in $\mathbb{R}^n$ and k centers in $\mathbb{R}^n$, form the $p \times k$ Euclidean Distance Matrix (EDM) A, such that

$$A_{ij} = dist(x^{(i)}, c^{(j)}) = ||x^{(i)} - c^{(j)}||$$

- Form the transfer matrix Φ: $\Phi = \phi(A)$. If biases are desired, add a column of 1's to the end of Φ, so that it is $p \times (k + 1)$.
- Solve the matrix equation:

$$\Phi \cdot W = Y$$

where W (which has size $(k+1)\times m$) contains the weights of the linear combination, and Y (which has size $p \times m$) contains the desired outputs.

To test the function on new domain points (now with fixed centers and weight matrix W):

- Form the EDM. With $\vec{p}$ new data points, this will be $\vec{p} \times k$.
- Form the transfer matrix Φ, which will be $\vec{p} \times (k+1)$.
- Perform the matrix product ΦW, producing the new output.

I. Evaluation of Classifier

1. K-Fold Cross Validation

In k-fold cross validation, the training set is partitioned into k subsets of equal size. Then, one subset is used as test data using the classification model, after being trained on the remaining $k-1$ subsets. This process is done with k iterations until each subset is used as a testing set [51]. An additional step that could be used for preventing both underfitting or overfitting problems is re-randomization, where there dataset is re-randomized prior to repeating the training and validation processes.

2. Model Evaluation Metrics

2..1 Accuracy

Accuracy is the number of correct predictions outputted divided by the total number of predictions made, converted to percentage [52].

2..2 Recall

Recall measures the ratio of actual positives that are correctly identified as such. It is also called the true positive rate, the recall, or probability of detection in some fields [53]. As an example, in a medical test used to identify a disease, the recall of the test is the proportion of people who test positive for the disease among those who have it.

2..3 Precision

Precision is defined as the number of true positives divided by the number of true positives plus the number of false positives. False positives are cases the model incorrectly labels as positive that are actually negative [54]. As an example, in a medical test for diagnosing a disease, the precision is the proportion of the true positive over the number of true positives plus the number of false positives.

2..4 F1-Score

F1-Score is the *harmonic mean* between precision and recall $(2 \times \frac{precision \times recall}{precision + recall})$

IV. Design and Implementation

A. Datasets

The datasets used for creating the SVM Classifier Model consist of mass spectrometry data of samples taken from pancreatic cancer patients and controls who are free from pancreatic cancer. The mass spectrometry data is acquired from the Global Natural Product Social Molecular Networking (GNPS) MS database website and the PRoteomics IDEntifications (PRIDE) Archive, all in *mzml* format. Each particular *mzml* file contains: metadata such as instrumentConfigurationList, aquisitionSettingsList, dataProcessingList, sampleList; and also the spectrum list which contains the spectrum description and the actual spectra data. Validation of file formats and mass spectrometry data pre-processing would be done using pymzML, SciPy, and NumPy.

The dataset to be used for the identification of chemical compounds from mass spectra is acquired from the Chemical Entities of Biological Interest (ChEBI) database. Included are three tables: *compounds*, *names*, and *chemical_data*.

B. System Architecture

The implementation of the system is defined in the following diagrams:

1. Development Flow Diagram

The first step is data collection. The quality and quantity of the data are crucial since it would determine how good the classifier model would be. A higher number of quantity of mass spectrometry data would be beneficial in order to create an accurate model. Moreover, the quality would affect the degree of difficulty the pre-processing would involve.

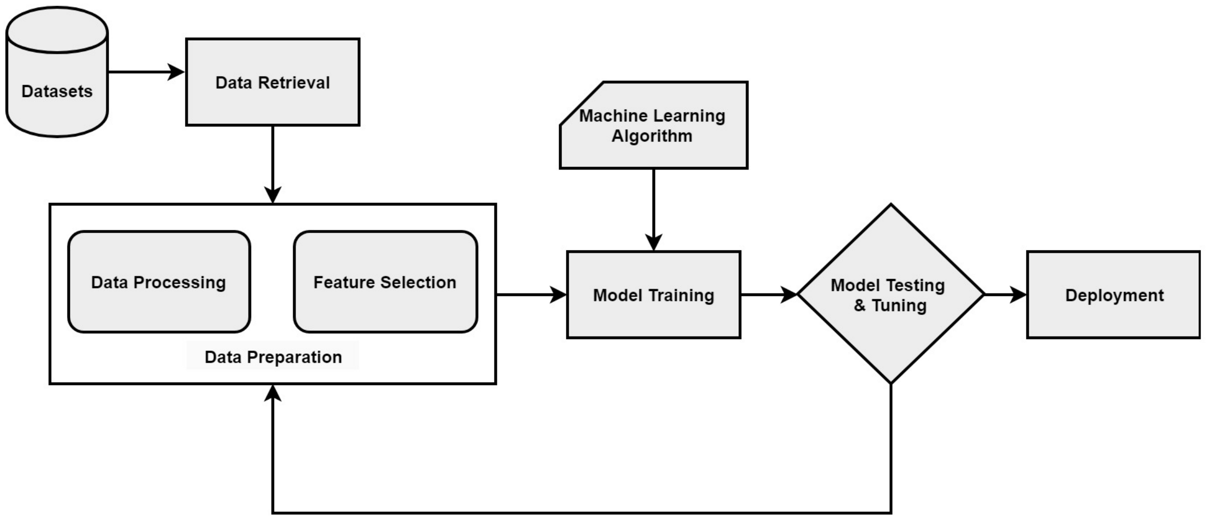


Figure 1: Machine Learning Development Flow Diagram

After gathering enough mass spectrometry data, the preparation (including data processing and feature selection) comes next. The mass spectrometry data pre-processing steps are as described in the preceding chapter. All steps shall be implemented with pymzML, SciPy, and NumPy. With regard to feature selection, since the number of features in a particular mass spectrum is low and all the features of the spectrumList data are considered relevant, all features would be used for the development of the model.

Once the data has been pre-processed, training of model starts. The algorithm to be used is the Support Vector Machines (Radial Basis Function Kernel) algorithm. The original input space would be mapped into a high-dimensional feature space using RBF where it becomes linearly separable. This allows a more efficient classification involving the mass-per-charge, intensity, and retention time values.

Consequently, model testing is done and the percentage values of accuracy, sensitivity and specificity are calculated. Tuning the model parameters is necessary whilst repeating the model training and testing processes iteratively. Subsequently, 10-fold cross validation would be performed.

2. Context Diagram

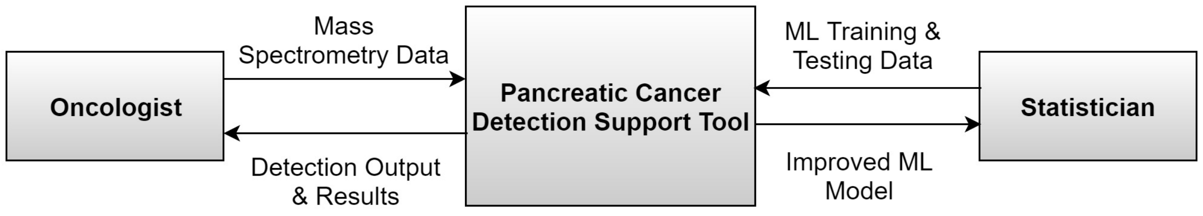

Mass Spectrometry Data
ML Training & Testing Data
Oncologist
Pancreatic Cancer Detection Support Tool
Statistician
Detection Output & Results
Improved ML Model

Figure 2: PCDST Context Diagram

The system would be utilized by two types of users: the researchers / oncologists and the statisticians/administrators. The researcher / oncologist would be performing the disease detection by inputting a patient's mass spectrometry data. After data pre-processing, the detection is executed and the results are generated together with the list of most common chemical compounds found in the mass spectrum. On the other hand, the statistician would be in-charge of the machine learning-related work the tool requires. He/she may input additional training datasets in order to improve the current model.

3. Use Case Diagrams

The researcher could perform all the steps in the mass spectrometry data pre-processing sequence. Each step could be selected individually as to provide the user the option whether to perform a particular step or not. The inputted MS data could also be viewed in a visual form (mass spectrum image). Detection of disease and results generation could be done after the preceding steps. Also, a user's manual for the tool would be included.

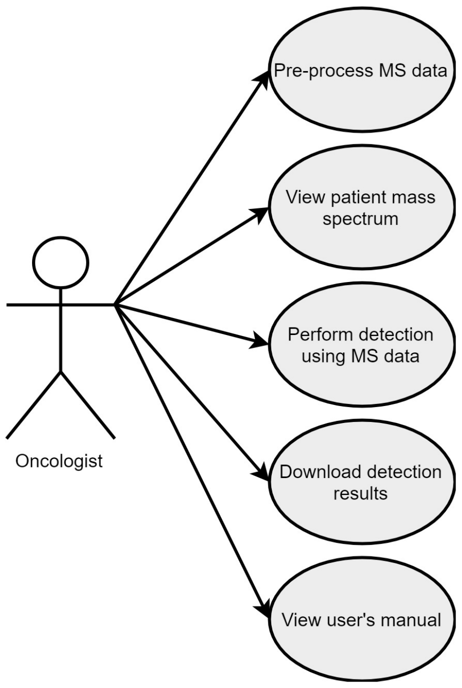

Pre-process MS data
View patient mass spectrum
Perform detection using MS data
Download detection results
View user's manual
Oncologist

Figure 3: Use Case Diagram of Researcher / Oncologist

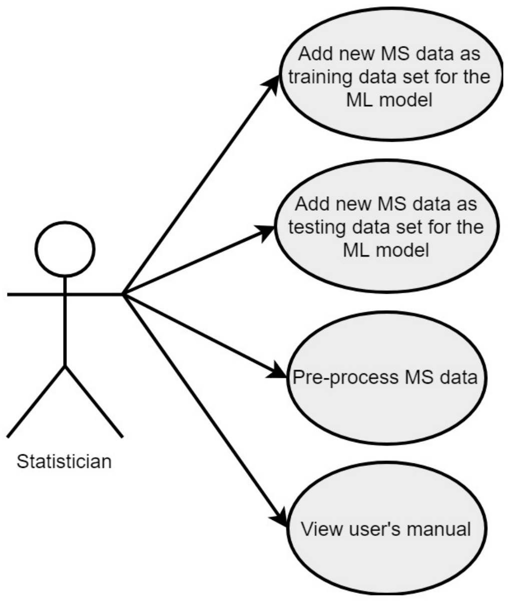

Add new MS data as training data set for the ML model
Add new MS data as testing data set for the ML model
Pre-process MS data
View user's manual
Statistician

Figure 4: Use Case Diagram of Statistician

The statistician would be able to add both training and testing datasets for the SVM model. Pre-processing of these datasets is recommended. A user's manual for the tool would be included.

4. Data Flow Diagrams

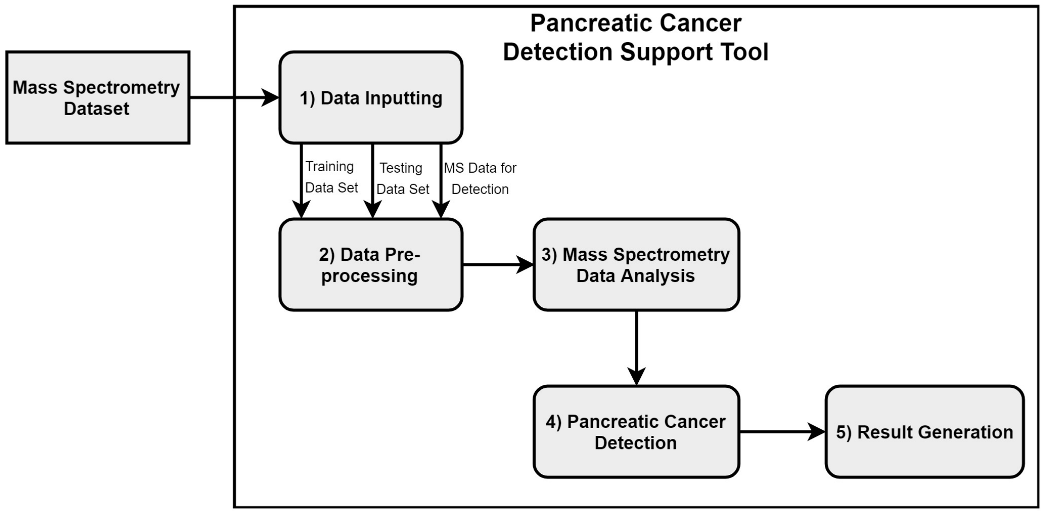

Pancreatic Cancer Detection Support Tool
Mass Spectrometry Dataset
1) Data Inputting
Training Data Set
Testing Data Set
MS Data for Detection
2) Data Pre-processing
3) Mass Spectrometry Data Analysis
4) Pancreatic Cancer Detection
5) Result Generation

Figure 5: Top-Level Dataflow Diagram of PCDST

For both users, the dataflow starts with the data inputting, where the user uploads the mass spectrum file (*mzml* format required) to the tool. Afterwards, data pre-processing with selected steps is executed. (The succeeding processes would only be performed by the oncologist).

The patient's mass spectrum could be then analyzed in graph form, complemented by the tabulated form of data. Detection is done subsequently, showing the percentage value of likelihood (based on data's distance to decision boundary). Finally, the results are generated and the options for result file download is presented.

The processes in the top-level dataflow diagram is illustrated in Figures 6-8 below:

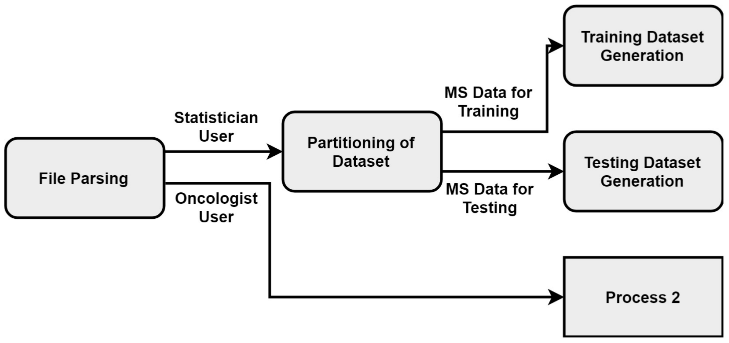


Figure 6: Dataflow Diagram of Process 1

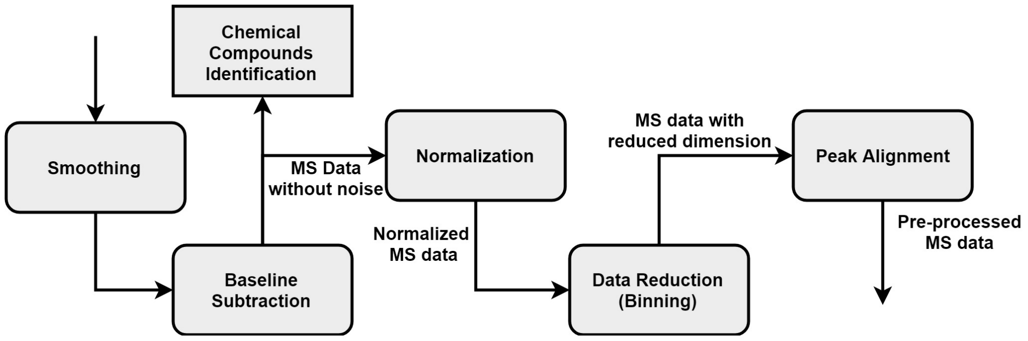


Figure 7: Dataflow Diagram of Process 2

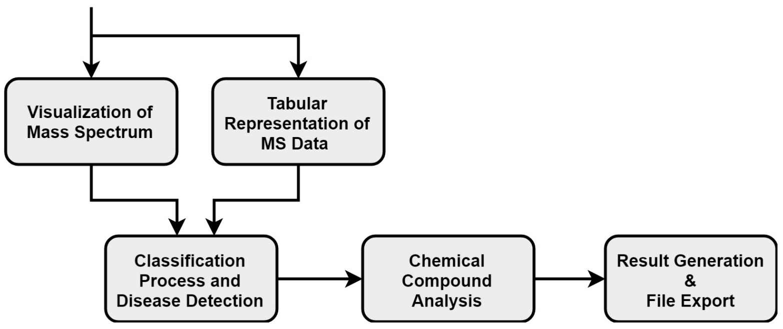

Visualization of Mass Spectrum
Tabular Representation of MS Data
Classification Process and Disease Detection
Chemical Compound Analysis
Result Generation & File Export

Figure 8: Dataflow Diagram of Processes 3-5

5. Entity Relationship Diagram

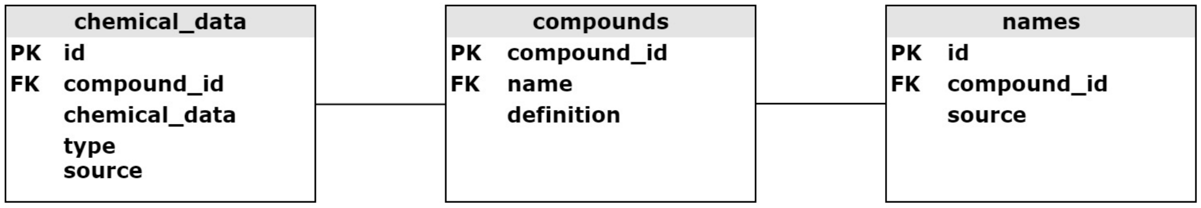


Figure 9: Entity Relationship Diagram of Chemical Compounds Database

The ERD of the chemical compounds database includes three tables: compounds, chemical_data, and names. These tables are utilized for determining the most abundant chemical compounds in a particular patient's mass spectrum, based on the mass-per-charge values with the highest intensities.

C. Technical Architecture

The system would be developed using the following:

- PyCharm IDE 2018.3
- Python 3.7
- NumPy 1.16
- SciPy 1.2
- pymzML 2.0.6
- scikit-learn LibSVM 3.23
- MySQL
- Windows 10 OS

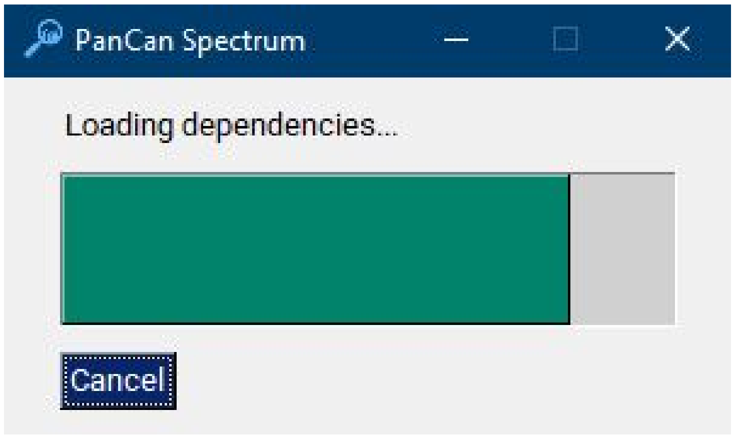
Minimum hardware requirements:

- At least 4 GB free hard-disk space
- At least 8 GB RAM
- Minimum processor speed of 2.6 GHz

V. Results

A. PanCan Spectrum Application

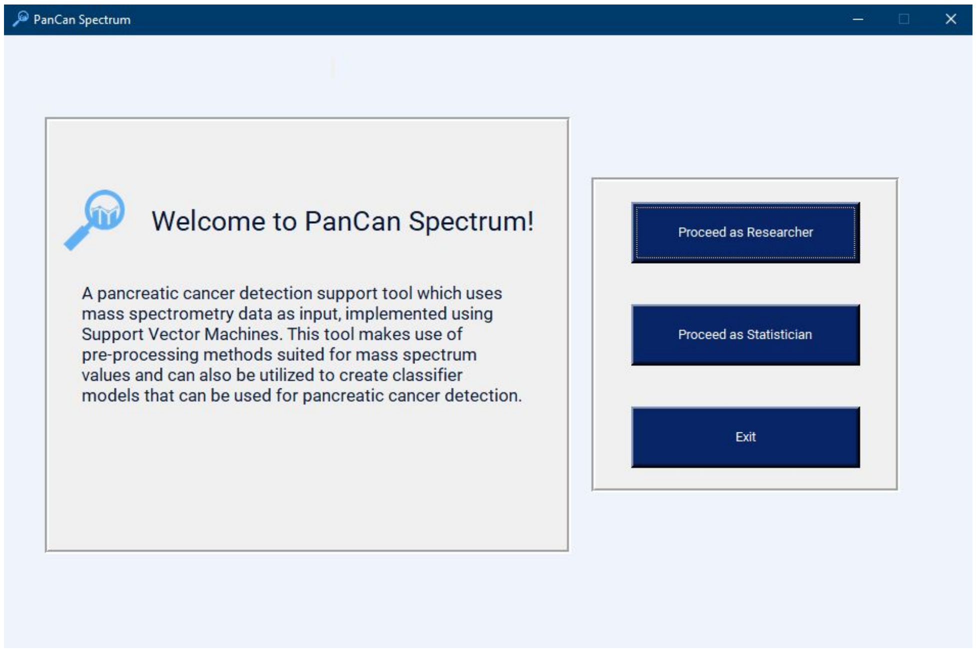
After opening the tool, PanCan Spectrum would notify the user with regard to loading of dependencies. A progress meter would be shown.



Loading dependencies...

Figure 10: PanCan Start Progress Meter

Subsequent to the setup of application, the main menu would be shown. The user then could choose to proceed as an oncologist or a statistician. The main application window, along with the functionalities, would be dependent on the chosen user type. Moreover, the panels would be different.



Welcome to PanCan Spectrum!

A pancreatic cancer detection support tool which uses mass spectrometry data as input, implemented using Support Vector Machines. This tool makes use of pre-processing methods suited for mass spectrum values and can also be utilized to create classifier models that can be used for pancreatic cancer detection.

Figure 11: PanCan Spectrum User Choice Window

If the user proceeds as an oncologist, the following would be shown:

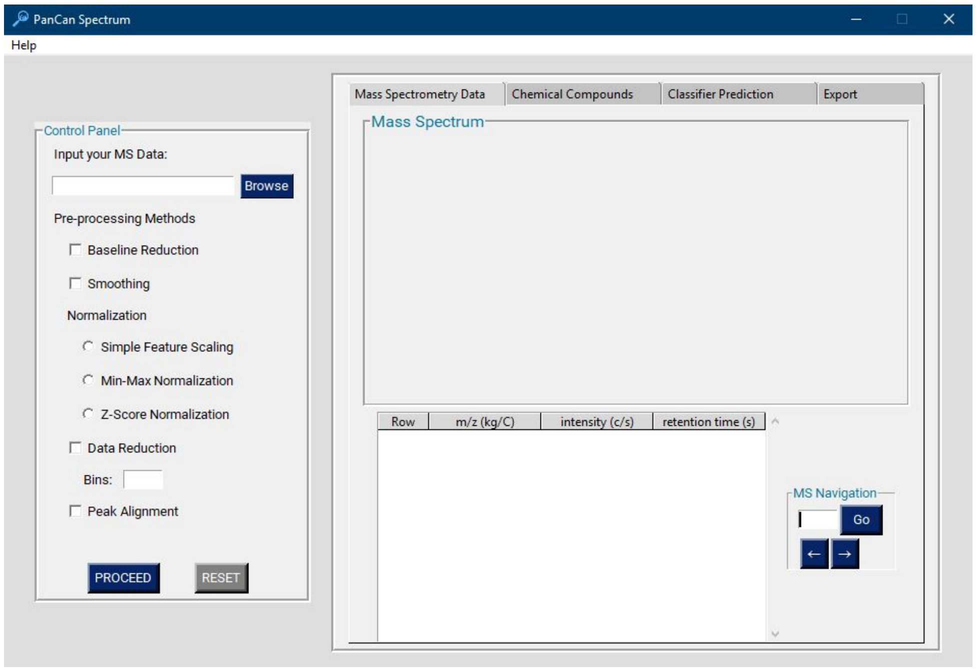


Figure 12: PanCan Spectrum Oncologist Window

The application's main window consists of a control panel, a multi-tab panel for mass spectrometry data visualization, chemical compounds list, prediction, export option, and a user's manual found in the window menu bar. The researcher then may start inputting a mass spectrometry dataset. By using the browse button found in the control panel, the user would be prompted to choose the location of the dataset.

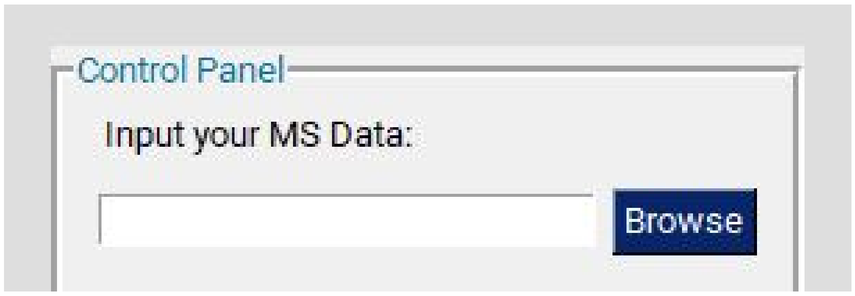
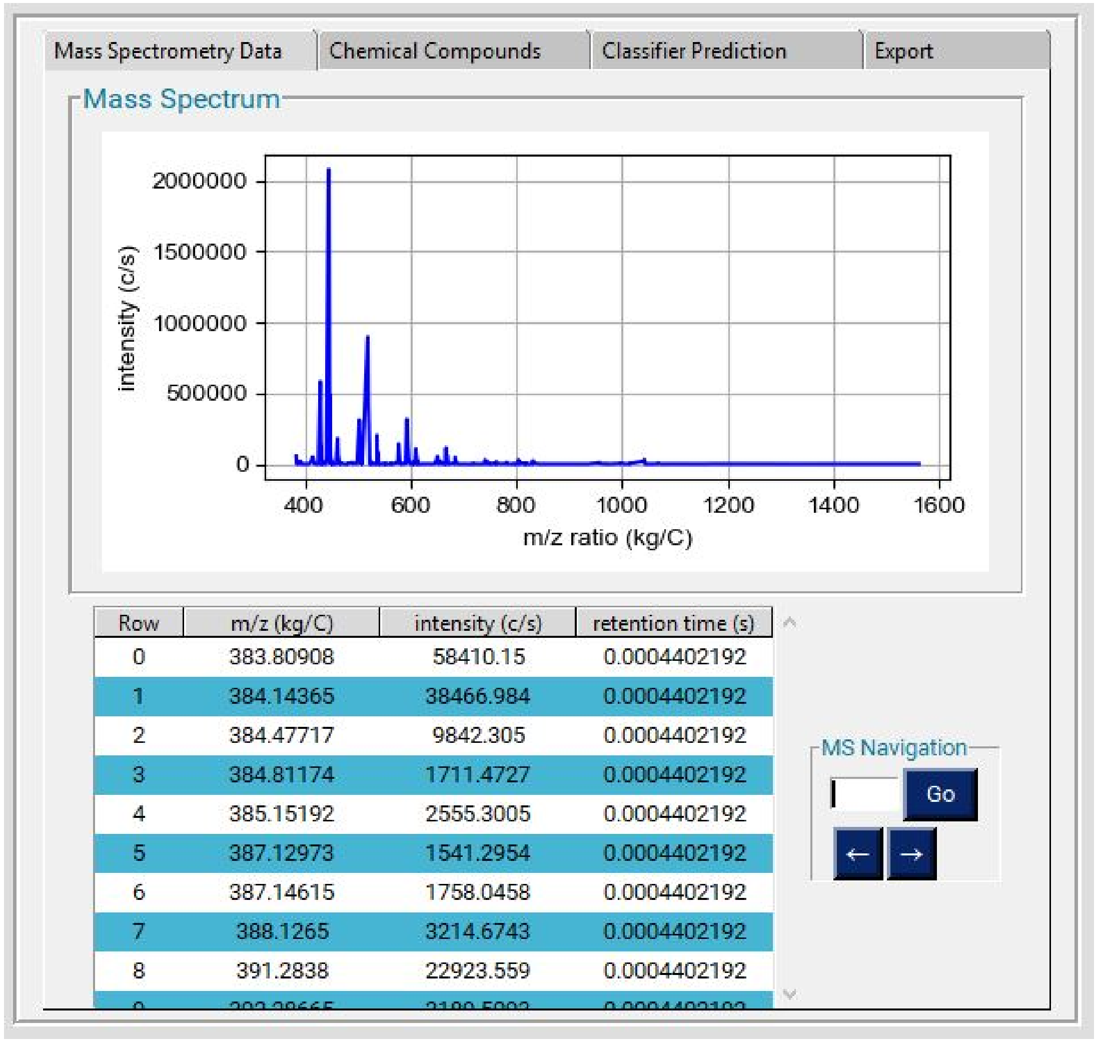


Figure 13: PanCan Spectrum Dataset Selection Window

Afterwards, the user may choose the pre-processing steps that would be applied on the data. Once the dataset location and pre-processing steps are chosen, the user may click the proceed button to display the mass spectra visualization, tabular data, list of most abundant chemical compounds, and prediction results. Resetting the dataset location and processing parameters would clear all the values in the control panel and multi-tab panel.

The MS Data Tab shows the plot visualization and tabular data representation of the mass spectrum. The plot shows the peaks found in the data, based on the m/z and intensity values. The table found below the plot shows the exact values of the MS data The user then may navigate through the list of mass spectrum (the entirety of the mass spectra), by using the buttons found in the lower right of the main window, beside the MS data table. Doing so would update both the plot and table in the ms data tab. This navigation functionality would provide the option for the user to check the other mass spectra found in the inputted dataset (see next two figures).



Row	m/z (kg/C)	intensity (c/s)	retention time (s)
0	383.80908	58410.15	0.0004402192
1	384.14365	38466.984	0.0004402192
2	384.47717	9842.305	0.0004402192
3	384.81174	1711.4727	0.0004402192
4	385.15192	2555.3005	0.0004402192
5	387.12973	1541.2954	0.0004402192
6	387.14615	1758.0458	0.0004402192
7	388.1265	3214.6743	0.0004402192
8	391.2838	22923.559	0.0004402192

Figure 14: PanCan Spectrum Mass Spectrometry Data Tab

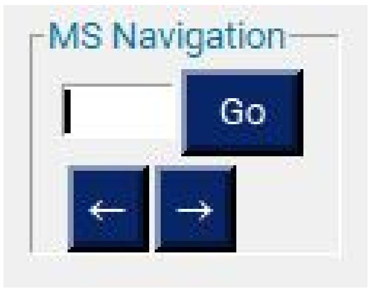
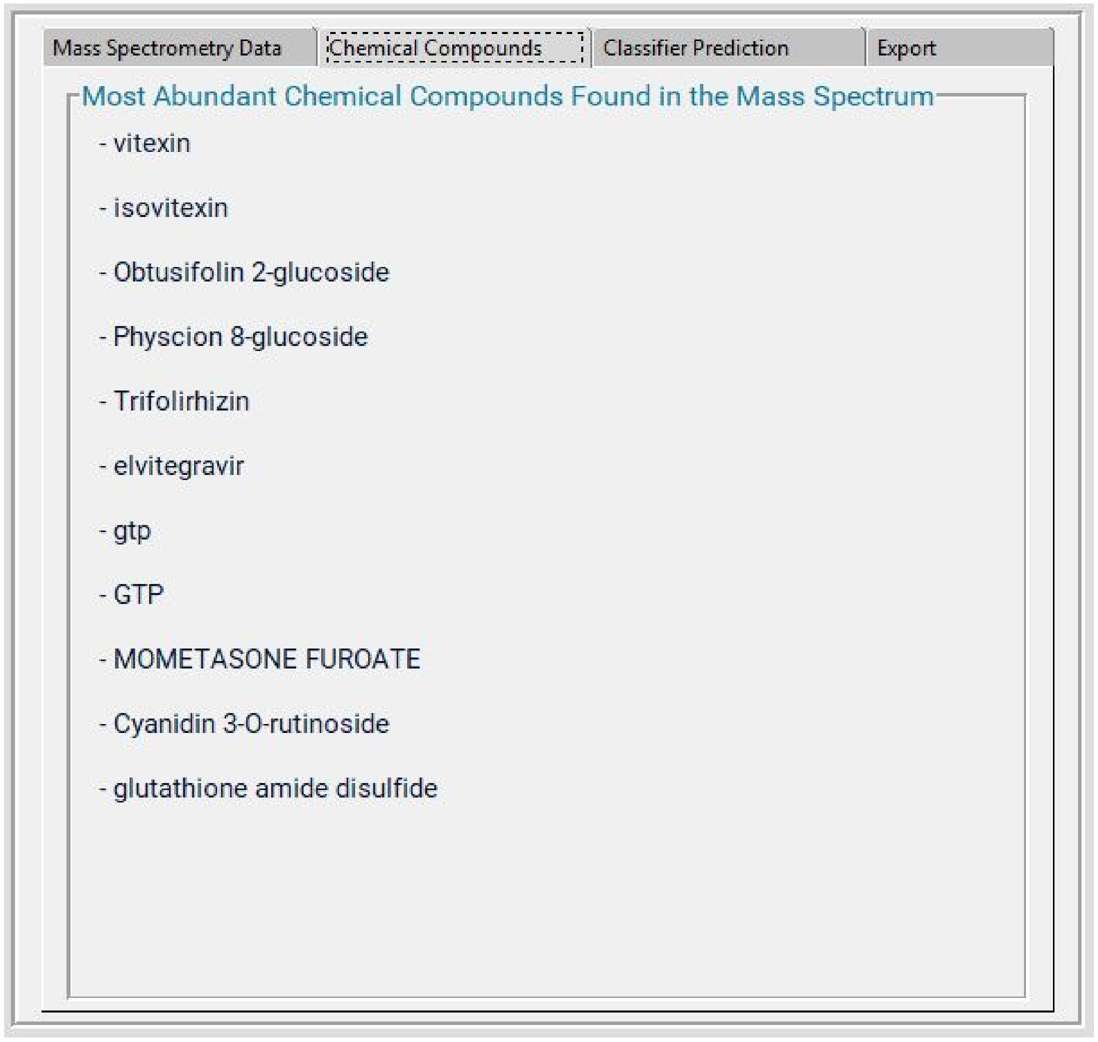


Figure 15: PanCan Spectrum Navigation for MS Data

The Chemical Compounds Tab shows the list of the most abundant chemical compounds found in the current mass spectrum, based on the m/z and intensity values. The results of the list are gathered from the query results on the chemical compounds database acquired from Chemical Entities of Biological Interest (ChEBI) database.

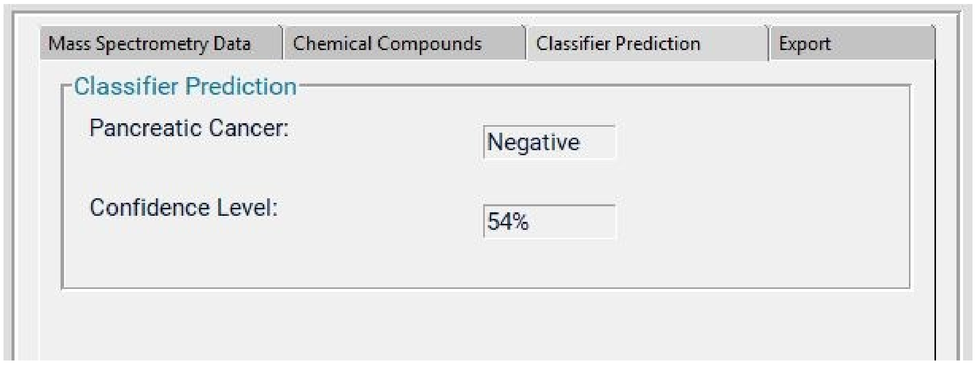


Most Abundant Chemical Compounds Found in the Mass Spectrum

- vitexin
- isovitexin
- Obtusifolin 2-glucoside
- Physcion 8-glucoside
- Trifolirhizin
- elvitegravir
- gtp
- GTP
- MOMETASONE FUROATE
- Cyanidin 3-O-rutinoside
- glutathione amide disulfide

Figure 16: PanCan Spectrum Chemical Compounds Tab

The prediction tab shows the classification result for the current mass spectrum, together with the confidence level.



Pancreatic Cancer: Negative

Confidence Level: 54%

Figure 17: PanCan Spectrum Prediction Tab

Finally, the oncologist may export to a PDF file all the mass spectrum details, including the MS plot, MS tabular data, list of most abundant chemical compounds, and the classification results. The input fields found in the export tab include the folder location of where to download the PDF, and the actual file name. These fields must be filled out in order for the export button to function properly.

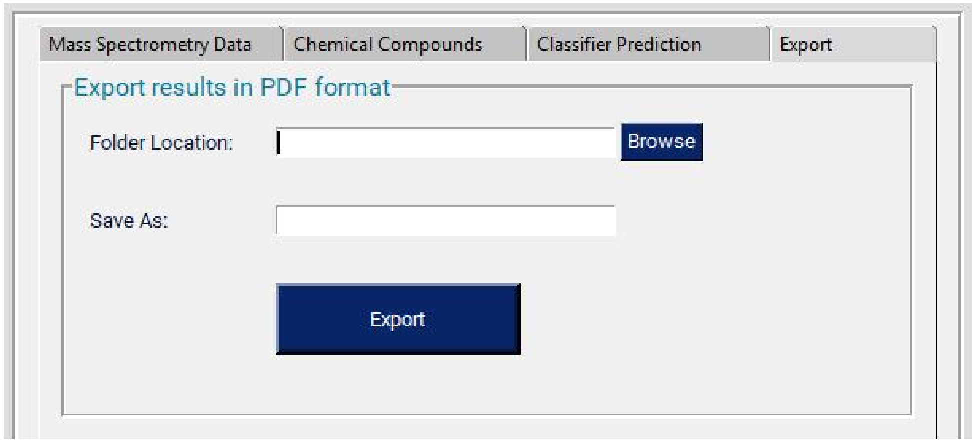
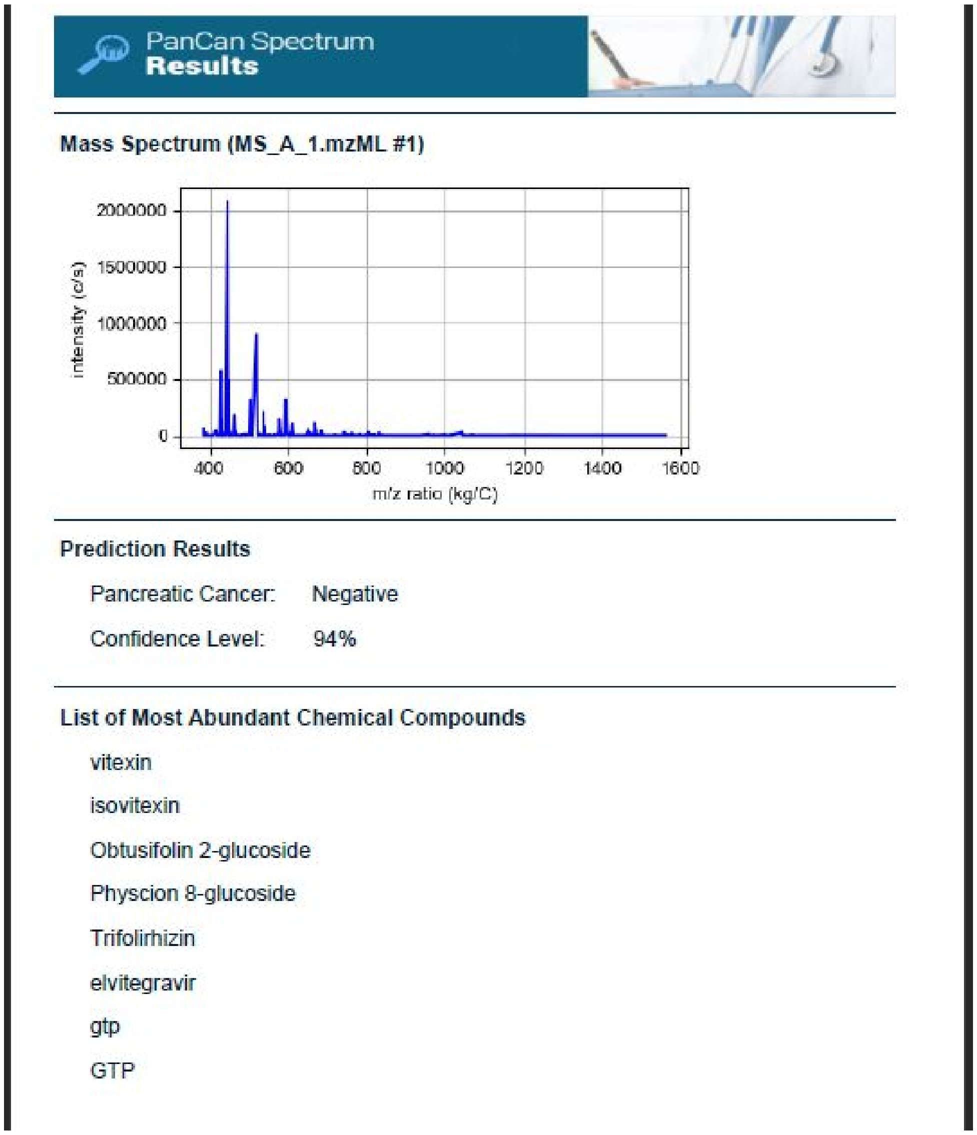


Figure 18: PanCan Spectrum Export Tab

An example of an exported PDF file from the PanCan Spectrum:



Mass Spectrum (MS_A_1.mzML #1)

Prediction Results

Pancreatic Cancer:	Negative
Confidence Level:	94%

List of Most Abundant Chemical Compounds

vitexin

isovitexin

Obtusifolin 2-glucoside

Physcion 8-glucoside

Trifolirhizin

elvitegravir

gtp

GTP

Figure 19: PanCan Spectrum PDF Sample

If the user proceeds as a statistician, the following would be shown.

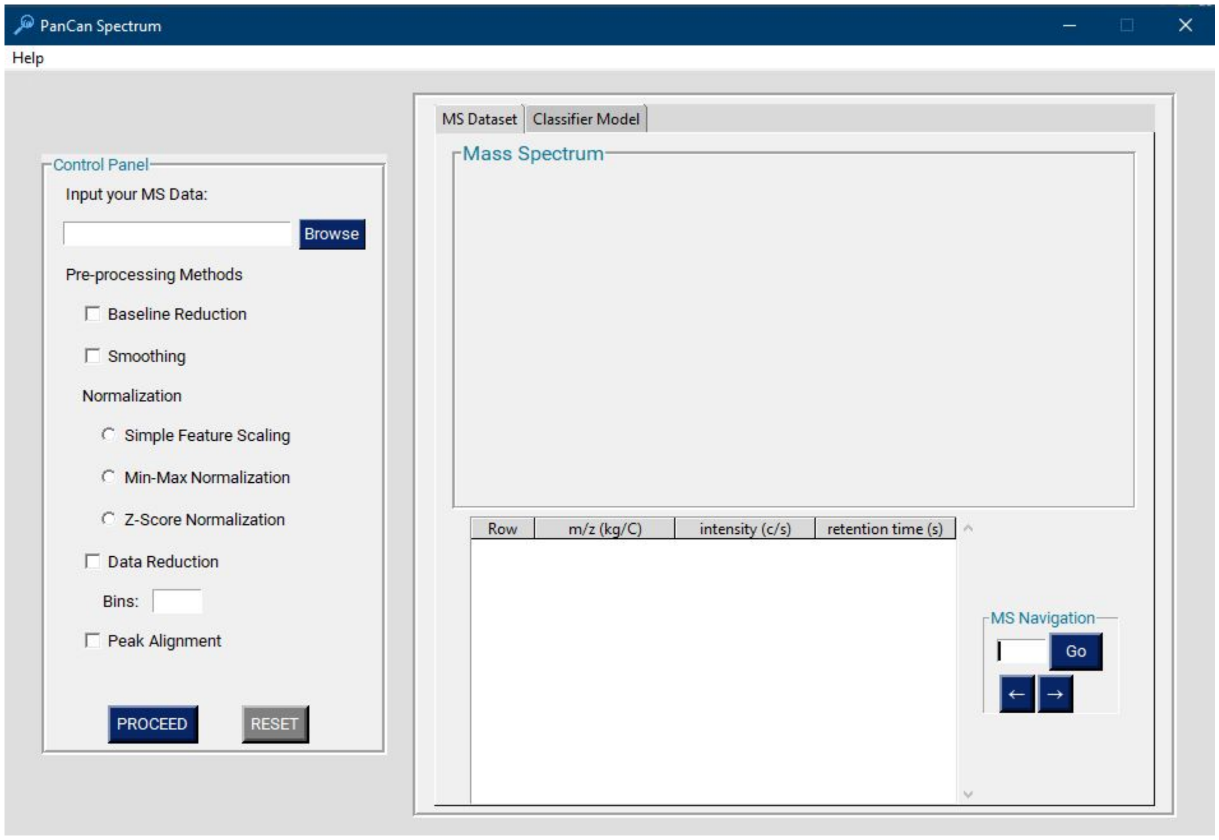
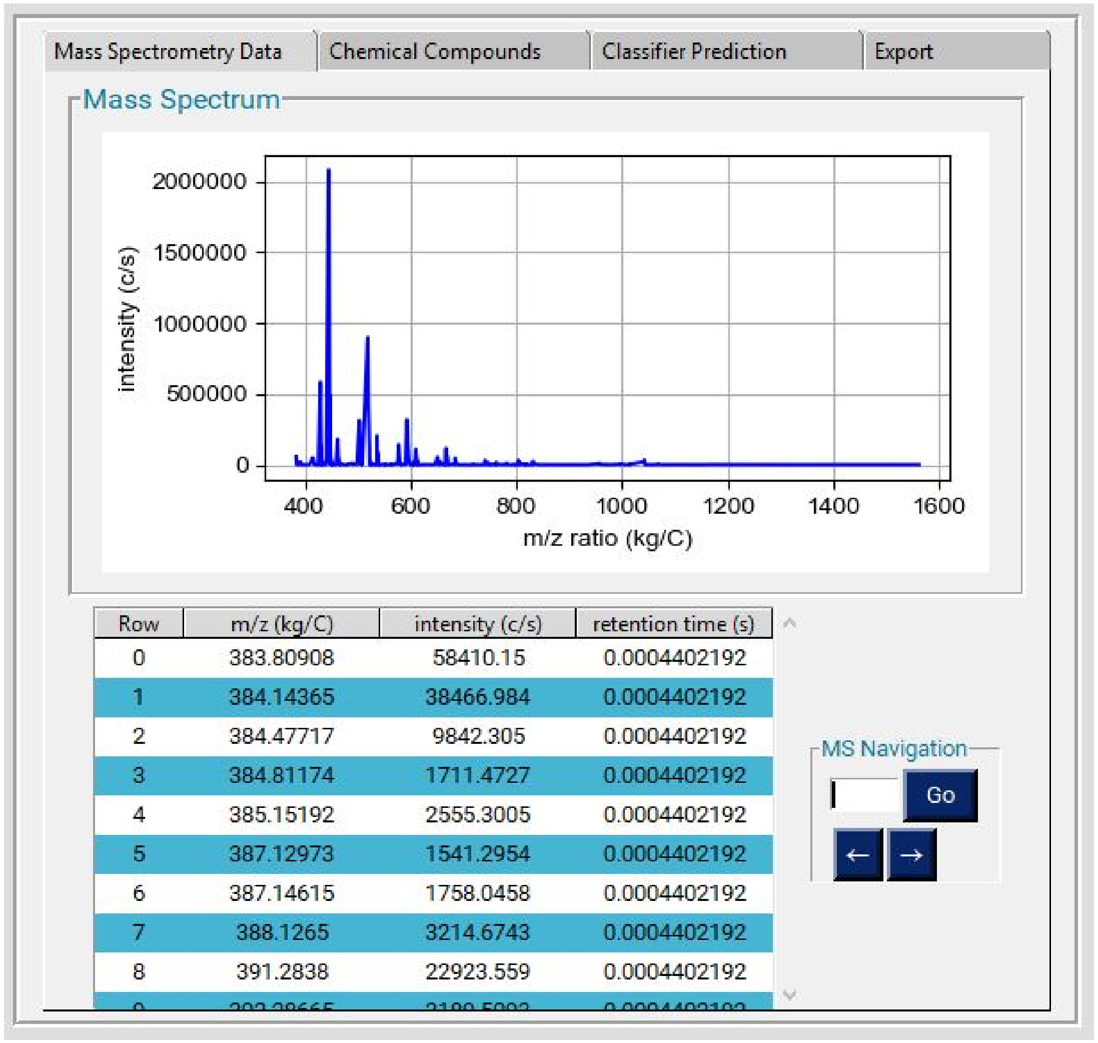


Figure 20: PanCan Spectrum Statistician Window

Similar to the main window shown to the oncologist, the application's main window for the statistician also consists of a control panel and a multi-tab panel for mass spectrometry data visualization and MS tabular data. Instead of a prediction and export tab, the other tabs for this case would include the classifier improvement and performance tab.

Figure 21: PanCan Spectrum Statistician Window Tabs

After choosing the *mzml* file to be inputted and the pre-processing steps, the mass spectrum visualization and tabular data would be shown. Navigation through the mass spectra is also possible for the statistician.



Row	m/z (kg/C)	intensity (c/s)	retention time (s)
0	383.80908	58410.15	0.0004402192
1	384.14365	38466.984	0.0004402192
2	384.47717	9842.305	0.0004402192
3	384.81174	1711.4727	0.0004402192
4	385.15192	2555.3005	0.0004402192
5	387.12973	1541.2954	0.0004402192
6	387.14615	1758.0458	0.0004402192
7	388.1265	3214.6743	0.0004402192
8	391.2838	22923.559	0.0004402192
9	393.28665	[illegible]	0.0004402192

Figure 22: PanCan Spectrum Statistician MS Data Tab

The classifier and performance tab allows the statistician to use the inputted MS dataset as training and testing data to create an SVM model. The type of data must be specified prior to improvement of the model. The classifier model that would be created corresponds to the included pre-processing steps in the control panel. A total of six types of classifiers can be created. After training, the classification metrics results of the new model would be shown to the user.

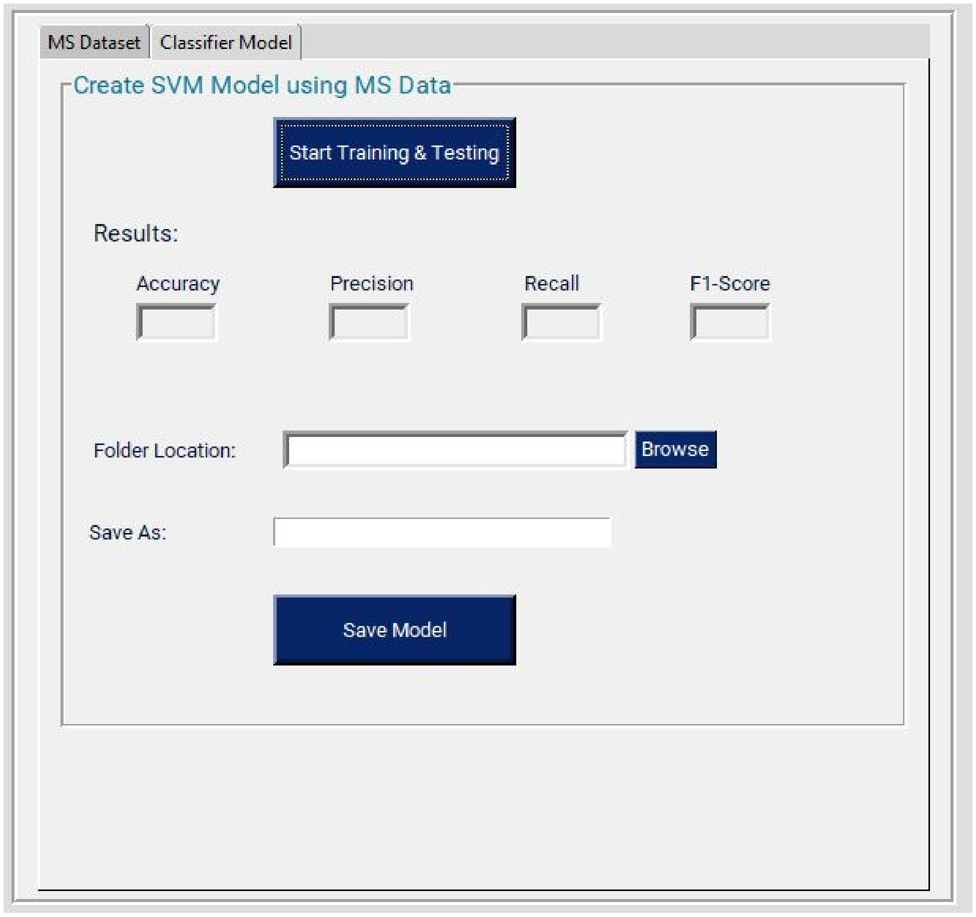
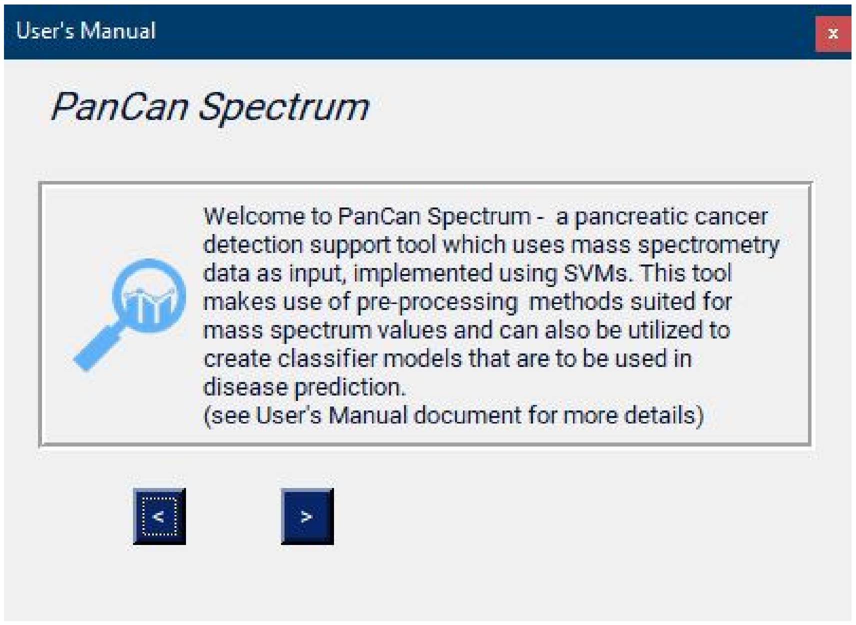


Figure 23: PanCan Spectrum Statistician Model Tab

For both the researcher and statistician user, a user's manual (found on the menu bar in the upper part of the window) would be provided.



Welcome to PanCan Spectrum - a pancreatic cancer detection support tool which uses mass spectrometry data as input, implemented using SVMs. This tool makes use of pre-processing methods suited for mass spectrum values and can also be utilized to create classifier models that are to be used in disease prediction.
(see User's Manual document for more details)

Figure 24: PanCan Spectrum User Manual

B. SVM Models

A total of six RBF-SVM models were constructed for this project. For each model, a final 5-fold Grid Search cross validation is done to ensure that the hyperparameters are optimized.

Table 1: Hyperparameter Grid

Hyperparameter	**Values**
C	1, 10, 100
kernel	RBF
gamma	0.1, 0.01, 0.001, 0.0001

The chosen parameters for the six SVM models are shown in the following table.

Table 2: Best Hyperparameters for the RBF-SVM Models

Hyperparameter	**Values**
C	1
kernel	RBF
gamma	0.001

The final metrics scores achieved from classifying the test sets using each SVM models is as follows:

Table 3: SVM Models Metrics Results on Test Sets

Model	**Accuracy**	**Precision**	**Recall**	**F1-Score**
SVM_model_1	0.8326	0.7633	0.4413	0.5593
SVM_model_2	0.8523	0.8271	0.5146	0.6345
SVM_model_3	0.8359	0.7514	0.5118	0.6089
SVM_model_4	0.8194	0.9162	0.0667	0.1243
SVM_model_5	0.9034	0.9026	0.5764	0.7035
SVM_model_6	0.7368	0.6147	0.4912	0.5461

VI. Discussion

The main objective of this study is the creation of RBF-SVM classifier models that would help determine whether a patient, based on his/her mass spectrometry data, has pancreatic cancer or not. The MS datasets were acquired from the GNPS site and PRIDE Archive. Due to scarcity of *mzml* files acquired from PC-diagnosed patients' biological samples, the quantity of data corresponding to the two classes (with PC or not) are imbalanced. Moreover, the mass spectrometry data are derived from human samples of various types (blood, urine, pancreatic tissue). This somehow affects the process of model training since there could be a bias for the class of greater data quantity. As a work-around, tuning of model parameters had been done for quite a long period of time in order to properly develop the classifier.

One of the most demanding processes done in the creation of the tool is the pre-processing of data. Besides the actual data de-noising, reduction, and processing, the implementation of the algorithms was onerous since it required multiple adjustments of parameters especially when it comes to baseline reduction and smoothing. The noise reduction effect of these two methods are greatly reliant on the number of dimensions a particular mass spectrum has, hence finding the optimal denosing parameters takes a lot of time due to the dissimilar lengths of m/z and intensity lists of the spectra in an *mzml* file.

On the other hand, no problems were encountered in the normalization part of the data. As for the data reduction and peak alignment, the time needed and degree of difficulty depended entirely on the amount of data in each *mzml* file. Since each *mzml* file contains a huge quantity of spectra, the datasets had to be partitioned so that the pre-processing methods could be applied on all of them without having problems with regard to computer performance, lagging, and overall processing time.

Among all the possible combinations of pre-processing methods, it was observed that selecting all to be in-use is the most optimal choice. The SVM_model_five which is the one trained, tested, and evaluated on datasets with complete pre-processing relatively had the highest scores in accuracy, precision, recall, and f1-score after final evaluation on the datasets. Furthermore, GridCVSearch was used with regard to the entire process of cross validation and hyperparameter optimization.

Although some pre-processing methods resulted to significantly lower metric scores than the others, these particular ones have an advantage when it comes to the required time of processing. Since *mzml* files are commonly high-dimensional, this could be a topic of argument regarding the trade-off between time and accuracy.

PanCan Spectrum is a pancreatic cancer detection support tool in the form of a desktop GUI application, developed for the sake of providing a fast method of determining whether a patient has pancreatic cancer or not, based on his or her MS data. The whole application was written with the Python language only, using libraries such as NumPy, SciPy, PySimpleGUI, pymzML, and others which are greatly beneficial for data science-related projects. PySimpleGUI was chosen for the front-end part since it eases the development of user interface elements which mostly utilizes TkInter widgets. For the parsing of *mzml* files, pymzML is used due to its automatic mapping functionality with regard to m/z, intensity, and retention time values of mass spectrometry data. Both NumPy and SciPy are heavily utilized in the pre-processing parts of the inputted MS datasets. Although these two libraries provide a very wide array of data manipulation methods, those that were employed in the tool are mostly for list to array conversions (vice-versa), baseline reduction, peak smoothing, and data reduction.

For the process of training, testing, and evaluation of each classifier model, scikit-learn (sklearn) is used. Persisting of each model of was done with joblib, a library which basically used for saving and loading machine learning models. A past option with regard to this part was the library pickle, but due to the nature of the datasets used, the former one is chosen. Another feature of the tool, which is the PDF export option, was implemented with pyFPDF. Each set of details from the various panels of PanCan Spectrum are individually outputted to a pdf object before being downloaded.

VII. Conclusion

PanCan Spectrum is a simple mass spectrometry-focused tool which utilizes Support Vector Machines algorithm in order to classify MS data based on mass-per-charge values, its corresponding intensities, and retention time. It allows the user to perform varying types of pre-processing methods which are recommended specifically for MS data. Furthermore, it provides the visualization of a mass spectrum, complemented by its data in tabular form. Using the tool, the list of most abundant chemical compounds present based on the highest intensity values can be known. This allows the determination of which appropriate options of treatment are possible in the case that the patient MS data tests positive for pancreatic cancer. The results of each application run can also be recorded due to the PDF export capability of the tool. This allows the user to keep track of past detections and analyses.

The creation of PanCan Spectrum involved the process of training, testing, and evaluating several RBF-SVM models. After cross validation, it is observed that the most efficient model has an accuracy of 0.9034, precision of 0.9026, recall of 0.5764, and f1-score of 0.7035. Moreover, since classifier models can be created by a statistician using PanCan Spectrum, more efficient disease detection can be done as additional datasets are acquired and utilized.

Ultimately, based on the entirety of the study and all acquired datasets, it is found that the values most correlated to pancreatic cancer are in the ranges 400-600 m/z. Also, the chemical compounds that are usually identified in MS data of PC-diagnosed patients are delanzomib, obtusin 2-glucoside, physcion 8-glucoside,trifolirhizin, and elvitegravir. These information are discovered through the use of a chemical compounds database, and queries with the m/z & intensity values as input values.

VIII. Recommendation

PanCan Spectrum has a lot of areas in which improvements could be done. Additional datasets, especially those consisting of mass spectrometry data from biological samples of PC-diagnosed patients, are needed so that further validation could be performed. The accuracy of the models used for classifying MS data are highly dependent on the pre-processing methods chosen by the user, hence it is imperative that the data should be analyzed efficiently using the mass spectrum visualization plot and table. Furthermore, the list of most abundant chemical compounds found in each spectrum is limited to the compounds database acquired from Chemical Entities of Biological Interest (CHeBI). Acquisition of more data regarding the chemical compounds can also be done so that even more possible values of mass-per-charge can be checked.

With regard to the detection support capability of PanCan Spectrum, its integration with an actual in-use mass spectrometer can also be done to make the process of pancreatic cancer detection quicker and involve less transmission and processing of mass spectrometry data.

IX. Bibliography

[1] W. C. R. Fund, "Pancreatic cancer statistics," 2018. [Online]. Available: https://www.wcrf.org/dietandcancer/cancer-trends/pancreatic-cancer-statistics?fbclid=IwAR3R4-G3KzUh6sXmDCbWXIchsGYVQvXGdpqUgjxGZ90driz4p8NlmteTIXA

[2] M. Ilic and I. Ilic, "Epidemiology of pancreatic cancer," 2016. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/ONLINEs/PMC5124974/

[3] A. C. Society, "Pancreatic cancer risk factors," 2016. [Online]. Available: https://www.cancer.org/cancer/pancreatic-cancer/causes-risks-prevention/risk-factors.html

[4] L. Martin, "An overview of pancreatic cancer," 2017. [Online]. Available: https://www.webmd.com/cancer/pancreatic-cancer/digestive-diseases-pancreatic-cancer

[5] A. Ashcroft, "An introduction to mass spectrometry," n.d. [Online]. Available: http://www.astbury.leeds.ac.uk/facil/MStut/mstutorial.htm

[6] O. Mamer, "Medical applications of mass spectrometry," 2017. [Online]. Available: https://www.sciencedirect.com/topics/agricultural-and-biological-sciences/mass-spectrometry

[7] U. Pancreatic Cancer Organization, "How is pancreatic cancer diagnosed?" 2018. [Online]. Available: https://www.pancreaticcancer.org.uk/diagnosis

[8] A. C. Society, "Can pancreatic cancer be found early?" 2016. [Online]. Available: https://www.cancer.org/cancer/pancreatic-cancer/detection-diagnosis-staging/detection.html

[9] A. Swan, A. Mobasheri, D. Allaway, S. Liddell, and J. Bacardit, "Application of machine learning to proteomics data: Classification and biomarker identification in postgenomics biology," *OMICS: A Journal of Integrative Biology*, p. 595–610, 2013, doi: 10.1089/omi.2013.0017.

[10] J. Rajapakse, K. Duana, and W. K. Yeo, "Proteomic cancer classification with mass spectrometry data," *American Journal of PharmacoGenomics*, p. 281–292, 2005, doi: 10.2165/00129785-200505050-00001.

[11] K. Coombes *et al.*, "Understanding the characteristics of mass spectrometry data through the use of simulation," 2007. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/ONLINEs/PMC2657656/

[12] C. Smith, E. Want, G. O'Maille, R. Abagyan, and G. Siuzdak, "Xcms: Processing mass spectrometry data for metabolite profiling using nonlinear peak alignment, matching, and identification," p. 779–787, 2006, doi: 10.1021/ac051437y.

[13] Datta, "Feature selection and machine learning with mass spectrometry data. methods in molecular biology," p. 237–262, 2013, doi: 10.1007/978-1-62703-392-3_10.

[14] S. Ahmed, M. Zhang, and L. Peng, "Feature selection and classification of high dimensional mass spectrometry data: A genetic programming approach," p. 43–55, 2013, doi: 10.1007/978-3-642-37189-9_5.

[15] K.Kim, S.Ahn, J.Lim, B.C.Yoo, J.H.Hwang, and W.Jang, "Detection of pancreatic cancer biomarkers using mass spectrometry," p. 43–55, 2013, doi: 10.1007/978-3-642-37189-9_5.

[16] T. Nguyen, S. Nahavandi, D. Creighton, and A. Khosravi, "Mass spectrometry cancer data classification using wavelets and genetic algorithm," p. 3879–3886, 2015, doi: 10.1016/j.febslet.2015.11.019.

[17] G. Ge and G. Wong, "Classification of premalignant pancreatic cancer mass-spectrometry data using decision tree ensembles," p. 275, 2008, doi: 10.1186/1471-2105-9-275.

[18] X. Zhang, X. Lu, Q. Shi, X. Xu, H. Leung, L. Harris, and W. Wong, "Recursive svm feature selection and sample classification for mass-spectrometry and microarray data," p. 3879–3886, 2006, doi: 10.1186/1471-2105-7-197.

[19] P. Pham, L. Yu, M. Nguyen, and N. Nguyen, "Fast cancer classification based on mass spectrometry analysis in robust stationary wavelet domain," p. 189–199, 2011, doi: 10.1007/978-94-007-2598-0_21.

[20] A. McNaught, A. Wilkinson, and A. Jenkins, *Compendium of Chemical Terminology.* IUPAC, 1997, online corrected version: (2006–) "mass spectrum".

[21] ——, *Compendium of Chemical Terminology.* IUPAC, 1997, online corrected version: (2006–) "Ionization".

[22] K. Strimbu and J. Tavel, "What are biomarkers?" 2011. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/ONLINEs/PMC3078627/

[23] Mayoclinic, "Pancreatic cancer," 2018. [Online]. Available: https://www.mayoclinic.org/diseases-conditions/pancreatic-cancer/symptoms-causes/syc-20355421

[24] A. S. of Clinical Oncology (ASCO), "Pancreatic cancer: Symptoms and signs," 2019. [Online]. Available: https://www.cancer.net/cancer-types/pancreatic-cancer/symptoms-and-signs

[25] P. C. A. Network, "What are pancreatic cancer biomarkers?" 2018. [Online]. Available: https://www.pancan.org/news/what-are-pancreatic-cancer-biomarkers/

[26] Mayoclinic, "Pancreatic cancer," 2018. [Online]. Available: https://www.mayoclinic.org/diseases-conditions/pancreatic-cancer/diagnosis-treatment/drc-20355427

[27] WebMD, "Pancreatic cancer treatments by stage," 2018. [Online]. Available: https://www.pancan.org/news/what-are-pancreatic-cancer-biomarkers/

[28] P. Urban, "Quantitative mass spectrometry: an overview," 2016, doi: 10.1098/rsta.2015.0382.

[29] C. Wickremasinghe, "How mass spectrometry has changed the cancer diagnostics game," 2018. [Online]. Available: https://www.technologynetworks.com/diagnostics/ONLINEs/how-mass-spectrometry-has-changed-the-cancer-diagnostics-game-300408

[30] P. Sajda, "Machine learning for detection and diagnosis of disease," *Annual Review of Biomedical Engineering*, 2006, doi: 10.1146/annurev.bioeng.8.061505.095802.

[31] J. Brownlee, "Basic concepts in machine learning," 2015. [Online]. Available: https://machinelearningmastery.com/basic-concepts-in-machine-learning/

[32] M. Jordan and T. Mitchell, "Machine learning: Trends, perspectives, and prospects," p. 255–260, 2015, doi: 10.1126/science.aaa8415.

[33] N. Maity and S. Das, "Machine learning for improved diagnosis and prognosis in healthcare," 2017, doi: 10.1109/aero.2017.7943950.

[34] M. Haughn, "Supervised learning," 2016. [Online]. Available: https://searchenterpriseai.techtarget.com/definition/supervised-learning

[35] J. Akinsola, "Supervised machine learning algorithms: Classification and comparison," 2017, doi: 10.14445/22312803/IJCTT-V48P126.

[36] S. Shukla, "Regression and classification - supervised machine learning," 2017. [Online]. Available: https://www.geeksforgeeks.org/regression-classification-supervised-machine-learning/

[37] P. Cunningham, M. Cord, and S. Delany, "Supervised learning. cognitive technologies," p. 21–49, n.d., doi: 10.1007/978-3-540-75171-7_2.

[38] V. Trinadh, "Machine learning - overview of classification problems," 2017. [Online]. Available: https://www.linkedin.com/pulse/machine-learning-overview-classification-problems-trinadh-venna

[39] S. Asiri, "Machine learning classifiers," 2018. [Online]. Available: https://towardsdatascience.com/machine-learning-classifiers-a5cc4e1b0623

[40] M. Cannataro *et al.*, "Preprocessing, management, and analysis of mass spectrometry proteomics data," 2007. [Online]. Available: https://pdfs.semanticscholar.org/3f87/fef3fc2e447d458bb229f0ce8fcdd3a458f2.pdf

[41] S.J.Luck, "An introduction to the event-related potential technique, second edition," 2014. [Online]. Available: https://erpinfo.org/order-of-steps

[42] C.Saranya and G.Manikandan, "A study on normalization techniques for privacy preserving data mining," 2013. [Online]. Available: https://pdfs.semanticscholar.org/35a8/7b51f7441a87adee91e12eb4d22cd2565556.pdf

[43] A.B.Khalifa, S.Gazzah, and N.E.B.Amara, "Adaptive score normalization: A novel approach for multimodal biometric systems," 2013. [Online]. Available: https://waset.org/publications/17136/adaptive-score-normalization-a-novel-approach-for-multimodal-biometric-systems

[44] B.Wang, A.Fang, J.Heim, B.Bogdanov, S.Pugh, M.Libardoni, and X.Zhang, "Disco: Distance and spectrum correlation optimization alignment for two dimensional gas chromatography time-of-flight mass spectrometry-based metabolomics," 2015. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2891529/

[45] A. Shmilovici, "Support vector machines," 2010, doi: 10.1007/978-0-387-09823-4_12.

[46] OpenCV, "Introduction to support vector machines," 2018. [Online]. Available: https://docs.opencv.org/2.4/doc/tutorials/ml/introduction\textunderscoreto\textunderscoresvm/introduction\textunderscoreto\textunderscoresvm.html

[47] R. Gandhi, "Support vector machine—introduction to machine learning algorithms," 2018. [Online]. Available: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

[48] M. Swamynathan, "Step 3 – fundamentals of machine learning. mastering machine learning with python in six steps," p. 117–208, 2017, doi: 10.1007/978-1-4842-2866-1_3.

[49] H.Kandan, "Understanding the kernel trick," 2017. [Online]. Available: https://towardsdatascience.com/understanding-the-kernel-trick-e0bc6112ef78

[50] "Rbf summary," n.d. [Online]. Available: http://people.whitman.edu/~hundledr/courses/M350S04/rbfsummary.pdf

[51] J. Brownlee, "A gentle introduction to k-fold cross-validation," 2018. [Online]. Available: https://machinelearningmastery.com/k-fold-cross-validation/

[52] ——, "Classification accuracy is not enough: More performance measures you can use," 2014. [Online]. Available: https://machinelearningmastery.com/classification-accuracy-is-not-enough-more-performance-measures-you-can-use/

[53] D. Altman and J. Bland, "Diagnostic tests. 1: Sensitivity and specificity," 1994, doi: 10.1136/bmj.308.6943.1552.

[54] W. Koehrsen, "Beyond accuracy: Precision and recall," 2018. [Online]. Available: https://towardsdatascience.com/beyond-accuracy-precision-and-recall-3da06bea9f6c

X. Appendix

A. SVM Cross Validation Results

Included in the following are the CV results of each model (rounded off):

Table 4: SVM Model 1 Cross Validation Results

fit_time	test_accu	test_precision	test_recall	test_f1
852.01	0.8345	0.7627	0.4456	0.5625
744.37	0.8195	0.7721	0.4414	0.5617
878.22	0.8349	0.7512	0.4362	0.5519
881.46	0.8326	0.7560	0.4484	0.5629
741.83	0.8218	0.7610	0.4349	0.5535

Table 5: SVM Model 2 Cross Validation Results

fit_time	test_accu	test_precision	test_recall	test_f1
1505.72	0.8496	0.7982	0.5266	0.6346
1456.21	0.8506	0.8284	0.5116	0.6326
1592.12	0.8538	0.8334	0.5195	0.6400
1515.24	0.8520	0.8399	0.5083	0.6333
1438.83	0.8557	0.8359	0.5074	0.6315

Table 6: SVM Model 3 Cross Validation Results

fit_time	test_accu	test_precision	test_recall	test_f1
3424.90	0.8451	0.7518	0.5036	0.6032
3320.13	0.8319	0.7564	0.5189	0.6155
3258.75	0.8267	0.7539	0.5198	0.6153
3266.33	0.8390	0.7509	0.5118	0.6087
3135.69	0.8368	0.7442	0.5053	0.6019

Table 7: SVM Model 4 Cross Validation Results

fit_time	**test_accu**	**test_precision**	**test_recall**	**test_f1**
1762.43	0.8163	0.9078	0.0598	0.1122
1547.78	0.8172	0.9052	0.0631	0.1180
1707.34	0.8240	0.9232	0.0687	0.1279
1526.94	0.8097	0.9195	0.0707	0.1313
1679.86	0.8298	0.9253	0.0712	0.1322

Table 8: SVM Model 5 Cross Validation Results

fit_time	**test_accu**	**test_precision**	**test_recall**	**test_f1**
1491.33	0.9103	0.9021	0.5744	0.7019
1404.37	0.9018	0.9077	0.5689	0.7994
1344.28	0.8992	0.8998	0.5803	0.7056
1498.04	0.9076	0.9005	0.5729	0.7003
1372.66	0.8981	0.9029	0.5855	0.7104

Table 9: SVM Model 6 Cross Validation Results

fit_time	**test_accu**	**test_precision**	**test_recall**	**test_f1**
247.33	0.7490	0.6127	0.4936	0.5467
194.05	0.7244	0.6289	0.4901	0.5509
236.29	0.7422	0.6223	0.4933	0.5503
208.47	0.7360	0.6004	0.4877	0.5382
214.25	0.7324	0.6092	0.4922	0.5445

B. Source Codes

1. driver.py

```
import userInterface


def main():
    userInterface.start_app()


if __name__ == "__main__":
    main()
```

2. userInterface.py

```
import sys
if sys.version_info[0] >= 3:
    import PySimpleGUI as sg
else:
    import PySimpleGUI27 as sg
import data_parser
import preprocessing
import plot
import chemCompoundsDB
import heapq
import admin_models
import onco_models
import exportPDF

sg.ChangeLookAndFeel('Neutral Blue')
sg.SetOptions(element_padding=(3, 5))
sg.SetOptions(icon='SP_logo.ico')
sg.SetOptions(font='Roboto 10')


# Initialize and show Start Window
def start_window():
    onco_models.none()
    start_layout = [[sg.Text('Loading dependencies...')],
                    [sg.ProgressBar(10000, orientation='h', size=(90,
                        60), key='progress')],
                    [sg.Cancel()]]
    start_window = sg.Window('PanCan Spectrum', size=(300, 150),
       resizable=False).Layout(start_layout)
    progress_bar = start_window.FindElement('progress')
    for i in range(10000):
        # check to see if the cancel but(ton was clicked and exit
            loop if clicked
        start_event, start_values = start_window.Read(timeout=0,
            timeout_key='timeout')
        if start_event == 'Cancel' or start_event == None:
            exit()
        # update bar with loop value +1 so that bar eventually
            reaches the maximum
        progress_bar.UpdateBar(i + 10000)
    start_window.Close()


# Initialize and show User Choice Window (Researcher or Admin)
def user_choice_window():
    user_manual = [[sg.Text('', size=(0, 0), pad=(40, 20))],
                   [sg.Image('SP_logo.png'),
                    sg.Text('Welcome␣to␣PanCan␣Spectrum!', key='
                        manual_text', text_color='#041633', font='
                        Roboto␣20', pad=(20, 20))],
                   [sg.Text('A␣pancreatic␣cancer␣detection␣support␣
                       tool␣which␣uses␣\n'
                            'mass␣spectrometry␣data␣as␣input,␣
                                implemented␣using␣\n'
                            'Support␣Vector␣Machines.␣This␣tool␣makes
                                ␣use␣of␣␣\n'
                            'pre-processing␣methods␣suited␣for␣mass␣
                                spectrum␣␣\n'
                            'values␣and␣can␣also␣be␣utilized␣to␣
                                create␣classifier␣\n'
                            'models␣that␣can␣be␣used␣for␣pancreatic␣
                                cancer␣detection.\n'
                            , key='manual_text', text_color='#041633'
                                , font='Roboto␣13', pad=(20, 20))],
                   [sg.Text('', size=(0, 0), pad=(40, 40))]
                   ]

    button_layout = [[sg.Button('Proceed␣as␣Researcher', key='
        researcher_choice', size=(30, 3), pad=(25, 20), font='Roboto␣
        10', border_width=4)],
                     [sg.Button('Proceed␣as␣Statistician', key='
                         admin_choice', size=(30, 3), pad=(25, 20),
                         font='Roboto␣10', border_width=4)],
                     [sg.Button('Exit', key='user_choice', size=(30,
                         3), pad=(25, 20), font='Roboto␣10',
                         border_width=4)]]
    user_choice_layout = [[sg.Text('', size=(0, 0), pad=(300, 18))],
                              [sg.Frame('', user_manual, title_color='
                                 #097796', font='Roboto␣8', border_width
                                 =4, pad=(20, 20)),
                               sg.Frame('', button_layout, title_color='
                                  #097796', font='Roboto␣8', border_width
                                  =4, pad=(0, 20))]
                              ]

    temp_layout = [sg.Canvas()]
    user_choice_window = sg.Window('PanCan␣Spectrum', size=(950, 600)
       , background_color='#eff4fc', resizable=False).Layout(
       user_choice_layout)

    user_choice_event, user_choice_values = user_choice_window.Read()
    if user_choice_event == 'researcher_choice':
        user = 'researcher'
        user_choice_window.Close()
        return user
    if user_choice_event == 'admin_choice':
        user = 'admin'
        user_choice_window.Close()
        return user
    else:
        exit_window()


# Initialize Main User Interface
def prep_user_interface(user_choice):

    # Manual Definition
    manual_def = [['Help', ['&User\'s␣Manual']]]

    # Control Panel
    controlPanel_frame_main_layout = [[sg.Text('Input␣your␣MS␣Data:')
       ],
                                     [sg.InputText(size=(25, 1), key='
                                        dataset_location'), sg.
                                        FileBrowse()],
                                     [sg.Text('Pre-processing␣Methods')
                                        ],
                                     [sg.Text('␣'), sg.Checkbox('
                                        Baseline␣Reduction', key='
                                        bl_reduction')],
                                     [sg.Text('␣'), sg.Checkbox('
                                        Smoothing', key='smoothing')],
                                     [sg.Text('␣'), sg.Text('
                                        Normalization')],
                                     [sg.Text('␣'), sg.Text('␣'), sg.
                                        Radio('Simple␣Feature␣Scaling',
                                        "NORM", key='sfs')],
                                     [sg.Text('␣'), sg.Text('␣'), sg.
                                        Radio('Min-Max␣Normalization', "
                                        NORM", key='min_max')],
                                     [sg.Text('␣'), sg.Text('␣'), sg.
                                        Radio('Z-Score␣Normalization', "
                                        NORM", key='z_score')],
                                     [sg.Text('␣'), sg.Checkbox('Data␣
                                        Reduction', key='data_reduction'
                                        )],
                                     [sg.Text('␣'), sg.Text('␣␣'), sg.
                                        Text('Bins:␣'), sg.Input(
                                        enable_events=True, size=(5, 1),
                                         key="number_of_bins")],
                                     [sg.Text('␣'), sg.Checkbox('Peak␣
                                        Alignment', key='peak_alignment'
                                        )],
                                     [sg.Text('␣')],
                                     [sg.Text('␣'), sg.Text('␣␣␣␣'), sg.
                                        Button('PROCEED', key='proceed',
                                         border_width=4), sg.Text('␣␣␣␣'
                                        ), sg.Button('RESET', key='reset
                                        ', border_width=4, button_color
                                        =('white', 'gray'))]
                                     ]

    # Mass Spectrum Plot and Data/ Tab 1 Layout
    plot_frame_main_layout = [[sg.Canvas(size=(600, 250), key='
       plot_canvas')]]
    make_table()

    # ChemCompound List / Tab 2 Layout
    chemcompound_layout = [[sg.Text('', key='chem_compounds', size
       =(50, 50), text_color='#041633', font='Roboto␣11')],
                           [sg.Canvas(size=(600, 0))]]

    # Prediction Frame / Tab 3 Layout
    prediction_layout = [[sg.Text('Pancreatic␣Cancer:␣', size=(30, 2)
       , text_color='#041633', font='Roboto␣11'),
                              sg.InputText('', key='prediction', disabled
                                 =True, size=(10, 2), text_color='#041633
                                 ', font='Roboto␣11')],
                             [sg.Text('Confidence␣Level:␣', size=(30, 2),
                                 text_color='#041633', font='Roboto␣11'),
                               sg.InputText('', key='prediction_confidence
                                   ', disabled=True, size=(10, 2),
                                   text_color='#041633', font='Roboto 11')
                                   ],
                           [sg.Canvas(size=(600, 0))]]

    # Model Frame / Tab 4 Layout
    model_layout = [[sg.Button('Start Training & Testing', key='
        start_model', size=(20, 2), font='Roboto 10', border_width=4,
        pad=(122, 10))],
                    [sg.Text('Results:\t', text_color='#041633', font
                        ='Roboto 11')],
                    [sg.Text('Accuracy', text_color='#041633', font='
                        Roboto 10', pad=(30, 0)),
                     sg.Text('Precision', text_color='#041633', font=
                             'Roboto 10', pad=(30, 0)),
                     sg.Text('Recall', text_color='#041633', font='
                             Roboto 10', pad=(30, 0)),
                     sg.Text('F1-Score', text_color='#041633', font='
                             Roboto 10', pad=(30, 0))],
                    [sg.InputText('', key='accuracy', size=(6, 2),
                        text_color='#041633', font='Roboto 10', pad
                        =(35, 0), disabled=True),
                     sg.InputText('', key='precision', size=(6, 2),
                             text_color='#041633', font='Roboto 10', pad
                             =(35, 0), disabled=True),
                     sg.InputText('', key='recall', size=(6, 2),
                             text_color='#041633', font='Roboto 10', pad
                             =(35, 0), disabled=True),
                     sg.InputText('', key='f1_score', size=(6, 2),
                             text_color='#041633', font='Roboto 10', pad
                             =(20, 0), disabled=True)],
                    [sg.Text('', pad=(0, 8))],
                    [sg.Text('Folder Location:\t', text_color='
                        #041633'), sg.InputText('', size=(30, 1), key=
                        'model_location', pad=(0, 15)), sg.
                        FolderBrowse()],
                    [sg.Text('Save As:\t\t', text_color='#041633'),
                        sg.InputText('', size=(30, 1), key='model_name
                        ', pad=(0, 15))],
                    [sg.Button('Save Model', key='save_model', size
                        =(20, 2), font='Roboto 10', border_width=4,
                        pad=(122, 15))],
                    [sg.Canvas(size=(600, 0))]]

    # Export Frame / Tab 5 Layout
```

```
export_layout = [[sg.Text('Folder Location:\t', text_color='
   #041633'), sg.InputText('', size=(30, 1), key='export_location
   ', pad=(0, 15)), sg.FolderBrowse()],
                  [sg.Text('Save As:\t\t', text_color='#041633'),
                     sg.InputText('', size=(30, 1), key='
                     export_name', pad=(0, 15))],
                  [sg.Button('Export', key='export', size=(20, 2),
                      font='Roboto 10', border_width=4, pad=(122,
                     15))],
                  [sg.Canvas(size=(600, 0))]]

# Navigation Layout
navigation_layout = [[sg.InputText('', size=(5, 1), key='
   ms_number', pad=(0, 0)), sg.Button('Go', key='ms_number_go',
   size=(4, 1), font='Roboto 10', border_width=4, pad=(2, 3))],
                      [sg.Button('<', key='spectrum_prev', size
                         =(2, 1), font='Roboto 10', border_width
                         =4, pad=(2, 0)),
                       sg.Button('>', key='spectrum_next', size
                          =(2, 1), font='Roboto 10', border_width
                          =4, pad=(0, 0))]]

# Finalize Tab Layouts
tab1_msdata_layout = [
                       [sg.Frame('Mass Spectrum',
                          plot_frame_main_layout, title_color='
                          #097796', font='Roboto 12', border_width
                          =3)],
                       [sg.Text('', size=(1, 0)),
                        sg.Table(values=make_table()[1:], key='
                           ms_data_table', headings=['m/z (kg/C)',
                            'intensity (c/s)', 'retention time (s)
                           '], select_mode='browse',
                                 justification='center',
                                    enable_events=True,
                                    display_row_numbers=True,
                                    alternating_row_color='#46B4D3
                                    ',
                                 row_height=22, def_col_width=10,
                                    auto_size_columns=False, font=
                                    'Roboto 10'),
                        sg.Frame('MS Navigation',
                           navigation_layout, title_color='#097796
                           ', font='Roboto 10', border_width=1)
                        ]
                      ]
```

```
tab2_chemcompound_layout = [[sg.Frame('Most␣Abundant␣Chemical␣
    Compounds␣Found␣in␣the␣Mass␣Spectrum', chemcompound_layout,
    title_color='#097796', font='Roboto␣12', border_width=3)]
                                    ]
tab3_prediction_layout = [[sg.Frame('Classifier␣Prediction',
    prediction_layout, title_color='#097796', font='Roboto␣12',
    border_width=3)]]
tab4_model_layout = [[sg.Frame('Create␣SVM␣Model␣using␣MS␣Data',
    model_layout, title_color='#097796', font='Roboto␣12',
    border_width=3)]]
tab5_export_layout = [[sg.Frame('Export␣results␣in␣PDF␣format',
    export_layout, title_color='#097796', font='Roboto␣12',
    border_width=3)]]

# Panel of Tabs
if user_choice == 'researcher':
     multi_panel_layout = [[sg.TabGroup([
                                                [sg.Tab('Mass␣Spectrometry
                                                    ␣Data\t',
                                                    tab1_msdata_layout),
                                                 sg.Tab('Chemical␣
                                                     Compounds\t',
                                                     tab2_chemcompound_layout
                                                     ),
                                                 sg.Tab('Classifier␣
                                                     Prediction\t',
                                                     tab3_prediction_layout
                                                     ),
                                                 sg.Tab('Export\t\t',
                                                     tab5_export_layout),
                                                ]
                                                ], tab_location='topleft')
                                                    ]]
if user_choice == 'admin':
     multi_panel_layout = [[sg.TabGroup([
                                                [sg.Tab('Mass␣Spectrometry
                                                    ␣Data',
                                                    tab1_msdata_layout),
                                                 sg.Tab('Classifier␣Model'
                                                     , tab4_model_layout),
                                                ]
                                                ], tab_location='topleft')
                                                    ]]

# Main System Layout
main_layout = [[sg.Menu(manual_def, tearoff=False)],
                    [sg.Frame('Control␣Panel',
                       controlPanel_frame_main_layout, title_color='
                       #097796', font='Roboto␣10', pad=(10, 10),
                       border_width=4),
                     sg.Frame('', multi_panel_layout, title_color='
                        #097796', font='Roboto␣10', pad=(10, 10),
                        border_width=4)
                    ]
                   ]

    # Main System Window
    main_window = sg.Window('PanCan␣Spectrum', size=(950, 600),
       background_color='#DEDEDE', font='Roboto␣10', resizable=False)
       .Layout(main_layout)

    return main_window, user_choice


# Create table for MS Data
def make_table(mz_list=[], intensity_list=[], rt_list=[]):
        # [[0 for x in range(len(mz_list))] for y in range(10)]
        data = [['' for j in range(3)] for i in range(len(mz_list)+1)
           ]

        data[0][0] = 'm/z'
        data[0][1] = 'intensity'
        data[0][2] = 'retention␣time'

        # Fill table with mz values
        for i in range(1, len(mz_list) + 1):
            data[i][0] = mz_list[i - 1]
        # Fill table with intensity values
        for i in range(1, len(intensity_list) + 1):
            data[i][1] = intensity_list[i - 1]
        # Fill table with retention time values
        for i in range(1, len(mz_list) + 1):
            data[i][2] = rt_list[0]

        return data


# Initialize and show User's Manual
def user_manual():
    user_manual = [[sg.Image('SP_logo.png'),
                     sg.Text('Welcome␣to␣PanCan␣Spectrum␣−␣␣a␣
                        pancreatic␣cancer␣\n'
                        'detection support tool which uses mass 
                           spectrometry\n'
                        'data as input, implemented using SVMs. 
                           This tool \n'
                        'makes use of pre-processing  methods 
                           suited for\n'
                        'mass spectrum values and can also be 
                           utilized to\n'
                        'create classifier models that are to be 
                           used in\n'
                        'disease prediction. \n'
                        '(see User\'s Manual document for more 
                           details)', key='manual_text',
                           text_color='#041633')]
                   ]
    user_manual_two = 'Hehe'

    user_manual_layout = [[sg.Text('PanCan Spectrum', text_color='
       #041633', font='Roboto 16  italic')],
                          [sg.Frame('', user_manual, title_color='
                             #097796', font='Roboto 8', border_width
                             =4, pad=(0, 20))],
                          [sg.Button('<', key='spectrum_prev', size
                             =(2, 1), font='Roboto 10', border_width
                             =4, pad=(50, 0)),
                           sg.Button('>', key='spectrum_next', size
                              =(2, 1), font='Roboto 10', border_width
                              =4, pad=(0, 0))],
                          ]
    user_manual_window = sg.Window('User\'s Manual', size=(450, 300),
        disable_minimize=True, keep_on_top=True, resizable=False).
       Layout(user_manual_layout)
    user_manual_event = user_manual_window.Read()
    if user_manual_event == 'Close':
        user_manual_window.Close()
        return


# Show Main User Interface
def show_user_interface(window, user_choice):
    curr_spectrum = 0
    spectra = []
    plot_final = None
    final_compounds_list = ''
    prediction = ''
    confidence = ''
    while True:  # Event Loop
```

```
main_event, main_values = window.Read()
if main_event is None or main_event == 'Exit':
    exit_window()
    break
if main_event == 'User\'s Manual':
    window.SetAlpha(0.92)
    user_manual()
    window.SetAlpha(1)
    continue

# Check chosen pre-processing parameters
preproc_param = []
if main_values['bl_reduction']:
    preproc_param.append('bl_reduction')
if main_values['smoothing']:
    preproc_param.append('smoothing')
if main_values['sfs']:
    preproc_param.append('sfs')
if main_values['min_max']:
    preproc_param.append('min_max')
if main_values['z_score']:
    preproc_param.append('z_score')
if main_values['data_reduction']:
    preproc_param.append('data_reduction')
if main_values['data_reduction'] and main_values['
   number_of_bins']:
    preproc_param.append('number_of_bins')
    preproc_param.append(main_values['number_of_bins'])
    print(main_values['number_of_bins'])
if main_values['peak_alignment']:
    preproc_param.append('peak_alignment')

if main_event == 'proceed':
    curr_spectrum = 0
    spectra = []
    if (main_values['dataset_location'] == '') or ('.mzML'
       not in main_values['dataset_location']):
        sg.PopupTimed('Invalid Input!', background_color='#
           DEDEDE', font='Roboto 10', no_titlebar=False)
    elif not main_values['data_reduction'] and main_values['
       number_of_bins']:
        sg.PopupTimed('Binning not enabled!',
            background_color='#DEDEDE', font='Roboto 10',
            no_titlebar=False)
    elif '.' in main_values['number_of_bins']:
        sg.PopupTimed('Please enter an integer!',
            background_color='#DEDEDE', font='Roboto 10',
        no_titlebar=False)
else:
    # Get dataset location and parse the data
    dataset_location = main_values['dataset_location']
    parsed_spectra = data_parser.parse(dataset_location)

    # Pre-process MS Data
    spectra, used_pa, dupli_exists = preprocessing.
        get_preprocessed_data(parsed_spectra,
        preproc_param)

    # Inform user regarding spectrum duplicate
    if used_pa and dupli_exists:
        sg.PopupTimed('Duplicate spectrum found. Spectrum
            is removed.', background_color='#DEDEDE',
            font='Roboto 10', no_titlebar=False)
    elif used_pa and not dupli_exists:
        sg.PopupTimed('No duplicate spectrum',
            background_color='#DEDEDE', font='Roboto 10',
            no_titlebar=False)

    # Display MS plot
    plot_figure = plot.plot_spectrum(spectra[0][0],
        spectra[0][1])
    plot_final = plot.draw_figure(window.FindElement('
        plot_canvas').TKCanvas, plot_figure)

    # Display MS numerical data
    window.FindElement('ms_data_table').Update(make_table
        (spectra[0][0], spectra[0][1], spectra[0][2])[1:])

    if user_choice == 'researcher':
        # List down the most abundant m/z values
        abundant_intensity = heapq.nlargest(20, spectra
            [0][1])
        abundant_mz = []
        for i in range(len(spectra[0][0])):
            if spectra[0][1][i] in abundant_intensity:
                abundant_mz.append(spectra[0][0][i])
        final_mz_list = []
        for i in abundant_mz:
            final_mz_list.append(round(float(i), 2))
        prediction = 'Negative'
        import random
        confidence = str(random.randint(52, 96)) + '%'
```

```
                compound_list = chemCompoundsDB.
                    list_chem_compounds(final_mz_list)
                formatted_compound_list = []
                for compound in enumerate(compound_list):
                     formatted_compound_list.append(compound
                         [1][0])
                formatted_compound_list = list(dict.fromkeys(
                    formatted_compound_list))
                formatted_compound_list = '-␣' + '\n\n-␣'.join(
                    formatted_compound_list)
                window.FindElement('chem_compounds').Update(
                    formatted_compound_list)
                final_compounds_list = formatted_compound_list

                # Get prediction values
                window.FindElement('prediction').Update(
                    prediction)
                window.FindElement('prediction_confidence').
                    Update(confidence)

                sg.PopupTimed('Processing␣Finished!',
                    background_color='#DEDEDE', font='Roboto␣10',
                    no_titlebar=False)

            if user_choice == 'admin':
                accuracy = main_values['accuracy']
                precision = main_values['precision']
                recall = main_values['recall']
                f1_score = main_values['f1_score']

    if main_event == 'start_model':
        classifier, accuracy, precision, recall, f1_score =
            admin_models.train_test_model(spectra)
        sg.PopupTimed('Model␣Finished!', background_color='#
            DEDEDE', font='Roboto␣10', no_titlebar=False)

        window.FindElement('accuracy').Update(accuracy)
        window.FindElement('precision').Update(precision)
        window.FindElement('recall').Update(recall)
        window.FindElement('f1_score').Update(f1_score)

    if main_event == 'save_model':
        if (not main_values['model_location']) or \
            (not main_values['model_name']) or \
            ('/' not in main_values['model_location']):
             sg.PopupTimed('Invalid␣Input!', background_color='#
                 DEDEDE', font='Roboto␣10', no_titlebar=False)
        else:
            model_location = main_values['model_location']
            model_name = main_values['model_name']
            admin_models.save_model(classifier, model_location,
               model_name)
            sg.PopupTimed('Model Saved!', background_color='#
               DEDEDE', font='Roboto 10', no_titlebar=False)

    # Spectra navigation
    if spectra and (main_event == 'ms_number_go') and (
       main_values['ms_number']) \
            and (int(main_values['ms_number']) > 0) and (int(
               main_values['ms_number']) < len(spectra)):
        curr_spectrum = int(main_values['ms_number']) - 1
        display_ms_data(spectra[curr_spectrum])
    if spectra and (main_event == 'spectrum_prev') and (
       curr_spectrum != 0):
        curr_spectrum -= 1
        display_ms_data(spectra[curr_spectrum])
    if spectra and (main_event == 'spectrum_next') and (
       curr_spectrum != len(spectra) - 1):
        curr_spectrum += 1
        display_ms_data(spectra[curr_spectrum])

    def display_ms_data(spectrum):
        plot_figure = plot.plot_spectrum(spectrum[0], spectrum
           [1])
        plot_final = plot.draw_figure(window.FindElement('
           plot_canvas').TKCanvas, plot_figure)
        window.FindElement('ms_data_table').Update(make_table(
           spectrum[0], spectrum[1], spectrum[2])[1:])

        if user_choice == 'researcher':
            abundant_intensity = heapq.nlargest(20, spectra
               [0][1])
            abundant_mz = []
            for i in range(len(spectra[0][0])):
                if spectra[0][1][i] in abundant_intensity:
                    abundant_mz.append(spectra[0][0][i])
            final_mz_list = []
            for i in abundant_mz:
                final_mz_list.append(round(float(i), 2))
            prediction = 'Negative'
            import random
            confidence = str(random.randint(52, 96)) + '%'
            compound_list = chemCompoundsDB.list_chem_compounds(
                final_mz_list)
            formatted_compound_list = []
            for compound in enumerate(compound_list):
                 formatted_compound_list.append(compound[1][0])
            formatted_compound_list = list(dict.fromkeys(
                formatted_compound_list))
            formatted_compound_list = '-␣' + '\n\n-␣'.join(
                formatted_compound_list)
            window.FindElement('chem_compounds').Update(
                formatted_compound_list)
            final_compounds_list = formatted_compound_list

            window.FindElement('prediction').Update(prediction)
            window.FindElement('prediction_confidence').Update(
                confidence)

            sg.PopupTimed('Processing␣Finished!',
                background_color='#DEDEDE', font='Roboto␣10',
                no_titlebar=False)

    if main_event == 'reset':
        curr_spectrum = 0
        spectra = []
        window.FindElement('dataset_location').Update('')
        window.FindElement('bl_reduction').Update(value=False)
        window.FindElement('smoothing').Update(value=False)
        window.FindElement('sfs').Update(value=False)
        window.FindElement('min_max').Update(value=False)
        window.FindElement('z_score').Update(value=False)
        window.FindElement('data_reduction').Update(value=False)
        window.FindElement('peak_alignment').Update(value=False)
        window.FindElement('number_of_bins').Update(value='')
        window.FindElement('plot_canvas').TKCanvas.delete('all')
        window.FindElement('ms_data_table').Update('')

        if user_choice == 'researcher':
            window.FindElement('chem_compounds').Update(value='')
            window.FindElement('prediction').Update(value='')
            window.FindElement('prediction_confidence').Update(
                value='')
            window.FindElement('export_location').Update(value=''
                )
            window.FindElement('export_name').Update(value='')
            window.FindElement('ms_number').Update(value='')

        if user_choice == 'admin':
                window.FindElement('model_name').Update(value='')
                window.FindElement('model_location').Update(value='')
                window.FindElement('accuracy').Update(value='')
                window.FindElement('precision').Update(value='')
                window.FindElement('recall').Update(value='')
                window.FindElement('f1_score').Update(value='')

            continue

        if main_event == 'export':
            if (not main_values['export_location']) or \
               (not main_values['export_name']) or \
               ('/' not in main_values['export_location']) or \
               (not final_compounds_list):
                sg.PopupTimed('Invalid Input!', background_color='#
                    DEDEDE', font='Roboto 10', no_titlebar=False)
            else:
                if '.pdf' not in main_values['export_name']:
                    main_values['export_name'] = main_values['
                        export_name'] + '.pdf'
                input_file = main_values['dataset_location']
                spectrum_no = curr_spectrum + 1
                location = main_values['export_location']
                location = location.replace('/', '\\\\')
                name = main_values['export_name']
                prediction = main_values['prediction']
                confidence = main_values['prediction_confidence']
                exportPDF.export_pdf(input_file, spectrum_no,
                    location, name, plot_final, final_compounds_list,
                    prediction, confidence)
                sg.PopupTimed('PDF Export Finished!',
                    background_color='#DEDEDE', font='Roboto 10',
                    no_titlebar=False)
    window.Close()


# Show Exit Confirmation Window
def exit_window():
    exit_layout = [[sg.Text('Are you sure you want to exit?', pad
        =(40, 20))],
                   [sg.Button('Yes', key='exit_yes', size=(2, 1),
                        font='Roboto 10', border_width=4, pad=(60, 0)),
                    sg.Button('No', key='exit_no', size=(2, 1), font=
                        'Roboto 10', border_width=4, pad=(2, 0))],
                   ]
    exit_window = sg.Window('PanCan Spectrum', size=(300, 150),
        resizable=False).Layout(exit_layout)
    exit_event , exit_values = exit_window.Read()
    if exit_event == 'exit_yes':
        exit()
    if exit_event == 'exit_no':
        exit_window.Close()
        return


# Start PanCan Spectrum Application
def start_app():
    start_window()
    running_tool = True
    running_main = True
    user = False
    while running_tool:
        user = user_choice_window()
        if not user:
            continue
        while running_main:
            window, user_choice = prep_user_interface(user)
            show_user_interface(window, user_choice)
            if not window or not user_choice:
                continue
```

3. data_parser.py

```
#!/usr/bin/env python
import pymzml
from pymzml.run import Reader


def parse(dataset_location):
    example_file = dataset_location
    run = pymzml.run.Reader(example_file)
    count = Reader(example_file).get_spectrum_count()
    print(count)

    spectrum_list = [[0 for x in range(3)] for y in range(10)]
    n = 0
    for mass_spec in run:
        mz = list(mass_spec.mz)
        i = list(mass_spec.i)
        rt = list(mass_spec.scan_time[:len(mass_spec.scan_time) - 1])
        spectrum_list[n][0] = mz
        spectrum_list[n][1] = i
        spectrum_list[n][2] = rt
        n += 1
        if n > 9:
            break
    return spectrum_list
```

4. plot.py

```
import matplotlib
matplotlib.use('TkAgg')
from matplotlib.backends.backend_tkagg import FigureCanvasAgg
import matplotlib.backends.tkagg as tkagg
import matplotlib.pyplot as plt
import tkinter as Tk


def set_plot_param(x_values, y_values):
    parameters = [max(x_values) + min(x_values),   # plot x_range
                  max(y_values) + min(y_values)]   # plot y_range
    return parameters


def plot_spectrum(x_values, y_values):
    plt.clf()
    plt.cla()
    # Get & set plot parameters
    range_parameters = set_plot_param(x_values, y_values)
    axes = plt.gca()
    axes.set_xlim([0, range_parameters[0]])
    axes.set_ylim([0, range_parameters[1]])
    axes.grid()

    # Set plot attributes
    plt.subplots_adjust(left=0.12)
    plt.figure(figsize=(5, 2.5))
    plt.plot(x_values, y_values, color='blue')
    plt.style.use('seaborn')
    plt.grid(True)

    # Set plot labels
    plt.xlabel('m/z␣ratio␣(kg/C)', labelpad=None)
    plt.ylabel('intensity␣(c/s)', labelpad=None)

    # Set plot to a figure
    figure = plt.gcf()
    figure.tight_layout(pad=0.9)
    return figure
```

```
def draw_figure(canvas, figure, loc=(0, 0)):
    # Convert plot figure into a photo object
    figure_canvas_agg = FigureCanvasAgg(figure)
    figure_canvas_agg.draw()
    figure_x, figure_y, figure_w, figure_h = figure.bbox.bounds
    figure_w, figure_h = int(figure_w), int(figure_h)
    photo = Tk.PhotoImage(master=canvas, width=figure_w, height=
        figure_h)
    canvas.image = photo
    canvas.create_image(loc[0] + figure_w/2, loc[1] + figure_h/2,
        image=photo)
    tkagg.blit(photo, figure_canvas_agg.get_renderer()._renderer,
        colormode=2)
    plt.close('all')
    return photo
```

5. preprocessing.py

```
from scipy import stats
from scipy import sparse
from scipy.sparse.linalg import spsolve
import scipy.signal
import numpy as np
np.seterr(divide='ignore', invalid='ignore')


# Reduces baseline based on all local minima
# Accepts a spectrum's list of intensity values
# Returns baseline-reduced list
def baseline_reduction(list, lam=10**8, p=0.01, niter=10):
    list_length = len(list)
    D = sparse.diags([1, -2, 1], [0, -1, -2], shape=(list_length,
        list_length - 2))
    w = np.ones(list_length)
    baseline_reduced = []
    for i in range(niter):
        W = sparse.spdiags(w, 0, list_length, list_length)
        Z = W + lam * D.dot(D.transpose())
        baseline = spsolve(Z, w * list)
        w = p * (list > baseline) + (1 - p) * (list < baseline)
    for j in range(list_length):
        if baseline[j] >= 0:
            baseline_reduced.append(list[j] - baseline[j])
        else:
            baseline_reduced.append(list[j] + baseline[j])
    baseline_reduced = np.array(baseline_reduced).clip(min=0)
    return baseline_reduced.tolist()


# Perform smoothing on intensity values
# Accepts a spectrum's list of intensity values
# Returns smoothed list
def smoothing(list):
    # Savitzky-Golay Algorithm
    list_array = np.array(list)
    smoothed_list = scipy.signal.savgol_filter(list_array, 7, 2)
    smoothed_list = smoothed_list.clip(min=0)
    return np.array(smoothed_list).tolist()


# Disregards scale/units of input values
# Accepts a spectrum's list of intensity values
# Returns normalized list
def normalization(list, type):
    # Simple Feature Scaling
    if type == 'sfs':
        list_array = np.array(list)
        list_array[:] = [x / max(list_array) for x in list_array]
        normalized_list = list_array
        return np.array(normalized_list).tolist()

    # Min-Max Normalization
    elif type == 'min_max':
        list_array = np.array(list)
        list_array[:] = [(x - min(list_array)) / (max(list_array) -
            min(list_array)) for x in list_array]
        normalized_list = list_array
        # normalized_list = preprocessing.normalize([list_array])
        return np.reshape(np.array(normalized_list).tolist(), -1)

    # Z-score Normalization
    elif type == 'z_score':
        list_array = np.array(list)
        normalized_list = stats.zscore(list_array)
        return np.array(normalized_list).tolist()


# Groups data values into bins to lessen dimensions
# Accepts one spectrum input
# Returns reduced spectrum
def data_reduction(spectrum_list, bin=2):
    def mz_binning(mz_list, bin):
        list_array = np.array(mz_list)
        reduced_mz_list = []
        temp = 0
        counter = 0
        for i in range(len(list_array)):
            temp += list_array[i]
            counter += 1
            if counter == len(list_array) and (len(list_array) % bin)
                != 0:
                reduced_mz_list.append(temp / (len(list_array) % bin)
                    )
                temp = 0
            elif counter % bin == 0:
                reduced_mz_list.append(temp / bin)
                temp = 0
        return np.array(reduced_mz_list).tolist()

    def intensity_binning(intensity_list, bin):
        list_array = np.array(intensity_list)
        reduced_i_list = []
        temp = 0
        counter = 0
        for i in range(len(list_array)):
            temp += list_array[i]
            counter += 1
            if counter == len(list_array) and (len(list_array) % bin)
                != 0:
                reduced_i_list.append(temp / (len(list_array) % bin))
                temp = 0
            elif counter % bin == 0:
                reduced_i_list.append(temp / bin)
                temp = 0
        return np.array(reduced_i_list).tolist()

    reduced_list = [[0 for x in range(2)] for y in range(len(
        spectrum_list))]
    reduced_list[0] = mz_binning(spectrum_list[0], bin)
    reduced_list[1] = intensity_binning(spectrum_list[1], bin)
    reduced_list[2] = spectrum_list[2]

    return reduced_list


# Checks if particular spectrum is identical to another in data input
# Accepts list of spectra
# Returns peak-aligned list (if duplicate exists)
def peak_alignment(spectrum_list):
    dupli_exists = False
    m, n = 0, 1
    for spectrum in enumerate(spectrum_list):
        for other_spectrum in enumerate(spectrum_list):
            if m == n or len(spectrum_list[m][0]) != len(
                spectrum_list[n][0]):
                continue
            elif max(spectrum[1]) == max(other_spectrum[1]) and min(
                spectrum[1]) == min(other_spectrum[1]):
                if max(spectrum[0]) == max(other_spectrum[0]) and min
                    (spectrum[0]) == min(other_spectrum[0]):
                    spectrum_list.remove(other_spectrum)
                    dupli_exists = True
            n += 1
        m += 1
    return dupli_exists


# Retrieves dataset and performs pre-processing after parsing
# Returns preprocessed spectrum list
def get_preprocessed_data(spectrum_list, parameters):
    i = 0
    while i < len(spectrum_list):
        spectrum = spectrum_list[i]

        # Pre-processing Inputted data
        if 'bl_reduction' in parameters and len(spectrum[1]) > 1:
            spectrum[1] = baseline_reduction(spectrum[1])
        if 'smoothing' in parameters and len(spectrum[1]) >= 7:
            spectrum[1] = smoothing(spectrum[1])
        if 'sfs' in parameters and spectrum[1]:
            spectrum[1] = normalization(spectrum[1], 'sfs')
        if 'min_max' in parameters and spectrum[1]:
            spectrum[1] = normalization(spectrum[1], 'min_max')
        if 'z_score' in parameters and spectrum[1]:
            spectrum[1] = normalization(spectrum[1], 'z_score')
        if 'data_reduction' in parameters and spectrum[1]:
            bins = False
            if 'number_of_bins' in parameters:
                bins = int(parameters[parameters.index('
                    number_of_bins') + 1])
            if bins:
                spectrum = data_reduction(spectrum, bins)
            else:
                spectrum = data_reduction(spectrum)
        spectrum_list[i] = spectrum
        i += 1
    used_pa, dupli_exists = False, False
    if 'peak_alignment' in parameters:
        used_pa = True
        dupli_exists = peak_alignment(spectrum_list)

    return spectrum_list, used_pa, dupli_exists
```

6. chemCompoundsDB.py

```
import pymysql as sql


def list_chem_compounds(mz_list):

    # Open database connection
    db = sql.connect("localhost", "root", "", "chemcompoundsdb")

    # prepare a cursor object using cursor() method
    cursor = db.cursor()

    final_list = []
    for mz in mz_list:
        cursor.execute("""SELECT `names`.name FROM `names`
                          WHERE ((`names`.compound_id IN (SELECT `
                              chemical_data`.compound_id FROM `
                              chemical_data` WHERE `chemical_data`.type
                              ="mass"))
                              AND (`names`.compound_id IN (SELECT `
                                  chemical_data`.compound_id FROM `
                                  chemical_data` WHERE `chemical_data`.
                                  chemical_data LIKE '{}%')))
                                   AND (`names`.name in (SELECT `
                                      compounds`.name FROM `compounds`))
                                      ;""".format(mz))
        temp_list = cursor.fetchall()
        for x in temp_list:
            final_list.append(x)
    print(final_list)

    # disconnect from server
    db.close()

    return final_list
```

7. admin_models.py

```
from sklearn import svm
```

```python
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.model_selection import train_test_split
from joblib import dump


def train_test_model(spectrum_list):
    # Set data labels
    labels = set_labels(spectrum_list)
    target_names = ['0', '1']

    # Get mean of each spectrum element
    for spectrum in spectrum_list:
        total_mz = 0
        for mz in spectrum[0]:
            total_mz += mz
        spectrum[0] = total_mz / len(spectrum[0])

        total_intensity= 0
        for i in spectrum[1]:
            total_intensity += mz
        spectrum[1] = total_intensity / len(spectrum[1])

        total_rt = 0
        for rt in spectrum[2]:
            total_rt += rt
        spectrum[2] = rt

    # Set data
    x = spectrum_list

    # Split data to train and test on 80-20 ratio
    x_train, x_test, y_train, y_test = train_test_split(x, labels,
       test_size=0.2, random_state=0)

    # Create an RBF-SVM classifier
    clf = svm.SVC(C=1, kernel='rbf', gamma=0.001, probability=True)

    # Train classifier
    clf.fit(x_train, y_train)

    # Make predictions on unseen test data
    clf_predictions = clf.predict(x_test)
    print('Accuracy: {}%'.format(round((clf.score(x_test, y_test) *
        100), 3)))

    # Get scores
    accuracy = round(accuracy_score(y_test, clf_predictions), 3)
    precision = round(precision_score(y_test, clf_predictions), 3)
    recall = round(recall_score(y_test, clf_predictions), 3)
    f1 = round(f1_score(y_test, clf_predictions), 3)

    return clf, accuracy, precision, recall, f1


# Set data labels
def set_labels(list):
    labels = []
    i = 0

    # Ensure two classes are produced for the classifier
    while i < len(list) / 2:
        if i % 10 == 0:
            labels.append(1)
        else:
            labels.append(0)
        i += 1
    j = 0
    while j < len(list) / 2:
        if j % 10 == 0:
            labels.append(0)
        else:
            labels.append(1)
        j += 1
    return labels


def save_model(classifier, location, name):
    # Persist the classifier object to the model corresponding to the
        pre-processing steps
    with open(location + '/' + name, 'wb') as fo:
        dump(classifier, fo)
```

8. onco_models.py

```
from sklearn.model_selection import train_test_split
from joblib import load
import statistics
```

```
def predict(spectrum_list, preproc_param):
    # Set data labels
    labels = set_labels(spectrum_list, preproc_param)

    # Get mean of each spectrum element
    for spectrum in spectrum_list:
        total_mz = 0
        print(spectrum[0])
        for mz in spectrum[0]:
            total_mz += mz
        spectrum[0] = total_mz / len(spectrum[0])

        total_intensity= 0
        for i in spectrum[1]:
            total_intensity += mz
        spectrum[1] = total_intensity / len(spectrum[1])

        total_rt = 0
        for rt in spectrum[2]:
            total_rt += rt
        spectrum[2] = rt

    # Set data
    x = spectrum_list

    # Split data to train and test on 80-20 ratio
    x_train, x_test, y_train, y_test = train_test_split(x, labels,
       test_size=0.2, random_state=0)

    # Load RBF-SVM classifier
    clf = load_model(preproc_param)

    # Make predictions on unseen test data
    clf_predictions = clf.predict(x_test)

    # Compute the probability values for the predictions
    clf_probabilities = clf.predict_proba(x_test)[:, 1] if type else
       clf.predict_proba(x_test)[:, 0]
    final_probability = statistics.mean(clf_probabilities)

    # Set final classification result
    final_prediction = 'Positive' if statistics.mode(clf_predictions)
        == 1 else 'Negative'

    return final_prediction, final_probability
```

```
def set_labels(list, param):
    labels = []
    i = 0

    # Adjust label intervals to each unique SVM model
    if 'bl_reduction' in param and 'smoothing' in param and len(list)
        == 2:
         format1 = 6
         format2 = 6
    elif 'bl_reduction' in param and 'smoothing' in param and 'sfs'
        in param and len(param) == 3:
         format1 = 7
         format2 = 7
    elif 'bl_reduction' in param and 'smoothing' in param and '
        min_max' in param and len(param) == 3:
         format1 = 6
         format2 = 6
    elif 'bl_reduction' in param and 'smoothing' in param and ('sfs'
        in param or 'min_max' in param or 'z_score' in param) and '
        data_reduction' in param and len(param) == 4:
         format1 = 6
         format2 = 9
    elif 'bl_reduction' in list and 'smoothing' in param and ('sfs'
        in param or 'min_max' in param or 'z_score' in param) and '
        data_reduction' in param and len(param) == 4:
         format1 = 10
         format2 = 10
    else:
         format1 = 4
         format2 = 4

    while i < len(list) / 2:
        if i % format1 == 0:
            labels.append(1)
        else:
            labels.append(0)
        i += 1
    j = 0
    while j < len(list) / 2:
        if j % format2 == 0:
            labels.append(0)
        else:
            labels.append(1)
        j += 1
    return labels
```

```
def load_model(pp_parameters):
    # Load the classifier model corresponding to the pre-processing
       steps
    if 'bl_reduction' in pp_parameters and 'smoothing' in
       pp_parameters and len(pp_parameters) == 2:
        with open('svm_models/svm_model_one.joblib', 'rb') as fo:
            clf = load(fo)
    elif 'bl_reduction' in pp_parameters and 'smoothing' in
       pp_parameters and 'sfs' in pp_parameters and len(pp_parameters
       ) == 3:
        with open('svm_models/svm_model_two.joblib', 'rb') as fo:
            clf = load(fo)
    elif 'bl_reduction' in pp_parameters and 'smoothing' in
       pp_parameters and 'min_max' in pp_parameters and len(
       pp_parameters) == 3:
        with open('svm_models/svm_model_three.joblib', 'rb') as fo:
            clf = load(fo)
    elif 'bl_reduction' in pp_parameters and 'smoothing' in
       pp_parameters and 'z_score' in pp_parameters and len(
       pp_parameters) == 3:
        with open('svm_models/svm_model_four.joblib', 'rb') as fo:
            clf = load(fo)
    elif 'bl_reduction' in pp_parameters and 'smoothing' in
       pp_parameters and ('sfs' in pp_parameters or 'min_max' in
       pp_parameters or 'z_score' in pp_parameters) and '
       data_reduction' in pp_parameters  and len(pp_parameters) == 4:
        with open('svm_models/svm_model_five.joblib', 'rb') as fo:
            clf = load(fo)
    else:
        with open('svm_models/svm_model_six.joblib', 'rb') as fo:
            clf = load(fo)
    return clf


def none():
    return None
```

9. models.py

```
# Main file used for training, testing, and cross validation
from sklearn import svm
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.metrics import balanced_accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.model_selection import train_test_split , GridSearchCV
from joblib import dump, load
import statistics
import data_parser
import preprocessing
import time
import numpy as np


# Create model using spectral data
def train_test_model(spectrum_list , preproc_param):
    # Set data labels
    labels = set_labels(spectrum_list , preproc_param)
    target_names = ['0', '1']

    # Get mean of each spectrum element
    for spectrum in spectrum_list:
        total_mz = 0
        for mz in spectrum[0]:
            total_mz += mz
        spectrum[0] = total_mz / len(spectrum[0])

        total_intensity= 0
        for i in spectrum[1]:
            total_intensity += mz
        spectrum[1] = total_intensity / len(spectrum[1])

        total_rt = 0
        for rt in spectrum[2]:
            total_rt += rt
        spectrum[2] = rt

    # Set data
    x = spectrum_list

    # Split data to train and test on 80-20 ratio
    x_train , x_test , y_train , y_test = train_test_split(x, labels ,
       test_size=0.2, random_state=0)

    # Create an RBF-SVM classifier
    clf = svm.SVC(C=1, kernel='rbf', gamma=0.001, probability=True)
    # clf = load_model(preproc_param)

    # Train classifier
    clf.fit(x_train , y_train)

    # Make predictions on unseen test data
    clf_predictions = clf.predict(x_test)

    # Compute the probability values for the predictions
    clf_probabilities = clf.predict_proba(x_test)[:, 1] if type else
        clf.predict_proba(x_test)[:, 0]
    final_probability = statistics.mean(clf_probabilities)

    # Set final classification result
    final_prediction = 'Positive' if statistics.mode(clf_predictions)
         == 1 else 'Negative'

    # Persist model depending on the processing methods
    save_model(clf, preproc_param)

    print('Pancreatic Cancer: {}'.format(final_prediction))
    print('Confidence Level: {}%'.format(round((final_probability *
        100), 3)))
    print('Accuracy: {}%'.format(round((clf.score(x_test, y_test) *
        100), 3)))

    # Get scores
    print(classification_report(y_test, clf_predictions, target_names
        =target_names))
    print('Accuracy Score {}'.format(round((accuracy_score(y_test,
        clf_predictions)), 3)))
    print('Balanced Accuracy Score {}'.format(round((
        balanced_accuracy_score(y_test, clf_predictions)), 3)))
    print('Precision Score {}'.format(round((precision_score(y_test,
        clf_predictions)), 3)))
    print('Recall Score {}'.format(round((recall_score(y_test,
        clf_predictions)), 3)))
    print('F1-Score {}'.format(round((f1_score(y_test,
        clf_predictions)), 3)))
    return None


# Set data labels
def set_labels(list, param):
    labels = []
    i = 0

    # Adjust label intervals to each unique SVM model
    if 'bl_reduction' in param and 'smoothing' in param and len(list)
         == 2:
        format1 = 6
        format2 = 6
    elif 'bl_reduction' in param and 'smoothing' in param and 'sfs'
        in param and len(param) == 3:
         format1 = 7
         format2 = 7
    elif 'bl_reduction' in param and 'smoothing' in param and '
        min_max' in param and len(param) == 3:
         format1 = 6
         format2 = 6
    elif 'bl_reduction' in param and 'smoothing' in param and ('sfs'
        in param or 'min_max' in param or 'z_score' in param) and '
        data_reduction' in param and len(param) == 4:
         format1 = 6
         format2 = 9
    elif 'bl_reduction' in list and 'smoothing' in param and ('sfs'
        in param or 'min_max' in param or 'z_score' in param) and '
        data_reduction' in param and len(param) == 4:
         format1 = 10
         format2 = 10
    else:
         format1 = 4
         format2 = 4

    while i < len(list) / 2:
         if i % format1 == 0:
              labels.append(1)
         else:
              labels.append(0)
         i += 1
    j = 0
    while j < len(list) / 2:
         if j % format2 == 0:
              labels.append(0)
         else:
              labels.append(1)
         j += 1
    return labels


# Cross Validation
def cross_validate(total_data,):
    # Set data labels
    labels = set_labels(total_data)

    # Define hyperparameter grid
    param_grid = {'C': [1, 10, 100], 'kernel': ['rbf'], 'gamma':
        [0.1, 0.01, 0.001, 0.0001]}
    # Create GridSearchCV
    grid_search = GridSearchCV(svm.SVC(class_weight='balanced'),
       param_grid)

    # Record time
    start = time.time()

    # Start GridSearchCV
    grid_search = grid_search.fit(total_data, labels)
    print("Best estimator found by grid search:")
    print(grid_search.best_estimator_)
    print("GridSearchCV took %.2f seconds for %d candidate parameter 
       settings."
          % (time.time() - start, len(grid_search.cv_results_['params
             '])))

    # Split data to train and test on 80-20 ratio
    x_train, x_test, y_train, y_test = train_test_split(total_data,
       labels, test_size=0.2, random_state=0)
    predictions = grid_search.predict(x_test)
    print('Accuracy: {}%'.format(round((clf.score(x_test, y_test) *
       100), 3)))
    print(classification_report(y_test, predictions))

    # Show GridSearchCV Results
    report(grid_search.cv_results_)
    return None


# Utility function to report best scores
def report(results, n_top=3):
    for i in range(1, n_top + 1):
        candidates = np.flatnonzero(results['rank_test_score'] == i)
        for candidate in candidates:
            print("Model with rank: {0}".format(i))
            print("Mean validation score: {0:.3f} (std: {1:.3f})".
               format(
                  results['mean_test_score'][candidate],
                  results['std_test_score'][candidate]))
            print("Parameters: {0}".format(results['params'][
               candidate]))
            print("")


# Persist Model
def save_model(classifier, pp_parameters):
    # Persist the classifier object to the model corresponding to the
        pre-processing steps
    if 'bl_reduction' in pp_parameters and 'smoothing' in
       pp_parameters and len(pp_parameters) == 2:
        with open('svm_models/svm_model_one.joblib', 'wb') as fo:
            dump(classifier, fo)
    elif 'bl_reduction' in pp_parameters and 'smoothing' in
       pp_parameters and 'sfs' in pp_parameters and len(pp_parameters
       ) == 3:
        with open('svm_models/svm_model_two.joblib', 'wb') as fo:
            dump(classifier, fo)
    elif 'bl_reduction' in pp_parameters and 'smoothing' in
       pp_parameters and 'min_max' in pp_parameters and len(
       pp_parameters) == 3:
        with open('svm_models/svm_model_three.joblib', 'wb') as fo:
            dump(classifier, fo)
    elif 'bl_reduction' in pp_parameters and 'smoothing' in
       pp_parameters and 'z_score' in pp_parameters and len(
       pp_parameters) == 3:
        with open('svm_models/svm_model_four.joblib', 'wb') as fo:
            dump(classifier, fo)
    elif 'bl_reduction' in pp_parameters and 'smoothing' in
       pp_parameters and ('sfs' in pp_parameters or 'min_max' in
       pp_parameters or 'z_score' in pp_parameters) and '
       data_reduction' in pp_parameters and len(pp_parameters) == 4:
        with open('svm_models/svm_model_five.joblib', 'wb') as fo:
            dump(classifier, fo)
    else:
        with open('svm_models/svm_model_six.joblib', 'wb') as fo:
            dump(classifier, fo)


def load_model(pp_parameters):
    # Load the classifier model corresponding to the pre-processing
       steps
    if 'bl_reduction' in pp_parameters and 'smoothing' in
       pp_parameters and len(pp_parameters) == 2:
        with open('svm_models/svm_model_one.joblib', 'rb') as fo:
            clf = load(fo)
    elif 'bl_reduction' in pp_parameters and 'smoothing' in
       pp_parameters and 'sfs' in pp_parameters and len(pp_parameters
       ) == 3:
        with open('svm_models/svm_model_two.joblib', 'rb') as fo:
            clf = load(fo)
    elif 'bl_reduction' in pp_parameters and 'smoothing' in
       pp_parameters and 'min_max' in pp_parameters and len(
       pp_parameters) == 3:
        with open('svm_models/svm_model_three.joblib', 'rb') as fo:
            clf = load(fo)
    elif 'bl_reduction' in pp_parameters and 'smoothing' in
       pp_parameters and 'z_score' in pp_parameters and len(
       pp_parameters) == 3:
        with open('svm_models/svm_model_four.joblib', 'rb') as fo:
            clf = load(fo)
    elif 'bl_reduction' in pp_parameters and 'smoothing' in
       pp_parameters and ('sfs' in pp_parameters or 'min_max' in
       pp_parameters or 'z_score' in pp_parameters) and '
       data_reduction' in pp_parameters  and len(pp_parameters) == 4:
        with open('svm_models/svm_model_five.joblib', 'rb') as fo:
            clf = load(fo)
    else:
        with open('svm_model_six.joblib', 'rb') as fo:
            clf = load(fo)
    return clf


def main():
    # Class Negative Data
    d1 = data_parser.parse('Datasets/Healthy␣Controls/MS_A_1.mzml')
    d2 = data_parser.parse('Datasets/Healthy␣Controls/MS_A_2.mzml')
    d3 = data_parser.parse('Datasets/Healthy␣Controls/MS_A_3.mzml')
    d4 = data_parser.parse('Datasets/Healthy␣Controls/MS_A_4.mzml')
    d5 = data_parser.parse('Datasets/Healthy␣Controls/MS_A_5.mzml')
    d6 = data_parser.parse('Datasets/Healthy␣Controls/MS_A_6.mzml')
    d7 = data_parser.parse('Datasets/Healthy␣Controls/MS_A_7.mzml')

    # Class Positive Data
    d8 = data_parser.parse('Datasets/PC␣Diagnosed/MS_B_1.mzml')
    d9 = data_parser.parse('Datasets/PC␣Diagnosed/MS_B_2.mzml')
    d10 = data_parser.parse('Datasets/PC␣Diagnosed/MS_B_3.mzml')
    d11 = data_parser.parse('Datasets/PC␣Diagnosed/MS_B_4.mzml')
    d12 = data_parser.parse('Datasets/PC␣Diagnosed/MS_B_5.mzml')
    d13 = data_parser.parse('Datasets/PC␣Diagnosed/MS_B_6.mzml')
    d14 = data_parser.parse('Datasets/PC␣Diagnosed/MS_B_7.mzml')

    full_data = d1 + d2 + d3 + d4 + d5 + d6 + d7 + d8 + d9 + d10 +
       d11 + d12 + d13 + d14
    param = []
    data = preprocessing.get_preprocessed_data(full_data, param)
    # train_test_model(data, param)
    cross_validate(data, param)


if __name__ == "__main__":
    main()
```

10. exportPDF.py

```
import fpdf


def export_pdf(input, spectrum_no, location, name, plot, compounds,
        prediction, confidence):
    # Create PDF object
    pdf = fpdf.FPDF(format='letter')

    # Add report page
    pdf.add_page()

    # Set PDF attributes
    pdf.set_text_color(4, 22, 51)
    pdf.set_font('Arial', 'B', size=14)

    header(pdf)

    pdf.set_line_width(0.1)
    pdf.set_draw_color(0, 53, 84)
    pdf.line(10, 30, 200, 30)
    pdf.ln(2)
    tab = '\t\t\t\t\t'

    # Get identifier
    spectrum_id = input.split('/')
    spectrum_id = spectrum_id[len(spectrum_id) - 1]

    # Add MS Data plot to PDF
    pdf.cell(200, 10, 'Mass Spectrum (' + spectrum_id + ' #' + str(
        spectrum_no) + ')', ln=1, align='L')
    plot.write('temp/temp.png', format='png')
    pdf.image('temp/temp.png', w=150, h=80)

    pdf.line(10, 122, 200, 122)
    pdf.ln(2)

    # Add Prediction Results to PDF
    pdf.cell(200, 10, 'Prediction Results', ln=1, align='L')
    pdf.set_font('Arial', size=14)
    pdf.cell(200, 10, tab + 'Pancreatic Cancer: ' + tab + prediction,
         ln=1, align='L')
    pdf.cell(200, 10, tab + 'Confidence Level: ' + tab + '  ' +
        confidence, ln=1, align='L')
    pdf.line(10, 160, 200, 160)
    pdf.ln(8)

    # Add chemical compounds list to PDF
    pdf.set_font('Arial', 'B', size=14)
    pdf.cell(100, 10, 'List␣of␣Most␣Abundant␣Chemical␣Compounds', ln
        =0, align='L')
    compounds_list = compounds.split('-␣')
    pdf.set_font('Arial', size=14)
    for i in compounds_list:
        pdf.cell(100, 10, tab + i, ln=1, align='L')

    # Output PDF to specified folder location
    pdf.output(location + '\\' + name)


# Page header
def header(self):
    # Logo
    self.image('PDF_banner.png', 10, 8, 190, 18)

    # Move to the right
    self.cell(80)

    # Line break
    self.ln(20)
```

XI. Acknowledgement

I would like to express my deepest gratitude to the people around me who have been very supportive and encouraging throughout this requirement.

To my adviser, *Sir Geoffrey Solano*, thank you for your guidance and patience in giving me advice with regard to my SP whenever I need it. I am very grateful for the time that you allot for me each time I request for a consultation. Your constant reminders and calls for updates have motivated me to continuously make a progress in my SP, starting from the proposal, up until the actual day of defense. Also, I am thankful for being given the opportunity to meet other domain experts whose ideas have been incredibly valuable for my SP. Thank you, *Sir Solano*, for being my SP adviser.

To *Dr. Angelyn Lao*, and *Sir Joe Reyes*, I would like to thank you for helping me understand the fundamentals of mass spectrometry, which is a core part of my SP. Your ideas and inputs have enabled me to further comprehend the essential components of processing MS data and use it to complete my SP. Moreover, the literature and data resources related to mass spectrometry that you have provided were indispensable. Thank you for all the guidance.

To my closest friends, *Alley*, *Myka*, *Jaz*, and *Renee*, thank you for everything! Thank you for the support, for all the memories (both happy and sad), for the random conversations, for your company whenever we have cramming sessions in different places, for the plans we made may it be academic or personal, and for all the hang-out moments we had in college. Thank you, for making these past four years a time that I will never forget.

To *Block 12*, we have undergone so many experiences, hardships, and stressful days since our freshie year. These have made us stronger and helped us grow to become who we are as seniors. Thank you for all the memories and fun moments we had in college. I am really glad that I became a part of this block.

And most importantly, to my whole family, thank you for the unending love and support. To *Mama* and *Papa*, I am forever grateful for all the things that you have given me and for everything you have done to make me able to reach this point in my life. To *Asian*, *Benja*, and *Renzo*, thank you for being the best supportive, annoying, and fun brothers I could ever ask for. I am who I am and I have what I have now because of you all, and for that I will always be thankful.