University of the Philippines – Manila

College of Arts and Sciences

Department of Physical Sciences and Mathematics

PARKINSON'S DISEASE
DECISION SUPPORT SYSTEM

A Special Problem in Partial Fulfillment

Of the Requirements for the Degree of

Bachelor of Science in Computer Science

Aemilia Leupoldine III P. Tibayan

June 2015

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 “Parkinson’s Disease Decision Support System” prepared and submitted by Aemilia Leupoldine III P. Tibayan in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science has been examined and is recommended for acceptance.

____

Perlita E. Gasmen, M. Sc. (*candidate*)
Adviser

EXAMINERS:

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

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

____

Ma. Sheila A. Magboo, M.Sc.
Unit Head
Mathematical and Computing Sciences Unit
Department of Physical Sciences
and Mathematics

____

Marcelina B. Lirazan, Ph.D.
Chair
Department of Physical Sciences
and Mathematics

____

Alex C. Gonzaga, Ph.D., Dr.Eng.
Dean
College of Arts and Sciences

Abstract

Parkinson’s Disease is a neurodegenerative disorder that is slowly progressive. Among the common symptoms of PD are tremor, shaking and dysarthria. This refers to a group of speech disorders usually resulting from neurological damage and is characterized by some degree to having weak, slow, uncoordinated or changed tone of speech muscles. It is stated in some researches that speech deficiency is one of the good indicators of PD. Therefore, Parkinson’s Disease Decision Support System makes use of the speech signal dataset to create a model that would be used for future classifications of PD subjects. Data preprocessing techniques and data reduction algorithms are implemented together with Support Vector Machines. The highest accuracy obtained through 10-fold cross validation is 98.24%. PD DSS proves to aid students and researchers in the field of PD.

Keywords: Parkinson’s disease, Support Vector Machine, data reduction technique, data preprocessing algorithms, decision support system

Table of Contents

List of Figures

CHAPTER I

INTRODUCTION

A. Background of the Study

Neurodegeneration is the process of impairment of brain cells [1]. Dopamine is a neurotransmitter that helps in brain regulation of movement and emotional responses [2]. When the dopamine-producing cells in the brain become damaged, and failed to produce enough dopamine, the motor symptoms of Parkinson's disease appear.

Parkinson's disease (PD) is a neurodegenerative brain disorder that is slowly progressive. This means that it may not be incapacitating for many years [3]. One of the main symptoms of PD is tremor, which may pertain to shaking or trembling. The body parts that may be affected include hands, arms, legs, or head. This may also bring about stiff muscles, slow movement, difficulty with balancing or walking and problems with speech. Other symptoms may include depression and other emotional changes [4]. At present, there is no proven fact why PD occurs to a patient but some researches would conclude that stress, dysfunctional cell processes and genetics are most likely the greatest factors why dopamine amount in the brain drops.

Though Parkinson's disease is not fatal by itself, the complications brought about by PD is rated by Center for Disease Control and Prevention to be the 14th leading cause of death in America [5] and is the second most common neurodegenerative disease [6]. These fatal complications include pneumonia, falling-related injuries, and choking. In local context, it is estimated that PD affects 120,000 people or one percent of the fifty-year old and above bracket [7]. There is no current cure for Parkinson's disease however; there are medications that are prescribed to help increase dopamine levels in the brain [8]. Early diagnosis of PD are important to help minimize dopamine loss and to slow the progression of neurodegeneration. By the time the disease is diagnosed due to motor symptoms, some sixty percent of nigrostriatal neurons are degenerated, and eighty percent of striatal dopamine is depleted. Thus, studies have been conducted to determine biomarkers that will aid in detecting early stage of PD [9].

Speech abnormalities has been acknowledged as one of the earliest indicators of PD[10]. It is suggested to be present even before the classical symptoms of the disease such as muscle rigidity and rest tremor appear [10,11]. Voice analysis has been used in various researches to aid in the detection of illnesses specifically the Parkinson's disease. Researchers used various speech signals to detect PD. This is through voice signal recording which is both easy and non-invasive [12].

The integration of electronic health systems to various medical departments has taken place over the last decade. [13] Additionally, the study on developing computer-aided diagnosis has also played its role especially in opt of making a more accurate diagnosis of a certain disease. Machine learning algorithms were originally designed in the hope of analyzing medical datasets [14]. In this study, various data pre processing and feature selection techniques will be used to diagnose Parkinson's disease. In line with this, the classifier that will be used in this study is Support Vector Machine.

In the work of Shahbakhi et.al. [15] on Speech Analysis for Diagnosis of Parkinson's Disease Using Genetic Algorithm and Support Vector Machine, SVM has been proven to be a good classifier for the specific dataset which garnered the highest accuracy of 94.50%. Other works that deal with SVM and Parkinson's include the work of Gil et. al. [16] entitled Diagnosing Parkinson by using Artificial Neural Networks and Support Vector Machines. In this study, SVM showed highest classification accuracy of 93.33% as compared to Multilayer Perceptron which obtained a 92.31% accuracy. Finally, Tsanas et. al. [17] researched on Random Forests and SVM for binary classification of PD subjects. The results show that SVM outperformed RF which garnered 93.5% accuracy compared to the 98.6% of SVM. All the aforementioned works made use of the Parkinson's Disease Dataset provided by the UCI repository of machine learning [18].

B. Statement of the Problem

In the clinical setting, PD is commonly missed or inaccurately diagnosed. Misdiagnosis appears to be greatly affected by the process of applying diagnostic criteria from clinical guidelines and the specialist conducting the diagnosis. Nearly half (47%) of PD diagnosis are incorrect when performed in the primary care setting. Moreover, there are a number of common indicators that usually causes misdiagnosis since these symptoms are not exclusive to PD. Among these are motor related symptoms. Speech, on the other hand, is not [19] making speech or voice signals more reliable than motor symptoms.

The error percentage brought about, whether due to misdiagnosis of the experts or lack of standard clinical procedure, needs to be addressed and supported by a system that would aid in PD diagnosis. Since the indicators that affect the quality of the diagnosis could not be totally eliminated, this study only aims to provide a decision support system that would at least lessen the common indicators that causes misdiagnosis of PD and serves as an addition to the existing PD diagnosis tools.

C. Objectives of the Study

This study aims to develop a diagnostic system that supports the classification of Parkinson's disease whether positive or not. Taking the speech signal dataset as input to the system, preprocessing and feature selection techniques are applied and processed through the classifier. Accuracy is identified based on the result of the classification.

In detail, the user who trains the system can execute the following:

1. The user can input the PD dataset in *.csv* format.
2. The user can select the data preprocessing techniques as well as if data reduction will be applied to the data.
3. The user can change the default parameters for each data preprocessing technique selected and SVM parameters.
4. The user can specify the directory where the model file will be saved as well as the filename of it.
5. The user can train the system and can view the training results pane. This pane contains the general information of the dataset, the preprocessing

information applied and the accuracy of the model.

Finally, the user who classifies with a test data can execute the following:

1. The user can input the model file in *.bin* format.
2. The user can input the test set file in *.csv* format.
3. The user can classify the data and view the classification results pane. This pane contains the general model information, the count of how many instances were correctly classified out of the total test count and the tabular representation of the PD subjects with the predicted class and the corresponding probability.

Both type of user can view the details of the system in the About option as well as the user guide from the Tutorial option.

D. Significance of the Study

Researchers, particularly students or trainees on the field of PD, is the main beneficiary of this study since the output serves as a tool for familiarization on PD and speech signals. Furthermore, since the tool allows for parameters experimentation, researchers are also taught on machine learning concepts.

Parkinson's disease is a field that proves to still have a lot of aspects to improve on. Through this study, PD diagnosis is put closer to the goal of having a more standardized process of PD classification. Moreover, the outcome of this study is beneficial to researchers since it provides extra aid in conducting researches in the field of PD.

Since the dataset being concentrated upon in this study is speech signals, this initiative also promotes early detection of PD. Error rate brought about by misdiagnosis has been taken into account in various studies and proves to be one of the weak points that must be dealt. By implementing a system that minimizes this error rate, this study aims to produce a reliable tool that would act as a classification support system. Finally, with the use of various machine learning algorithms, the study seeks to provide a faster means for diagnosis without compromising the quality and accuracy.

E. Scope and Limitations

The topic focuses on developing a decision support system for diagnosing Parkinson’s disease. The following are the scope and limitations of the proposed system:

1. The system can only classify instances as either PD positive or PD negative. PD type classification is not included in this study.
2. The user can choose which data preprocessing technique to be applied to the dataset. If no data preprocessing is selected, Min Max Normalization is the default data preprocessing method.
3. The user can choose just one feature reduction technique – PCA.
4. The system has a predetermined default parameters set but the user can still change the parameters if needed.

5. The training and testing set comes from the original dataset provided. The system uses 10-fold cross validation in computing for accuracy.
6. The input data is an extracted form already; that is – the extraction of features from raw voice signals in audio format is not be included in this study.

F. Assumptions

The following are assumed in the study:

1. The input dataset comprises the speech signal features in numerical format. The input is fed to the preprocessing algorithms and classifier that work with data in numerical format.
2. It is assumed that the dataset has no missing values.
3. The system is used by researchers or students that have basic knowledge of Parkinson's disease and machine learning systems.
4. The result of the system only serves as a decision support in diagnosis and must be verified by a specialist or expert.

CHAPTER II

REVIEW OF RELATED LITERATURE

One of the driving forces for machine learning to arise is the need for processing large medical datasets and analyzing its results. Today, machine learning proves to be indispensable in the field of intelligent data analysis. With relatively inexpensive and reliable means to collect and store data, the digital revolution grew to the point when modern hospitals are well equipped with tools that enable gathering and sharing of data in large information systems. Machine learning technology is suited for medical data analysis particularly in diagnosis. In basic tools, patient records with known diagnosis serves as an input to a program that runs a learning algorithm. In principle, the diagnosis can be concluded from previous cases of the same illness or disease. The classification, whatever the result may be, can then support the decision that will be made by the specialist, expert or physician when diagnosing a patient. This system can also serve as a learning tool for researchers about the specific fields [14].

An example of a machine learning tool is the work of Salvatore, et al. [20] on brain MRI for differential diagnosis of Parkinson's Disease and Progressive Supranuclear Palsy (PSP). In this study, supervised learning has been proposed to be an approach for gathering medical image biomarkers that would allow automated diagnosis of a subject. Principal Component Analysis together with Support Vector Machine were the algorithms used in extracting features and classifying MRIs from 28 PD and PSP patients. The study acquired an accuracy of >90% making supervised machine learning algorithm feasible and suitable for diagnosing PD and PSP. The algorithm provides discriminants for the PD and PSP subjects thus promoting the application of computer-based diagnosis in clinical practice.

Machine learning in speech data has been proven to aid the early detection of PD in the work of Hazan, et al. [9]. In this research, vowel formants were used in diagnosis taking two different datasets. These datasets are gathered in USA and Germany from both healthy controls and patients with mild or in early stage of PD. At first, the researchers hypothesized that the optimal features are language dependent but were proven wrong by the findings that such features are dataset dependent. Having stated that, speech signals and acoustic metrics data can be combined with other datasets of the same category. The development of a diagnostic tool that is not region dependent is then supported by this study. Also, since only vowel formants have been used as input, it has been recommended that other acoustic metrics such as vocal intensity, vocal decay, speech prosody and voice quality be experimented on in future related works.

In other fields of medicine like in the work of Shashikant et. al.,[21] machine learning algorithms were also applied to build a decision support system for heart disease. Taking the heart disease dataset as input, the study produced tool that aims to help doctors give diagnosis without seeking consultation from the specialists. The algorithms that were investigated upon are Support Vector Machine (SVM) and Radial Basis Function (RBF) Network SVM with sequential minimal optimization algorithm is applied to the India based patient’s dataset. The same data was applied to the RBF Network trained by Orthogonal Least Square algorithm. It was proven at the latter that SVM obtained successful results.

Face recognition has also been a field where SVM proved its success. Le et. al., [22] compared the performance of three classification algorithm that were applied to AT&T and FERET face database. The algorithms include SVM, Multilayer Perceptron and K-Nearest Neighbor. The feature vectors were extracted using PCA and 2DPCA and were fed to the classifiers. SVM combined with 2DPCA garnered the highest accuracy of 95.1% which outperformed the traditional classification algorithms significantly.

The combination of PCA and SVM has been popular not only in medical field but also in many other various applications such as surveillance and weapon detection. The study of Poh et. Al. [23] deals with imaging surveillance and focused on determining the algorithm combination that produces the most accurate prediction with low computational cost. The study took advantage of PCA's capability to explore features and reduce multidimensionality of the data to a simplified form without compensating results. Like other researches, a number of algorithms has been investigated and compared. Among these methods are Neural Network, Decision Trees, Naïve Bayes and K-Nearest Neighbor. The results show that SVM improved the accuracy and worked well with PCA.

PCA and SVM algorithm combination has also been theoretically applied to Parkinson's dataset by Shahbakhti et. al. [12] using MATLAB software and served as a basis for this study. This time, PCA transformed the dataset to optimized features. The first 3 optimized features were used in classifying with SVM. The performance of the algorithm was then measured by using 3 statistical measures; (1) specificity, (2) sensitivity and (3) total classification accuracy. A 91.5% accuracy was achieved through cross validation.

The success of an algorithm in machine learning and statistical classification is not solely dependent on the classifier used. In most cases, several data preprocessing techniques also aid in producing better accuracy.

In the work of Shawkat, et al., normalization for Support Vector Machines has been considered. The normalization has been applied to 112 classification problems with SVM as the classifier. A significant increase in the classification due to normalization has been observed [24].

The problem of imbalanced datasets can also be addressed through data preprocessing. A dataset is imbalanced if the classification categories are not approximately equally represented. Synthetic Minority Over-sampling Technique (SMOTE) is an over-sampling approach wherein the minority class is over-sampled by creating “synthetic” examples rather than by over-sampling with replacement [25]. Han, et al. research about modified SMOTE as a new over-sampling method proves to solve imbalanced dataset problems [26].

SMOTE has also been combined with SVM in the hope of solving SVM's limitation when learning form imbalanced dataset. The work of Akbani et. al. discussed the factors for the this limitation and explains why the common solution of undersampling the dataset is not be the best solution [27].

Classification through noisy data shows great performance deterioration of conventional machine learning techniques. Eliminating noise got the attention of some researchers like Natarajan et. al. who worked with theoretically studying the effect of noise in binary classification. The classifier obtains class labels that have been randomly flipped resulting to the noisy labels [28].

Due to noise in real world datasets and its effect on classification algorithms, detecting and eliminating noise have come to be crucial in data preprocessing. It was also mentioned by Quilan (1986), that when there is a high level of noise in classes, cleaning the training data will improve the accuracy of the prediction made by the classifier. Daza et. al. [29] proposed an algorithm for detecting noise on supervised classification. The algorithm is called QCleanNoise with the sole responsibility of detecting and eliminating noisy instances through computing the quality measure of the instance. Three classifiers were used in validating the effectiveness of the algorithm. The accuracy of the prediction produced by LDA, KNN and RPART has been measured. As a result, QCleanNoise proved to have outperformed other proposed noise eliminating algorithms. Furthermore, the misclassification of the classifiers reduced significantly.

There are several decision support systems to date that sought to provide diagnosis for PD patients. One of which is Hadjahmadi’s et. al. [30] work on DSS using classification and regression tree. The study used the Oxford Parkinson's Disease Detection Dataset provided by the UCI repository of machine learning databases to train and test the system. Sensitivity analysis was also conducted to determine the importance of the input variables.

The Oxford Parkinson's Disease Detection Dataset was also applied to soft computing in attempt to develop an intelligent DSS for PD. Bhande et. al. [31] implemented Multilayer Perceptron Neural Network in the said application.

While this study focuses on speech signals, other DSS for PD made use of imaging methods to diagnose the disease. Ericsson et. al. [32] made use of Single Photon Emission Computerized Tomography (SPECT) images that were loaded to the system for feature extraction. The features were classified using Support Vector Machines and gave a very promising accuracy.

CHAPTER III

THEORETICAL FRAMEWORK

A. Parkinson's Disease [33]

PD is classified as a type of movement disorder caused by loss of dopamine producing brain cells. Dopamine is a brain chemical that acts as a messenger between two brain areas - the substantia nigra and the corpus striatum. This substance allows for smooth and controlled movements. When dopamine amount in the brain drops, body movements become impaired and some brain cells are affected leading to degeneration of some non-movement related symptoms of PD. At present, there is no clear reason why the dopamine-producing cells deteriorate but based from researches done, stress, inflammation and dysfunctional cell processes may also contribute. Therefore, scientists concur that dopamine loss is a combination of genetics and environmental factors.

PD is the second most common neurodegenerative disorder and the most common movement disorder. Some of the symptoms that PD subjects display are as follows [34]:

- Trembling of hands, arms, legs, jaw and face
- Stiffness of the arms, legs and trunk
- Slowness of movement
- Poor balance and coordination

Generally, PD is characterized as loss of muscle control and as symptoms worsen basic actions like walking and talking may also become difficult. Furthermore, PD is a chronic and progressive disease, meaning that it persists over a long period of time and it grows worse over time.

Currently, there is no known cure for PD but some medications such as levodopa combined with carbidopa provide temporary relief for the symptoms. Levodopa acts as a substitute for dopamine that is converted in the brain. Carbidopa, on the other hand, delays the conversion of this substance until it reaches the brain [35].

B. Speech Deficiency in PD [36]

The basal ganglia, also affected by dopamine loss in the brain, are believed to be responsible for the direction, speed and force of voluntary movements. These movements include speech.

Sixty to eighty percent of PD subjects have been estimated to develop speech impairment as the disease progress. Deficiencies in communication often start with reduced vocal loudness, fast or fluctuating speech rate, short rushes of speech. This is later followed by changes in speech rate, imprecise consonant articulation or pronunciation and intelligibility. Dysarthria is also exhibited among PD individuals. This refers to a group of speech disorders usually resulting from neurological damage and is characterized by some degree to having weak, slow, uncoordinated or changed tone of speech muscles. PD patients also display signs of jumbled speech sounds and stuttering.

Stutter-like symptoms as well as vocal loudness degradation are addressed in speech therapy for PD patients and results in significant improvements in communication functions of the individuals.

C. Decision Support System

A Decision Support System (DSS) is a computer-based application that aims to help decision makers in giving analysis, diagnosis or findings to various fields and industries [37]. DSS in medical field, sometimes referred to as Clinical Decision Support Systems (CDSS), provide clinicians and medical staffs information that enhances quality of health care and diagnosis. Application of information systems in medical field has long been recognized but it must always be taken into consideration that these applications are merely a means to support and improve health care quality and not totally replace the diagnosis given by experts [38].

D. Machine Learning

Machine Learning (ML) is a type of artificial intelligence that provides applications the ability to learn without being explicitly programmed [39]. It aims at providing computational methods for changing and updating body of knowledge in intelligent systems. ML is beneficial in cases where given concrete algorithmic solutions are not suitable, there is lack of formal models, or the knowledge about the application domain is poorly defined. It is proven that ML methods and techniques provide solutions to diagnostic and prognostic problems in various medical domains. It also supports analysis in the significance of clinical parameters combinations for prognosis [40].

For a machine learning system to be efficient in providing diagnosis, the following features are desired [14]:

a. good overall performance - fast, efficient, accurate,

b. the ability to appropriately deal with missing and/or noisy data,

c. the ability to provide transparency in diagnostic knowledge,

d. the ability to explain decisions,

e. and the ability of the algorithm to reduce the number of tests necessary to obtain reliable diagnosis.

The basic flow of supervised ML process [41] is given by Fig 1.

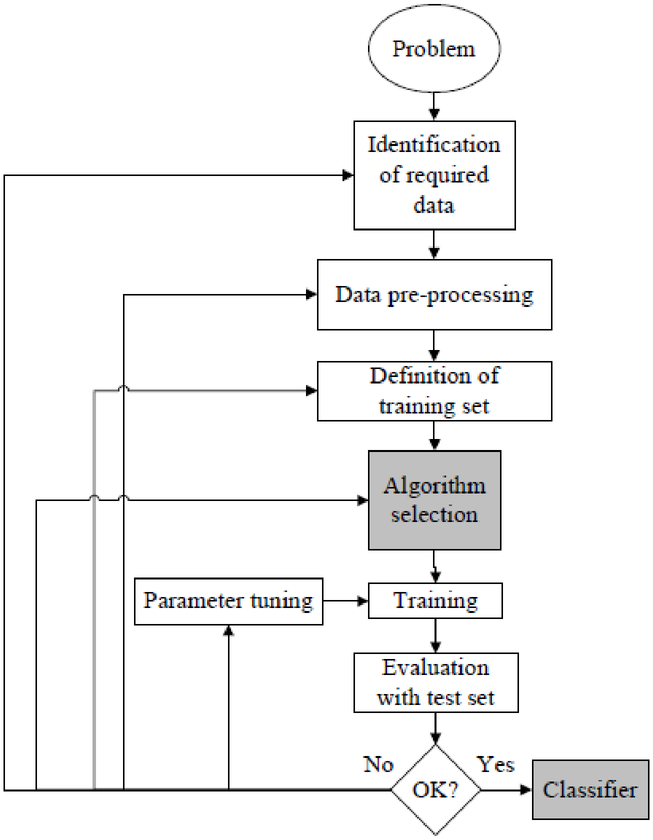


Figure 1. Process of supervised machine learning

Supervised machine learning is the type of ML that takes a known set of instances with known classifications and outputs a predictor that will be used for future classification of new instances [42]. This is explained better using the following guide

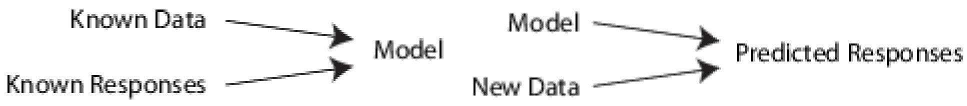


Figure 2. Supervised ML concept

Supervised ML is subdivided into two categories. The first is *classification* and is primarily for responses that can have a few know values such as “true” or “false”. *Regression*, on the other hand, is for real number responses.

E. Data Preprocessing

Any type of processing technique performed on raw data to prepare it for another processing procedure sums up what data preprocessing is about. This is commonly done in early stages of data mining and aims to optimize the data before it is input to the classifier. Data preprocessing has different forms which seeks to address problems related to data collected in real world [43]. Among these problems are: (i) data with missing, out of range or corrupt elements, (ii) noisy data, (iii) data from several levels of granularity, (iv) large data sets, data dependency, and irrelevant data, and (v) multiple sources of data [44].

The following are the major tasks in data preprocessing

a. *data cleaning* - deals with missing and noisy data and corrects inconsistencies

b. *data integration* - deals with redundancies from integrated data through correlation analysis

c. *data transformation* - deals with transforming data to a form suitable for data mining

d. *data reduction* - deals with reducing dataset which is smaller in volume but will not compensate results

F. QcleanNoise

Daza et. al. [29] proposed an algorithm called QcleanNoise that aims to detect and eliminate noisy instances from a given dataset. Class noise as defined by Zhu et. al., pertains to instances that are badly located in the cloud of points defining each of the classes as illustrated in Fig. 3.

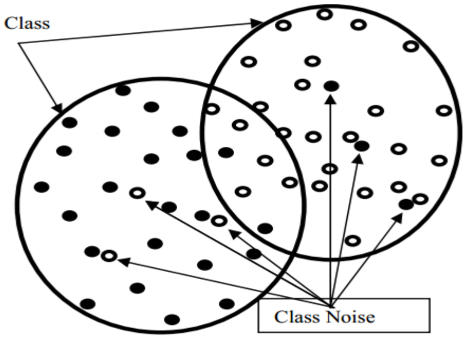


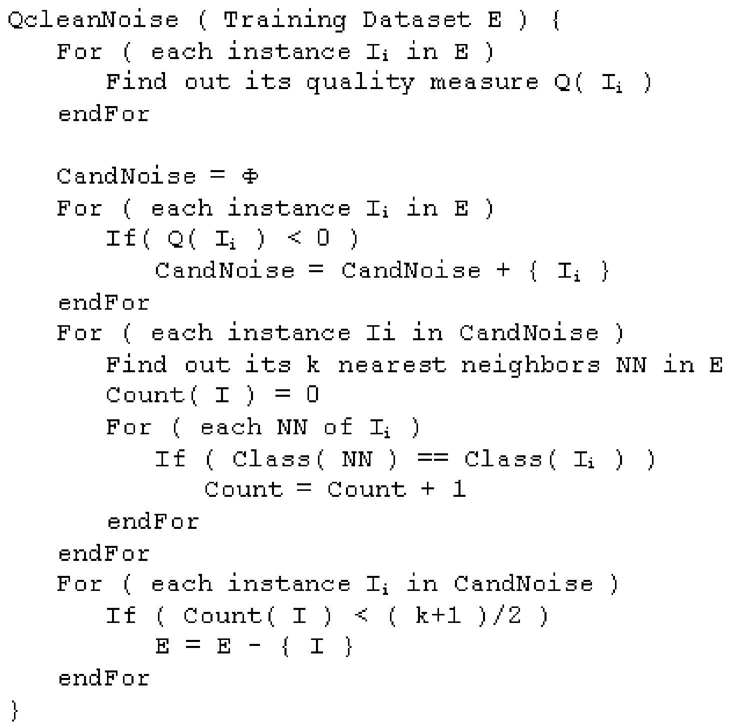
Figure 3. Graphical representation of class noise

A measure to evaluate the quality of an instance denoted by Q_i is introduced.

$$Q_i = {r_i - d_i} \big/ {max(d_i, r_i)}$$

Where d_i is the distance of the i-th instance to the centroid of its class and r_i is the minimum distance of the i-th instance to the centroid of the classes where it does not belong to. A noisy instance will have negative values for the quality measure *Q*. However, small negative quality values can also be computed for instances near the boundaries of two or more classes. This ambiguity is also addressed by the QcleanNoise algorithm. The goal is to identify and delete the noisy instances and preserving the class distribution and boundaries such that neither separability of the classes nor the discriminant power of the classification algorithm is altered.

The algorithm for detecting and removing noise from the data as given in [29] is shown below.



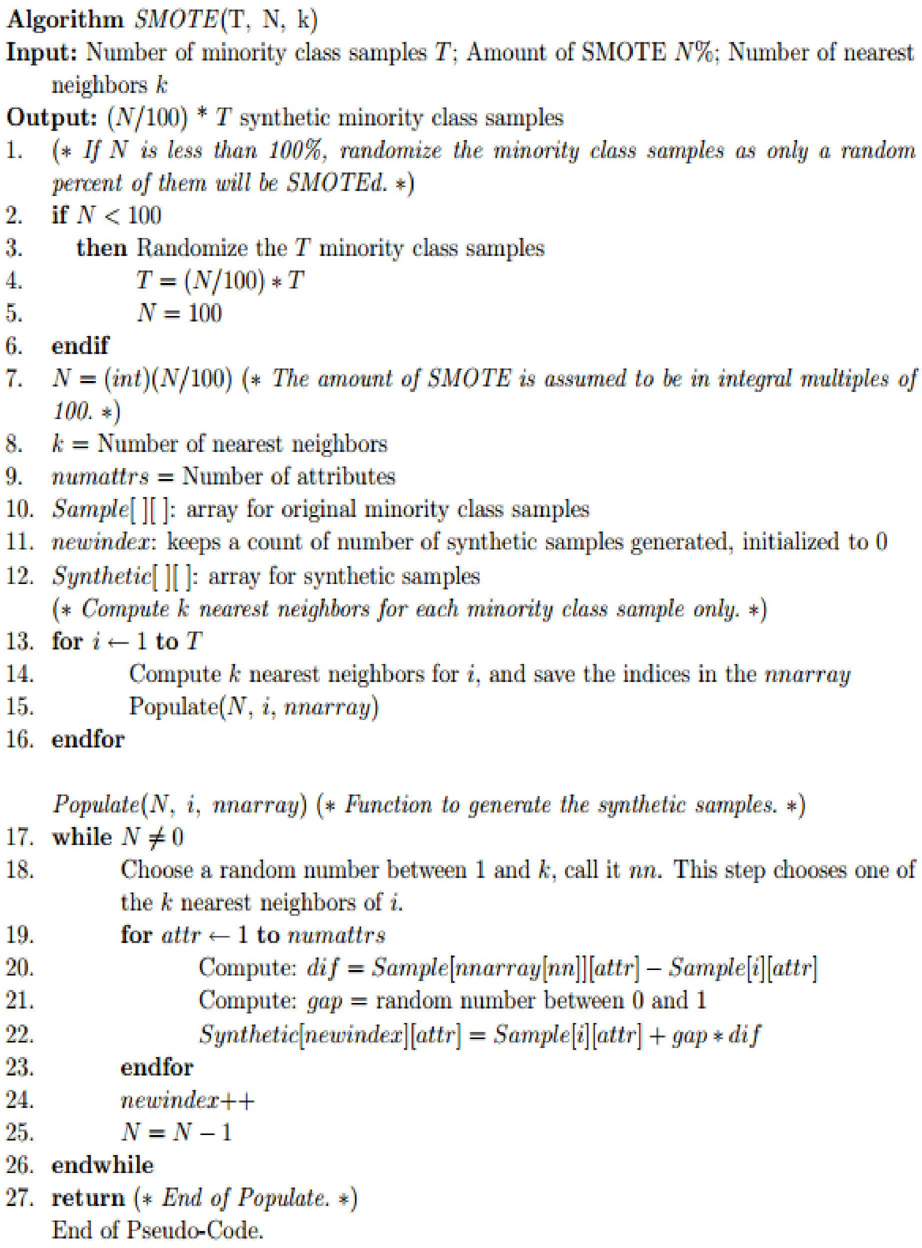
```
QcleanNoise ( Training Dataset E ) {
    For ( each instance $I_i$ in E )
        Find out its quality measure Q( $I_i$ )
    endFor

    CandNoise = $\Phi$
    For ( each instance $I_i$ in E )
        If( Q( $I_i$ ) < 0 )
            CandNoise = CandNoise + { $I_i$ }
    endFor
    For ( each instance Ii in CandNoise )
        Find out its k nearest neighbors NN in E
        Count( I ) = 0
        For ( each NN of $I_i$ )
            If ( Class( NN ) == Class( $I_i$ ) )
                Count = Count + 1
        endFor
    endFor
    For ( each instance $I_i$ in CandNoise )
        If ( Count( I ) < ( k+1 )/2 )
            E = E - { I }
    endFor
}
```

Algorithm 1. QCleanNoise

G. Synthetic Minority Over-sampling Technique

A dataset is imbalanced if the classes are not approximately equally represented. The performance of machine learning algorithm is evaluated using predictive accuracy and is affected negatively when the dataset is imbalanced.

Synthetic Minority Over-sampling Technique (SMOTE) [25] is an over-sampling approach that aims to solve problems with imbalanced dataset by generating “synthetic” examples rather than over-sampling with replacement. The generation of synthetic examples is operated in feature space of the minority class. This is done by taking each minority class sample and introducing the synthetic examples along the line segment joining any or the entire *k* minority nearest neighbors. Neighbors are chosen randomly from the *k* nearest neighbors depending on the amount of over-sampling required. Algorithm 2 shows the pseudocode for SMOTE algorithm.



```
Algorithm SMOTE(T, N, k)
Input: Number of minority class samples $T$; Amount of SMOTE $N\%$; Number of nearest
    neighbors $k$
Output: $(N/100) * T$ synthetic minority class samples
(∗ If $N$ is less than 100%, randomize the minority class samples as only a random
percent of them will be SMOTEd. ∗)
if $N < 100$
  then Randomize the $T$ minority class samples
       $T = (N/100) * T$
       $N = 100$
endif
$N = (int)(N/100)$ (∗ The amount of SMOTE is assumed to be in integral multiples of
100. ∗)
$k$ = Number of nearest neighbors
$numattrs$ = Number of attributes
$Sample[\,][\,]$: array for original minority class samples
$newindex$: keeps a count of number of synthetic samples generated, initialized to 0
$Synthetic[\,][\,]$: array for synthetic samples
(∗ Compute k nearest neighbors for each minority class sample only. ∗)
for $i \leftarrow 1$ to $T$
      Compute $k$ nearest neighbors for $i$, and save the indices in the $nnarray$
      Populate($N$, $i$, $nnarray$)
endfor

Populate($N$, $i$, $nnarray$) (∗ Function to generate the synthetic samples. ∗)
while $N \neq 0$
      Choose a random number between 1 and $k$, call it $nn$. This step chooses one of
      the $k$ nearest neighbors of $i$.
      for $attr \leftarrow 1$ to $numattrs$
            Compute: $dif = Sample[nnarray[nn]][attr] - Sample[i][attr]$
            Compute: $gap$ = random number between 0 and 1
            $Synthetic[newindex][attr] = Sample[i][attr] + gap * dif$
      endfor
      $newindex{+}{+}$
      $N = N - 1$
endwhile
return (∗ End of Populate. ∗)
End of Pseudo-Code.
```

Algorithm 2. SMOTE

H. Min Max Normalization

Min-max normalization is a data transformation technique that aims to scale data to a certain range *[Y_{new_min}, Y_{new_max}]*. Due to this bounded range, the data will now have smaller standard deviation and suppress the effect of outliers [45]. Suppose that Y_{min} and Y_{max} are the minimum and maximum values of an attribute. Min-max normalization maps a value X_i in Y to the transformed value X'_i in the range by computing the following equation given in [46]

$$X'_i = (\frac{X_i - Y_{min}}{Y_{max} - Y_{min}}) * (Y_{new\_max} - Y_{new\_min}) + Y_{new\_min}$$

Min-max normalization preserves the relationships among the original dataset.

I. Z-Score Normalization

Another data transformation technique is the Z-score normalization or zero-mean normalization [46]. The mean and standard deviation are the key parameters for normalizing X_i values in an attribute Y and is given by the following equation taking the transformed value as X'_i

$$X'_i = \frac{X_i - \overline{Y}}{\sigma_X}$$

Where $\bar{Y}$ and σ are the mean and standard deviation of attribute Y, respectively. This method is useful when the actual minimum and maximum values in Y are unknown. Also, it would suppress outliers that dominate after min-max normalization.

J. Principal Components Analysis

In data analysis, it is often necessary to reduce dimensionality of data to speed up processing of information or to make applications more efficient by making computational cost low [47].

Principal Components Analysis or PCA is a data reduction technique that aims to reduce the dimensionality of a dataset which consists of possibly interrelated variables without losing much of the variation present in the original dataset [12,48]. Each of the uncorrelated components produced by the algorithm is a linear combination of the original variables. These uncorrelated components are called principal components (PC) and are estimated from the eigenvalues and eigenvectors of the correlation or covariance matrix of the original dataset. Eigenvalue is the indicator of how much variation is explained by each PC. The first PC will have the largest eigenvalue and smaller for the succeeding PCs. Eigenvectors, on the other hand, are the linear combination of the centered original variables. These will act as the weights for arriving at the uncorrelated PCs [12]. For this study, covariance matrix will be used as stated in the algorithm in [49].

The following describes the steps in reducing the original dataset to the principal components [49].

(a) Propose X as the data with i number of attributes and j number of instances

(b) Calculate the mean value of Xi and subtract this from each of the j instances, let this be the data adjusted

(c) Calculate the covariance matrix C

$$C^{n\times n} = (c_{i,j},\ c_{i,j} = cov(Dim_i, Dim_j)),$$

where $$cov(X,Y) = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{(n-1)}$$

(d) Calculate the eigenvectors and eigenvalues of the covariance matrix

(e) Sort the eigenvalues from highest to lowest

(f) Form a feature vector from the eigenvectors

$$FeatureVector = (eig_1\ eig_2\ eig_3 \ldots.\ eig_n)$$

(g) Finally, derive the new dataset

$$FinalData = RowFeatureVector \times RowDataAdjust,$$

K. Support Vector Machine

A Support Vector Machine is a discriminative classifier used for classification and regression. The algorithm is based on the statistical learning theory and the Vapnik-Chervonenkis (VC) dimension introduced by Vladimir Vapnik and Alexey Chervonenkis [50-51]. SVM is formally defined by a separating hyperplane. It takes a labeled training data as input and outputs an optimal hyperplane that is used for categorizing new samples.

The goal of the SVM algorithm is to find the hyperplane that separates the classes and gives the largest minimum distance to the training examples. The optimal separating hyperplane maximizes the margin of the training data thus the term Maximum Margin Classifier for SVM [51-52]. Fig. 4a shows a linearly separable set of 2-D Points, each comprising a class, with separating lines while Fig. 4b shows the optimal hyperplane as well as the maximum margin. Additionally, Fig. 4a shows the *bad* lines that separate the classes. A line is bad if it passes too close to the points because it will be noise sensitive [52]. As stated, the goal of SVM is to find the line passing as far as possible from all points as shown in Fig. 4b.

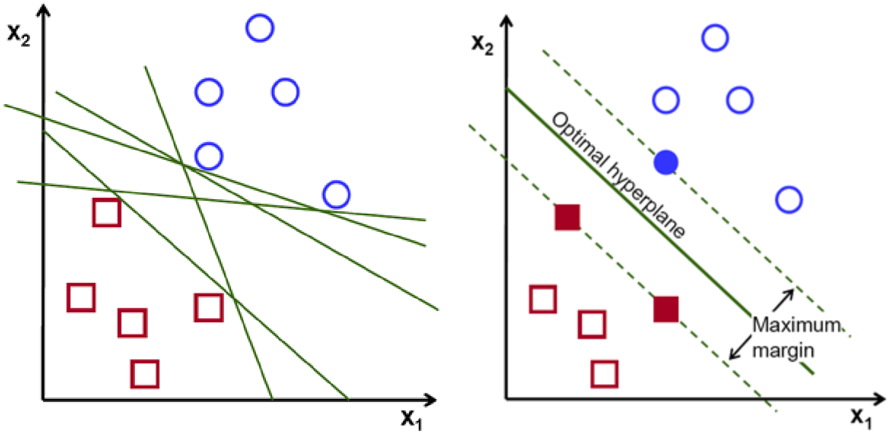


Figure 4. (a) Linearly separable training points; (b) optimal hyperplane

Mathematically, the separating hyperplane is given by [53]

$$w \cdot x + b = 0$$

Where w is normal to the hyperplane and $\frac{b}{\|\mathbf{w}\|}$ is the perpendicular distance from the hyperplane to the origin shown in Fig. 5. The examples closest to the separating hyperplane are called *support vectors*.

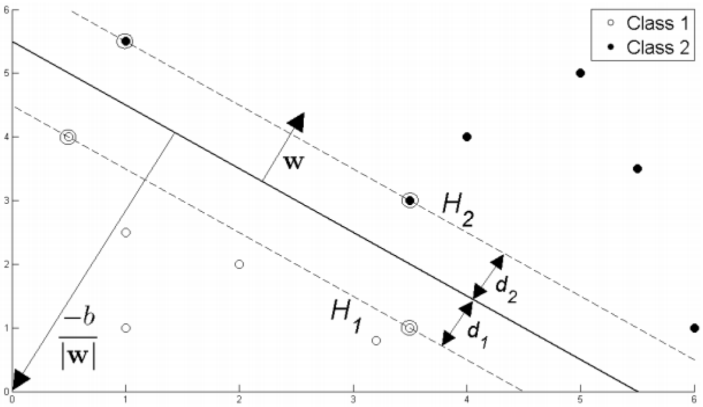


Figure 5. Hyperplane through two linearly separable classes

Since the variable of interest is the maximum margin, parallel hyperplanes must be constructed and is given by [52]

$$\begin{aligned} &w.x + b = 1 \\ &w.x + b = -1 \end{aligned}$$

Given a linearly separable training data, these hyperplanes can be selected such that there are no points between them and the distance from the first hyperplane to the second must be maximized. The distance is given by and must be $\frac{2}{|w|}$ minimized $|w|$ thus

$$y_i\ (\ w.\ x_i - b) \geq 1 \quad ,\ 1 \leq i \leq n$$

To orient the hyperplane to be as far from the support vectors, SVM margin must be maximized by minimizing ||w|| and is done using the following equation.

$$L_p = \frac{1}{2}\,||w\,||^2 - \sum_{i=1}^{L} \alpha_i[y_i(x_i \cdot w + b) - 1]$$

The final form of the hyperplane that will be used in predicting the classification of new samples is called the decision and is given by

$$f(x_{new}) = sign(\sum_{i=1}^{n} \alpha_i y_i \Box x_i, x_{new} \Box + b^*)$$

Kernel Selection of SVM [52]

The process of mapping the training vectors to a higher dimensional space is performed by the function Φ or the kernel function. There are a number of kernel functions in SVM and it will take research to determine which is the best function to take. However, Srivastava et. al. named a few popular functions as follows:

(a) Linear Kernel

$$K\ (x_i\ ,\ x_j) = x_i^T\ x_j.$$

(b) Polynomial Kernel

$$K\ (x_i\ ,\ x_j) = (\gamma\ x_i^T\ x_j + r)^d \quad , \qquad \gamma \underline{>} 0$$

(c) Radial Basis Function (RBF) Kernel

$$K\ (x_i\ ,\ x_j) = \exp(-\gamma\ \|x_i - x_j\|^2)\ ,\quad \gamma > 0$$

(d) Sigmoid Kernel

$$K\ (x_i\ ,\ x_j) = \tanh(\gamma\ x_i^T\ x_j + r)$$

Where γ, r and d are kernel parameters. Among these kernels, RBF is commonly used since (1) it nonlinearly maps samples into a higher dimensional space, (2) it has less hyperparameters compared to the other kernel functions and (3) it has less numerical difficulties.

Recent studies have proven that SVM, in general, is capable of performing better than other classification algorithms in terms of accuracy. However, the performance of SVM is very sensitive in some datasets. This varies depending on the cost and kernel parameters. Therefore, a careful study is needed to find the optimal parameters [52].

LIBSVM [54] is an integrated software for support vector classification and supports multi-class classification. The methods provided in this library will be used in training the system and predicting the class. LIBSVM is an open source machine learning library.

CHAPTER IV

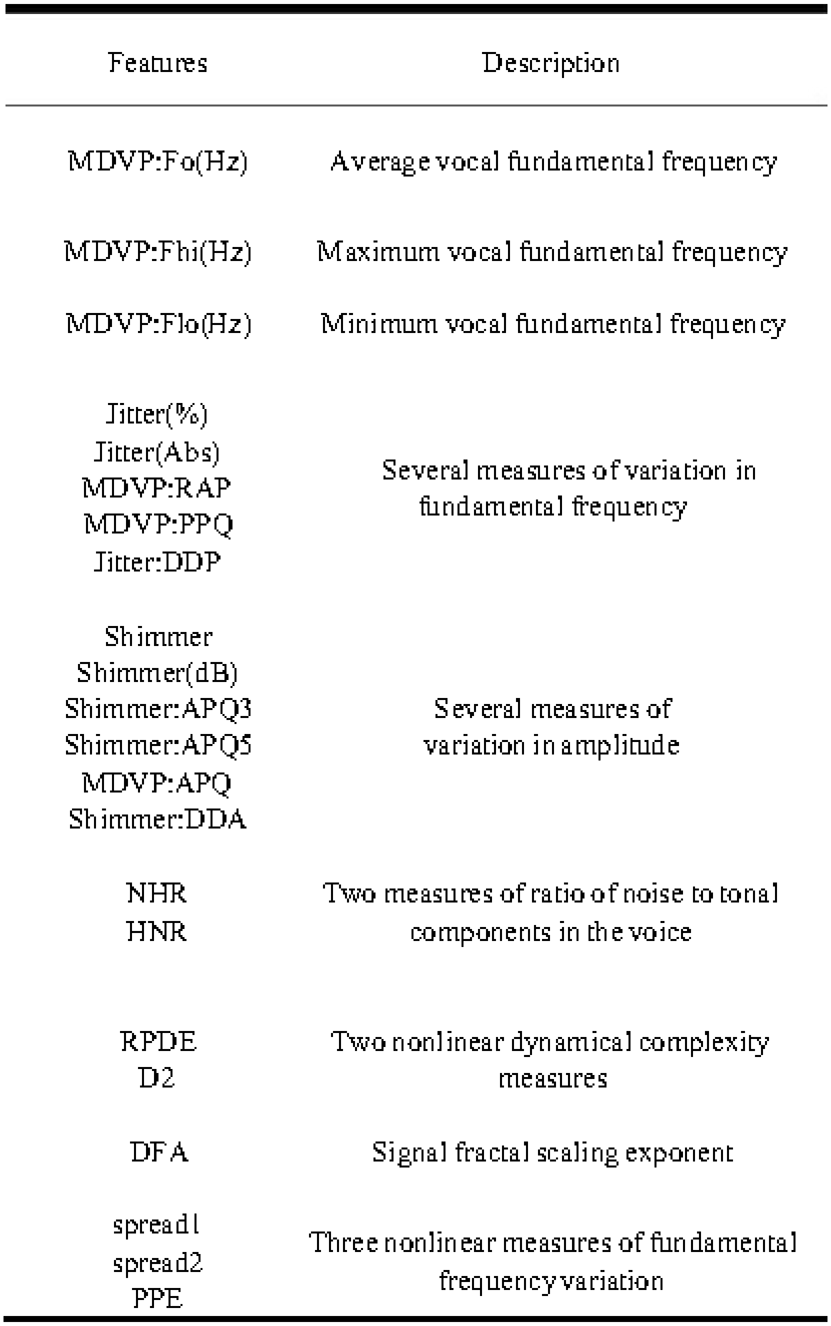
DESIGN AND IMPLEMENTATION

A. Parkinson’s Disease Dataset

Oxford Parkinson's Disease Detection Dataset [55] was created by Max Little of the University of Oxford and the National Centre for Voice and Speech in Denver, Colorado who recorded the voice signals.

A total of 31 people participated in the study, 23 of which are diagnosed to have Parkinson’s disease. The dataset is composed of biomedical measurements from 195 instances of voice recordings. 147 of which are PD positive and 48 are PD negative. The status column is a nominal attribute which is set to ‘0’ if the patient is healthy and ‘1’ otherwise.

The following figure summarizes the features that is used in this study.



Features	Description
MDVP:Fo(Hz)	Average vocal fundamental frequency
MDVP:Fhi(Hz)	Maximum vocal fundamental frequency
MDVP:Flo(Hz)	Minimum vocal fundamental frequency
Jitter(%) Jitter(Abs) MDVP:RAP MDVP:PPQ Jitter:DDP	Several measures of variation in fundamental frequency
Shimmer Shimmer(dB) Shimmer:APQ3 Shimmer:APQ5 MDVP:APQ Shimmer:DDA	Several measures of variation in amplitude
NHR HNR	Two measures of ratio of noise to tonal components in the voice
RPDE D2	Two nonlinear dynamical complexity measures
DFA	Signal fractal scaling exponent
spread1 spread2 PPE	Three nonlinear measures of fundamental frequency variation

Figure 6. Features and feature description [12]

B. System

The proposed system classifies two classes, PD+ or PD-. It is implemented using Java.

Basically, in training, the system must take the Parkinson's Disease dataset and the system allows the user to select data preprocessing and input the parameters for each. Preprocessing and data reduction techniques are implemented and SVM is used as the classifier. Also, the user can specify where to save the output model as well as its filename. Finally, performance are evaluated using 10-fold cross validation. The training results includes the dataset information, the preprocessing methods used and the accuracy. A user guide is also provided by the system and can be accessed by both the trainer and the tester.

For classifying, the user must input the model and the test set that comprise the instances to be classified. Finally, the classification results contain the model information, the accuracy and the tabular presentation of the instances, its actual and predicted class and the corresponding probability which the model obtained. Fig. 7 shows the context-diagram of the system.

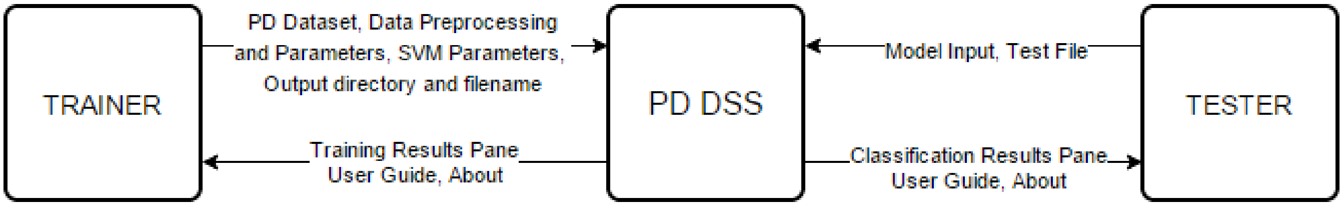
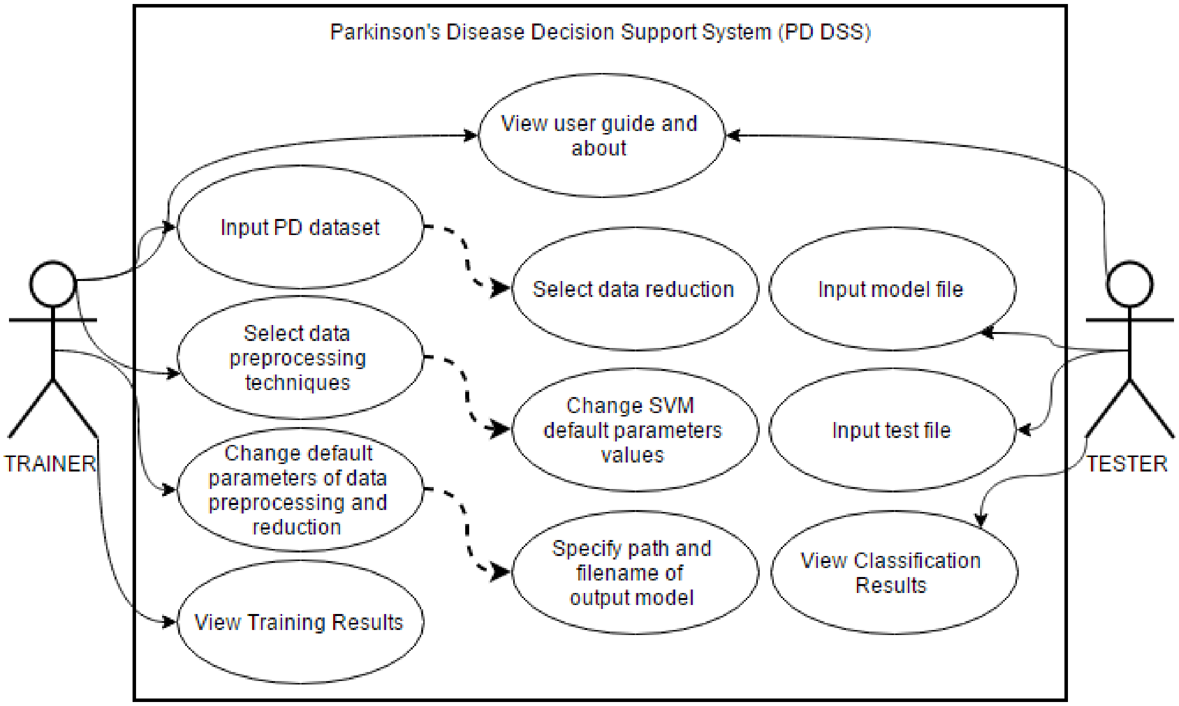

TRAINER
PD Dataset, Data Preprocessing and Parameters, SVM Parameters, Output directory and filename
Training Results Pane User Guide, About
PD DSS
Model Input, Test File
Classification Results Pane User Guide, About
TESTER

Figure 7. Context diagram, PD DSS

The specific functionalities of the users are given in the use-case diagram shown in Fig. 8. As illustrated, the trainer can input the PD dataset to the system, choose the preprocessing techniques that are applied to the data, choose the data reduction PCA if needed, change the default parameters for the data preprocessing and reduction, train the classifier which in this case is SVM and finally, view the training results. On the other hand, the tester can input the model and test file then view the classification results. Both users can view the user guide and about windows.



Parkinson's Disease Decision Support System (PD DSS)
View user guide and about
Input PD dataset
Select data reduction
Input model file
Select data preprocessing techniques
Change SVM default parameters values
Input test file
Change default parameters of data preprocessing and reduction
Specify path and filename of output model
View Classification Results
View Training Results
TRAINER
TESTER

Figure 8. Use case diagram, PD DSS

Fig. 9 illustrates the top-level data flow diagram of the system. As shown, there are 5 main processes namely (1) Data Input, (2) Data Preprocessing and Reduction, (3) Contructing Classification Model, (4) Classification Prediction and (5) Result Generation. Each process are illustrated in the succeeding visualizations.

In the first process, the user can input a comma-separated file (.*csv*) to the system. This file is parsed and stored to a matrix. The user can also choose which preprocessing technique to use. The system can implement four preprocessing techniques namely, (1) SMOTE, (2) QCleanNoise, (3) Min-Max Normalization and (4) Z-Score Normalization. If no preprocessing technique is selected, Min-Max Normalization is the default. The user can also input data reduction parameters that is used for PCA. The data matrix is then separated to two sets, training and testing. The testing sample comprises 15 randomly selected test subjects and the remaining is for training. The samples are selected at random and are not replaced. Fig. 10 illustrates process 1.

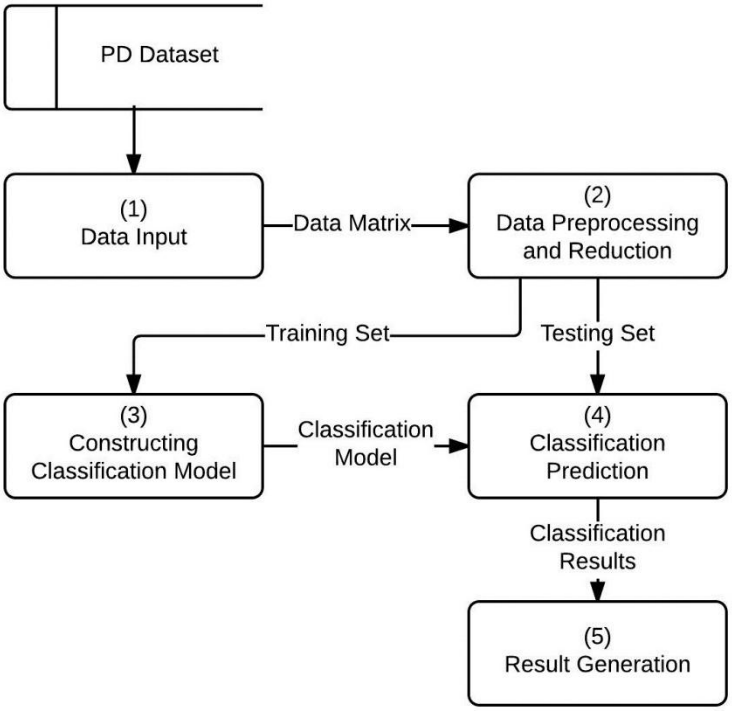


Figure 9. Top-level data flow diagram

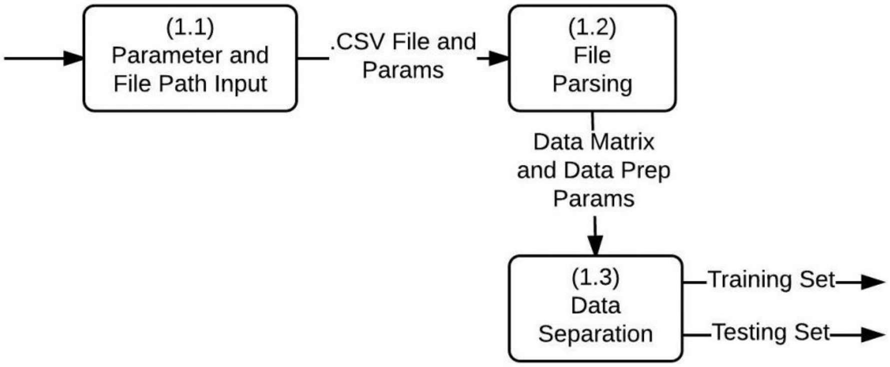


Figure 10. Sub-explosion of process 1

After data input, the training and testing sets undergoes data preprocessing and reduction better illustrated in Fig. 11. The training data undergoes the SMOTE and QCleanNoise algorithm first. The reason why the testing data is not part of this step is because of the instance addition and reduction these algorithms do. Since the object of interest for the testing data are the samples selected at random, additional instances should not be introduced. The next algorithm is Min-Max Normalization and Z-Score Normalization. Both training and testing set can undergo these techniques. Data preprocessing is done to remove outliers and noise and also to prepare data for dimension reduction.

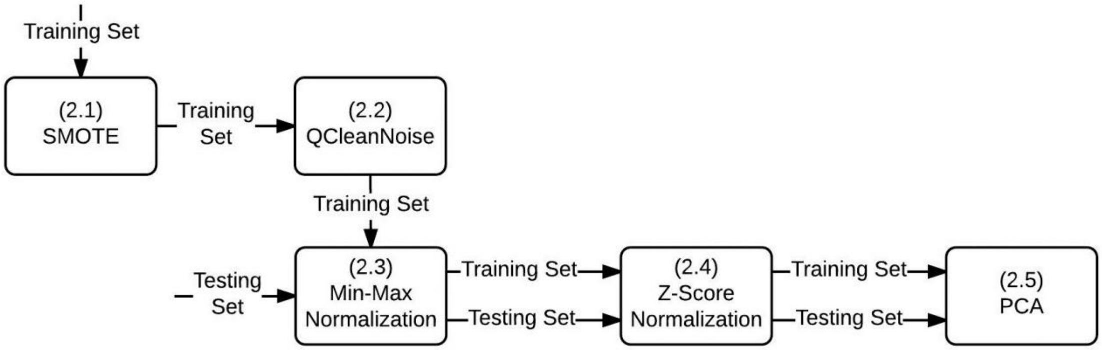


Figure 11. Sub-explosion of process 2

PCA is the dimension reduction technique that is used by the proposed system shown in Fig. 12. The process for applying PCA in training set reduces the dimension of the data. The mean per attribute calculated, as well as the eigenvectors produced is stored in a container and is used for transforming the testing set. From the user input given in process 1, the number of optimized features N is used in transforming the instances to N number of features.

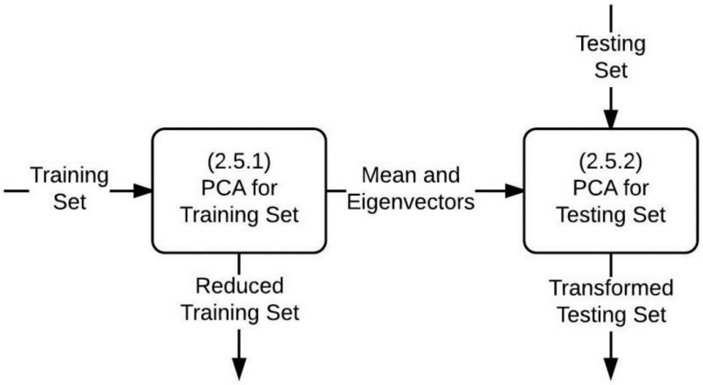


Figure 12. Sub-explosion of process 2.5

The next process involves training the system and creating the classification model that is used for the classification prediction stage. The input for this step is the reduced dataset from the previous process. The system builds a model based from the training set. This model, together with the testing set is then used for the next process which is classification prediction. The system predicts the class of each instance in the testing set. The algorithm to be used in this phase is SVM and is implemented using LIBSVM. Fig. 13 shows the general flow of this process.

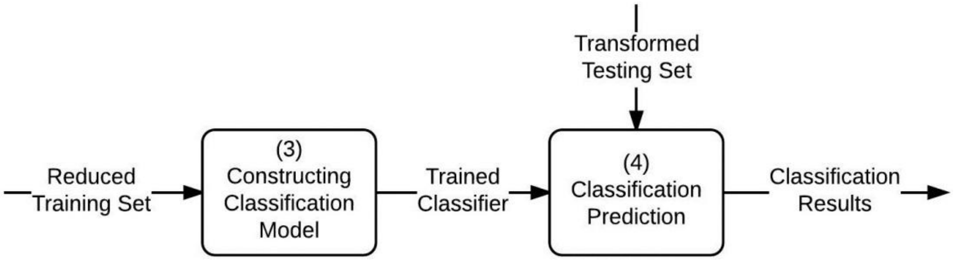


Figure 13. Process 3 and 4

Finally, the results of the classification is stored and presented in the last phase. The results stage outputs the result of the classification and performs performance evaluation. Accuracy is calculated through 10-fold cross validation and the classification results are presented.

The user also has the option to view the user manual or tutorial which includes basic information on how to navigate through the application as well as a short description of how to interpret results.

C. Technical Architecture

The proposed system is written in Java. Java Runtime Environment is a prerequisite for running the system. The system is built using the following tools:

a. Eclipse Kepler IDE

b. Java Runtime Environment (JRE7)

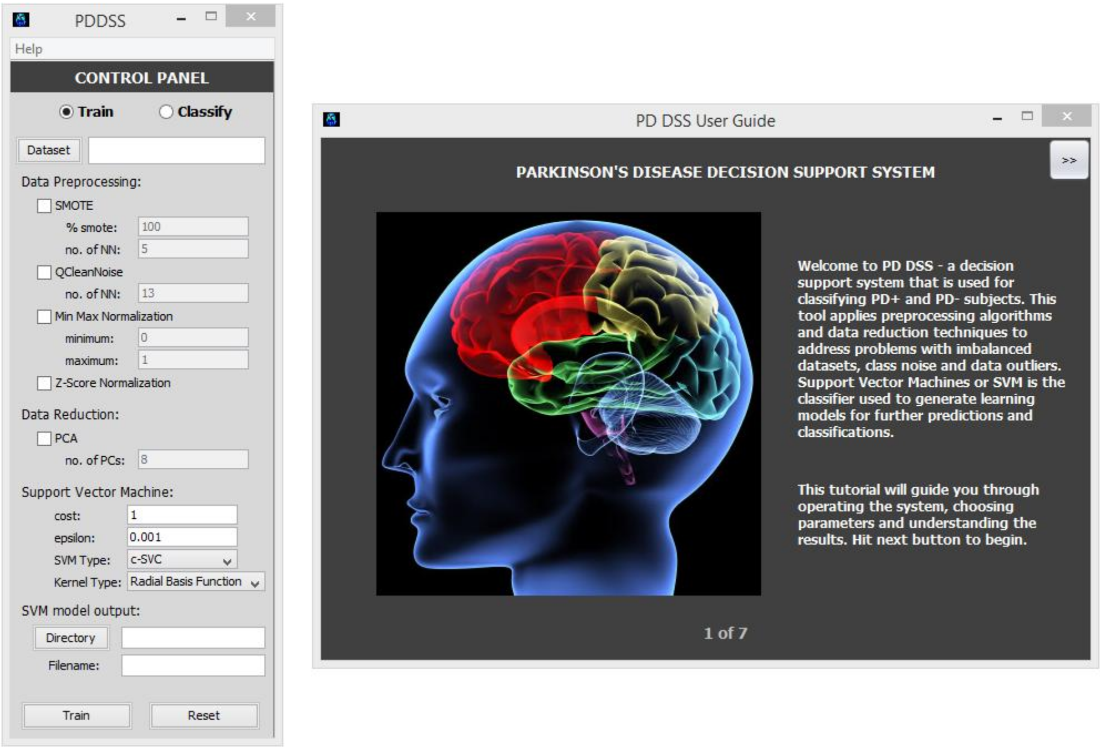
c. Java Development Kit (JDK 7)

d. LibSVM version 3.20

CHAPTER V

RESULTS

The first view of the system consists of the *Control Panel* and the *User Guide* as shown in Fig. 14. The *User Guide* frame contains general information about the system, the instructions on how to use the program and the required parameters as well as how to analyze the results. The *Control Panel* frame, on the other hand, is the main frame which contains the navigations and functionalities that the system can do.



● Train ○ Classify

Data Preprocessing:

□ SMOTE
% smote: 100
no. of NN: 5
□ QCleanNoise
no. of NN: 13
□ Min Max Normalization
minimum: 0
maximum: 1
□ Z-Score Normalization

Data Reduction:

□ PCA
no. of PCs: 8

Support Vector Machine:

cost: 1
epsilon: 0.001
SVM Type: c-SVC
Kernel Type: Radial Basis Function

SVM model output:

Filename:

PARKINSON'S DISEASE DECISION SUPPORT SYSTEM

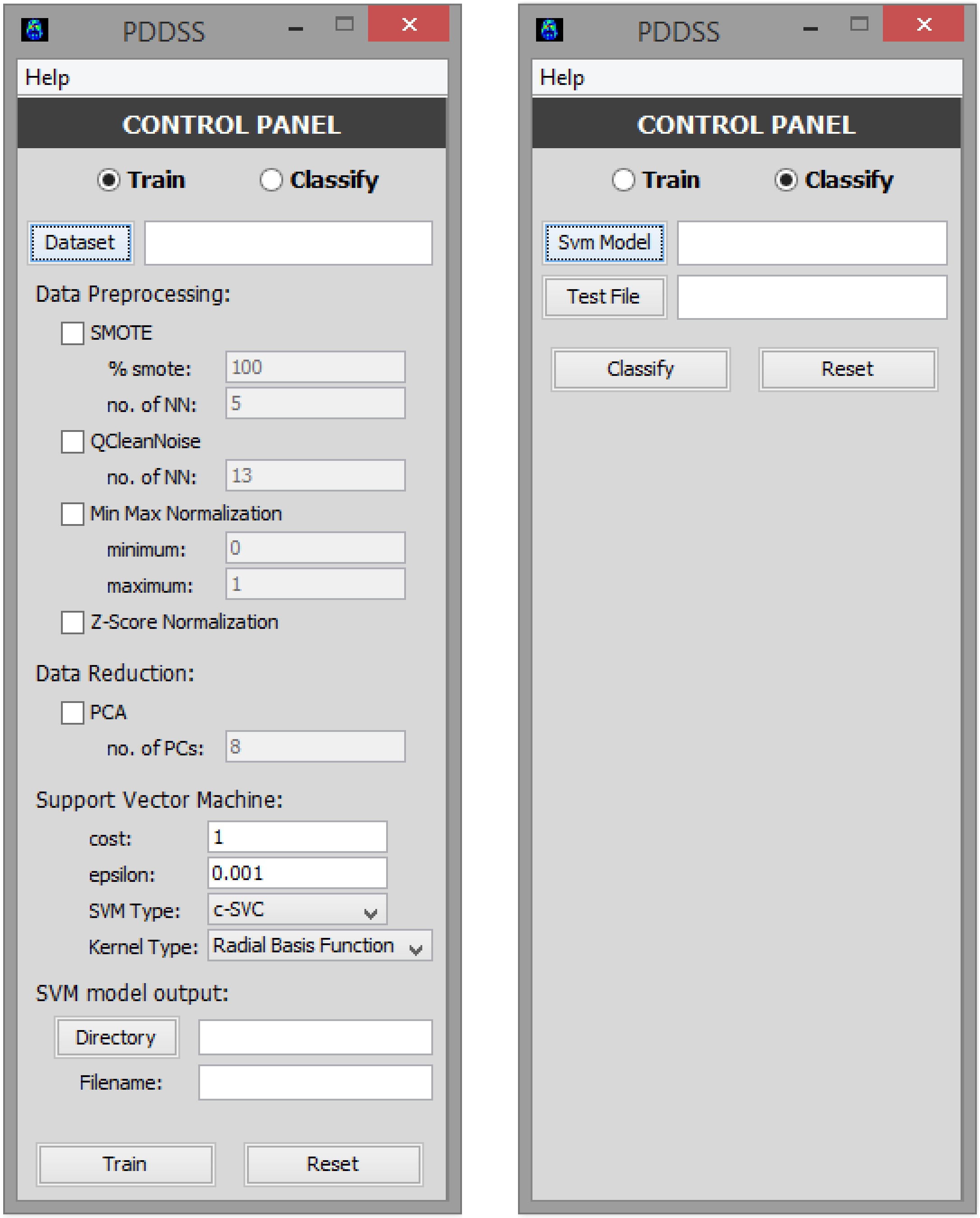
Welcome to PD DSS - a decision support system that is used for classifying PD+ and PD- subjects. This tool applies preprocessing algorithms and data reduction techniques to address problems with imbalanced datasets, class noise and data outliers. Support Vector Machines or SVM is the classifier used to generate learning models for further predictions and classifications.

This tutorial will guide you through operating the system, choosing parameters and understanding the results. Hit next button to begin.

1 of 7

Figure 14. Main View, PDDSS

As shown in Fig. 15, the *Control Panel* contains the menu bar and has two views. The first is for training and the second is for classifying.



● Train ○ Classify

Data Preprocessing:

□ SMOTE

% smote: 100

no. of NN: 5

□ QCleanNoise

no. of NN: 13

□ Min Max Normalization

minimum: 0

maximum: 1

□ Z-Score Normalization

Data Reduction:

□ PCA

no. of PCs: 8

Support Vector Machine:

cost: 1

epsilon: 0.001

SVM Type: c-SVC

Kernel Type: Radial Basis Function

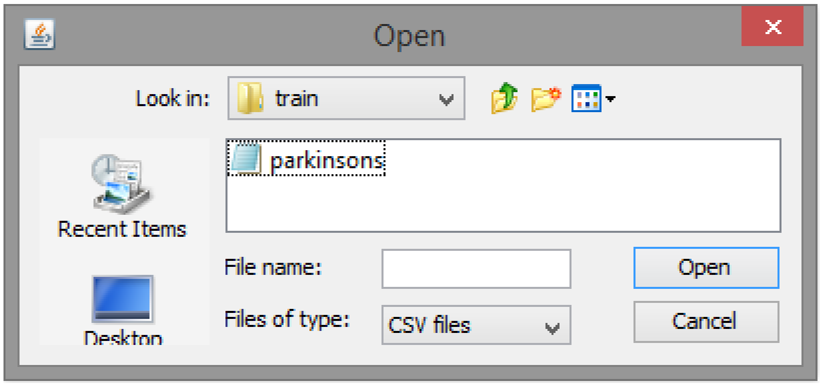
SVM model output:

Filename:

○ Train ● Classify

Figure 15. Control Panel, PDDSS

To start the system, the user can opt to train the system first or classify directly. This depends if there is an existing model for classification already. If the user wishes to train the system first, the user can start through choosing an input dataset by clicking *Dataset* button. This button will open a file chooser shown in Fig. 16 with required file extension of *.csv*.



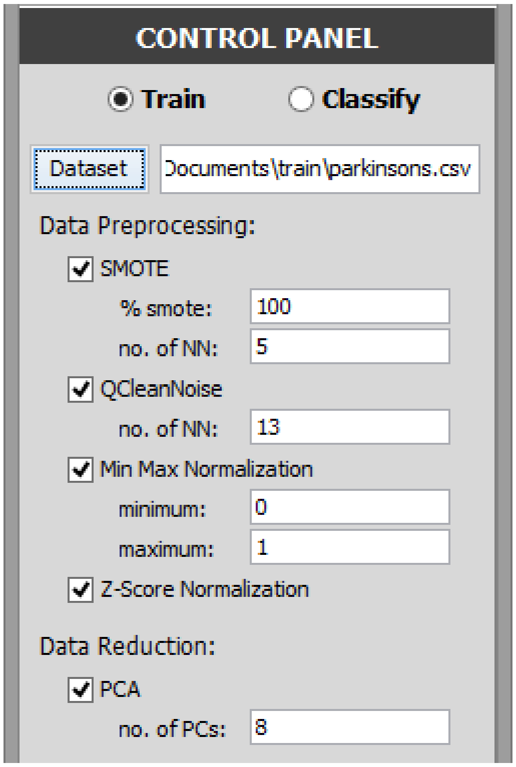
Look in: train

File name:

Files of type: CSV files

Figure 16. Dataset File Chooser, PDDSS

When the training dataset has already been input, the user can then choose the data preprocessing techniques that will be applied to the dataset. There are default values set for the preprocessing already but the user can still change the parameters for each selected data preprocessing. This section is shown in Fig. 17.



CONTROL PANEL

● Train ○ Classify

Dataset: Documents\train\parkinsons.csv

Data Preprocessing:

☑ SMOTE
- % smote: 100
- no. of NN: 5

☑ QCleanNoise
- no. of NN: 13

☑ Min Max Normalization
- minimum: 0
- maximum: 1

☑ Z-Score Normalization

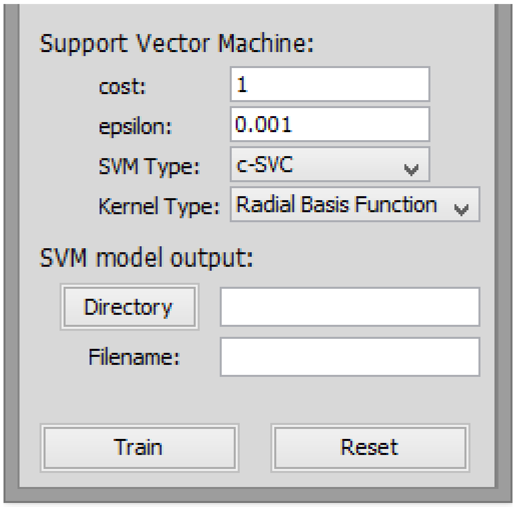
Data Reduction:

☑ PCA
- no. of PCs: 8

Figure 17. Data Preprocessing, Training, Control Panel, PDDSS

After choosing the data preprocessing techniques, the user can also check the default values set for the SVM parameters. Just like in the data preprocessing, these fields can still be changed by the user. When the parameters have been finalized, the user should also specify the directory and the filename where the model must be saved. By clicking the *Directory* button, the file directory chooser will pop up.

The user can then select where to save the model. Finally, at the bottom of the training pane, the user can see the *Train* button and the *Reset* button. To see the results of the training, the user can click the *Train* button but if the user would like to reset the parameters to the default values, then the user can select the *Reset* button. The second section of the training pane is shown in Fig. 18.



Support Vector Machine:

cost: 1

epsilon: 0.001

SVM Type: c-SVC

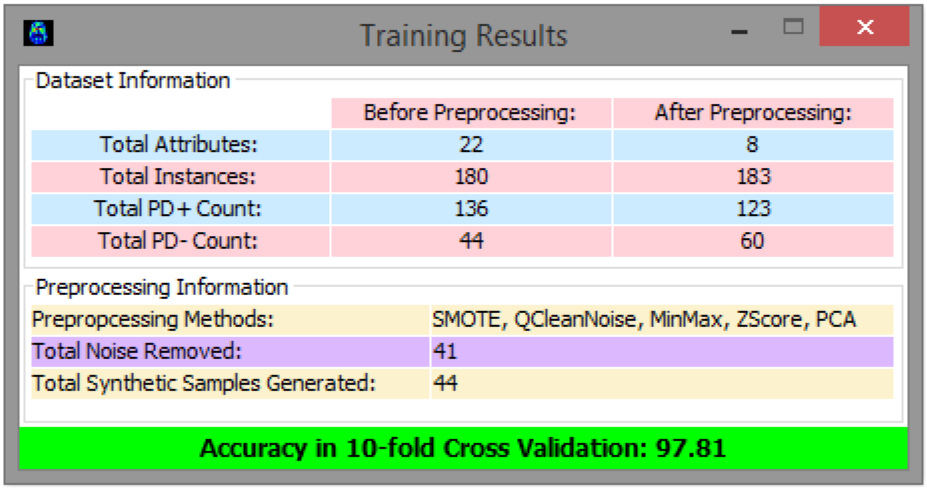
Kernel Type: Radial Basis Function

SVM model output:

Filename:

Figure 18. SVM Parameters, SVM Model Output, Training, Control Panel, PDDSS

The *Training Results* illustrated in Fig. 19 is divided into 3 sections - *Dataset Information*, *Preprocessing Information* and the *Accuracy*. The *Dataset Information* contains the general information about the dataset before and after preprocessing. It basically provides the total attributes and instances per class. The *Preprocessing Information* section contains the general information about what preprocessing techniques were applied to the data. The *Total Noise Removed* and the *Total Synthetic Samples Generated* indicates how much class noise was removed by the QCleanNoise algorithm and how many synthetic samples were introduced by the SMOTE algorithm. These two fields will have a 0 value if SMOTE or QCleanNoise is not initially selected by the user. Lastly, the *Accuracy* section provides the percentage accuracy of the model that is validated through 10-fold cross validation.



Training Results

Dataset Information

	Before Preprocessing:	After Preprocessing:
Total Attributes:	22	8
Total Instances:	180	183
Total PD+ Count:	136	123
Total PD- Count:	44	60

Preprocessing Information

Prepropcessing Methods:	SMOTE, QCleanNoise, MinMax, ZScore, PCA
Total Noise Removed:	41
Total Synthetic Samples Generated:	44

Accuracy in 10-fold Cross Validation: 97.81

Figure 19. Training Results, PDDSS

When the training phase is finished, the system serializes the data information, the preprocessing used, the parameters and the model to a *.bin* file with filename and path provided by the user. This *.bin* file cannot be viewed normally in any text editor since it is a java object written in a file. It is meant to be deserialized in the classification phase of the system. This will comprise all the parameters that will be used in classification.

In the case when a model already exists and the user would like to classify new instances, the user can tick the *Classify* button from the *Control Panel* and the *Classify pane* will appear. To classify, model file in *.bin* extension and the input test set in *.csv* extension must be chosen through the file chooser pop up. These panes will open through the *Svm Model* and *Test File* buttons shown in Fig. 20. When the files have been input, the user can click *Classify* button to show the *Classification Results* pane shown in Fig. 21. Otherwise, the user can click the *Reset* button to remove the previously selected files.

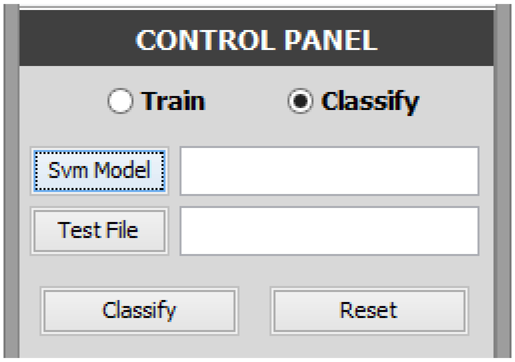
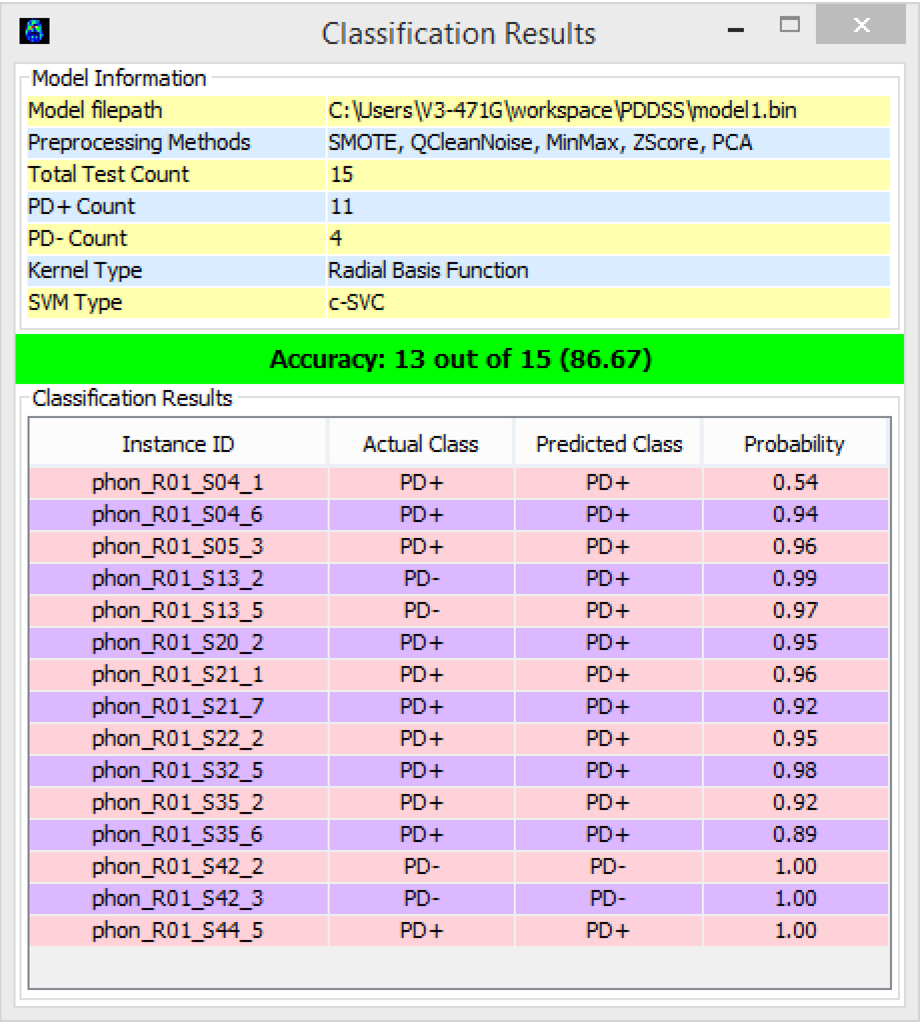


Figure 20. Classify Pane, Control Panel, PDDSS

The *Classification Results* frame has 3 sections - *Model Information, Accuracy* and the *Classification Results* table. The *Model Information* section provides information regarding the parameters used in training the model. These parameters include the preprocessing techniques used, the kernel type and SVM type. The total count of PD+ and PD- subjects of the test set are also included. *Accuracy* section, on the other hand, shows how many instances were correctly classified out of the total test count. Finally, the *Classification Results* table shows the prediction made by the model to each instance, shown in column 3, with the probability at the last column.



Classification Results

Model Information

Model filepath	C:\Users\V3-471G\workspace\PDDSS\model1.bin
Preprocessing Methods	SMOTE, QCleanNoise, MinMax, ZScore, PCA
Total Test Count	15
PD+ Count	11
PD- Count	4
Kernel Type	Radial Basis Function
SVM Type	c-SVC

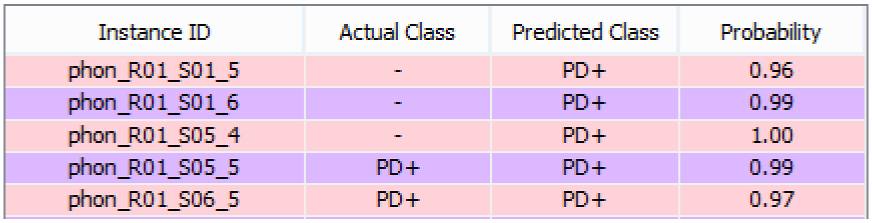
Accuracy: 13 out of 15 (86.67)

Classification Results

Instance ID	Actual Class	Predicted Class	Probability
phon_R01_S04_1	PD+	PD+	0.54
phon_R01_S04_6	PD+	PD+	0.94
phon_R01_S05_3	PD+	PD+	0.96
phon_R01_S13_2	PD-	PD+	0.99
phon_R01_S13_5	PD-	PD+	0.97
phon_R01_S20_2	PD+	PD+	0.95
phon_R01_S21_1	PD+	PD+	0.96
phon_R01_S21_7	PD+	PD+	0.92
phon_R01_S22_2	PD+	PD+	0.95
phon_R01_S32_5	PD+	PD+	0.98
phon_R01_S35_2	PD+	PD+	0.92
phon_R01_S35_6	PD+	PD+	0.89
phon_R01_S42_2	PD-	PD-	1.00
phon_R01_S42_3	PD-	PD-	1.00
phon_R01_S44_5	PD+	PD+	1.00

Figure 21. Classification Results, PDDSS

Since the system's aim is to just give a prediction and not totally replace the diagnosis of an expert, the test subjects are required to have a class already – whether PD+ or PD-. If in case the subjects doesn't have a known class yet, the classification results *Actual Class* column will have a "-" value just like in Fig. 22.



Instance ID	Actual Class	Predicted Class	Probability
phon_R01_S01_5	-	PD+	0.96
phon_R01_S01_6	-	PD+	0.99
phon_R01_S05_4	-	PD+	1.00
phon_R01_S05_5	PD+	PD+	0.99
phon_R01_S06_5	PD+	PD+	0.97

Figure 22. Actual class display of instances with unknown class

There can be more than one instance of the *Classification Results* frame and the *Training Results* frame depending whether the user wants to run multiple training or classifying procedure.

Finally, the *Help Menu* shown in Fig. 23 contains the *Tutorial* for the user guide and *About* for the details of the system.

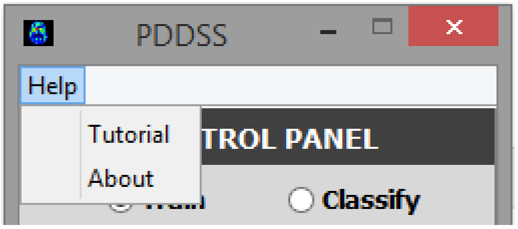
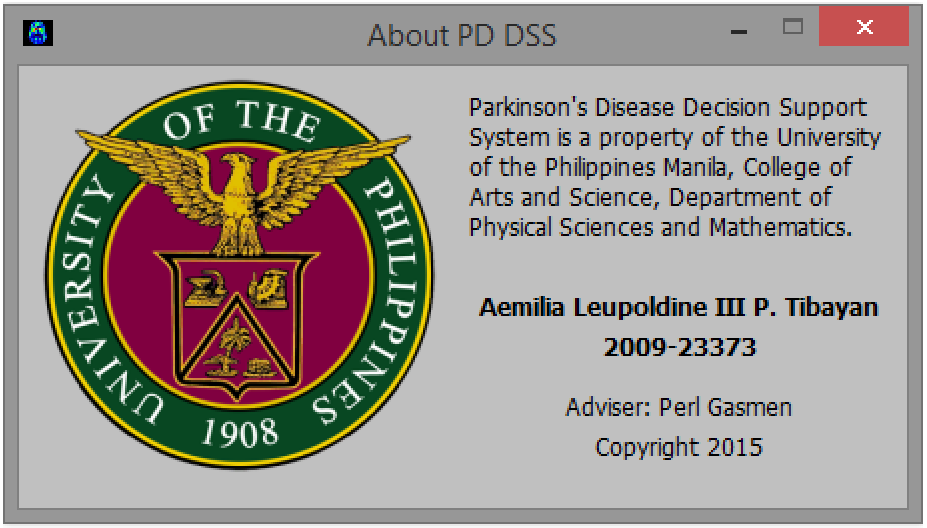


Figure 23. Help Menu, Control Panel, PDDSS

Fig. 24 shows the *About* pane. This window primarily shows general information of the system – the developer, the adviser and the year in which the system was implemented.



Parkinson's Disease Decision Support System is a property of the University of the Philippines Manila, College of Arts and Science, Department of Physical Sciences and Mathematics.

Aemilia Leupoldine III P. Tibayan
2009-23373

Adviser: Perl Gasmen
Copyright 2015

Figure 24. About Window, PDDSS

CHAPTER VI

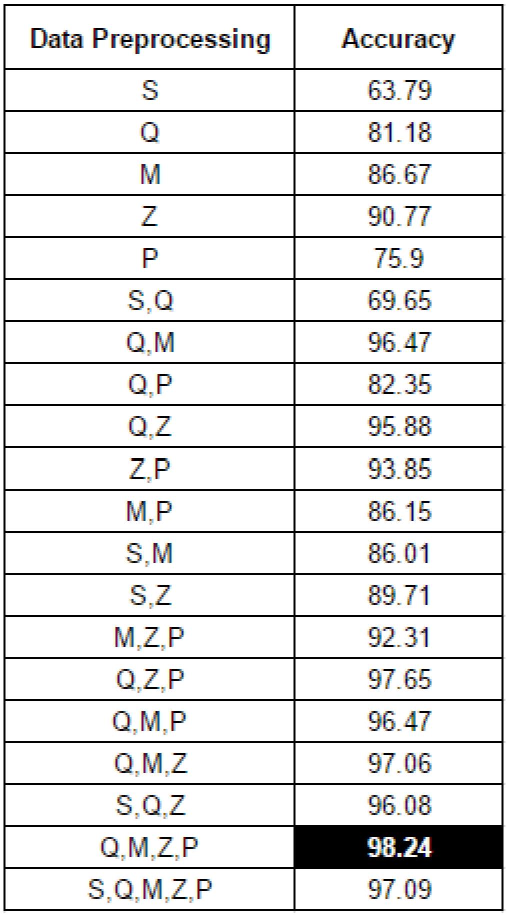
DISCUSSION

Parkinson’s Disease Decision Support System is a tool used for classifying PD+ and PD- instances based on the PD dataset provided in UCI Machine Learning Repository. The system has two major views – the train view and the classification view. The training pane allows the user to input dataset for training, choose data preprocessing and reduction techniques, enter valid parameters that will be used for the algorithms and state where to put the testing model output. For classification, the user can input the testing model produced by training and input the test set that contains the subjects to be classified.

In training, the system provides four data preprocessing algorithms to choose from – SMOTE, QCleanNoise, Min Max Normalization and ZScore Normalization. Each of these data preprocessing, when selected, gives the user the option to input valid parameters. A data reduction technique – PCA, is also available if in case the user prefers the dimensionality of the data to be smaller than the original data. In addition, the classifier algorithm by default is SVM.

The total number of instance in the PD dataset provided is 195. PD+ subjects comprise 147 of the total instances and the rest is PD-. For testing purposes, before the data is fed to the data preprocessing algorithms and classifier, the system randomly selects 15 test data and saves it with filename demo.csv. This file can then be used later when the user tries classifying data.

For determining the accuracy, the system uses 10-fold cross validation applied to the different data preprocessing combinations. The default values were used for each data preprocessing. Some important combinations are shown in Fig. 25.



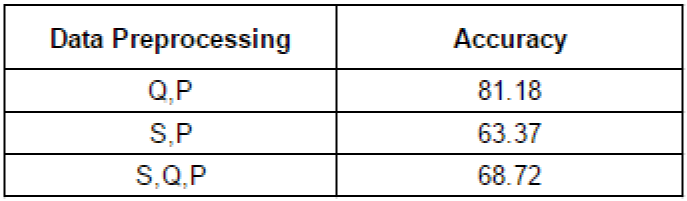
Data Preprocessing	Accuracy
S	63.79
Q	81.18
M	86.67
Z	90.77
P	75.9
S,Q	69.65
Q,M	96.47
Q,P	82.35
Q,Z	95.88
Z,P	93.85
M,P	86.15
S,M	86.01
S,Z	89.71
M,Z,P	92.31
Q,Z,P	97.65
Q,M,P	96.47
Q,M,Z	97.06
S,Q,Z	96.08
Q,M,Z,P	**98.24**
S,Q,M,Z,P	97.09

Figure 25. Data preprocessing combination with accuracy (RBF)

The results show that the accuracy varies depending on the preprocessing techniques applied to the data. This entails that the accuracy also depends on how the user will train the data – by selecting the right combination of preprocessing techniques.

From the highlighted row in the table above, the combination of QCleanNoise, Min Max Normalization, ZScore Normalization and PCA yields the highest accuracy with 98.24%. Though looking at these methods separately, the methods did not perform as well as when combined. It can also be noted that the highest accuracy is not so much different from other combinations like when SMOTE is added (S,Q,M,Z,P) that garnered 97.09%. Also, when Min Max Normalization is removed (Q, Z, P) that obtained the second highest accuracy of 97.65%.

From these observations, the highest accuracies achieved always incorporate scaling, normalizing or standardizing the data. Since the dataset is biased due to outlying values, the data shows poor accuracy when no standardization is applied shown in Fig. 26.

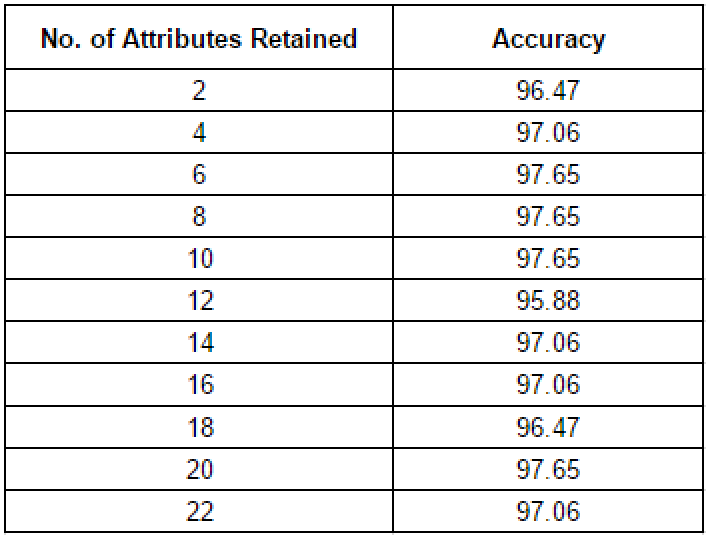


Data Preprocessing	Accuracy
Q,P	81.18
S,P	63.37
S,Q,P	68.72

Figure 26. Data preprocessing without scaling

In the table above, QCleanNoise and PCA combined yields the highest accuracy with 81.18%. The other combinations fall in <70% range. The results of data preprocessing without scaling proves that the dataset needs a normalizing algorithm before being fed to the classifier.

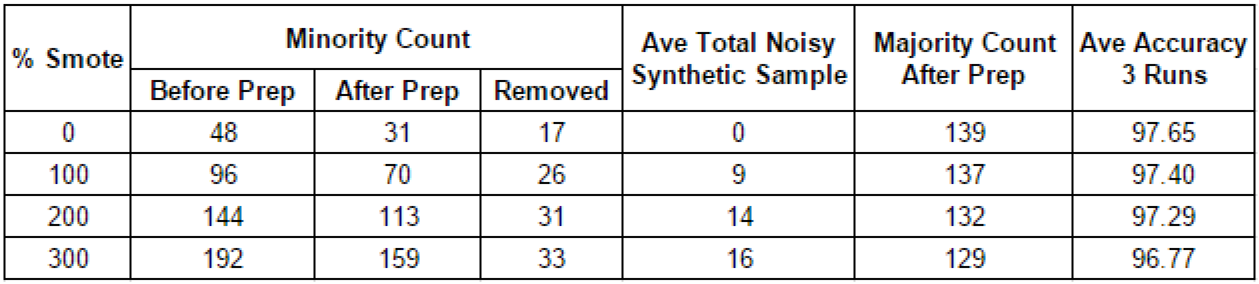
Another claim that needs to be addressed is the concept of Principal Components Analysis stating that the algorithm reduces the dimensionality of the data without compromising the validity of the data. Fig. 27 summarizes the accuracies obtained from selecting different counts of attributes to retain. From the figure, the range of accuracy is between 95.88% and 97.65% which varies since 10-fold cross validation picks instances at random.



No. of Attributes Retained	Accuracy
2	96.47
4	97.06
6	97.65
8	97.65
10	97.65
12	95.88
14	97.06
16	97.06
18	96.47
20	97.65
22	97.06

Figure 27. PCA algorithm with varying no. of attributes retained

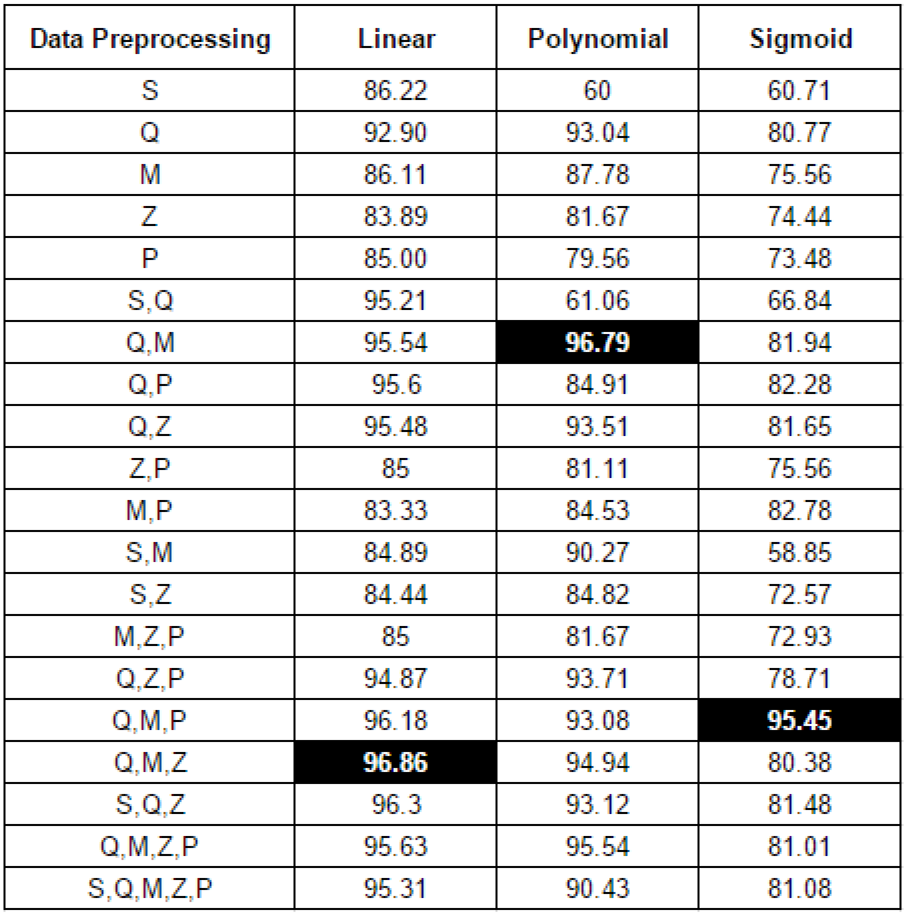
Additionally, the synthetic samples introduced if SMOTE algorithm is selected were observed together with QCleanNoise. Fig. 28 summarizes the data results.



% Smote	Minority Count			Ave Total Noisy Synthetic Sample	Majority Count After Prep	Ave Accuracy 3 Runs
	Before Prep	After Prep	Removed			
0	48	31	17	0	139	97.65
100	96	70	26	9	137	97.40
200	144	113	31	14	132	97.29
300	192	159	33	16	129	96.77

Figure 28. Smote algorithm with noise reduction

SMOTE algorithm was tested against varying percentage smote. The minority count before and after preprocessing were observed and the final count of removed minority instances were also calculated. From these values, the total noisy synthetic samples can be predicted. It can be observed that the higher the percentage of smote is, the greater the noisy synthetic samples were introduced. Therefore, it may be concluded that SMOTE algorithm is somewhat cancelled by the QCleanNoise algorithm. This is because most of the synthetic samples are based from originally noisy minority class instance. Furthermore, the majority class instances are also affected as more synthetic samples are introduced. As shown in column 6, the majority class is reduced more as the synthetic samples introduced increase. Though SMOTE introduces noisy samples, the accuracy is still good regardless of how much the smote percentage is since QCleanNoise eliminates these noisy instances.

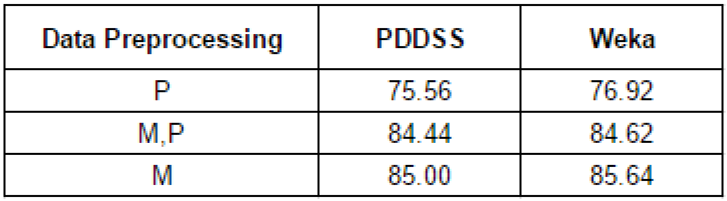


Data Preprocessing	Linear	Polynomial	Sigmoid
S	86.22	60	60.71
Q	92.90	93.04	80.77
M	86.11	87.78	75.56
Z	83.89	81.67	74.44
P	85.00	79.56	73.48
S,Q	95.21	61.06	66.84
Q,M	95.54	**96.79**	81.94
Q,P	95.6	84.91	82.28
Q,Z	95.48	93.51	81.65
Z,P	85	81.11	75.56
M,P	83.33	84.53	82.78
S,M	84.89	90.27	58.85
S,Z	84.44	84.82	72.57
M,Z,P	85	81.67	72.93
Q,Z,P	94.87	93.71	78.71
Q,M,P	96.18	93.08	**95.45**
Q,M,Z	**96.86**	94.94	80.38
S,Q,Z	96.3	93.12	81.48
Q,M,Z,P	95.63	95.54	81.01
S,Q,M,Z,P	95.31	90.43	81.08

Figure 29. Data preprocessing with accuracy of other kernel types

While Radial Basis Function has been the default kernel type used in the system and Q,M,Z,P combination proves to produce the highest accuracy with RBF, other kernel types namely Linear, Polynomial and Sigmoid have also been explored. The resulting accuracies are shown in Fig. 29. The highlighted values are the highest accuracy obtained per kernel type. For linear type, (Q,M,Z) achieved 96.86%. Polynomial, on the other hand, garnered 96.79% with (Q,M) combination. Finally, (Q,M,P) achieved 95.45% with sigmoid type. From these data, it can be deduced that the kernel type greatly influence the performance of the system. Similarly, the data preprocessing combination results vary depending on which kernel type is selected. In conclusion, it still holds true that (Q,M,Z,P) with RBF as kernel type garnered the highest accuracy.

Finally, for validation of the implementation, some open source libraries and software were used to check if the results are accurate. Weka [56], a data mining software written in Java offers libraries for data preprocessing and classifier. The PD dataset was transformed to *.arff* file that served as an input to Weka. The software has Normalization, PCA and SVM option. Unfortunately, other algorithms included in this study are still not available in Weka. The results of Normalization, PCA and SVM with RBF are shown in Fig. 30. Weka and PDDSS seems to have the same performance. Therefore, it shows that the algorithms were implemented correctly.



Data Preprocessing	PDDSS	Weka
P	75.56	76.92
M,P	84.44	84.62
M	85.00	85.64

Figure 30. Comparison of accuracy with Weka

CHAPTER VII

CONCLUSION

Parkinson’s Disease Decision Support System is a tool that makes use of data preprocessing techniques, data reduction algorithms and machine learning concepts to create a system that will aid students and researchers in understanding PD better and in classifying PD+ and PD- subjects. The system provides algorithms that will clean the data, standardize it and reduce it in a way that will generate the highest possible accuracy through 10-fold cross validation. Furthermore, the system allows the user to manipulate the data that will be used in training the system to give way to researches that will be conducted in the field of PD in the future. The resulting views consist of the training view and the classification view. These views primarily contain general information about the dataset, the preprocessing algorithms, the model and the accuracy. Finally, the system also includes a user guide that will be beneficial for unfamiliar users in the field of PD and machine learning.

The main objective of this study is to give decision support to researchers and students and to have a standardized way of arriving at possible diagnoses. This is achieved by the classification view that includes the predicted class of each instance fed to the system. It also aims to have a high accuracy in classification. This is proven by the highest accuracy obtained equal to 98.24%. But due to the fact that the available dataset is only limited, the system can only represent that of the 195 instances provided. Thus, further researches are still necessary when a larger dataset is already available.

CHAPTER VIII

RECOMMENDATIONS

PD DSS proves to have efficient and accurate algorithms that classify PD subjects but there are some aspects in this study that can still be improved on. Among these is the exploration on different modes of acquiring data.

The current dataset available is already in extracted form. The use of speech signals or vowel formants in raw version can be researched on and certain features used in this study can be extracted from those then fed to PDDSS. This can aid in maximizing the capabilities of each data preprocessing that is included in this study.

While there are a number of data preprocessing techniques already available, further improvements can include more algorithms. As proved on the discussions, some data preprocessing techniques work well if combined with others. Perhaps, there can be some algorithms that can address the noise brought about by generating synthetic samples or it may be that there is an existing improved version of the preprocessing algorithms in this study.

CHAPTER IX

BIBLIOGRAPHY

[1] "What Is Parkinson's Disease?" National Parkinson Foundation. Ed. Nina Browner. N.p., n.d. Web. Dec. 2014. <www.parkinson.org/parkinson-s-disease/pd-101/what-is-parkinson-s-disease>.

[2] "What Is Dopamine?" Psychology Today. Sussex Publishers, n.d. Web. Dec. 2014. <www.psychologytoday.com/basics/dopamine>.

[3] "Parkinson's Disease." WebMD. N.p., n.d. Web. Dec. 2014. <www.webmd.com/parkinsons-disease/parkinsons-disease-nord>.

[4] "Parkinson's Disease Symptoms." Mayo Clinic. Mayo Foundation for Medical Education and Research, n.d. Web. Dec. 2014. <www.mayoclinic.org/diseases-conditions/parkinsons-disease/basics/symptoms/con-20028488>.

[5] "CDC Lists Parkinson’s as the 14th Leading Cause of Death in America." Parkinson’s Action Network. N.p., Jan. 2012. Web. Dec. 2014. <parkinsonsaction.org/cdc-lists-parkinsons-14th-leading-cause-death-america/>.

[6] Men, Yifei. The Parkinson’s Voice Initiative: Early Diagnosis for Parkinson’s Disease through Speech Recognition (n.d.): n. pag. The Parkinson's Voice Initiative: Early Diagnosis for Parkinson's Disease through Speech Recognition | Stanford Journal of Public Health. Mar. 2013. Web. Dec. 2014.

[7] Subido, Joy. "Moving on after Parkinson's Disease." Philstar. N.p., May 2011. Web. Dec. 2014. <www.philstar.com/health-and-family/681539/moving-after-parkinsons-disease>.

[8] Simon, Harvey. "Parkinson's Disease." University of Maryland Medical Center. N.p., Oct. 2012. Web. Dec. 2014. <umm.edu/health/medical/reports/articles/parkinsons-disease>.

[9] Hazan, Hananel, Dan Hilu, Larry Manevitz, Lorraine Ramig, and Shimon Sapir. "Early Diagnosis of Parkinson’s Disease via Machine Learning on Speech Data." (2012): n. pag. IEEE 27-th Convention of Electrical and Electronics Engineers in Israel. Web. Dec. 2014.

[10] Tsanas, Athanasios, Max Little, Patrick McSharry, and Lorraine Ramig. "Accurate Telemonitoring of Parkinson's Disease Progression by Non-invasive Speech Tests." (2009): n. pag. IEEE Transactions on Biomedical Engineering. Web. Dec. 2014.

[11] Ruzickova, H., E. Ruzicka, J. Rusz, and R. Cmejla. "Quantitative Acoustic Measurements for Characterization of Speech and Voice Disorders in Early Untreated Parkinson's Disease." (2011): 350-67. Web. Dec. 2014.

[12] Shahbakhti, Mohammad, Danial Taherifar, and Zahra Zareei. "Combination Of PCA And SVM For Diagnosis Of Parkinson's Disease." (2013): n. pag. The 2nd International Conference on Advances in Biomedical Engineering. Web. Dec. 2014.

[13] Mair, Frances, Carl May, Catherine O'Donnell, Tracy Finch, Frank Sullivan, and Elizabeth Murray. "Factors That Promote or Inhibit the Implementation of E-health Systems: An Explanatory Systematic Review." 90 (2012): 357-64. Bulletin of the World Health Organization. Web. Dec. 2014.

[14] Kononenko, Igor. "Machine Learning for Medical Diagnosis: History, State of the Art and Perspective." 23.1 (2001): 89-109. Elsevier Science B.V. Web. Dec. 2014.

[15] Shahbakhi, Mohammad, Danial Taheri Far, and Ehsan Tahami. "Speech Analysis for Diagnosis of Parkinson's Disease Using Genetic Algorithm and Support Vector Machine." *J. Biomedical Science and Engineering* 7 (2014): 147-56. *Scientific Research*. Web. June 2015.

[16] Gil, David, and Magnus Johnson. "Diagnosing Parkinson by Using Artificial Neural Networks and Support Vector Machines." *Global Journal of Computer Science and Technology* (n.d.): 63-71. Web. June 2015.

[17] Tsanas, Athanasios, Max Little, Patrick McSharry, Jennifer Spielman, and Lorraine Ramig. "Novel Speech Signal Processing Algorithms for High-Accuracy Classification of Parkinson's Disease." *IEEE Transactions on Biomedical Engineering* 59.5 (2012): 1264-271. Web. June 2015.

[18] Little, Max. "Parkinsons Data Set." *UCI Machine Learning Repository*. National Science Foundation, n.d. Web. Dec. 2014. <https://archive.ics.uci.edu/ml/datasets/Parkinsons>.

[19] Pagan, Fernando L. "Improving Outcomes Through Early Diagnosis of Parkinson's Disease." The American Journal of Managed Care (2012): n. pag. PubMed. Web. Dec. 2014.

[20] Salvatore, Christian, Antonio Cerasa, Isabella Castiglioni, and Antonio Augimeri. "Machine Learning on Brain MRI Data for Differential Diagnosis of Parkinson's Disease and Progressive Supranuclear Palsy." Journal of Neuroscience Methods (2013): 230-37. Web. Dec. 2014.

[21] Ghumbre, Shashikant, Chetan Patil, and Ashok Ghatol. "Heart Disease Diagnosis Using Support Vector Machine." International Conference on Computer Science and Information Technology (2011): n. pag. Web. Dec. 2014.

[22] Hoang Le, Thai, and Len Bui. "Face Recognition Based on SVM and 2DPCA." International Journal of Signal Processing, Image Processing and Pattern Recognition 4.3 (2011): n. pag. Web. Dec. 2014.

[23] Poh, Tan Chue, Nur Fateha Muhamad Lani, and Lai Weng Kin. "Multi-dimensional Features Reduction of PCA on SVM Classifier for Imaging Surveillance Application." International Journal of Systems Applications, Engineering & Development 3.1 (2007): n. pag. Web. Dec. 2014.

[24] Ali, Shawkat, and Kate A. Smith-Miles. "Improved Support Vector Machine Generalization Using Normalized Input Space." (2006): 352-71. Springer-Verlag Berlin Heidelberg. Web. Dec. 2014.

[25] Chawla, Nitesh V., Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. "SMOTE: Synthetic Minority Over-sampling Technique." Journal of Artificial Intelligence Research 16 (2002): 321-57. Web. Dec. 2014.

[26] Han, Hui, Wen-Yuan Wang, and Bing-Huan Mao. "Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning." Lecture Notes in Computer Science 3644 (2005): 878-87. Web. Dec. 2014.

[27] Akbani, Rehan, Stephen Kwek, and Nathalie Japkowicz. "Applying Support Vector Machines to Imbalanced Datasets." Lecture Notes in Computer Science 3201 (2004): 39-50. Web. Dec. 2014.

[28] Natarajan, Nagarajan, Inderjit S. Dhillon, and Pradeep Ravikumar. "Learning with Noisy Labels." Advances in Neural Information Processing Systems 26 (2013): n. pag. Web. Dec. 2014.

[29] Daza, Luis, and Edgar Acuna. "An Algorithm for Detecting Noise on Supervised Classification." Proceedings of the World Congress on Engineering and Computer Science (2007): n. pag. Web. Dec. 2014.

[30] Hadjahmadi, A.H., and Taiebeh J. Askari. "A Decision Support System for Parkinson's Disease Diagnosis Using Classification and Regression Tree." The Journal of Mathematics and Computer Science 4.2 (2012): 257-63. Web. Dec. 2014.

[31] Bhande, Sanjivani, and RanjanaRaut, Dr. "Intelligent Decision Support System for Parkinson Diseases Using Softcomputing." IOSR Journal of Electrical and Electronics Engineering (2014): 17-22. Web.

[32] Ericsson, Anders, Markus Nowak Lonsdale, Kalle Astrom, Lars Edenbrandt, and Lars Friberg. "Decision Support System for the Diagnosis of Parkinson's Disease." Lecture Notes in Computer Science Volume 3540 (2005): 740-49. Web. Dec. 2014.

[33] "Parkinson's Disease." *MedicineNet*. N.p., n.d. Web. May 2015. <http://www.medicinenet.com/parkinsons_disease/article.htm>.

[34] "Parkinson's Disease." *Medline Plus*. N.p., n.d. Web. May 2015. <http://www.nlm.nih.gov/medlineplus/parkinsonsdisease.html>.

[35] "What Is Parkinson's Disease?" *National Institute of Neurological Disorders and Stroke*. N.p., n.d. Web. May 2015. <http://www.ninds.nih.gov/disorders/parkinsons_disease/parkinsons_disease.htm>.

[36] Blanchet, Paul. "Speech Disorders in Individuals with Parkinson’s Disease." *New York State Speech-Language-Hearing Association, Inc.* N.p., n.d. Web. May 2015.

[37] Janssen, Cory. "Decision Support System (DSS)." Techopedia. Janalta Interactive Inc., n.d. Web. Dec. 2014. <www.techopedia.com/definition/770/decision-support-system-dss>.

[38] Berner, ES. "Clinical decision support systems: State of the Art. " *AHRQ Publication No. 09-0069-EF. Rockville, Maryland: Agency for Healthcare Research and Quality*. June 2009.

[39] Rouse, Margaret. "What Is Machine Learning? - Definition from WhatIs.com." WhatIs.com. Techtarget, Jan. 2011. Web. Dec. 2014. <whatis.techtarget.com/definition/machine-learning>.

[40] Magoulas, George D., and Andriana Prentza. "Machine Learning In Medical Applications." *Lecture Notes In Computer Science* 2049 (2001): 300-07. Web. Dec. 2014.

[41] Kotsiantis, S. B. "Supervised Machine Learning: A Review of Classification Techniques." *Informatica* 31 (2007): 249-68. Web. Dec. 2014.

[42] "Supervised Learning Workflow and Algorithms." *MathWorks*. The MathWorks, Inc., n.d. Web. Dec. 2014. <se.mathworks.com/help/stats/supervised-learning-machine-learning-workflow-and-algorithms.html>.

[43] Rouse, Margaret. "Data Preprocessing Definition." SearchSQLServer. TechTarget, n.d. Web. Dec. 2014. <searchsqlserver.techtarget.com/definition/data-preprocessing>.

[44] Malik, Jasdeep Singh, Prachi Goyal, and Akhilesh K Sharma. "A Comprehensive Approach Towards Data Preprocessing Techniques & Association Rules." Proceedings of The4th National Conference (2010): n. pag. Web. Dec. 2014.

[45] Rascha, Sebastian. "About Feature Scaling and Normalization." Sebastian Racha. Disques, n.d. Web. Dec. 2014. <sebastianraschka.com/Articles/2014_about_feature_scaling.html#about-min-max-scaling>.

[46] Kamber, Micheline. "Example 3.4 Min-max Normalization." Data Mining: Concepts and Techniques. By Jiawei Han. N.p.: n.p., n.d. N. pag. Web. Dec. 2014.

[47] Boersma, Paul, and David Weenink. "Principal Component Analysis." Praat: Doing Phonetics by Computer. N.p., May 2012. Web. Dec. 2014. <www.fon.hum.uva.nl/praat/manual/Principal_component_analysis.html>.

[48] Fernandez, George. "Principal Component Analysis." University of Nevada, Reno. Web. Dec. 2014. <www.cabnr.unr.edu/saito/classes/ers701/pca2.pdf>.

[49] Smith, Lindsay I. "A Tutorial on Principal Components Analysis." Cornell University, USA, Feb. 2002. Web. Dec. 2014. <www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf>.

[50] "SVM - Support Vector Machines." Support Vector Machines. Ovidiu Ivanciuc, 2005. Web. Dec. 2014. <support-vector-machines.org/>.

[51] "Introduction to Support Vector Machines." OpenCV. N.p., n.d. Web. Dec. 2014. <docs.opencv.org/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html>.

[52] Srivastava, Durgesh K., and Lekha Bhambhu. "Data Classification Using Support Vector Machine." Journal of Theoretical and Applied Information Technology (2009): n. pag. Web. Dec. 2014.

[53] Fletcher, Tristan. "www.tristanfletcher.co.uk/SVM%20Explained.pdf." Dec. 2014. Reading.

[54] Chang, Chih-Chung, and Chih-Jen Lin. LibSVM. Computer software. LIBSVM -- A Library for Support Vector Machines. Vers. 3.2. National Science Council of Taiwan, n.d. Web. Dec. 2014. <www.csie.ntu.edu.tw/~cjlin/libsvm/>.

[55] Little, Max A., Patrick E. McSharry, Eric J. Hunter, and Lorraine O. Ramig. "Suitability of Dysphonia Measurements for Telemonitoring of Parkinson's Disease." IEEE Transactions on Biomedical Engineering 56.4 (2009): 1015-022. Web. Dec. 2014.

[56] Fletcher, Dale, and Peter Reutemann. *Weka 3: Data Mining Software in Java*. Computer software. *Machine Learning Group at the University of Waikato*. Vers. 3-7-12. N.p., n.d. Web. Dec. 2014.

CHAPTER X

APPENDIX

Source Code

A. Launcher.java

```java
import gui.mainUI;
/**
 *
 * @author      Aemilia Leupoldine III P. Tibayan
 *              2009-23373
 *              University of the Philippines Manila
 *              Department of Physical Sciences and Mathematics
 *
 */

public class Launcher {
        public static void main (String[] args) {
                javax.swing.SwingUtilities.invokeLater(new Runnable () {
                        public void run () {
                                new mainUI ();
                        }
                });
        }
}
```

B. gui/AboutFrame.java

```java
package gui;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Image;
import java.awt.event.FocusEvent;
import java.awt.event.FocusListener;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;
import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JLayeredPane;
import javax.swing.JPanel;
import javax.swing.SwingConstants;

@SuppressWarnings("serial")
public class AboutFrame extends JFrame implements FocusListener {
        private JFrame aboutFrame;
        private JPanel mainPanel;
        private GridBagLayout gridBag = new GridBagLayout();
private GridBagConstraints cons = new GridBagConstraints();

AboutFrame() {
	aboutFrame = new JFrame("About PD DSS");
	aboutFrame.addFocusListener(this);
	aboutFrame.setSize(450, 250);
	aboutFrame.setLocation(400, 200);
	aboutFrame.setResizable(false);
	aboutFrame.setVisible(true);
	ImageIcon img = new ImageIcon("./img/icon.jpg");
	aboutFrame.setIconImage(img.getImage());
	aboutFrame.setBackground(Color.decode("#D8D8D8"));

	mainPanel = new JPanel(new BorderLayout());
	mainPanel.setPreferredSize(new Dimension(450,250));
	mainPanel.setBackground(Color.LIGHT_GRAY);
	mainPanel.setLayout(gridBag);
	createMainPanel();

	aboutFrame.getContentPane().add(mainPanel, BorderLayout.WEST);
	aboutFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
}

private void createMainPanel() {
	cons.weightx = 1;
	cons.weighty = 1;
	cons.gridx = 0;
	cons.gridy = 0;
	cons.gridwidth = 1;
	cons.gridheight = 1;
	cons.fill = GridBagConstraints.BOTH;

	JLayeredPane imagePane = new JLayeredPane();
	BufferedImage img1 = null;
	try {
		img1 = ImageIO.read(new File("./img/upm.png"));
	} catch(IOException e) {
		e.printStackTrace();
	}

	Image dimg1 = img1.getScaledInstance(200, 200, Image.SCALE_SMOOTH);

	JLabel imgLabel = new JLabel();
	imgLabel.setIcon(new ImageIcon(dimg1));
	imgLabel.setBounds(10, 5, 200, 200);
	imagePane.add(imgLabel);

	imagePane.setBounds(0, 0, 200, 200);
	gridBag.setConstraints(imagePane, cons);
	mainPanel.add(imagePane);

	cons.weightx = 1;
	cons.weighty = 1;
	cons.gridx = 1;
	cons.gridy = 0;
	cons.gridwidth = 1;
	cons.gridheight = 1;
	cons.fill = GridBagConstraints.BOTH;

	JLayeredPane infoPane = new JLayeredPane();
```

```
		JLabel infoPaneLabel1 = new JLabel("<html><align=justify> Parkinson's 
Disease Decision Support System is a "
				+ "property of the University of the Philippines Manila, 
College of Arts and Science, Department of "
				+ "Physical Sciences and Mathematics.</html>");
		infoPaneLabel1.setHorizontalAlignment(SwingConstants.CENTER);
		infoPaneLabel1.setBounds(0, -100, 210, 300);
		infoPaneLabel1.setForeground(Color.black);
		infoPaneLabel1.setFont(new Font("Tahoma", Font.PLAIN, 12));

		JLabel infoPaneLabel2 = new JLabel("<html><align=justify> Aemilia 
Leupoldine III P. Tibayan </html>");
		infoPaneLabel2.setHorizontalAlignment(SwingConstants.CENTER);
		infoPaneLabel2.setBounds(0, -30, 210, 300);
		infoPaneLabel2.setForeground(Color.black);
		infoPaneLabel2.setFont(new Font("Tahoma", Font.BOLD, 12));

		JLabel infoPaneLabel3 = new JLabel("<html><align=justify> 2009-23373 
</html>");
		infoPaneLabel3.setHorizontalAlignment(SwingConstants.CENTER);
		infoPaneLabel3.setBounds(0, -10, 210, 300);
		infoPaneLabel3.setForeground(Color.black);
		infoPaneLabel3.setFont(new Font("Tahoma", Font.BOLD, 12));

		JLabel infoPaneLabel4 = new JLabel("<html><align=justify> Adviser: Perl 
Gasmen </html>");
		infoPaneLabel4.setHorizontalAlignment(SwingConstants.CENTER);
		infoPaneLabel4.setBounds(0, 20, 210, 300);
		infoPaneLabel4.setForeground(Color.black);
		infoPaneLabel4.setFont(new Font("Tahoma", Font.PLAIN, 12));

		JLabel infoPaneLabel5 = new JLabel("<html><align=justify> Copyright 2015 
</html>");
		infoPaneLabel5.setHorizontalAlignment(SwingConstants.CENTER);
		infoPaneLabel5.setBounds(0, 40, 210, 300);
		infoPaneLabel5.setForeground(Color.black);
		infoPaneLabel5.setFont(new Font("Tahoma", Font.PLAIN, 12));

		infoPane.add(infoPaneLabel1);
		infoPane.add(infoPaneLabel2);
		infoPane.add(infoPaneLabel3);
		infoPane.add(infoPaneLabel4);
		infoPane.add(infoPaneLabel5);
		infoPane.setBounds(0, 0, 235, 500);
		infoPane.setBackground(Color.LIGHT_GRAY);
		gridBag.setConstraints(infoPane, cons);
		mainPanel.add(infoPane);
	}

	@Override
	public void focusGained(FocusEvent e) {}

	@Override
	public void focusLost(FocusEvent e) {
		if (e.getSource() == aboutFrame) {
			aboutFrame.dispose();
		}
	}
}
```

C. gui/ControlPanel.java

```
package gui;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import javax.swing.BorderFactory;
import javax.swing.ButtonGroup;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JLayeredPane;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JRadioButton;
import javax.swing.JTextField;
import javax.swing.SwingConstants;
import proc.Dataset;
import proc.ProcParams;
import proc.TestProcessor;
import proc.TrainProcessor;
import util.Util;

@SuppressWarnings("serial")
public class ControlPanel extends JPanel implements ActionListener, ItemListener {
	private JFrame aFrame;
	private GridBagLayout gridBag = new GridBagLayout();
	private GridBagConstraints cons = new GridBagConstraints();
	private JLayeredPane buttonOptionPane, trainPane, testPane;
	private JButton classifyBtn, resetTestBtn, testSetBtn, svmModelBtn, dirBtn, 
resetTrainBtn, trainBtn, openDatasetBtn;
	private JRadioButton trainRBtn, classifyRBtn;
	private JCheckBox pcaChk, smoteChk, qcleanChk, minmaxChk, zscoreChk;
	private JComboBox<String> svmTypeCombo, svmKernelTypeCombo;
	private JTextField testSetTextField, svmModelTextField, 
svmModelDirectoryTextField, svmModelFileTextField, costSVMTextField, epsSVMTextField, 
attrAmtTextField, minTextField,
	maxTextField, nnQCleanTextField, nnSmoteTextField, amtSmoteTextField, 
fileTextField;
	private Dataset dataModel;

	public ControlPanel(JFrame aFrame) {
		this.aFrame = aFrame;
		setPreferredSize(new Dimension(237,24));
		setOpaque(true);
		setBackground(Color.decode("#D8D8D8"));
		setLayout(gridBag);
		createSidePanel();
	}
	private void createSidePanel() {
		cons.weightx = 0;
		cons.weighty = 2;
```

```
        cons.fill = GridBagConstraints.BOTH;
        JPanel labPanel1 = new JPanel(new BorderLayout());
        labPanel1.setBackground(Color.DARK_GRAY);
        JLabel lab1 = new JLabel("CONTROL PANEL");
        lab1.setHorizontalAlignment(SwingConstants.CENTER);
        lab1.setForeground(Color.WHITE);
        lab1.setFont(new Font("Tahoma", Font.BOLD, 14));
        labPanel1.add(lab1, BorderLayout.CENTER);
        cons.gridx = 0;
        cons.gridy = 0;
        cons.gridwidth = 1;
        cons.gridheight = 1;
        gridBag.setConstraints(labPanel1, cons);

        add(labPanel1);

        cons.weightx = 1000;
        cons.weighty = 100;
        cons.fill = GridBagConstraints.BOTH;

        JLayeredPane controlPanel = new JLayeredPane();

        buttonOptionPane = createButtonOptionPane();
        buttonOptionPane.setBounds(0, 0, 235, 500);

        trainPane = createTrainPane();
        trainPane.setBounds(0, 0, 235, 600);

        testPane = createTestPane();
        testPane.setBounds(0, 0, 235, 600);

        controlPanel.add(buttonOptionPane);
        controlPanel.add(trainPane);
        controlPanel.add(testPane);
        controlPanel.setBorder(BorderFactory.createEmptyBorder());
        controlPanel.setRequestFocusEnabled(false);
        controlPanel.setBackground(Color.decode("#87CEFA"));
        cons.gridx = 0;
        cons.gridy = 1;
        cons.gridwidth = 1;
        cons.gridheight = 1;
        gridBag.setConstraints(controlPanel, cons);
        add(controlPanel);
}

private JLayeredPane createButtonOptionPane() {
        JLayeredPane buttonOptionPane = new JLayeredPane();

        trainRBtn = new JRadioButton("Train");
        trainRBtn.setBounds(40, 5, 100, 25);
        trainRBtn.setOpaque(false);
        trainRBtn.addItemListener(this);
        trainRBtn.setSelected(true);
        trainRBtn.setFont(new Font("Tahoma", Font.BOLD, 13));
        trainRBtn.setFocusable(false);

        classifyRBtn = new JRadioButton("Classify");
        classifyRBtn.setBounds(130, 5, 100, 25);
        classifyRBtn.setOpaque(false);
        classifyRBtn.addItemListener(this);
        classifyRBtn.setFont(new Font("Tahoma", Font.BOLD, 13));
        classifyRBtn.setFocusable(false);

        ButtonGroup btnGrp = new ButtonGroup();
        btnGrp.add(trainRBtn);
        btnGrp.add(classifyRBtn);

        buttonOptionPane.add(classifyRBtn);
        buttonOptionPane.add(trainRBtn);

        return buttonOptionPane;
}

private JLayeredPane createTestPane() {
        JLayeredPane testPane = new JLayeredPane();
        testPane.setPreferredSize(new Dimension(237, 700));
        testPane.setVisible(false);

        svmModelBtn = new JButton("Svm Model");
        svmModelBtn.setBorder(BorderFactory.createLineBorder(
                        Color.LIGHT_GRAY));
        svmModelBtn.setBounds(5, 40, 70, 25);
        svmModelBtn.addActionListener(this);
        svmModelTextField = new JTextField();
        svmModelTextField.setBounds(80, 40, 150, 25);

        testSetBtn = new JButton("Test File");
        testSetBtn.setBorder(BorderFactory.createLineBorder(
                        Color.LIGHT_GRAY));
        testSetBtn.setBounds(5, 70, 70, 25);
        testSetBtn.addActionListener(this);
        testSetTextField = new JTextField();
        testSetTextField.setBounds(80, 70, 150, 25);

        classifyBtn = new JButton("Classify");
        classifyBtn.addActionListener(this);
        classifyBtn.setBorder(BorderFactory.createLineBorder(Color.LIGHT_GRAY));
        classifyBtn.setBounds(10, 110, 100, 25);

        resetTestBtn = new JButton("Reset");
        resetTestBtn.addActionListener(this);
        resetTestBtn.setBorder(BorderFactory.createLineBorder(Color.LIGHT_GRAY));
        resetTestBtn.setBounds(125, 110, 100, 25);


        testPane.add(svmModelBtn);
        testPane.add(svmModelTextField);
        testPane.add(testSetBtn);
        testPane.add(testSetTextField);
        testPane.add(classifyBtn);
        testPane.add(resetTestBtn);
        return testPane;
}

private JLayeredPane createTrainPane() {
        JLayeredPane trainPane = new JLayeredPane();
        trainPane.setPreferredSize(new Dimension(237, 700));
        trainPane.setVisible(true);

        openDatasetBtn = new JButton("Dataset");
        openDatasetBtn.setBorder(BorderFactory.createLineBorder(
                        Color.LIGHT_GRAY));
```

```
		openDatasetBtn.setBounds(5, 40, 60, 25);
		openDatasetBtn.addActionListener(this);
		fileTextField = new JTextField();
		fileTextField.setBounds(70, 40, 160, 25);

		JLabel dpLabel = new JLabel("Data Preprocessing:");
		dpLabel.setBounds(10, 68, 200, 25);
		dpLabel.setFont(new Font("Tahoma", Font.PLAIN, 12));

		smoteChk = new JCheckBox("SMOTE");
		smoteChk.setBounds(20, 90, 200, 25);
		smoteChk.setOpaque(false);
		smoteChk.addItemListener(this);
		smoteChk.setFocusable(false);

		JLabel amtSmoteLabel = new JLabel("% smote: ");
		amtSmoteLabel.setBounds(50, 110, 100, 25);

		amtSmoteTextField = new JTextField();
		amtSmoteTextField.setBounds(115, 112, 100, 18);
		amtSmoteTextField.setEnabled(false);
		amtSmoteTextField.setText("100");
		amtSmoteTextField.setToolTipText("must not be less than 10 or amount of
nn");

		JLabel nnSmoteLabel = new JLabel("no. of NN: ");
		nnSmoteLabel.setBounds(50, 130, 100, 25);

		nnSmoteTextField = new JTextField();
		nnSmoteTextField.setBounds(115, 132, 100, 18);
		nnSmoteTextField.setEnabled(false);
		nnSmoteTextField.setText("5");
		nnSmoteTextField.setToolTipText("must not be less than 1");

		qcleanChk = new JCheckBox("QCleanNoise");
		qcleanChk.setBounds(20, 150, 200, 25);
		qcleanChk.setOpaque(false);
		qcleanChk.addItemListener(this);
		qcleanChk.setFocusable(false);

		JLabel nnQCleanLabel = new JLabel("no. of NN: ");
		nnQCleanLabel.setBounds(50, 170, 100, 25);

		nnQCleanTextField = new JTextField();
		nnQCleanTextField.setBounds(115, 172, 100, 18);
		nnQCleanTextField.setEnabled(false);
		nnQCleanTextField.setText("13");
		nnQCleanTextField.setToolTipText("must not be less than 1");

		minmaxChk = new JCheckBox("Min Max Normalization");
		minmaxChk.setBounds(20, 190, 200, 25);
		minmaxChk.setOpaque(false);
		minmaxChk.addItemListener(this);
		minmaxChk.setFocusable(false);

		JLabel minLabel = new JLabel("minimum: ");
		minLabel.setBounds(50, 210, 100, 25);

		minTextField = new JTextField();
		minTextField.setBounds(115, 212, 100, 18);
		minTextField.setEnabled(false);
```

```
               minTextField.setText("0");

               JLabel maxLabel = new JLabel("maximum: ");
               maxLabel.setBounds(50, 230, 100, 25);

               maxTextField = new JTextField();
               maxTextField.setBounds(115, 232, 100, 18);
               maxTextField.setEnabled(false);
               maxTextField.setText("1");

               zscoreChk = new JCheckBox("Z-Score Normalization");
               zscoreChk.setBounds(20, 250, 200, 25);
               zscoreChk.setOpaque(false);
               zscoreChk.addItemListener(this);
               zscoreChk.setFocusable(false);

               JLabel drLabel = new JLabel("Data Reduction:");
               drLabel.setBounds(10, 278, 200, 25);
               drLabel.setFont(new Font("Tahoma", Font.PLAIN, 12));

               pcaChk = new JCheckBox("PCA");
               pcaChk.setOpaque(false);
               pcaChk.setBounds(20, 300, 200, 25);
               pcaChk.addItemListener(this);
               pcaChk.setFocusable(false);

               JLabel attrAmtLabel = new JLabel("no. of PCs: ");
               attrAmtLabel.setBounds(50, 320, 150, 25);

               attrAmtTextField = new JTextField();
               attrAmtTextField.setBounds(115, 322, 100, 18);
               attrAmtTextField.setEnabled(false);
               attrAmtTextField.setText("8");
               attrAmtTextField.setToolTipText("must not be less than 1 or greater than
22");

               JLabel svmLabel = new JLabel("Support Vector Machine:");
               svmLabel.setBounds(10, 348, 200, 25);
               svmLabel.setFont(new Font("Tahoma", Font.PLAIN, 12));

               JLabel costSVMLabel = new JLabel("cost: ");
               costSVMLabel.setBounds(40, 370, 150, 25);

               costSVMTextField = new JTextField();
               costSVMTextField.setBounds(105, 372, 100, 18);
               costSVMTextField.setText("1");

               JLabel epsSVMLabel = new JLabel("epsilon: ");
               epsSVMLabel.setBounds(40, 390, 150, 25);

               epsSVMTextField = new JTextField();
               epsSVMTextField.setBounds(105, 392, 100, 18);
               epsSVMTextField.setText("0.001");

               JLabel typeSVMLabel = new JLabel("SVM Type: ");
               typeSVMLabel.setBounds(40, 410, 150, 25);

               String[] svmTypeChoices = {"nu-SVC", "c-SVC"};
               svmTypeCombo = new JComboBox<String>(svmTypeChoices);
               svmTypeCombo.setBounds(105, 412, 100, 18);
               svmTypeCombo.setSelectedIndex(1);
```

```
              svmTypeCombo.setFocusable(false);

              JLabel kernelSVMLabel = new JLabel("Kernel Type: ");
              kernelSVMLabel.setBounds(40, 430, 150, 25);

              String[] svmKernelTypeChoices = {"Linear", "Polynomial", "Radial Basis
Function", "Sigmoid"};
              svmKernelTypeCombo = new JComboBox<String>(svmKernelTypeChoices);
              svmKernelTypeCombo.setBounds(105, 432, 125, 18);
              svmKernelTypeCombo.setSelectedIndex(2);
              svmKernelTypeCombo.setFocusable(false);

              JLabel svmModelOutputLabel = new JLabel("SVM model output: ");
              svmModelOutputLabel.setBounds(10, 455, 150, 25);
              svmModelOutputLabel.setFont(new Font("Tahoma", Font.PLAIN, 12));

              dirBtn = new JButton("Directory");
              dirBtn.addActionListener(this);
              dirBtn.setBorder(BorderFactory.createLineBorder(Color.LIGHT_GRAY));
              dirBtn.setBounds(20, 480, 70, 24);

              svmModelDirectoryTextField = new JTextField();
              svmModelDirectoryTextField.setBounds(100, 482, 130, 20);

              JLabel svmModelFileLabel = new JLabel("Filename: ");
              svmModelFileLabel.setBounds(35, 505, 150, 25);

              svmModelFileTextField = new JTextField();
              svmModelFileTextField.setBounds(100, 507, 130, 20);

              trainBtn = new JButton("Train");
              trainBtn.addActionListener(this);
              trainBtn.setBorder(BorderFactory.createLineBorder(Color.LIGHT_GRAY));
              trainBtn.setBounds(10, 550, 100, 25);

              resetTrainBtn = new JButton("Reset");
              resetTrainBtn.addActionListener(this);

       resetTrainBtn.setBorder(BorderFactory.createLineBorder(Color.LIGHT_GRAY));
              resetTrainBtn.setBounds(125, 550, 100, 25);

              trainPane.add(openDatasetBtn);
              trainPane.add(fileTextField);
              trainPane.add(dpLabel);
              trainPane.add(smoteChk);
              trainPane.add(amtSmoteLabel);
              trainPane.add(amtSmoteTextField);
              trainPane.add(nnSmoteLabel);
              trainPane.add(nnSmoteTextField);
              trainPane.add(qcleanChk);
              trainPane.add(nnQCleanLabel);
              trainPane.add(nnQCleanTextField);
              trainPane.add(minmaxChk);
              trainPane.add(minLabel);
              trainPane.add(minTextField);
              trainPane.add(maxLabel);
              trainPane.add(maxTextField);
              trainPane.add(zscoreChk);
              trainPane.add(drLabel);
              trainPane.add(pcaChk);
              trainPane.add(attrAmtLabel);
```

```
        trainPane.add(attrAmtTextField);
        trainPane.add(svmLabel);
        trainPane.add(costSVMLabel);
        trainPane.add(costSVMTextField);
        trainPane.add(epsSVMLabel);
        trainPane.add(epsSVMTextField);
        trainPane.add(typeSVMLabel);
        trainPane.add(svmTypeCombo);
        trainPane.add(kernelSVMLabel);
        trainPane.add(svmKernelTypeCombo);
        trainPane.add(svmModelOutputLabel);
        trainPane.add(dirBtn);
        trainPane.add(svmModelDirectoryTextField);
        trainPane.add(svmModelFileLabel);
        trainPane.add(svmModelFileTextField);
        trainPane.add(trainBtn);
        trainPane.add(resetTrainBtn);
        return trainPane;
}

public void itemStateChanged(ItemEvent e) {
        if(e.getStateChange() == ItemEvent.SELECTED) {
                if(e.getSource() == trainRBtn) {
                        if(trainPane != null) {
                                trainPane.setVisible(true);
                        }
                }
                else if(e.getSource() == classifyRBtn) {
                        if(testPane != null) {
                                testPane.setVisible(true);
                        }
                }
                else if(e.getSource() == smoteChk) {
                        amtSmoteTextField.setEnabled(true);
                        nnSmoteTextField.setEnabled(true);
                }
                else if(e.getSource() == qcleanChk) {
                        nnQCleanTextField.setEnabled(true);
                }
                else if(e.getSource() == minmaxChk) {
                        minTextField.setEnabled(true);
                        maxTextField.setEnabled(true);
                }
                else if(e.getSource() == pcaChk) {
                        attrAmtTextField.setEnabled(true);
                }

        }

        if(e.getStateChange() == ItemEvent.DESELECTED) {
                if(e.getSource() == trainRBtn) {
                        trainPane.setVisible(false);
                }
                else if(e.getSource() == classifyRBtn) {
                        testPane.setVisible(false);
                }
                else if(e.getSource() == smoteChk) {
                        amtSmoteTextField.setEnabled(false);
                        nnSmoteTextField.setEnabled(false);
                }
                else if(e.getSource() == qcleanChk) {
```

```
                nnQCleanTextField.setEnabled(false);
            }
            else if(e.getSource() == minmaxChk) {
                minTextField.setEnabled(false);
                maxTextField.setEnabled(false);
            }
            else if(e.getSource() == pcaChk) {
                attrAmtTextField.setEnabled(false);
            }
        }
    }

    public void actionPerformed(ActionEvent event) {
        Object src = event.getSource();

        if(src == openDatasetBtn) {
            Util.openChooserWindow(fileTextField, false, true);
        }
        else if(src == dirBtn) {
            Util.openChooserWindow(svmModelDirectoryTextField, true, true);
        }
        else if(src == svmModelBtn) {
            Util.openChooserWindow(svmModelTextField, false, false);
        }
        else if(src == testSetBtn) {
            Util.openChooserWindow(testSetTextField, false, true);
        }
        else if(src == resetTrainBtn) {
            fileTextField.setText("");
            smoteChk.setSelected(false);
            smoteChk.setEnabled(false);
            amtSmoteTextField.setText("100");
            nnSmoteTextField.setText("5");
            qcleanChk.setSelected(false);
            qcleanChk.setEnabled(false);
            nnQCleanTextField.setText("13");
            minmaxChk.setSelected(false);
            minmaxChk.setEnabled(false);
            minTextField.setText("0");
            maxTextField.setText("1");
            zscoreChk.setSelected(false);
            zscoreChk.setEnabled(false);
            pcaChk.setSelected(false);
            attrAmtTextField.setText("8");
            costSVMTextField.setText("1");
            epsSVMTextField.setText("0.001");
            svmTypeCombo.setSelectedIndex(1);
            svmKernelTypeCombo.setSelectedIndex(2);
            svmModelDirectoryTextField.setText("");
            svmModelFileTextField.setText("");
        }

        else if(src == resetTestBtn) {
            svmModelTextField.setText("");
            testSetTextField.setText("");
        }

        else if(src == trainBtn) {
            if(fileTextField.getText().equals("")) {
                showWarningMsg("Choose an input dataset!");
            }
            else if(svmModelDirectoryTextField.getText().equals("")) {
                showWarningMsg("Choose directory where svm model will be
saved!");
            }
            else if(svmModelFileTextField.getText().equals("")) {
                showWarningMsg("Input filename where svm model will be
saved!");
            }
            else {
                try {
                    dataModel = new Dataset(fileTextField.getText(),
aFrame, "train");
                    ProcParams proc = new ProcParams();
                    boolean setProc = false;

                    if(smoteChk.isSelected()) {
                        int smote =
Integer.parseInt(amtSmoteTextField.getText());
                        int nn =
Integer.parseInt(nnSmoteTextField.getText());
                        if(smote<10 || smote<nn) {
                            showWarningMsg("Smote % is less than
10 or less than NN value. Changing to default.");
                            amtSmoteTextField.setText("100");
                            nnSmoteTextField.setText("5");
                        }

                        if(nn < 1) {
                            showWarningMsg("Smote nn is less
than 1. Changing to default.");
                            nnSmoteTextField.setText("5");
                        }


    proc.setSmoteParam(Integer.parseInt(amtSmoteTextField.getText()),

    Integer.parseInt(nnSmoteTextField.getText()));
                        setProc = true;
                    }

                    if(qcleanChk.isSelected()) {
                        int nn =
Integer.parseInt(nnQCleanTextField.getText());
                        if(nn < 1) {
                            showWarningMsg("QCleanNoise nn is
less than 1. Changing to default value.");
                            nnQCleanTextField.setText("13");
                        }

    proc.setQCleanParam(Integer.parseInt(nnQCleanTextField.getText()));
                        setProc = true;
                    }

                    if(minmaxChk.isSelected()) {

    proc.setMinMaxParam(Integer.parseInt(minTextField.getText()),
Integer.parseInt(maxTextField.getText()));
                        setProc = true;
                    }
```

```
                                        if(zscoreChk.isSelected()) {
                                                setProc = true;
                                        }

                                        if(pcaChk.isSelected()) {
                                                int attrCount =
Integer.parseInt(attrAmtTextField.getText());
                                                if(attrCount > 22 || attrCount < 1) {
                                                        showWarningMsg("Attribute count is
less than 1 or greater than the max amount of attributes. Changing to default value.");
                                                        attrAmtTextField.setText("8");
                                                }

        proc.setPCAParam(Integer.parseInt(attrAmtTextField.getText()));
                                                setProc = true;
                                        }

                                        if(setProc) {
                                                proc.setProc(smoteChk.isSelected(),
qcleanChk.isSelected(), minmaxChk.isSelected(),
                                                                zscoreChk.isSelected(),
pcaChk.isSelected());
                                        }


        proc.setSVMParam(Double.parseDouble(costSVMTextField.getText()),

        Double.parseDouble(epsSVMTextField.getText()),

        svmTypeCombo.getSelectedItem().toString(),

        svmKernelTypeCombo.getSelectedItem().toString());

                                        StringBuilder fileBuilder = new StringBuilder();

        fileBuilder.append(svmModelDirectoryTextField.getText().toString());
                                        fileBuilder.append("\\");

        fileBuilder.append(svmModelFileTextField.getText().toString());
                                        fileBuilder.append(".bin");

                                        proc.setSVMModelFile(fileBuilder.toString());

                                        JFrame trainFrame = createNewResultPanel("Training
Results");
                                        trainFrame.setSize(450, 230);
                                        TrainProcessor processor = new
TrainProcessor(dataModel, proc, trainFrame);
                                        processor.execute();
                                }
                                catch(NumberFormatException e) {
                                        JOptionPane.showMessageDialog(aFrame,
                                                        "Wrong input! View tutorial for
details.",
                                                        "Error",
                                                        JOptionPane.ERROR_MESSAGE);
                                }
                        }
                }
                else if(src == classifyBtn) {
                        if(svmModelTextField.getText().equals("")) {
```

```
                JOptionPane.showMessageDialog(aFrame,
                        "Choose an svm model file!",
                        "Warning",
                        JOptionPane.WARNING_MESSAGE);
            }
            else if(testSetTextField.getText().equals("")) {
                JOptionPane.showMessageDialog(aFrame,
                        "Choose an input test set!",
                        "Warning",
                        JOptionPane.WARNING_MESSAGE);
            }

            else {
                try {
                    dataModel = new Dataset(testSetTextField.getText(),
aFrame, "test");
                    JFrame testFrame =
createNewResultPanel("Classification Results");
                    testFrame.setSize(450, 500);
                    TestProcessor processor = new
TestProcessor(dataModel, svmModelTextField.getText().toString(), testFrame);
                    processor.execute();
                }
                catch(NumberFormatException e) {
                    JOptionPane.showMessageDialog(aFrame,
                            "Wrong input! View tutorial for
details.",
                            "Error",
                            JOptionPane.ERROR_MESSAGE);
                }
            }
        }
    }

    private void showWarningMsg(String msg) {
        JOptionPane.showMessageDialog(aFrame,
                msg,
                "Warning",
                JOptionPane.WARNING_MESSAGE);
    }

    private JFrame createNewResultPanel(String frameTitle) {
        JFrame resultPanelFrame = new JFrame(frameTitle);
        resultPanelFrame.setLocation(380, 100);
        resultPanelFrame.setResizable(false);
        resultPanelFrame.setVisible(true);
        ImageIcon img = new ImageIcon("./img/icon.jpg");
        resultPanelFrame.setIconImage(img.getImage());
        return resultPanelFrame;
    }
}
```

D. gui/mainUI.java

```
package gui;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JPanel;
import javax.swing.SwingUtilities;
import javax.swing.UIManager;

public class mainUI implements ActionListener {
	private JFrame aFrame;
	public final int width = 244, locX = 100;
	public final int height = 660, locY = 10;
	private JMenuItem tutorialMenu;
	private JMenuItem aboutMenu;
	JPanel mainPanel;
	GridBagLayout gridBag;
	GridBagConstraints gridBagCons;
	private UIManager.LookAndFeelInfo[] looks;

	public mainUI() {
		aFrame = new JFrame("PDDSS");
		aFrame.setSize(width, height);
		aFrame.setLocation(locX, locY);
		ImageIcon img = new ImageIcon("./img/icon.jpg");
		aFrame.setIconImage(img.getImage());
		createMenuBar();
		looks = UIManager.getInstalledLookAndFeels();
		changeLookAndFeel(3);

		mainPanel = new JPanel();
		gridBag = new GridBagLayout();
		gridBagCons = new GridBagConstraints();

		mainPanel.setLayout(gridBag);

		gridBagCons.weightx = 1000;
		gridBagCons.weighty = 1;
		gridBagCons.fill = GridBagConstraints.BOTH;

		JPanel sidebar = new JPanel(new BorderLayout());
		sidebar.add(new ControlPanel(aFrame), BorderLayout.WEST);
		sidebar.setBackground(Color.gray);

		aFrame.getContentPane().add(sidebar, BorderLayout.CENTER);
		aFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		aFrame.setResizable(false);
		aFrame.setVisible(true);

		new TutorialFrame();
	}

	@Override
	public void actionPerformed(ActionEvent e) {
		if(e.getSource() == tutorialMenu) {
			new TutorialFrame();
		}
		else if(e.getSource() == aboutMenu) {
```

```
                        new AboutFrame();
                }
        }

        private void createMenuBar() {
                JMenu helpMenu = new JMenu("Help");
                helpMenu.setMnemonic('H');
                tutorialMenu = new JMenuItem("Tutorial");
                tutorialMenu.addActionListener(this);

                aboutMenu = new JMenuItem("About");
                aboutMenu.addActionListener(this);

                helpMenu.add(tutorialMenu);
                helpMenu.add(aboutMenu);

                JMenuBar menuBar = new JMenuBar();
                menuBar.add(helpMenu);
                aFrame.setJMenuBar(menuBar);
        }

        private void changeLookAndFeel(int value) {
                try {
                        UIManager.setLookAndFeel(looks[value].getClassName());
                        SwingUtilities.updateComponentTreeUI(aFrame);
                } catch(Exception e) {
                        e.printStackTrace();
                }
        }

}
```

E. gui/TestResultPanel.java

```
package gui;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Component;
import java.awt.Font;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import javax.swing.BorderFactory;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.SwingConstants;
import javax.swing.table.DefaultTableCellRenderer;
import javax.swing.table.TableCellRenderer;

import proc.DataPrep;
import proc.Dataset;
import svm.SvmEvaluator;

@SuppressWarnings("serial")
public class TestResultPanel extends JPanel {
        private DataPrep dataPrep;
        private Dataset dataModel;
        private SvmEvaluator svmEval;
```

```
	public TestResultPanel(DataPrep dataPrep, Dataset dataModel, SvmEvaluator 
svmEval) {
		this.dataPrep = dataPrep;
		this.dataModel = dataModel;
		this.svmEval = svmEval;
		
		setBackground(Color.white);
		
		GridBagLayout gridbag = new GridBagLayout();
		GridBagConstraints c = new GridBagConstraints();
		setLayout(gridbag);
		
		c.weightx = 0;
		c.weighty = 0;
		c.fill = GridBagConstraints.BOTH;;
		c.gridx = 0;
		c.gridy = 0;
		c.gridwidth = 1;
		c.gridheight = 1;
		
		JPanel modelInfoPane = createModelInfoTable();
		modelInfoPane.setBackground(Color.WHITE);
		modelInfoPane.setBorder(BorderFactory.createTitledBorder("Model 
Information"));
		gridbag.setConstraints(modelInfoPane, c);
		add(modelInfoPane);
		
		c.weightx = 1000;
		c.weighty = 40;
		c.fill = GridBagConstraints.BOTH;
		c.gridx = 0;
		c.gridy = 1;
		c.gridwidth = 1;
		c.gridheight = 1;
		
		JPanel accPanel = new JPanel(new BorderLayout());
		accPanel.setBackground(Color.GREEN);
		double acc = 
(double)svmEval.getCorrectlyClassifiedCount()/(double)dataModel.getFullDataset().size()
*100;
		JLabel accuracyLabel = new JLabel("Accuracy: " + 
Integer.toString(svmEval.getCorrectlyClassifiedCount())
				+ " out of " + 
Integer.toString(dataModel.getFullDataset().size()) + " (" + String.format("%.2f",acc)
+")");
		accuracyLabel.setFont(new Font("Tahoma", Font.BOLD, 12));
		accuracyLabel.setHorizontalAlignment(SwingConstants.CENTER);
		accPanel.add(accuracyLabel);
		gridbag.setConstraints(accPanel, c);
		add(accPanel);
		
		c.weightx = 1000;
		c.weighty = 1000;
		c.fill = GridBagConstraints.BOTH;
		c.gridx = 0;
		c.gridy = 2;
		c.gridwidth = 1;
		c.gridheight = 1;
		
		JPanel resultsPane = createClassificationResultsTable();
```

```
                gridbag.setConstraints(resultsPane, c);
                add(resultsPane);
        }

        private JPanel createModelInfoTable() {

                JPanel panel = new JPanel();
                panel.setLayout( new BorderLayout() );

                String columnNames[] = { "Column 1", "Column 2" };
                StringBuilder sb = new StringBuilder();

                if(dataPrep.getAlgo()[0])
                {
                        sb.append("SMOTE, ");
                }

                if(dataPrep.getAlgo()[1])
                {
                        sb.append("QCleanNoise, ");
                }

                if(dataPrep.getProc()[0])
                {
                        sb.append("MinMax, ");
                }

                if(dataPrep.getProc()[1])
                {
                        sb.append("ZScore, ");
                }

                if(dataPrep.getProc()[2])
                {
                        sb.append("PCA");
                }

                String dataValues[][] =
                        {
                                { "Model filepath", dataPrep.getModelFilePath() },
                                { "Preprocessing Methods", sb.toString() },
                                { "Total Test Count",
Integer.toString(dataModel.getFullDataset().size()) },
                                { "PD+ Count",
Integer.toString(dataModel.getClassCount(1.0)) },
                                { "PD- Count",
Integer.toString(dataModel.getClassCount(0.0)) },
                                { "Kernel Type", dataPrep.getKerneltype()},
                                { "SVM Type", dataPrep.getSVMType()}
                        };

                JTable table = new JTable( dataValues, columnNames )
                {
                        public Component prepareRenderer(TableCellRenderer renderer, int
row, int column)
                        {
                                Component returnComp = super.prepareRenderer(renderer,
row, column);
                                Color alternateColor = Color.decode("#FFFFAD");
                                Color whiteColor = Color.decode("#DAECFF");
        if(!returnComp.getBackground().equals(getSelectionBackground())){
                                        Color bg = (row % 2 == 0 ? alternateColor : 
whiteColor);
                                        returnComp.setBackground(bg);
                                        bg = null;
                                }
                                if(this.isRowSelected(row))
                                        returnComp.setBackground(new Color(210, 250, 210));
                                this.setCellSelectionEnabled(false);

                                return returnComp;
                        }

                        @Override
                        public boolean isCellEditable(int i, int i1) {
                                return false;
                        }
                };

                table.setOpaque(false);
                table.setTableHeader(null);
                table.setShowGrid(false);
                table.getColumnModel().getColumn(0).setPreferredWidth(150);
                table.getColumnModel().getColumn(0).setMaxWidth(150);
                table.setBackground(Color.white);
                table.setRowSelectionAllowed(false);
                table.setFocusable(false);
                panel.add( table, BorderLayout.CENTER );
                return panel;
        }

        private JPanel createClassificationResultsTable() {
                JPanel panel = new JPanel();
                panel.setLayout( new BorderLayout() );

                String columnNames[] = {"Instance ID", "Actual Class", "Predicted Class", 
"Probability"};
                String dataValues[][] = new String[dataModel.getFullDataset().size()][4];

                for(int i = 0; i < dataValues.length; i++)
                {
                        dataValues[i][0] = dataModel.getIds().get(i);
                        dataValues[i][1] = svmEval.getActuals()[i];
                        dataValues[i][2] = svmEval.getPredictions()[i];
                        dataValues[i][3] = String.format("%.2f", 
svmEval.getProbabilities()[i]);
                }


                JTable table = new JTable( dataValues, columnNames )
                {
                        public Component prepareRenderer(TableCellRenderer renderer, int 
row, int column)
                        {
                                Component returnComp = super.prepareRenderer(renderer, 
row, column);
                                Color alternateColor = Color.decode("#FFD1D8");
                                Color whiteColor = Color.decode("#DBB8FF");

        if(!returnComp.getBackground().equals(getSelectionBackground())){
                                        Color bg = (row % 2 == 0 ? alternateColor : 
```

```
whiteColor);
                    returnComp.setBackground(bg);
                    bg = null;
                }
                if(this.isRowSelected(row))
                    returnComp.setBackground(new Color(210, 250, 210));
                this.setCellSelectionEnabled(false);

                return returnComp;
            }

            @Override
            public boolean isCellEditable(int i, int i1) {
                return false;
            }
        };

        table.setOpaque(false);
        table.setShowGrid(false);
        table.getColumnModel().getColumn(0).setPreferredWidth(150);
        table.getColumnModel().getColumn(0).setMaxWidth(150);
        table.setFocusable(false);

        ((DefaultTableCellRenderer)table.getTableHeader().getDefaultRenderer())
        .setHorizontalAlignment(JLabel.CENTER);
        table.getTableHeader().setEnabled(false);

        DefaultTableCellRenderer centerRenderer = new DefaultTableCellRenderer();
        centerRenderer.setHorizontalAlignment( JLabel.CENTER );
        table.getColumnModel().getColumn(0).setCellRenderer( centerRenderer );
        table.getColumnModel().getColumn(1).setCellRenderer( centerRenderer );
        table.getColumnModel().getColumn(2).setCellRenderer( centerRenderer );
        table.getColumnModel().getColumn(3).setCellRenderer( centerRenderer );

        JScrollPane scrollPane = new JScrollPane(table,
                JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
        scrollPane.setBorder(null);
        scrollPane.setOpaque(false);
        panel.add( scrollPane, BorderLayout.CENTER );

        panel.setBackground(Color.WHITE);
        panel.setBorder(BorderFactory.createTitledBorder("Classification
Results"));
        return panel;
    }
}
```

F. gui/TrainResultPanel.java

```
package gui;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Component;
import java.awt.Font;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;

import javax.swing.BorderFactory;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTable;
import javax.swing.SwingConstants;
import javax.swing.table.DefaultTableCellRenderer;
import javax.swing.table.TableCellRenderer;

import proc.Dataset;
import proc.ProcParams;

@SuppressWarnings("serial")
public class TrainResultPanel extends JPanel {

        private Dataset dataModel;
        private ProcParams proc;

        public TrainResultPanel(Dataset dataModel, ProcParams proc, double accuracy) {
                this.dataModel = dataModel;
                this.proc = proc;
                setBackground(Color.white);

                GridBagLayout gridbag = new GridBagLayout();
                GridBagConstraints c = new GridBagConstraints();
                setLayout(gridbag);

                c.weightx = 0;
                c.weighty = 0;
                c.fill = GridBagConstraints.BOTH;;
                c.gridx = 0;
                c.gridy = 0;
                c.gridwidth = 1;
                c.gridheight = 1;

                JPanel datasetInfoPane = createDatasetInfo();
                datasetInfoPane.setBackground(Color.WHITE);
                datasetInfoPane.setBorder(BorderFactory.createTitledBorder("Dataset
Information"));
                gridbag.setConstraints(datasetInfoPane, c);
                add(datasetInfoPane);

                c.weightx = 1000;
                c.weighty = 10;
                c.fill = GridBagConstraints.BOTH;
                c.gridx = 0;
                c.gridy = 1;
                c.gridwidth = 1;
                c.gridheight = 1;

                JPanel prepropInfoPane = createPrePropInfo();
                prepropInfoPane.setBackground(Color.WHITE);

        prepropInfoPane.setBorder(BorderFactory.createTitledBorder("Preprocessing
Information"));
                gridbag.setConstraints(prepropInfoPane, c);
                add(prepropInfoPane);

                c.weightx = 1000;
                c.weighty = 10;
                c.fill = GridBagConstraints.BOTH;
                c.gridx = 0;
                c.gridy = 3;
```

```
		c.gridwidth = 1;
		c.gridheight = 1;

		JPanel accPanel = new JPanel(new BorderLayout());
		accPanel.setBackground(Color.GREEN);
		JLabel accuracyLabel = new JLabel("Accuracy in 10-fold Cross Validation: 
" + String.format("%.2f", accuracy));
		accuracyLabel.setFont(new Font("Tahoma", Font.BOLD, 12));
		accuracyLabel.setHorizontalAlignment(SwingConstants.CENTER);
		accPanel.add(accuracyLabel);
		gridbag.setConstraints(accPanel, c);
		add(accPanel);
	}

	private JPanel createPrePropInfo() {
		JPanel prepropPanel = new JPanel();
		prepropPanel.setLayout(new BorderLayout());

		StringBuilder sb = new StringBuilder();

		if(proc.getProc()[0]) {
			sb.append("SMOTE, ");
		}

		if(proc.getProc()[1]) {
			sb.append("QCleanNoise, ");
		}

		if(proc.getProc()[2]) {
			sb.append("MinMax, ");
		}

		if(proc.getProc()[3]) {
			sb.append("ZScore, ");
		}

		if(proc.getProc()[4]) {
			sb.append("PCA");
		}

		String columnNames[] = { "Column 1", "Column 2" };

		String dataValues[][] = {
				{ "Prepropcessing Methods:", sb.toString()},
				{ "Total Noise Removed:", 
Integer.toString(dataModel.getTotalNoiseRemoved())},
				{ "Total Synthetic Samples Generated:", 
Integer.toString(dataModel.getTotalSyntheticSamplesGenerated())},
		};

		JTable table = new JTable(dataValues, columnNames) {
			public Component prepareRenderer(TableCellRenderer renderer, int 
row, int column) {
				Component returnComp = super.prepareRenderer(renderer, 
row, column);
				Color alternateColor = new Color(252,242,206);
				Color whiteColor = Color.decode("#DBB8FF");

	if(!returnComp.getBackground().equals(getSelectionBackground())){
					Color bg = (row % 2 == 0 ? alternateColor : 
whiteColor);
```

```
                    returnComp .setBackground(bg);
                    bg = null;
                }
                return returnComp;
            }

            @Override
            public boolean isCellEditable(int i, int i1) {
                return false;
            }
        };

        table.setOpaque(false);
        table.setTableHeader(null);
        table.setShowGrid(false);
        table.setFocusable(false);
        table.setBackground(Color.white);
        table.getColumnModel().getColumn(0).setPreferredWidth(200);
        table.getColumnModel().getColumn(0).setMaxWidth(200);
        table.setRowSelectionAllowed(false);
        prepropPanel.add(table, BorderLayout.CENTER);

        return prepropPanel;
    }

    private JPanel createDatasetInfo() {
        JPanel datasetPanel = new JPanel();
        datasetPanel.setLayout(new BorderLayout());

        String columnNames[] = { "Column 1", "Column 2", "Column 3" };

        String dataValues[][] = {
                { "", "Before Preprocessing:", "After Preprocessing:"},
                { "Total Attributes:",
Integer.toString(dataModel.getTotalOriginalAttributes()),
                    Integer.toString(dataModel.getAttributeCount())},
                    { "Total
Instances:",Integer.toString(dataModel.getTotalOriginalInstance()),

    Integer.toString(dataModel.getStatuses().size())},
                        { "Total PD+ Count:",
Integer.toString(dataModel.getTotalOriginalPDPos()),

    Integer.toString(dataModel.getClassCount(1.0))},
                            { "Total PD- Count:",
Integer.toString(dataModel.getTotalOriginalPDNeg()),

    Integer.toString(dataModel.getClassCount(0.0))}
        };

        JTable table = new JTable(dataValues, columnNames) {
            public Component prepareRenderer(TableCellRenderer renderer, int
row, int column) {
                Component returnComp = super.prepareRenderer(renderer,
row, column);
                Color alternateColor = Color.decode("#FFD1D8");
                Color whiteColor = Color.decode("#CCEBFF");

    if(!returnComp.getBackground().equals(getSelectionBackground()) &&
!(row==0&&column==0)) {
```

```
                    Color bg = (row % 2 == 0 ? alternateColor : 
whiteColor);
                    returnComp .setBackground(bg);
                    bg = null;
                }

                if(row == 0 && column == 0) {
                    returnComp.setBackground(Color.white);
                }
                return returnComp;
            }

            @Override
            public boolean isCellEditable(int i, int i1) {
                return false;
            }
        };

        table.setOpaque(false);
        table.setTableHeader(null);
        table.setShowGrid(false);
        table.setRowSelectionAllowed(false);
        table.setFocusable(false);
        table.getColumnModel().getColumn(0).setPreferredWidth(150);
        table.getColumnModel().getColumn(0).setMaxWidth(150);
        DefaultTableCellRenderer centerRenderer = new 
DefaultTableCellRenderer();
        centerRenderer.setHorizontalAlignment(JLabel.CENTER);
        table.getColumnModel().getColumn(0).setCellRenderer(centerRenderer);
        table.getColumnModel().getColumn(1).setCellRenderer(centerRenderer);
        table.getColumnModel().getColumn(2).setCellRenderer(centerRenderer);

        datasetPanel.add(table, BorderLayout.CENTER);
        return datasetPanel;
    }
}
```

G. gui/TutorialFrame.java

```
package gui;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Image;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.FocusEvent;
import java.awt.event.FocusListener;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;

import javax.imageio.ImageIO;
import javax.swing.BorderFactory;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JLayeredPane;
import javax.swing.JPanel;
import javax.swing.SwingConstants;
import javax.swing.UIManager;

@SuppressWarnings("serial")
public class TutorialFrame extends JFrame implements ActionListener, FocusListener {
        private final JFrame tutorialFrame;
        private final int width = 700, height = 500, locX = 380, locY = 100;
        private GridBagLayout gridBag = new GridBagLayout();
        private GridBagConstraints cons = new GridBagConstraints();
        private JPanel mainPanel;
        private JButton nextButton, prevButton;
        private int iter = 0;
        private ArrayList<JLayeredPane> tutorialPanes;
        private JLabel titleLabel, pageLabel;
        private ArrayList<String> titles;

        public TutorialFrame() {
                this.tutorialPanes = new ArrayList<JLayeredPane>();
                this.iter = 0;
                this.titles = new ArrayList<String>();
                tutorialFrame = new JFrame("PD DSS User Guide");
                tutorialFrame.addFocusListener(this);
                tutorialFrame.setSize(width, height);
                tutorialFrame.setLocation(locX, locY);
                tutorialFrame.setResizable(false);
                tutorialFrame.setVisible(true);
                ImageIcon img = new ImageIcon("./img/icon.jpg");
                tutorialFrame.setIconImage(img.getImage());

                mainPanel = new JPanel(new BorderLayout());
                mainPanel.setPreferredSize(new Dimension(700,500));
                mainPanel.setBackground(Color.DARK_GRAY);
                mainPanel.setLayout(gridBag);
                createMainPanel();

                tutorialFrame.getContentPane().add(mainPanel, BorderLayout.CENTER);
                tutorialFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
                tutorialFrame.setVisible(true);
                tutorialFrame.setResizable(false);
        }

        private void createMainPanel() {
                cons.weightx = 1;
                cons.weighty = 3;
                cons.gridx = 0;
                cons.gridy = 0;
                cons.gridwidth = 1;
                cons.gridheight = 1;
                cons.fill = GridBagConstraints.BOTH;

                JLayeredPane buttonLayeredPane = createButtonLayeredPane();
                buttonLayeredPane.setBounds(0, 0, 235, 500);
                buttonLayeredPane.setBackground(Color.DARK_GRAY);
                gridBag.setConstraints(buttonLayeredPane, cons);
                mainPanel.add(buttonLayeredPane);
```

```
		cons.weightx = 1;
		cons.weighty = 20;
		cons.fill = GridBagConstraints.BOTH;
		cons.gridx = 0;
		cons.gridy = 1;
		cons.gridwidth = 1;
		cons.gridheight = 1;

		createTutorialPanes();

		for(JLayeredPane i : tutorialPanes) {
			i.setBounds(0, 0, 235, 500);
			i.setBackground(Color.DARK_GRAY);
			i.setVisible(false);
			gridBag.setConstraints(i, cons);
			mainPanel.add(i);
		}

		cons.weightx = 1;
		cons.weighty = 3;
		cons.gridx = 0;
		cons.gridy = 3;
		cons.gridwidth = 1;
		cons.gridheight = 1;
		cons.fill = GridBagConstraints.BOTH;

		JLayeredPane pagePane = new JLayeredPane();
		pagePane.setBounds(0, 0, 235, 500);
		pagePane.setBackground(Color.DARK_GRAY);

		pageLabel = new JLabel("");
		pageLabel.setHorizontalAlignment(SwingConstants.CENTER);
		pageLabel.setBounds(115, 10, 500, 40);
		pageLabel.setFont(new Font("Tahoma", Font.BOLD, 14));
		pageLabel.setForeground(Color.LIGHT_GRAY);
		pagePane.add(pageLabel);

		gridBag.setConstraints(pagePane, cons);
		mainPanel.add(pagePane);

		showPane(iter);
	}

	private void createTutorialPanes() {
		tutorialPanes.add(createFirstPane());
		tutorialPanes.add(createSecondPane());
		tutorialPanes.add(createThirdPane());
		tutorialPanes.add(createFourthPane());
		tutorialPanes.add(createFifthPane());
		tutorialPanes.add(createSixthPane());
		tutorialPanes.add(createSeventhPane());
	}

	private JLayeredPane createSeventhPane() {
		JLayeredPane seventhPane = new JLayeredPane();
		titles.add("CLASSIFICATION RESULTS");

		BufferedImage img1 = null;
		try {
			img1 = ImageIO.read(new
File("./img/ClassificationResultsOnly.png"));
```

```
		} catch(IOException e) {
			e.printStackTrace();
		}

		Image dimg1 = img1.getScaledInstance(370, 360, Image.SCALE_SMOOTH);

		JLabel imgLabel = new JLabel();
		imgLabel.setIcon(new ImageIcon(dimg1));
		imgLabel.setBounds(30, -40, 400, 450);

		JLabel seventhPaneLabel1 = new JLabel("<html><align=justify>This is the 
testing results. In here you will "
				+ "find 3 sections: Model Information, Accuracy and 
Classification Results.</html>");
		seventhPaneLabel1.setBounds(430, -90, 250, 300);
		seventhPaneLabel1.setForeground(Color.white);
		seventhPaneLabel1.setFont(new Font("Tahoma", Font.BOLD, 12));

		JLabel seventhPaneLabel2 = new JLabel("<html><align=justify>Model 
Information section contains "
				+ "the model file path, the preprocessing methods that are 
used by the model, total test count - PD+ and PD- "
				+ "count. As well as the SVM and Kernel Type.</html>");
		seventhPaneLabel2.setBounds(430, -10, 250, 300);
		seventhPaneLabel2.setForeground(Color.white);
		seventhPaneLabel2.setFont(new Font("Tahoma", Font.BOLD, 12));

		JLabel seventhPaneLabel3 = new JLabel("<html><align=justify>Accuracy 
section shows how many instances were correctly "
				+ "classified by the system out of the total test 
count.</html>");
		seventhPaneLabel3.setBounds(430, 70, 250, 300);
		seventhPaneLabel3.setForeground(Color.white);
		seventhPaneLabel3.setFont(new Font("Tahoma", Font.BOLD, 12));


		JLabel seventhPaneLabel4 = new 
JLabel("<html><align=justify>Classification Results show the instance ID of the 
subject, "
				+ "the actual class of the subject and the result of the 
evaluation with its probability.</html>");
		seventhPaneLabel4.setBounds(430, 150, 250, 300);
		seventhPaneLabel4.setForeground(Color.white);
		seventhPaneLabel4.setFont(new Font("Tahoma", Font.BOLD, 12));

		seventhPane.add(imgLabel);
		seventhPane.add(seventhPaneLabel1);
		seventhPane.add(seventhPaneLabel2);
		seventhPane.add(seventhPaneLabel3);
		seventhPane.add(seventhPaneLabel4);
		return seventhPane;
	}

	private JLayeredPane createSixthPane() {
		JLayeredPane sixthPane = new JLayeredPane();
		titles.add("CONTROL PANEL - CLASSIFICATION");

		JLabel imgLabel = new JLabel();
		imgLabel.setIcon(new ImageIcon("./img/classifyPanel.png"));
		imgLabel.setBounds(380, -20, 350, 350);
```

```
                JLabel sixthPaneLabel1 = new JLabel("<html><align=justify>This panel is
where you put the model file in .bin extension "
                              + "that is generated from training the system. You must
also choose the test set in .csv extension that will "
                              + "be classified by the system. The format of the test set
is also the same with that of the training set. You "
                              + "can consult UCI Parkinson's Disease repository for
further details.</html>");
                sixthPaneLabel1.setBounds(50, -10, 300, 300);
                sixthPaneLabel1.setForeground(Color.white);
                sixthPaneLabel1.setFont(new Font("Tahoma", Font.BOLD, 12));

                sixthPane.add(imgLabel);
                sixthPane.add(sixthPaneLabel1);
                return sixthPane;
        }

        private JLayeredPane createFifthPane() {
                JLayeredPane fifthPane = new JLayeredPane();
                titles.add("TRAINING RESULTS");

                JLabel imgLabel = new JLabel();
                imgLabel.setIcon(new ImageIcon("./img/TrainingResultsOnly.png"));
                imgLabel.setBounds(120, -40, 500, 350);

                JLabel fifthPaneLabel1 = new JLabel("<html><align=justify>The training
results panel has 3 sections: Dataset "
                              + "Information, Preprocessing Information and Accuracy.
The dataset information shows general info "
                              + "about the data before and after preprocessing.
Preprocessing shows the summary of what techniques "
                              + "have been applied to the dataset as well as how many
synthetic samples are generated or how many "
                              + "class noise was removed. Finally, the accuracy section
shows the % accuracy of the system that is "
                              + "validated using 10-fold cross validation.</html>");
                fifthPaneLabel1.setBounds(60, 30, 600, 600);
                fifthPaneLabel1.setForeground(Color.white);
                fifthPaneLabel1.setFont(new Font("Tahoma", Font.BOLD, 12));

                fifthPane.add(imgLabel);
                fifthPane.add(fifthPaneLabel1);
                return fifthPane;
        }

        private JLayeredPane createFourthPane() {
                JLayeredPane fourthPane = new JLayeredPane();

                titles.add("SVM PARAMETERS");

                JLabel imgLabel = new JLabel();
                imgLabel.setIcon(new ImageIcon("./img/svm.png"));
                imgLabel.setBounds(30, -20, 230, 350);

                JLabel fourthPaneLabel1 = new JLabel("<html><align=justify>Support Vector
Machine is a "
                              + "discriminative classifier used for classification and
regressionThe goal of "
                              + "the SVM algorithm is to find the hyperplane that
separates the classes and "
                              + "gives the largest minimum distance to the training
samples.</html>");
		fourthPaneLabel1.setBounds(270, -90, 400, 300);
		fourthPaneLabel1.setForeground(Color.white);
		fourthPaneLabel1.setFont(new Font("Tahoma", Font.BOLD, 12));

		JLabel fourthPaneLabel2 = new JLabel("<html><align=justify>The cost (C) 
parameter tells the "
				+ "SVM optimization how much you want to avoid 
misclassifying each training example. "
				+ "For large values of C, the optimization will choose a 
smaller-margin hyperplane "
				+ "if that hyperplane does a better job of getting all the 
training points classified correctly.</html>");
		fourthPaneLabel2.setBounds(270, -10, 400, 300);
		fourthPaneLabel2.setForeground(Color.white);
		fourthPaneLabel2.setFont(new Font("Tahoma", Font.BOLD, 12));

		JLabel fourthPaneLabel3 = new JLabel("<html><align=justify>The value of 
epsilon (eps) determines the level "
				+ "of accuracy of the approximated function. It relies 
entirely on the target values in the training "
				+ "set. If epsilon is larger than the range of the target 
values we cannot expect a good result. If "
				+ "epsilon is zero, we can expect overfitting.</html>");
		fourthPaneLabel3.setBounds(270, 80, 400, 300);
		fourthPaneLabel3.setForeground(Color.white);
		fourthPaneLabel3.setFont(new Font("Tahoma", Font.BOLD, 12));

		JLabel fourthPaneLabel4 = new JLabel("<html><align=justify>The range of 
C-SVC is from zero to infinity but nu-SVC "
				+ "is always between [0,1]</html>");
		fourthPaneLabel4.setBounds(270, 140, 400, 300);
		fourthPaneLabel4.setForeground(Color.white);
		fourthPaneLabel4.setFont(new Font("Tahoma", Font.BOLD, 12));

		fourthPane.add(imgLabel);
		fourthPane.add(fourthPaneLabel1);
		fourthPane.add(fourthPaneLabel2);
		fourthPane.add(fourthPaneLabel3);
		fourthPane.add(fourthPaneLabel4);
		return fourthPane;
	}

	private JLayeredPane createThirdPane() {
		JLayeredPane thirdPane = new JLayeredPane();
		titles.add("PREPROCESSING PARAMETERS");

		JLabel imgLabel = new JLabel();
		imgLabel.setIcon(new ImageIcon("./img/dataPreprop.png"));
		imgLabel.setBounds(450, 0, 210, 350);

		JLabel thirdPaneLabel1 = new JLabel("<html><align=justify>Synthetic 
Minority Oversampling Technique or SMOTE "
				+ "generates synthetic samples from the minority class by 
introducing samples along a line segment joining "
				+ "the k nearest neighbors of a minority instance. % smote 
refers to how much of the minority samples "
				+ "must be introduced while no. of NN refers to k-NN.
</html>");
		thirdPaneLabel1.setBounds(20, -100, 420, 300);
		thirdPaneLabel1.setForeground(Color.white);
```

```
                thirdPaneLabel1.setFont(new Font("Tahoma", Font.BOLD, 12));

                JLabel thirdPaneLabel2 = new JLabel("<html><align=justify>QCleanNoise is
a data cleaning technique that aims to "
                                + "remove class noise by calculating the quality measure
of each instance. A noisy instance will have a "
                                + "negative value. But a negative value may also be
achieved when an instance is near the boundaries of the "
                                + "classes. This is resolved through comparing the K-NN of
the instance.</html>");
                thirdPaneLabel2.setBounds(20, 0, 420, 300);
                thirdPaneLabel2.setForeground(Color.white);
                thirdPaneLabel2.setFont(new Font("Tahoma", Font.BOLD, 12));

                JLabel thirdPaneLabel3 = new JLabel("<html><align=justify>MinMax
Normalization is a type of data transformation "
                                + "that scales data to a new range given by the minimum
and maximum parameters. Z-Score Normalization "
                                + "also scales data but with the use of mean and standard
deviation.</html>");
                thirdPaneLabel3.setBounds(20, 90, 420, 300);
                thirdPaneLabel3.setForeground(Color.white);
                thirdPaneLabel3.setFont(new Font("Tahoma", Font.BOLD, 12));

                JLabel thirdPaneLabel4 = new JLabel("<html><align=justify>Principal
Components Analysis is a data reduction technique "
                                + "that reduces dimension of data through eigenvalues and
eigenvectors. Taking only the principal "
                                + "components, the technique reduces computational cost
without compromising data accuracy.</html>");
                thirdPaneLabel4.setBounds(20, 170, 420, 300);
                thirdPaneLabel4.setForeground(Color.white);
                thirdPaneLabel4.setFont(new Font("Tahoma", Font.BOLD, 12));

                thirdPane.add(imgLabel);
                thirdPane.add(thirdPaneLabel1);
                thirdPane.add(thirdPaneLabel2);
                thirdPane.add(thirdPaneLabel3);
                thirdPane.add(thirdPaneLabel4);
                return thirdPane;
        }

        private JLayeredPane createSecondPane() {
                JLayeredPane secondPane = new JLayeredPane();
                titles.add("CONTROL PANEL - TRAINING");

                BufferedImage img1 = null, img2 = null;
                try {
                        img1 = ImageIO.read(new File("./img/trainingPanel1.png"));
                        img2 = ImageIO.read(new File("./img/trainingPanel2.png"));
                } catch(IOException e) {
                        e.printStackTrace();
                }

                Image dimg1 = img1.getScaledInstance(210, 350, Image.SCALE_SMOOTH);
                Image dimg2 = img2.getScaledInstance(210, 280, Image.SCALE_SMOOTH);

                JLabel imgLabel = new JLabel();
                imgLabel.setIcon(new ImageIcon(dimg1));
                imgLabel.setBounds(20, 0, 210, 350);
                JLabel imgLabel2 = new JLabel();
                imgLabel2.setIcon(new ImageIcon(dimg2));
                imgLabel2.setBounds(250, 90, 210, 300);

                JLabel secondPaneLabel1 = new JLabel("<html><align=justify>This is the
training view of the control panel. "
                                + "In here, you can input the required fields before
hitting the TRAIN button. "
                                + "To start, you must input a .csv file that will serve as
your training set. "
                                + "You can refer to UCI Parkinson's Disease Dataset for
more details. </html>");
                secondPaneLabel1.setBounds(250, -100, 400, 300);
                secondPaneLabel1.setForeground(Color.white);
                secondPaneLabel1.setFont(new Font("Tahoma", Font.BOLD, 12));

                JLabel secondPaneLabel2 = new JLabel("<html><align=justify>Notice that
there are default values already. "
                                + "These are just recommendations since the dataset has
been processed through the default values already. "
                                + "However, you can still input your desired values. The
next parts will talk about the valid values for each section.</html>");
                secondPaneLabel2.setBounds(480, 60, 210, 250);
                secondPaneLabel2.setForeground(Color.white);
                secondPaneLabel2.setFont(new Font("Tahoma", Font.BOLD, 12));

                secondPane.add(imgLabel);
                secondPane.add(imgLabel2);
                secondPane.add(secondPaneLabel1);
                secondPane.add(secondPaneLabel2);
                return secondPane;
        }

        private JLayeredPane createFirstPane() {
                JLayeredPane firstPane = new JLayeredPane();
                titles.add("PARKINSON'S DISEASE DECISION SUPPORT SYSTEM");

                JLabel imgLabel = new JLabel();
                imgLabel.setIcon(new ImageIcon("./img/brain.jpg"));
                imgLabel.setBounds(50, 10, 350, 350);

                JLabel generalInfoLabel = new JLabel("<html><align=justify>Welcome to PD
DSS - "
                                + "a decision support system that is used for classifying
PD+ and PD- subjects. "
                                + "This tool applies preprocessing algorithms and data
reduction techniques to "
                                + "address problems with imbalanced datasets, class noise
and data outliers. "
                                + "Support Vector Machines or SVM is the classifier used
to generate "
                                + "learning models for further predictions and
classifications. </html>");
                generalInfoLabel.setBounds(430, 10, 250, 250);
                generalInfoLabel.setForeground(Color.white);
                generalInfoLabel.setFont(new Font("Tahoma", Font.BOLD, 12));

                JLabel additionalLabel = new JLabel("<html><align=justify>This tutorial
will guide you through "
                                + "operating the system, choosing parameters and
understanding the results. Hit next button to begin.</html>");
```

```
            additionalLabel.setBounds(430, 160, 250, 250);
            additionalLabel.setForeground(Color.white);
            additionalLabel.setFont(new Font("Tahoma", Font.BOLD, 12));

            firstPane.add(generalInfoLabel);
            firstPane.add(additionalLabel);
            firstPane.add(imgLabel);
            return firstPane;
    }

    private JLayeredPane createButtonLayeredPane() {
            JLayeredPane buttonOptionPane = new JLayeredPane();

            UIManager.put("Button.select", Color.DARK_GRAY);
            prevButton = new JButton("Previous");
            prevButton.setBounds(10, 10, 75, 30);
            prevButton.setForeground(Color.DARK_GRAY);
            prevButton.setBackground(Color.yellow);
            prevButton.setFont(new Font("Tahoma", Font.BOLD, 14));
            prevButton.setOpaque(true);
            prevButton.addActionListener(this);
            prevButton.setBorder(BorderFactory.createLineBorder(Color.DARK_GRAY));
            prevButton.setFocusable(false);
            prevButton.setVisible(false);

            nextButton = new JButton(" Next ");

            nextButton.setFont(new Font("Tahoma", Font.BOLD, 14));
            nextButton.setBounds(640, 10, 50, 30);
            nextButton.setOpaque(true);
            nextButton.setForeground(Color.DARK_GRAY);
            nextButton.setBackground(Color.yellow);
            nextButton.addActionListener(this);
            nextButton.setBorder(BorderFactory.createLineBorder(Color.DARK_GRAY));
            nextButton.setFocusable(false);

            titleLabel = new JLabel();
            titleLabel.setHorizontalAlignment(SwingConstants.CENTER);
            titleLabel.setForeground(Color.WHITE);
            titleLabel.setFont(new Font("Tahoma", Font.BOLD, 14));
            titleLabel.setBackground(Color.DARK_GRAY);
            titleLabel.setBounds(115, 10, 500, 40);

            buttonOptionPane.add(titleLabel);
            buttonOptionPane.add(prevButton);
            buttonOptionPane.add(nextButton);

            return buttonOptionPane;
    }

    @Override
    public void actionPerformed(ActionEvent event) {
            Object src = event.getSource();

            if(src == nextButton && iter!=tutorialPanes.size()-1) {
                    iter++;
                    showPane(iter);
            }

            else if(src == prevButton && iter!=0) {
                    iter--;
				showPane(iter);
			}
		}

		private void showPane(int iterCount) {
			for(int i = 0; i < tutorialPanes.size(); i++) {
				if(i == iterCount) {
					tutorialPanes.get(i).setVisible(true);
				}
				else {
					tutorialPanes.get(i).setVisible(false);
				}
			}

			pageLabel.setText(Integer.toString(iter + 1) + " of 7");
			titleLabel.setText(titles.get(iterCount));
			repaintButtons();
		}

		private void repaintButtons() {
			if(iter == 0) {
				prevButton.setVisible(false);
			}
			else {
				prevButton.setVisible(true);
			}

			if(iter == tutorialPanes.size()-1) {
				nextButton.setVisible(false);
			}
			else {
				nextButton.setVisible(true);
			}
		}

		@Override
		public void focusGained(FocusEvent e) {}

		@Override
		public void focusLost(FocusEvent e) {
			if(e.getSource() == tutorialFrame) {
				tutorialFrame.dispose();
			}
		}
}
```

H. dataprepop/MinMaxNormalization.java

```
package dataprepop;

import java.util.ArrayList;

public class MinMaxNormalization {
		private static ArrayList<double[]> preProcessedData;
		private static double thisMin, thisMax;
		private static double[] globalMin, globalMax;

		public MinMaxNormalization(ArrayList<double[]> preProcessedData, double min,
double max) {
			MinMaxNormalization.preProcessedData = preProcessedData;
```

```
                MinMaxNormalization.thisMin = min;
                MinMaxNormalization.thisMax = max;
        }

        public ArrayList<double[]> processTrainingData() {
                globalMin = new double[preProcessedData.get(0).length];
                globalMax = new double[preProcessedData.get(0).length];
                performMethod();
                return preProcessedData;
        }

        public ArrayList<double[]> processTestingData(double[] min, double[] max) {
                transformData(min, max);
                return preProcessedData;
        }

        private static void transformData(double[] gMin, double[] gMax) {
                for (int i = 0; i < preProcessedData.get(0).length; i++) {
                        for (int k = 0; k < preProcessedData.size(); k++) {
                                double a = preProcessedData.get(k)[i];
                                preProcessedData.get(k)[i] = (((a - gMin[i])/(gMax[i] - 
gMin[i])) * (thisMin - thisMax)) + thisMax;
                        }
                }
        }

        public static double[] getGlobalMin() {
                return globalMin;
        }

        public static double[] getGlobalMax() {
                return globalMax;
        }

        private static void performMethod() {
                for (int i = 0; i < preProcessedData.get(0).length; i++) {
                        double min = preProcessedData.get(0)[i];
                        double max = preProcessedData.get(0)[i];
                        for (int j = 0; j < preProcessedData.size(); j++) {
                                double a = preProcessedData.get(j)[i];
                                min = min > a ? a : min;
                                max = max < a ? a : max;
                        }

                        globalMin[i] = min;
                        globalMax[i] = max;
                }

                transformData(globalMin, globalMax);
        }
}
```

I. dataprepop/PCA.java

```
package datapreprop;

import java.util.ArrayList;
import Jama.*;

public class PCA {
	private static ArrayList<double[]> preProcessedData;
	private static ArrayList<Double> statuses;
	private static double[] globalMeans;
	private static double[][] dataAdjusted, eigenvector, reducedData, finalData,
globalFeatureVector;
	private static int pcCount, totalFeatures;

	public PCA(ArrayList<double[]> preprop, ArrayList<Double> stat, int
totalfeatures, int pc) {
		preProcessedData = preprop;
		statuses = stat;
		totalFeatures = totalfeatures;
		pcCount = pc;
		dataAdjusted = new double[preProcessedData.size()][totalFeatures];
		reducedData = new double[preProcessedData.size()][pcCount + 1];
	}

	public double[][] processTrainingData() {
		performMethod();
		return reducedData;
	}

	public double[][] processTestingData(double[] means, double[][] eigenvector) {
		adjustData(means);
		double [][] featureVector = eigenvector;
		generateFinalData(featureVector);
		return reducedData;
	}

	public static double[] getGlobalMeans() {
		return globalMeans;
	}

	public static double[][] getGLobalFeatureVector() {
		return globalFeatureVector;
	}

	private static void performMethod() {
		double[] means = new double[totalFeatures];
		means = getMeanPerFeature();
		globalMeans = means;
		adjustData(means);
		double [][] cov = generateCovarianceMatrix();
		double [][] featureVector = generateFeatureVector(cov);
		generateFinalData(featureVector);
	}

	private static void generateFinalData(double[][] featureVector) {
		Matrix m = new Matrix(featureVector);
		Matrix n = new Matrix(dataAdjusted);
		finalData = n.times(m).getArrayCopy();

		for(int k = 0; k < finalData.length; k++) {
			reducedData[k][0] = statuses.get(k);
			for(int j=1; j<=finalData[0].length; j++) {
				reducedData[k][j] = finalData[k][j-1];
			}
		}
	}

	private static double [][] generateFeatureVector(double[][] cov) {
        Matrix m = new Matrix(cov);
        EigenvalueDecomposition a = m.eig();
        eigenvector = a.getV().getArrayCopy();
        double [][] featureVector = new double[eigenvector.length][pcCount];

        for(int i = 0; i < eigenvector.length; i++) {
            int counter = pcCount;
            for(int j = 0; j < pcCount; j++) {
                featureVector[i][j] = eigenvector[i][eigenvector.length - 
counter];
                counter--;
            }
        }

        globalFeatureVector = featureVector;
        return featureVector;
    }

    private static double[][] generateCovarianceMatrix() {
        double[][] cov = new double[totalFeatures][totalFeatures];

        for(int i=0; i < totalFeatures; i++) {
            for(int j=0; j < totalFeatures; j++) {
                if(cov[j][i] == 0.0) {
                    cov[i][j] = variance(i,j);
                }
                else {
                    cov[i][j] = cov[j][i];
                }
            }
        }

        return cov;
    }

    private static double variance(int i, int j) {
        double var = 0;
        int totalInstance = preProcessedData.size();
        for(int k=0; k < totalInstance; k++) {
            var += preProcessedData.get(k)[i] * preProcessedData.get(k)[j];
        }

        return var/(totalInstance-1);
    }

    private static void adjustData(double[] means) {
        int totalInstance = preProcessedData.size();

        for(int i=0; i < totalFeatures; i++) {
            for(int j = 0; j < totalInstance; j++) {
                double a = preProcessedData.get(j)[i];
                preProcessedData.get(j)[i] = a - means[i];
                dataAdjusted[j][i] = a - means[i];
            }
        }
    }

    private static double[] getMeanPerFeature() {
        int totalInstance = preProcessedData.size();
        double[] means = new double[totalFeatures];
        for(int i=0; i < totalFeatures; i++)
        {
            means[i] = 0.0;

            for(int j = 0; j < totalInstance; j++)
            {
                means[i] += preProcessedData.get(j)[i];

            }

            means[i] /= totalInstance;
        }
        return means;
    }
}
```

J. dataprepop/QCleanNoise.java

```
package datapreprop;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import proc.Dataset;
import util.Util;

public class QCleanNoise {
    private ArrayList<double[]> preProcessedData;
    private double[] centroidNPD;
    private double[] centroidPD;
    private Dataset dataModel;
    private ArrayList <double[]> candNoise;
    private ArrayList <Double> candNoiseStat;
    private int nnCount, totalNoiseRemoved;

    private void calculateCentroids() {
        int numOfAttr = preProcessedData.get(0).length;
        int PD = 0, NPD = 0;
        centroidPD = new double[numOfAttr];
        centroidNPD = new double[numOfAttr];

        for(int i = 0; i < dataModel.statuses.size(); i++) {
            if(dataModel.statuses.get(i) == 1.0) {
                for(int j=0; j<numOfAttr; j++) {
                    centroidPD[j]+=preProcessedData.get(i)[j];
                }
                PD++;
            }
            else {
                for(int j=0; j<numOfAttr; j++) {
                    centroidNPD[j]+=preProcessedData.get(i)[j];
                }
                NPD++;
            }
        }

        for(int k=0; k<numOfAttr; k++) {
            centroidNPD[k] /= NPD;
            centroidPD[k] /= PD;
        }
    }

    private static double getDist(double[] inst, double[] centr) {
        double dist = 0;
        for(int i=0; i < inst.length; i++) {
            dist += Math.pow(inst[i] - centr[i], 2);
        }

        return Math.pow(dist, .5);
    }

    private double getQualityMeasure(double[] inst, boolean isPD) {
        double quaMes = 0;
        double distD = isPD ? getDist(inst, centroidPD) : getDist(inst,
centroidNPD);
        double distR = isPD ? getDist(inst, centroidNPD) : getDist(inst,
centroidPD);

        quaMes = (distR - distD)/Math.max(distR, distD);

        return quaMes;
    }

    @SuppressWarnings("rawtypes")
    private void removeNoise() {
        ArrayList<double[]> candNoideTemp = new ArrayList<double[]>();
        ArrayList<Integer> CNtempSt = new ArrayList<Integer>();
        for(int i=0; i < candNoise.size(); i++) {
            Map<Integer, Double> distances = new HashMap<Integer, Double>();
            Map sortedDistances;
            ArrayList<Integer> nnArray = new ArrayList<Integer>();
            int count = 0;

            for(int j = 0; j < preProcessedData.size(); j++) {
                if(!(candNoise.get(i).equals(preProcessedData.get(j)))) {
                    double distance = 0;
                    for(int k = 0; k < candNoise.get(0).length; k++) {
                        double iVal = candNoise.get(i)[k];
                        double jVal = preProcessedData.get(j)[k];
                        distance += Math.pow(iVal - jVal, 2);
                    }
                    distance = Math.pow(distance, .5);
                    distances.put(j, distance);
                }
            }

            sortedDistances = Util.sortByValue(distances);

            int k = 0;
            @SuppressWarnings("unchecked")
            List list = new LinkedList(sortedDistances.entrySet());
            for(Iterator it = list.iterator(); it.hasNext() && k < nnCount;)
{
                Map.Entry entry = (Map.Entry)it.next();
                nnArray.add((int)entry.getKey());
                k++;
            }

            for(int a = 0; a < nnArray.size(); a++) {
```

```
                                double x = dataModel.statuses.get(nnArray.get(a));
                                double y = candNoiseStat.get(i);
                                if(x == y) {
                                        count++;
                                }
                        }

                        int acceptanceVal = (nnCount+1)/2;
                        if(count < acceptanceVal) {
                                candNoideTemp.add(candNoise.get(i));
                                int q = preProcessedData.indexOf(candNoise.get(i));
                                CNtempSt.add(q);
                        }
                }

                preProcessedData.removeAll(candNoideTemp);
                totalNoiseRemoved = CNtempSt.size();

                for(int i = CNtempSt.size()-1 ; i >= 0; i--) {
                        int a = CNtempSt.get(i);
                        dataModel.statuses.remove(a);
                }
        }

        private void performMethod() {
                calculateCentroids();

                for(int i=0; i<preProcessedData.size(); i++) {
                        boolean isPD;
                        isPD = (dataModel.statuses.get(i) == 1.0) ? true : false;
                        if(getQualityMeasure(preProcessedData.get(i), isPD) < 0) {
                                candNoise.add(preProcessedData.get(i));
                                candNoiseStat.add(dataModel.statuses.get(i));
                        }
                }

                removeNoise();
        }

        public ArrayList<double[]> preProcessData(Dataset dataModel, ArrayList<double[]> 
preProcessedData, int nnCount) {
                this.preProcessedData = preProcessedData;
                this.nnCount = nnCount;
                this.candNoise = new ArrayList<double[]>();
                this.candNoiseStat = new ArrayList<Double>();
                this.dataModel = dataModel;
                this.totalNoiseRemoved = 0;
                this.centroidNPD = new double[22];
                this.centroidPD = new double[22];
                performMethod();
                this.dataModel.setTotalNoiseRemoved(totalNoiseRemoved);
                return preProcessedData;
        }
}
```

K. dataprepop/Smote.java

```
package dataprepop;

import java.util.Iterator;
import java.util.List;
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.Map;
import java.util.Random;
import util.Util;
import proc.Dataset;

public class Smote {
        private static Dataset dataModel;
        private static ArrayList <double[]> fullDataset;
        private static int numberOfAttributes, nn, amountOfSmote, syntheticGenerated =
0;
        private static double minorityClassCount;
        private static ArrayList<double[]> minorityClassArray;
        @SuppressWarnings("rawtypes")
        private static Map sortedDistances;
        private static ArrayList<Integer> nnArray;
        private static double minorityClass;


        private static void performMethod() {
                generateMinorityClassArray();
                if(amountOfSmote < 100) {
                        Collections.shuffle(minorityClassArray);
                        minorityClassCount = (amountOfSmote/100.0)*minorityClassCount;
                        amountOfSmote = 100;
                }

                amountOfSmote = (int)(amountOfSmote/100);
                numberOfAttributes = fullDataset.get(0).length;
                doSmote();
        }

        private static void doSmote() {
                for(int i = 0; i < minorityClassCount; i++)
                {
                        Map<Integer, Double> distances = new HashMap<Integer, Double>();
                        for(int j = 0; j < minorityClassCount; j++)
                        {
                                int instanceJ = j;
                                if(i != j)
                                {
                                        double distance = 0;
                                        for(int k = 0; k < numberOfAttributes-1; k++)
                                        {
                                                double iVal = minorityClassArray.get(i)[k];
                                                double jVal = minorityClassArray.get(j)[k];
                                                distance += Math.pow( iVal - jVal, 2 );
                                        }
                                        distance = Math.pow( distance, .5 );
                                        distances.put(instanceJ, distance);
                                }
                        }

                        sortedDistances = Util.sortByValue(distances);
                        populateNNArray(nn);
                        populateSyntheticMinorityInstances(i);
                }
        }

        private static void populateSyntheticMinorityInstances(int instance) {
                int N = amountOfSmote;
                double[] syntheticMinorityInstances = new double[numberOfAttributes];
                while(N > 0) {
                        Random rand = new Random();
                        int randNN = rand.nextInt(nn);
                        for(int j = 0; j < numberOfAttributes; j++) {
                                double dif =
minorityClassArray.get(nnArray.get(randNN))[j] - minorityClassArray.get(instance)[j];
                                double gap = rand.nextDouble();
                                syntheticMinorityInstances[j] =
minorityClassArray.get(instance)[j] + gap * dif;
                        }
                        fullDataset.add(syntheticMinorityInstances);
                        syntheticGenerated++;
                        dataModel.statuses.add(minorityClass);
                        N--;
                }

        }

        private static void populateNNArray(int nn) {
                int j = 0;
                @SuppressWarnings({ "rawtypes", "unchecked" })
                List list = new LinkedList(sortedDistances.entrySet());
                for (@SuppressWarnings("rawtypes")
                Iterator it = list.iterator(); it.hasNext() && j < nn;) {
                        @SuppressWarnings("rawtypes")
                        Map.Entry entry = (Map.Entry)it.next();
                        nnArray.add((int)entry.getKey());
                        j++;
                }
        }

        private static ArrayList<double[]> generateMinorityClassArray () {
                minorityClass = getMinorityClass();
                for(int i=0; i < dataModel.statuses.size(); i++){
                        if (dataModel.statuses.get(i) == minorityClass) {
                                minorityClassArray.add(fullDataset.get(i));
                        }
                }

                return minorityClassArray;
        }

        public static double getMinorityClass () {
                double PD = 0.0, NPD = 0.0;
                for(int i = 0; i < dataModel.statuses.size(); i++){
                        if (dataModel.statuses.get(i) == 1.0) {
                                PD++;
                        }
                        else NPD++;
                }
                minorityClassCount = (int)(PD > NPD ? NPD : PD);
                return PD > NPD ? 0.0 : 1.0;
        }

        public ArrayList<double[]> preProcessData (Dataset data, ArrayList <double[]>
dataset, int smote, int nearestNeighbors) {
		dataModel = data;
		nn = nearestNeighbors;
		fullDataset = dataset;
		amountOfSmote = smote;
		syntheticGenerated = 0;
		minorityClassArray = new ArrayList<double[]>();
		nnArray = new ArrayList<Integer>();
		performMethod();
		dataModel.setTotalSyntheticSamplesGenerated(syntheticGenerated);
		return fullDataset;
	}
}
```

L. dataprepop/ZScoreNormalization.java

```
package datapreprop;

import java.util.ArrayList;

public class ZScoreNormalization {
	private ArrayList<double[]> preProcessedData;
	private int totalFeatures;
	private static double[] globalMeans, globalSds;

	public ZScoreNormalization(ArrayList<double[]> preprop, int totalfeatures)
	{
		this.preProcessedData = preprop;
		this.totalFeatures = totalfeatures;
	}

	public ArrayList<double[]> processTrainingData() {
		ZScoreNormalization.globalMeans = new double[totalFeatures];
		ZScoreNormalization.globalSds = new double[totalFeatures];
		performMethod();
		return preProcessedData;
	}

	public ArrayList<double[]> processTestingData(double[] means, double[] sds) {
		transformData(means, sds);
		return preProcessedData;
	}

	public static double[] getGlobalMeans()
	{
		return globalMeans;
	}

	public static double[] getGlobalSds()
	{
		return globalSds;
	}

	private void performMethod() {
		globalMeans = getMeanPerFeature();
		globalSds = getSDPerFeature(globalMeans);

		transformData(globalMeans, globalSds);
	}

	private void transformData(double[] means, double[] sds) {
        int totalInstance = preProcessedData.size();

        for(int i=0; i < totalFeatures; i++)
        {
            for (int j = 0; j < totalInstance; j++)
            {
                double a = preProcessedData.get(j)[i];
                preProcessedData.get(j)[i] = ((a - means[i])/sds[i]);
            }
        }
    }

    private double[] getSDPerFeature(double[] means) {
        double[] sds = new double [totalFeatures];
        int totalInstance = preProcessedData.size();

        for(int i=0; i < totalFeatures; i++)
        {
            double meanDistance = 0.0;
            for (int j = 0; j < totalInstance; j++)
            {
                double distance = Math.pow(means[i] - 
preProcessedData.get(j)[i], 2);
                meanDistance += distance;
            }
            meanDistance /= totalInstance;
            sds[i] = Math.sqrt(meanDistance);
        }

        return sds;
    }

    private double[] getMeanPerFeature() {
        double[] means = new double [totalFeatures];
        int totalInstance = preProcessedData.size();

        for(int i=0; i < totalFeatures; i++)
        {
            means[i] = 0.0;

            for (int j = 0; j < totalInstance; j++)
            {
                means[i] += preProcessedData.get(j)[i];

            }

            means[i] /= totalInstance;
        }
        return means;
    }

}
```

M. proc/DataPrep.java

```
package proc;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.Serializable;
import dataprepro p.MinMaxNormalization;
import dataprepro p.PCA;
import dataprepro p.ZScoreNormalization;
import libsvm.svm_model;

@SuppressWarnings("serial")
public class DataPrep implements Serializable
{
        private double[] globalMin, globalMax, globalMeans, globalSds, globalPCAMeans;
        private double[][] globalFeatureVector;
        private double thisMin, thisMax;
        private boolean smote, qclean, minmax, zscore, pca;
        private int globalPcCount;
        private svm_model svmModel;
        private String modelFilePath, kernelType, svmType;

        private void setMinMaxParam(double[] globalMin, double[] globalMax, double
thisMin, double thisMax) {
                this.globalMin = globalMin;
                this.globalMax = globalMax;
                this.thisMin = thisMin;
                this.thisMax = thisMax;
        }

        private void setZScoreParam(double[] globalMeans, double[] globalSds) {
                this.globalMeans = globalMeans;
                this.globalSds = globalSds;
        }

        private void setPCAParam(double[] globalPCAMeans, double[][]
globalFeatureVector, int pcCount) {
                this.globalPCAMeans = globalPCAMeans;
                this.globalFeatureVector = globalFeatureVector;
                this.globalPcCount = pcCount;
        }

        private void setProc(boolean minmax, boolean zscore, boolean pca) {
                this.minmax = minmax;
                this.zscore = zscore;
                this.pca = pca;
        }

        private void setAlgo(boolean smote, boolean qclean) {
                this.smote = smote;
                this.qclean = qclean;
        }

        private void setSVMModel(svm_model svmModel) {
                this.svmModel = svmModel;
        }

        public void setKernelType(String kernelType) {
                this.kernelType = kernelType;
        }

        public void setSVMType(String svmType) {
                this.svmType = svmType;
        }

        public void writeData(String filePath, ProcParams proc, svm_model svmModel)
                        throws FileNotFoundException, IOException, ClassNotFoundException
{
                ObjectOutputStream objectOutputStream = new ObjectOutputStream(new
FileOutputStream(filePath));
                boolean smote = proc.getProc()[0];
                boolean qclean = proc.getProc()[1];
                boolean minmax = proc.getProc()[2];
                boolean zscore = proc.getProc()[3];
                boolean pca = proc.getProc()[4];

                this.setAlgo(smote, qclean);
                this.setProc(minmax, zscore, pca);

                if(minmax) {
                        this.setMinMaxParam(MinMaxNormalization.getGlobalMin(),
MinMaxNormalization.getGlobalMax(), proc.getMin(), proc.getMax());
                }

                if(zscore) {
                        this.setZScoreParam(ZScoreNormalization.getGlobalMeans(),
ZScoreNormalization.getGlobalSds());
                }

                if(pca) {
                        this.setPCAParam(PCA.getGlobalMeans(),
PCA.getGLobalFeatureVector(), proc.getNoPCs());
                }

                this.setSVMModel(svmModel);
                this.modelFilePath = filePath;

                objectOutputStream.writeObject(this);
                objectOutputStream.flush();
                objectOutputStream.close();
        }

        public DataPrep readData(String filePath) throws FileNotFoundException,
IOException, ClassNotFoundException {
                @SuppressWarnings("resource")
                ObjectInputStream objectInputStream = new ObjectInputStream(new
FileInputStream(filePath));
                return (DataPrep)objectInputStream.readObject();
        }

        public boolean[] getProc() {
                boolean[] procs = new boolean[3];
                procs[0] = this.minmax;
                procs[1] = this.zscore;
                procs[2] = this.pca;
                return procs;
        }

        public double[] getGlobalMin() {
                return this.globalMin;
        }

        public double[] getGlobalMax() {
```

```
		return this.globalMax;
	}

	public double getThisMin() {
		return this.thisMin;
	}

	public double getThisMax() {
		return this.thisMax;
	}

	public double[] getGlobalMeans() {
		return this.globalMeans;
	}

	public double[] getGlobalSds() {
		return this.globalSds;
	}

	public double[] getGlobalPCAMeans() {
		return this.globalPCAMeans;
	}

	public double[][] getGlobalFeatureVector() {
		return this.globalFeatureVector;
	}

	public int getGlobalPcCount() {
		return this.globalPcCount;
	}

	public svm_model getSVMModel() {
		return this.svmModel;
	}

	public String getModelFilePath() {
		return this.modelFilePath;
	}

	public String getKerneltype() {
		return this.kernelType;
	}

	public String getSVMType() {
		return this.svmType;
	}

	public boolean[] getAlgo() {
		boolean[] algo = new boolean[2];
		algo[0] = smote;
		algo[1] = qclean;
		return algo;
	}
}
```

N. proc/Dataset.java

```
package proc;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.LineNumberReader;
import java.util.ArrayList;
import java.util.Random;
import java.util.StringTokenizer;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import util.Util;

public class Dataset {
       private JFrame appFrame;
       private String file;
       private int totalFeatures, indexOfStatus;
       private String[] attributes;
       public ArrayList <Double> statuses;
       private ArrayList <double[]> fullDataset, preProcessedData;
       private final int testCount = 15;
       private String mode;
       private ArrayList<String> ids;
       private int totalSynthetic, totalNoise, attributeCount, totalOriginalInstance,
       totalOriginalAttributes, totalOriginalPDPos, totalOriginalPDNeg;
       private boolean error = false;
       public Dataset(String file, JFrame appFrame, String mode) {
              this.appFrame = appFrame;
              this.file = file;
              this.totalFeatures = 0;
              this.totalNoise = 0;
              this.totalSynthetic = 0;
              this.fullDataset = new ArrayList <double[]>();
              this.statuses = new ArrayList <Double>();
              this.mode = mode;
              this.ids = new ArrayList<String>();
              this.preProcessedData = new ArrayList<double[]>();
              doProcess();
              this.totalOriginalInstance = fullDataset.size();
              this.totalOriginalAttributes = fullDataset.get(0).length;
              this.totalOriginalPDPos = getClassCount(1.0);
              this.totalOriginalPDNeg = getClassCount(0.0);
       }

       public int getTotalOriginalInstance() {
              return this.totalOriginalInstance;
       }

       public int getTotalOriginalAttributes() {
              return this.totalOriginalAttributes;
       }

       public int getTotalOriginalPDPos() {
              return this.totalOriginalPDPos;
       }

       public int getTotalOriginalPDNeg() {
              return this.totalOriginalPDNeg;
       }

       public int getTotalFeatures() { return totalFeatures; }
       public ArrayList <double[]> getFullDataset() { return fullDataset; }
```

```
    public ArrayList< Double > getStatuses() { return statuses; }
    public String[] getAttributes() { return attributes; }
    public int getClassCount(double stat) {
        int count = 0;
        for(double i : statuses) {
            if(stat == i) {
                count++;
            }
        }

        return count;
    }

    public ArrayList<String> getIds() {
        return this.ids;
    }

    public void setPreProcessedData(ArrayList<double[]> preProcessedData) {
        this.preProcessedData = preProcessedData;
    }
    public double[][] getReducedData() {
        double[][] reduced = new
double[preProcessedData.size()][preProcessedData.get(0).length + 1];

        for(int k = 0; k < preProcessedData.size(); k++) {
            reduced[k][0] = statuses.get(k);
            for(int j=1; j<=preProcessedData.get(0).length; j++) {
                reduced[k][j] = preProcessedData.get(k)[j-1];
            }
        }
        return reduced;
    }

    private void doProcess() {
        BufferedReader br;
        try {
            br = new BufferedReader(new FileReader(file));
            String attrLabel = br.readLine();
            ArrayList<String> lines = new ArrayList<String>();
            attributes = getAttributeLabels(attrLabel);
            totalFeatures = attributes.length;
            String line = "";
            int lineCount = getLineCount();
            int iter = 0;
            ArrayList<Integer> randomIndexes = new ArrayList<Integer>();
            if(mode.equals("train")) {
                randomIndexes = generateRandomIndexes(0, lineCount);
            }

            while((line = br.readLine()) != null) {
                if(mode.equals("train") && randomIndexes.contains(iter)) {
                    lines.add(line);
                }
                else {
                    double[] instance = getInstances(line);
                    fullDataset.add(instance);
                }
                iter++;
            }

            if(mode.equals("train")) {
```

```
                        writeTestFile(attrLabel, lines);
                }
        }

        catch(FileNotFoundException e) {
                JOptionPane.showMessageDialog(appFrame,
                                "Dataset file not found!");
                error = true;
        }
        catch(IOException e) {
                error = true;
        }
}

private void writeTestFile(String attrLabel, ArrayList<String> testInstances) {
        BufferedWriter writer = null;
        try {
                writer = new BufferedWriter(new FileWriter("./demo.csv"));
                writer.write(attrLabel);
                writer.newLine();
                for(String i: testInstances) {
                        writer.write(i);
                        writer.newLine();
                }
        }
        catch(IOException e) {}
        finally {
                try {
                        if(writer != null)
                                writer.close();
                }
                catch(IOException e) {}
        }
}

private int getLineCount() throws IOException {
        LineNumberReader  lnr = new LineNumberReader(new FileReader(file));
        lnr.skip(Long.MAX_VALUE);
        lnr.close();
        return lnr.getLineNumber();
}

private ArrayList<Integer> generateRandomIndexes(int min, int max) {
        ArrayList<Integer> indexes = new ArrayList<Integer>();
        Random randomGenerator = new Random();
        while(indexes.size() < testCount) {
                int random = randomGenerator.nextInt((max - min) + 1) + min;
                if(!indexes.contains(random)) {
                        indexes.add(random);
                }
        }

        return indexes;
}

private double[] getInstances(String instance) {
        ArrayList <Double> instArray = new ArrayList <Double>();
        StringTokenizer stringTokenizer = new StringTokenizer(instance, ",");
        while(stringTokenizer.hasMoreTokens()) {
                String token = stringTokenizer.nextToken();
                try {
```

```
                                instArray.add(Double.parseDouble(token));
                        }
                        catch(Exception e) {
                                ids.add(token);
                        }
                }

                statuses.add(instArray.get(indexOfStatus));
                instArray.remove(indexOfStatus);
                return Util.convertArrayListToDoubleArray(instArray);
        }

        private String[] getAttributeLabels(String labels) {
                ArrayList <String> attrArray = new ArrayList <String>();
                StringTokenizer stringTokenizer = new StringTokenizer(labels, ",");
                while(stringTokenizer.hasMoreTokens()) {
                        attrArray.add(stringTokenizer.nextToken());
                }
                attrArray.remove(0); //remove name
                indexOfStatus = attrArray.indexOf("status");
                attrArray.remove(indexOfStatus);
                return Util.convertToStringArray(attrArray);
        }

        public boolean hasError() {
                return error;
        }

        public void setAttributeCount(int attrCount) {

                this.attributeCount = attrCount;
        }

        public int getAttributeCount() {
                return this.attributeCount;
        }

        public void setTotalNoiseRemoved(int totalNoise) {
                this.totalNoise = totalNoise;
        }

        public void setTotalSyntheticSamplesGenerated(int totalSynthetic) {
                this.totalSynthetic = totalSynthetic;
        }

        public int getTotalNoiseRemoved() {
                return this.totalNoise;
        }

        public int getTotalSyntheticSamplesGenerated() {
                return this.totalSynthetic;
        }
}
```

O. proc/ProcParams.java

```
package proc;

public class ProcParams {
        private boolean smote = false;
        private boolean qclean = false;
        private boolean minmax = true;
        private boolean zscore = false;
        private boolean pca = false;

        private int percentageSmote, nnSmote, nnQclean, min, max = 1, noPCs;
        private double cost, eps;
        private String svmType, kernelType, svmModelFile;

        public void setProc(boolean smote, boolean qclean, boolean minmax,
                        boolean zscore, boolean pca) {
                this.smote = smote;
                this.qclean = qclean;
                this.minmax = minmax;
                this.zscore = zscore;
                this.pca = pca;
        }

        public void setSmoteParam(int percentageSmote, int nnSmote) {
                this.percentageSmote = percentageSmote;
                this.nnSmote = nnSmote;
        }

        public void setQCleanParam(int nnQclean) {
                this.nnQclean = nnQclean;
        }

        public void setMinMaxParam(int min, int max) {
                this.min = min;
                this.max = max;
        }

        public void setPCAParam(int noPCs) {
                this.noPCs = noPCs;
        }

        public void setSVMParam(double cost, double eps, String svmType, String
kernelType) {
                this.cost = cost;
                this.eps = eps;
                this.svmType = svmType;
                this.kernelType = kernelType;
        }

        public void setSVMModelFile(String svmModelFile) {
                this.svmModelFile = svmModelFile;
        }

        public int getPercentageSmote() {
                return percentageSmote;
        }

        public int getNnSmote() {
                return nnSmote;
        }

        public int getNnQclean() {
                return nnQclean;
        }
        public int getMin() {
                return min;
        }

        public int getMax() {
                return max;
        }

        public int getNoPCs() {
                return noPCs;
        }

        public double getCost() {
                return cost;
        }

        public double getEps() {
                return eps;
        }

        public String getSVMType() {
                return svmType;
        }

        public String getKernelType() {
                return kernelType;
        }

        public String getSVMModelFile() {
                return svmModelFile;
        }

        public boolean[] getProc() {
                return new boolean [] { smote, qclean, minmax, zscore, pca };
        }
}
```

P. proc/TestProcessor.java

```
package proc;

import gui.TestResultPanel;
import java.util.ArrayList;
import javax.swing.JFrame;
import javax.swing.JTabbedPane;
import javax.swing.SwingWorker;
import dataprepop.MinMaxNormalization;
import dataprepop.PCA;
import dataprepop.ZScoreNormalization;
import svm.SvmEvaluator;

public class TestProcessor extends SwingWorker <JTabbedPane, Void> {
        private Dataset dataModel;
        private String dataPrepFilePath;
        private JFrame aFrame;
        private DataPrep dataPrep;
        private ArrayList<double[]> testingData;
        private double[][] reducedData;

        public TestProcessor(Dataset dataModel, String dataPrepFilePath, JFrame aFrame)
{
```

```
                this.dataModel = dataModel;
                this.dataPrepFilePath = dataPrepFilePath;
                this.aFrame = aFrame;
        }

        @Override
        protected JTabbedPane doInBackground() throws Exception {
                JTabbedPane resultTabbedPane = new JTabbedPane();
                boolean dataReduce = true;
                dataPrep = new DataPrep();
                dataPrep = dataPrep.readData(dataPrepFilePath);
                testingData = dataModel.getFullDataset();

                if(dataPrep.getProc()[0]) {
                        MinMaxNormalization minmax = new MinMaxNormalization(testingData, 
dataPrep.getThisMin(), dataPrep.getThisMax());
                        testingData = minmax.processTestingData(dataPrep.getGlobalMin(), 
dataPrep.getGlobalMax());
                }

                if(dataPrep.getProc()[1]) {
                        ZScoreNormalization zscore = new ZScoreNormalization(testingData, 
dataModel.getTotalFeatures());
                        testingData = 
zscore.processTestingData(dataPrep.getGlobalMeans(), dataPrep.getGlobalSds());
                }

                if(dataPrep.getProc()[2]) {
                        PCA pca = new PCA(testingData, dataModel.getStatuses(), 
dataModel.getTotalFeatures(), dataPrep.getGlobalPcCount());
                        reducedData = 
pca.processTestingData(dataPrep.getGlobalPCAMeans(), 
dataPrep.getGlobalFeatureVector());
                        dataReduce = false;
                }

                if(dataReduce) {
                        dataModel.setPreProcessedData(testingData);
                        reducedData = dataModel.getReducedData();
                }

                SvmEvaluator svmEval = new SvmEvaluator(reducedData, 
dataPrep.getSVMModel());
                aFrame.add(new TestResultPanel(dataPrep, dataModel, svmEval));
                aFrame.setVisible(true);

                return resultTabbedPane;
        }
}
```

Q. proc/TrainProcessor.java

```
package proc;

import gui.TrainResultPanel;
import java.util.ArrayList;
import javax.swing.JFrame;
import javax.swing.JTabbedPane;
import javax.swing.SwingWorker;
import datapreprop.MinMaxNormalization;
import datapreprop.PCA;
import dataprepop.QCleanNoise;
import dataprepop.Smote;
import dataprepop.ZScoreNormalization;
import svm.SvmParam;
import svm.SvmProc;
import libsvm.svm_model;

public class TrainProcessor extends SwingWorker <JTabbedPane, Void> {
	private Dataset dataModel;
	private ProcParams proc;
	private JFrame aFrame;
	private ArrayList <double[]> preProcessedData;
	private double[][] reducedData;
	private double accuracy;

	public TrainProcessor(Dataset dataModel, ProcParams proc, JFrame aFrame) {
		this.dataModel = dataModel;
		this.proc = proc;
		this.aFrame = aFrame;
	}

	@Override
	protected JTabbedPane doInBackground() throws Exception {
		JTabbedPane resultTabPane = new JTabbedPane();
		preProcessedData = dataModel.getFullDataset();
		boolean dataReduce = true;

		if(proc.getProc()[0]) {
			Smote smote = new Smote();
			preProcessedData = smote.preProcessData(dataModel, 
preProcessedData,
					proc.getPercentageSmote(), proc.getNnSmote());
		}

		if(proc.getProc()[1]) {
			QCleanNoise qclean = new QCleanNoise();
			preProcessedData = qclean.preProcessData(dataModel, 
preProcessedData, proc.getNnQclean());
		}

		if(proc.getProc()[2]) {
			MinMaxNormalization minMax = new 
MinMaxNormalization(preProcessedData, proc.getMin(), proc.getMax());
			preProcessedData = minMax.processTrainingData();
		}

		if(proc.getProc()[3]) {
			ZScoreNormalization zScore = new 
ZScoreNormalization(preProcessedData, dataModel.getTotalFeatures());
			preProcessedData = zScore.processTrainingData();
		}

		if(proc.getProc()[4]) {
			PCA pca = new PCA(preProcessedData, dataModel.getStatuses(), 
dataModel.getTotalFeatures(), proc.getNoPCs());
			reducedData = pca.processTrainingData();
			dataReduce = false;
		}

		if(dataReduce) {
			dataModel.setPreProcessedData(preProcessedData);
```

```
                        reducedData = dataModel.getReducedData();
                }

                dataModel.setAttributeCount(reducedData[0].length - 1);
                SvmParam param = new SvmParam(proc.getCost(), proc.getEps(),
proc.getSVMType(), proc.getKernelType());
                SvmProc svmProc = new SvmProc(param.getSvmParam(), reducedData);
                svm_model model = svmProc.getSVMModel();
                accuracy = svmProc.getModelAccuracy();

                DataPrep dp = new DataPrep();
                dp.setKernelType(proc.getKernelType());
                dp.setSVMType(proc.getSVMType());
                dp.writeData(proc.getSVMModelFile(), proc, model);

                aFrame.add(new TrainResultPanel(dataModel, proc, accuracy));
                aFrame.setVisible(true);

                return resultTabPane;
        }
}
```

R. svm/SvmEvaluator.java

```
package svm;

import libsvm.svm;
import libsvm.svm_model;
import libsvm.svm_node;

public class SvmEvaluator {
        private double accuracy;
        private double[][] reducedData;
        private svm_model svmModel;
        private double[] probabilities;
        private String[] predictions, actuals;
        private int correctlyClassifiedCount;

        public SvmEvaluator(double[][] reducedData, svm_model svmModel) {
                this.predictions = new String[reducedData.length];
                this.actuals = new String[reducedData.length];
                this.probabilities = new double[reducedData.length];
                this.reducedData = reducedData;
                this.svmModel = svmModel;
                this.correctlyClassifiedCount = 0;
                this.accuracy = evaluateAccuracy();
        }

        public double getAccuracy() {
                return this.accuracy;
        }

        public String[] getPredictions() {
                return this.predictions;
        }

        public String[] getActuals() {
                return this.actuals;
        }

        public double[] getProbabilities() {
```

```
        return this.probabilities;
}

public int getCorrectlyClassifiedCount() {
        return correctlyClassifiedCount;
}

private double evaluateAccuracy() {
        double iter = 0.0;
        double acc = 0.0;

        for(double[] i : reducedData) {
                double res = evaluate(i, svmModel, (int)iter);
                if(res == reducedData[(int)iter][0]) {
                        acc++;
                }
                iter++;
        }

        acc = acc/iter*100;
        return acc;
}

private double evaluate(double[] features, svm_model model, int iter) {
        svm_node[] nodes = new svm_node[features.length-1];
        for (int i = 1; i < features.length; i++) {
                svm_node node = new svm_node();
                node.index = i;
                node.value = features[i];

                nodes[i-1] = node;
        }

        int totalClasses = 2;
        int[] labels = new int[totalClasses];
        svm.svm_get_labels(model,labels);

        double[] prob_estimates = new double[totalClasses];
        double v = svm.svm_predict_probability(model, nodes, prob_estimates);

        double proby = 0;
        for (int i = 0; i < totalClasses; i++) {
                if(labels[i] == v) {
                        proby = prob_estimates[i];
                        break;
                }
        }

        String pred = "";
        String act = "";

        if(v == 1.0) {
                pred = "PD+";
        }
        else if(v == 0.0) {
                pred = "PD-";
        }
        else {
                pred = "-";
        }
        if(features[0] == 0.0) {
            act = "PD-";
        }
        else if(features[0] == 1.0) {
            act = "PD+";
        }
        else {
            act = "-";
        }

        predictions[iter] = pred;
        actuals[iter] = act;
        probabilities[iter] = proby;

        if(pred.equals(act)) {
            correctlyClassifiedCount++;
        }

        return v;
    }
}
```

S. svm/SvmParam.java

```
package svm;

import libsvm.svm_parameter;

public class SvmParam {

    private static svm_parameter param;

    public SvmParam(double cost, double eps, String svmType, String kernelType) {
        param = new svm_parameter();
        param.C = cost;
        param.eps = eps;
        param.svm_type = svmType.equals("nu-SVC") ? svm_parameter.NU_SVC :
svm_parameter.C_SVC;

        int kernel = 0;

        switch(kernelType) {
        case "Linear" :
            kernel = svm_parameter.LINEAR;
            break;
        case "Polynomial" :
            kernel = svm_parameter.POLY;
            break;
        case "Radial Basis Function" :
            kernel = svm_parameter.RBF;
            break;
        case "Sigmoid" :
            kernel = svm_parameter.SIGMOID;
            break;
        default :
            break;
        }

        param.kernel_type = kernel;
        param.probability = 1;
```

```
		param.gamma = 0.5;
		param.nu = 0.5;
		param.cache_size = 40;
		param.coef0 = 0.0;
		param.degree = 3;
		param.shrinking = 1;
	}

	public svm_parameter getSvmParam() {
		return param;
	}
}
```

T. svm/SvmProc.java

```
package svm;

import libsvm.svm;
import libsvm.svm_model;
import libsvm.svm_node;
import libsvm.svm_parameter;
import libsvm.svm_problem;

public class SvmProc {

	private svm_parameter svmParam;
	private double[][] reducedData;
	private svm_problem svmProb;

	public SvmProc(svm_parameter svmParam, double[][] reducedData) {
		this.svmParam = svmParam;
		this.reducedData = reducedData;
		this.svmProb = buildSvmProb();
	}

	private svm_problem buildSvmProb() {
		svm_problem prob = new svm_problem();
		int dataCount = reducedData.length;
		prob.y = new double[dataCount];
		prob.l = dataCount;
		prob.x = new svm_node[dataCount][];

		for (int i = 0; i < dataCount; i++) {
			double[] features = reducedData[i];
			prob.x[i] = new svm_node[features.length-1];
			for (int j = 1; j < features.length; j++) {
				svm_node node = new svm_node();
				node.index = j;
				node.value = features[j];
				prob.x[i][j-1] = node;
			}
			prob.y[i] = features[0];
		}
		return prob;
	}

	public double getModelAccuracy() {
		double[] target = new double[reducedData.length];
		svm.svm_cross_validation(svmProb, svmParam, 10, target);

		double dataCount = 0;
		double accuracy = 0;

		for(double i : target) {
			if(i == reducedData[(int)dataCount][0]) {
				accuracy++;
			}
			dataCount++;
		}

		accuracy = 100*accuracy/dataCount;
		return accuracy;
	}

	public svm_model getSVMModel() {
		svm_model model = svm.svm_train(svmProb, svmParam);
		return model;
	}
}
```

U. util/Util.java

```
package util;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.Iterator;
import java.util.LinkedHashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;

import javax.swing.JFileChooser;
import javax.swing.JTextField;
import javax.swing.filechooser.FileNameExtensionFilter;

public class Util {
	private static BufferedReader br;
	private static String emptyString = "Field is empty.";
	private static String notcsvFile =
			"File is in wrong format. File should be in .csv.";
	private static String fileNotFound = "File not found.";
	private static String ioException = "Error opening file.";
	private static String valid = "Valid.";

	public static void openChooserWindow (JTextField fileTextField, boolean
directory, boolean csvFile) {
		JFileChooser chooser = new JFileChooser ();
		chooser.setCurrentDirectory(new java.io.File("."));
		if(directory) {
			chooser.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
			chooser.setAcceptAllFileFilterUsed(false);
		}
		else {
			FileNameExtensionFilter filter;

			if(csvFile) {
```

```
                        filter = new FileNameExtensionFilter("CSV files", "csv");
                }
                else {
                        filter = new FileNameExtensionFilter("BIN files", "bin");
                }
                chooser.setFileFilter(filter);
        }
        int result = chooser.showOpenDialog(null);
        if (result == JFileChooser.APPROVE_OPTION) {
                String filePath = null;
                try {
                        filePath = chooser.getSelectedFile().toString();
                }
                catch (Exception ex) {
                        System.out.println("Could not open file");
                }
                if (filePath != null) {
                        fileTextField.setText(filePath);
                }
        }
}

public static String isValidFile (String file) {
        try {
                if ((file.length() > 0 && !file.equals(""))) {
                        if (!file.endsWith(".csv")) {
                                return notcsvFile;
                        }
                        br = new BufferedReader (new FileReader(file));
                        br.readLine();
                }
                else return emptyString;
        }
        catch (FileNotFoundException e) {
                return fileNotFound;
        }
        catch (IOException e) {
                return ioException;
        }
        finally {
                try {
                        if (br != null)
                                br.close();
                }
                catch (IOException e) {
                        return ioException;
                }
        }
        return valid;
}

public static String[] convertToStringArray (ArrayList <String> arraylist) {
        String[] stringArr = new String [arraylist.size()];
        for (int i=0; i<arraylist.size(); i++) {
                stringArr[i] = arraylist.get(i);
        }
        return stringArr;
}

public static String convertToString (double[] arr) {
        String stringArr = "";
```

```
		for (int i=0; i<arr.length; i++) {
			stringArr.concat(arr[i] + " ");
		}
		return stringArr;
	}

	public static double[] convertToDoubleArray (ArrayList <String> arraylist) {
		double[] doubleArr = new double [arraylist.size()];
		for (int i=0; i<arraylist.size(); i++) {
			doubleArr[i] = Double.parseDouble((arraylist.get(i)));
		}
		return doubleArr;
	}

	public static double[] convertArrayListToDoubleArray (ArrayList <Double> 
arraylist) {
		double[] doubleArr = new double [arraylist.size()];
		for (int i=0; i<arraylist.size(); i++) {
			doubleArr[i] = (arraylist.get(i));
		}
		return doubleArr;
	}

	@SuppressWarnings({ "unchecked", "rawtypes" })
	public static Map<Integer, Double> sortByValue(Map<Integer, Double> map) {
		List list = new LinkedList(map.entrySet());
		Collections.sort(list, new Comparator() {
			public int compare(Object o1, Object o2) {
				return ((Comparable) ((Map.Entry) (o1)).getValue())
						.compareTo(((Map.Entry) (o2)).getValue());
			}
		});

		Map result = new LinkedHashMap();
		for (Iterator it = list.iterator(); it.hasNext();) {
			Map.Entry entry = (Map.Entry)it.next();
			result.put(entry.getKey(), entry.getValue());
		}
		return result;
	}
}
```

CHAPTER XI

ACKNOWLEDGEMENT

First of all, I would like to thank God for giving me strength and wisdom while I finish my special project. Since I am working at the same time I am doing my SP, I sometimes encounter hard times in keeping my brain functioning after a long day of work. Also, I would like to thank Him for blessing me with enough resources that I used in the course of my college life and blessing me with the gift of family and friends. Every accomplishment I had was because of Him.

I want to thank my closest college friend, LoAnn for being helpful whenever I have questions regarding anything and to Joshua and Jayvee for always being there whenever I need a substitute for enrolment or when I am applying for residency. To May, for being so patient with all my anything-under-the-sun questions. Thanks guys!

I would also like to thank my best friends – MKING - who have encouraged me to finish my SP and told me the benefits I would have if I am already finished with all my requirements. They have inspired me since all of them are graduates already and for appreciating my course even if most of the time they don't understand what I am talking about. Thank you!

This project will never be done without the help of my very approachable and kind adviser, Mam Perl and Sir Solano. Thank you Mam for the effort of meeting me for consultation even if I am just your adopted advisee and you are always busy. Thank you for pushing me to finish everything in two weeks. If you did not encourage me, I think I would still be extending one more semester. I am glad you are my adviser. To Sir Solano, thank you for calling from time to time to check about the progress of my work. I appreciated it a lot Sir.

To Blue, this is for our NZ! You already know all the things I am thankful for.

And finally, to my Ate, my Kuya and my Mom, who are always proud of me regardless of what I do. Thank you for understanding my failures. Sorry if my diploma got delayed for a while but I know you understand. Ma, thank you for being my mother cause I could not imagine my life without you. This is for you guys! I love you all so much!