I. INTRODUCTION

A. BACKGROUND OF THE STUDY

Tuberculosis (TB) is a highly contagious infection caused by the bacterium called Mycobacterium tuberculosis. Tuberculosis can be transmitted via aerosols (droplets from the mouth and respiratory tract) which are expelled from the body to the surrounding environment through coughing, spitting or sneezing. When inhaled by a susceptible host, these droplets can bring infection and within weeks or months, the disease may begin to develop within the infected person's body [1].

TB is a global health problem and is amongst one of the world's leading cause of death from a single infectious disease, with 10 million new cases and over 3 million deaths per year [1]. The World Health Organization (WHO) estimates that over one-third of world´s population now carries the infectious bacteria [2].

The increase in prevalence of tuberculosis brings not only financial troubles to the affected families but economic repercussions to the country as well. Eighty percent of people afflicted with tuberculosis are in the most economically productive years of their lives, thus, sending many self-sustaining families into poverty. The rise in the incidence of tuberculosis could be associated with the low priority accorded to anti-tuberculosis activities by many countries. The unavailability of anti-TB drugs, insufficient laboratory networking, poor health infrastructures,

including a lack of trained health personnel, have also contributed to the rise in the incidence of the disease [3].

According to the WHO's Global TB Report in 2009, the Philippines ranks ninth on the list of 22 high-burden tuberculosis countries in the world. After China, it had the second highest number of cases in the WHO Western Pacific Region. Tuberculosis ranks sixth on the top causes of morbidity and mortality in the country. In 2007, approximately 100 Filipinos died each day because of the disease [4].

Significant strides have been made by the government to increase detection rate and treatment of tuberculosis in the country. In 2004, the country achieved a TB case detection rate of 72 percent, exceeding WHO's target of 70 percent, and reached 75 percent in 2007. The DOTS (the internationally recommended strategy for TB control) treatment success rate reached WHO's target of 85 percent in 1999 and has remained around 88 percent since then [4]. These numbers could still be increased using expert systems that could provide early diagnosis for a suspected patient without undergoing medical exams. Likewise, patients diagnosed with the disease could be classified by the system in order to tell the medical practitioners the nature of tuberculosis the patient has.

B. STATEMENT OF THE PROBLEM

Medical diagnosis is the process of identifying diseases and can be determined by the signs

and symptoms exhibited by the patient. Disease diagnosis is utilized for planning the disease control measures, prognosis, special precautions, to know the life threatening situations and prevention of spreading of disease to others especially for highly-communicable diseases like tuberculosis. Case detection of tuberculosis is considered a vital and intricate job, hence, must be given enough attention and importance.

In today's world, especially the emergence of drug-resistant tuberculosis, a fast and reliable TB diagnosis generation is of eminent importance. In order to achieve speed and accuracy in diagnosis of tuberculosis, expert systems can be used as support for the decision making process of healthcare providers. Laboratory findings and clinical data are sometimes too complex for manual analysis since the human mind is not trained to process large data sets, hence, could be very prone to error. Pattern recognition for large data sets could be difficult for humans to achieve, but could be child's play for a computer capable of machine learning, thus, the need for sophisticated modeling techniques focused on solving extremely sensitive, yet complex function like disease diagnosis. However, there are very few expert systems used by the medical practitioners for tuberculosis detection especially on rural areas where no doctors are available.

Aside from the limited amount of computer systems used, the biggest drawback of most proposals published so far is that they implement systems that cannot be extended easily and that are difficult to maintain. Rule-based expert systems are difficult to update because of the if-then structure of the knowledge base. However, this problem could be solved using machine learning through artificial neural networks.

Given above reasons, there is a need to create and implement a reliable, flexible, robust and user-friendly diagnosis system for tuberculosis to be used by the local healthcare providers.

C. OBJECTIVES

To develop a software system that will allow users to:

1. Create a project for classification of the disease
2. Input data set. Users can choose to:
    a) Manually input the data using a built-in Excel-like data sheet
    b) Import an external .CSV or .XLS file
3. Manage the data set.
    a) Convert nominal data to ordinal using oneOf mapping
    b) Input maximum and minimum values for ordinal data
    c) Choose which columns are the input and output data
    d) Normalize values
4. Edit training coefficients and network topology. These variables include:
    a) momentum
    b) learning rate
    c) maximum error
    d) epoch (maximum training cycles)
    e) number of hidden neurons
5. Choose the type of activation function. Activation functions can either be:

a) Sigmoid

b) Hyperbolic Tangent

6. Train the neural network using backpropagation algorithm and incremental pruning. The user can also choose to:

   a) Stop training period

   b) Pause in between training

   c) Continue training after pause

7. Evaluate the network created from the training process

   a) Import the data to be evaluated

   b) Test the accuracy of the network

   c) Correctly map patient data to one of the outputs

   d) Export the classified data into .CSV or .XLS file

   e) Create a chart for data visualization

8. Save the project into a system file

D. SIGNIFICANCE OF THE STUDY

Artificial neural networks provide a powerful tool to help practitioners analyze, model and make sense of complex clinical data across a broad range of medical applications. Most applications of artificial neural networks to medicine are classification problems, that is, the task is on the basis of the measured features to assign the patient to one of a small set of classes [5].

A variety of health-related indexes (e.g., a combination of heart rate, levels of various substances in the blood, respiration rate) can be monitored. The onset of a particular medical condition could be associated with a very complex combination of changes on a subset of the variables being monitored. Neural networks have been used to recognize this predictive pattern so that the appropriate treatment can be prescribed [6].

Advantages of using an intelligent system using artificial neural networks to medicine includes ease of use. Neural networks learn by example. The neural network user gathers representative data, and then invokes training algorithms to automatically learn the structure of the data. The level of user knowledge needed to successfully apply neural networks is much lower compared to other techniques like the traditional nonlinear statistical methods [6]. Trying to learn a certain model is not adviseable especially to a fast-paced field like medical diagnosis especially on communicable and infectious diseases like tuberculosis.

Medical efficiency could easily be attained using an intelligent system. A classical, early study in the year 1971 showed that using 600 patient data, a computer could achieve 91.1 percent accuracy in diagnosis, almost 12 percent greater than the best human diagnosis achieved at 79.7 percent accuracy [7]. The larger data set provided, lesser the error the system could commit.

The proposed system will speed up the process of decision making for the medical professionals. They could easily decide on the problem solution since the system provides

support to a given medical hypothesis. Classification task for the neural network could provide the local healthcare provider an idea on the existence as well as the nature or type of tuberculosis, hence, they could easily provide the necessary treatments and medications needed by the patient.

Lastly, the system provides flexibility through the variety of data set that can be inputted by the user. The system could be used to predict the risk factor of tuberculosis on a certain person. It could also tell the existence or non-existence of the infection on a suspected patient. Furthermore, given a patient is already infected with the disease, the system could classify the type of tuberculosis and as well as the proper treatment that the patient needs. Recurrence of TB infection could also be tested for patients who have already been treated.

E. SCOPE AND LIMITATION

The running time and the convergence or divergence of an artificial neural network depends on several factors:

Data Set. The size of the data set will affect on how an artificial neural network will behave. A large data set will provide better approximations to the ideal output of the network but will surely increase the training time needed to reach convergence. On the other hand, smaller data sets, though provides faster training, might not give the desired output.

Network Topology. The size of a feedforward network depends on the number of nodes in the input layer, hidden layer and output layer. The number of nodes in the input layer is defined by the input elements in the input vector while the corresponding output vector defines the number of output nodes in the output layer. The more neurons in the a given layer, the more computations will be made, hence, increases the running time of the network to finish training. Network topology might also lead to underfitting and overfitting. Under-fitting occurs when there are too few neurons in the hidden layers of a network to adequately detect the signals in a complicated data set. Conversely, overfitting occurs when the neural network has so much information processing capacity that the limited amount of information contained in the training set is not enough to train all of the neurons in the hidden layers [8].

Training Cycles and Maximum Error. Training basically involves feeding training samples as input vectors through a neural network, calculating the error of the output layer, and then adjusting the weights of the network to minimize the error [8]. A training cycle or an epoch is the number of times traversing the network and computing for its weights. Increasing the number of epochs also mean the longer processing time for the given neural network. This behavior also applies to the maximum error. The greater the value of error provided by the user, the slower the rate of convergence of the system.

Learning Rate. A concept in many neural network training algorithms that specifies how radically the weight matrix should be updated based on training results. Properly setting the learning rate can have a profound impact on the speed with which the neural network learns.

Setting it too low will impede performance; too high may cause the neural network to behave randomly and never converge on a solution [8].

Momentum. Momentum could be introduced to the network in order to accelerate the convergence of the process. When no momentum term is used, it takes a long time before the minimum could be reached.

II. REVIEW OF RELATED LITERATURE

Several proposals have been made to speed up the diagnosis of tuberculosis. N.H. Phuong et al proposed a diagnosis system for tuberculosis called TUBERDIAG, which combines both positive and negative knowledge. Knowledge base of the system consists of a set of IF-THEN rules for diagnosis of tuberculosis where each rule assigns its weight in [0,1]. The inference machine can produce the final diagnosis as the total degree of confirmation and exclusion of tuberculosis diagnosis [9].

A.A. Imianvan and J.C. Obi explored the application of Fuzzy Cluster Means (FCM or Fuzzy C-Mean) analysis to the identification of different types of tuberculosis. Application of cluster analysis involves a sequence of methodological and analytical decision steps that enhances the quality and meaning of the clusters produced. The uncertainties often associated with analysis of tuberculosis test data are eliminated by the proposed system and it was concluded that the proposed model appears to be a more natural and intelligent way of classification, verification and matching of symptoms to the tuberculosis groups [1].

The use of Bayesian Networks and Rough Sets to predict the existence of mycobacterium tuberculosis was also considered by T. Uçar, D. Karahoca and A. Karahoca. 503 different patient records having 30 separate input parameters was obtained from a private clinic and was used in the entire process of the research. The Bayesian Network model classifies the instances with root mean square error of 22% whereas Rough Set algorithm does the same classification with 37%.

As a result, they concluded that Bayesian Network is a more accurate and reliable method when compared with Rough Set method for classification of tuberculosis patients [10].

T. Asha et al tried using different ensemble methods on classifying tuberculosis. They classified tuberculosis data into two groups named pulmonary tuberculosis (PTB) and retroviral PTB using ensemble classifiers such as Bagging, AdaBoost and Random forest trees. The data is obtained from the state hospital which mainly includes twelve preliminary symptoms. Evaluation measures such as sensitivity, specificity and accuracy are used for comparison. Random forest is found to be weak with 93% accuracy against 97% that of Bagging and 96% of Adaboost [11].

Several image processing techniques has also been applied to the diagnosis and detection of tuberculosis bacilli in patient data. M. Sotaquir´a, L. Rueda and R. Narvaez proposed a novel algorithm for the segmentation, detection and quantification of bacilli and clusters present in a digital image of sputum smear samples prepared with the Ziehl-Neelsen technique. Algorithms for color space segmentation, quantification and automatic diagnosis are described. Different color spaces were considered in order to develop an efficient and robust algorithm against noise and varying illumination conditions in the image. The combination of YcbCr and Lab color spaces offered the best results, with a sensitivity of 90.9% and a specificity of 100%. The accuracy of the algorithm was also measured, and results show that in 85.7% of the cases the algorithm provides a correct diagnosis in terms of infection level [12].

Another image processing technique was used by V. Makkapati, R. Agrawal and R. Acharya.

They created a hue color component based approach to segment tuberculosis bacilli by adaptive choice of the hue range. The bacilli are declared to be valid or invalid depending on the presence of beaded structure inside them. The beaded structure is segmented by thresholding the saturation component of the bacilli pixels. Clumps of bacilli and other artifacts are removed by thresholding the area, thread length and thread width parameters of the bacilli. Results presented for several images taken from different patients show that the scheme detects the presence of TB accurately [13].

Another image processing method for automating the detection of tubercle bacilli in sputum specimens was described by K. Veropoulos, C. Campbell and G. Learmonth. A fluorescence microscope with an attached digital camera is used to manually locate and capture images of tubercle bacilli. The method comiprises of two phases: image processing and analysis techniques are applied to the images for enhancement and feature extraction; and object recognition techniques are used for the automatic identification of tubercle bacilli in the images. The eventual implementation of the system would be semi-automatic where the best candidate images containing bacilli would be presented to the medical technologist together with a bacillus count, confidence measures and recommended diagnosis [14].

Using artificial neural networks is also popular algorithm for most researchers. Y. Benfu et al created a model to study the use of artificial neural networks in the diagnosis of the smear negative pulmonary tuberculosis. Participants for this study were from TB dispensaries in Jining City, counties including Sishui, Yanzhou and Jinxiang and hospitals affiliated with Jining City. In

total, there were 291 TB patients and 298 non-TB patients with various respiratory illnesses. When the model was applied to the validating sample, the area the model achieved accuracy, sensitivity and specificity at 93.10%, 88.89% and 100%, respectively [15].

A.A. El-Solh et al also developed an artificial neural network using clinical and radiographic information to predict active pulmonary TB. A general regression neural network (GRNN) was used to develop the predictive model. A derivation group of 563 isolation episodes and a validation group of 119 isolation episodes were included in the study. Predictive accuracy of the neural network was compared with clinicians' assessment and the proposed method was able to identify patients with active pulmonary TB better than that of the clinicians' [16].

Aside from above algorithms, different hybrid techniques were also applied. M.K. Osman, M.Y. Mashor and H. Jaafar proposed an approach which employs image processing technique and neural network for the segmentation and detection of tuberculosis bacilli. First, images of tuberculosis bacilli in tissue samples are captured using light microscope after stained with Ziehl-Neelsen staining method. Then colour image segmentation using moving K-mean clustering is used to extract tuberculosis bacilli from the tissue image. Two colour spaces, RGB and C-Y colour, were utilised in order to improve the quality of segmentation and robust against various staining condition. Next, geometrical features of Zernike moments are calculated. From these features, the best features that could detect tuberculosis bacilli with higher accuracy were selected using hybrid multilayered perceptron network. Experimental results demonstrate that the proposed method is efficient and accurate to detect the tubercle bacilli in tissue [17].

A hybrid technique of rough sets and conventional neural networks called rough neural networks was applied by A.A. Bakar and F. Febriyani. A total of 233 tuberculosis patients data was collected from Unit of Health Service Mandau Riau Indonesia. The data set was then carried out for modeling the classification of categories of tuberculosis patients. The experimental results show that average of ten splits accuracy of RNN reaches 92.29% while 90.44% accuracy was achieved through executing conventional neural networks. The rough set classifier, on the other hand, achieved a rate of 92.14% [18].

A novel fuzzy-neural based medical diagnosis system was created by S. Moein et al. The hybrid system does not concern about how to calculate the best membership function for each fuzzy data since the neural network takes care for this job. The real procedure of medical diagnosis which usually is employed by physicians was analyzed and converted to a machine implementable format. Then after selecting some symptoms of eight different diseases, a data set contains the information of a few hundreds cases was configured and applied to a multi layer perceptron neural network. The best performance of 97.5% correct diagnosis was achieved using fuzzified symptoms and an optimized ANN with 15 hidden nodes and after 500 training epochs [19].

Lastly, M. K. Osman et al presented a method using Hu's moment invariants combined with genetic algorithm approach for identification and classification of tubercle bacilli in ZN-stained tissue images. A set of seven Hu's moment has been extracted and used as feature representation for tubercle bacilli. They were able to identify significant input features and the further classify

into two classes; ‘true TB’ and ‘possible TB’. The proposed approach was able to produce good classification performance with an accuracy of 89.64% [20].

III. THEORETICAL FRAMEWORK

3.1. TUBERCULOSIS (TB)

Tuberculosis is a potentially fatal contagious disease that is caused by a bacterial microorganism, the tubercle bacillus or *Mycobacterium tuberculosis*. Although TB can be treated, cured, and can be prevented if persons at risk take certain drugs, scientists have never come close to wiping it out.

The lungs are the primary site of infection but the disease can spread to almost any other organ such as kidneys, bladder, bones, spine, liver, spleen and brain. Tuberculosis is more common in people with immune system problems. People who are HIV-positive face a much higher risk being infected with TB bacilli because their immune systems are compromised by the HIV. Other indicators which may favor the prevalence of TB are bad nourishment, insufficient hygienic circumstances combined with nonexistent medical measures [2].

Diagnosis may be made by skin test, which if positive should be followed by a chest X-ray to determine the status of the infection [21]. Tubercles (tiny lumps) are characteristic findings in TB. Symptoms include low grade fever, coughing, fatigue, and a loss of appetite. Later, hemoptysis or coughing up blood, may occur [1]. Treatment of active tuberculosis involves a course of antibiotics and vitamins that lasts about six months. It is important to finish the entire treatment, both to prevent reoccurrence and to prevent the development of drug-resistant tuberculosis [21].

3.1.1. **Types of Tuberculosis.** Tuberculosis can be classified depending on the target organ is. It can be categorized into pulmonary and extrapulmonary. Tuberculosis can alsbo be categorized depending on the resistance of infection against anti-TB drugs.

(1) *Pulmonary Tuberculosis.* There are several types of pulmonary tuberculosis. Tuberculosis Pneumonia is an uncommon type of TB presents as pneumonia and is very infectious. It occurs most often in extremely young children and the elderly. Cavitary TB involves the upper lobes of the lung. The bacteria cause progressive lung destruction by forming cavities, or enlarged air spaces. This type of TB occurs in reactivation disease. On the other hand, Miliary TB is a form of disseminated TB. Miliary describes the appearance on chest x-ray of very small nodules throughout the lungs that look like millet seeds. Laryngeal TB is the tuberculosis that infect the larynx, or the vocal chord area [1].

(2) *Extrapulmonary Tuberculosis*. Tuberculosis lymphadenitis is TB of the lymph nodes, usually along the neck. The symptoms are the formation of masses along the neck, and if the disease is advanced the mass may burst and form a draining sinus. Cutaneous tuberculosis is TB of the skin or mucous membrane from an external source of mycobacteria. Tuberculosis Pericarditis is the a condition in which the membrane surrounding the heart (the pericardium) is affected. Gastrointestinal tuberculosis is TB of the gastrointestinal tract: mouth, oesophagus, stomach, small and large intestine, and the anus. Lastly, Tuberculosis Meningitis infects the meninges (the main membrane surrounding the brain and spinal cord) which can lead to permanent impairment and death. This form of TB can be difficult to discern from a brain tumor

because it may present as a focal mass in the brain with focal neurological signs [1].

3.2. ARTIFICIAL NEURAL NETWORK (ANN)

An Artificial Neural Network is a computer algorithm that attempts to simulate biological neurons [8]. A neural network is an interconnected group of artificial neurons that uses a mathematical or computational model for information processing based on a connectionistic approach to computation. In most cases an ANN is an adaptive system that changes its structure based on external or internal information that flows through the network [22].

The idea of "neural networks" has been introduced since the early 1940s, when Warren McCulloch and Walter Pitts presented the first model of artificial neurons. They developed a simplified model of the neuron, called the MP neuron, centered on the idea that a nerve will fire only if its threshold value is exceeded. The MP neuron functioned as a sort of scanning device that read predefined input and output associations to determine the final output. The MP neuron does not have the ability to learn and is very limited compared to the actual biological neurons from where it was modeled. Hence, a type of artificial neural network capable for learning called the perceptron was invented in 1957 at the Cornell Aeronautical Laboratory by Frank Rosenblatt. The perceptron was able to learn by adjusting the weights on its connections between neurons and was able to solve simple problems [8].

Since then, newer and more sophisticated neural network architectures have been

presented. Through the years, neural network applications matured from simple to more complex and computationally expensive problems like time series analysis, data mining, image processing etc. Machine learning using neural networks has proven to be particularly useful in applications where the complexity of the data or task makes solving or computing them by hand impractical, or nearly improbable. Some common application of neural network includes classification, pattern recognition and prediction. Pattern recognition is perhaps the most common use for neural networks. For this type of problem, the neural network is presented a pattern then the system attempts to determine if the input data matches a pattern that it has been trained to recognize [8]. Classification, on the other hand, is a technique used by neural networks to organize input samples and categorize them into groups.

A neuron is an information-processing unit that is fundamental to the operation of a neural network [23]. A neuron is a member of one layer of the network and has connections to the next layer. Figure 3.1 shows the structure of a neuron, which will form the basis for a neural network.

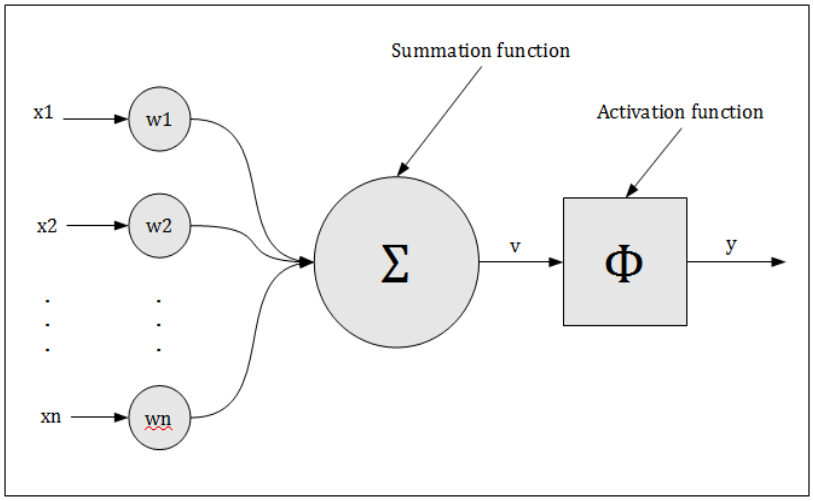


Figure 1. Graphical Representation of a Neuron

Mathematically, a neuron can be described by the following equation:

$$v = \sum_{i=1}^{N} X_i W_i \quad \text{and} \quad y = f(v)$$

where *v* is the weighted sum of all the inputs $X_1, X_2, \ldots, X_n$; *f or* Φ as the activation function; and *y* as the final output of the neuron.

3.2.1. **Activation function**. Activation functions are mathematical functions where the output of a neuron layer is passed through. The activation function ensures that the output is in the correct range [8]. Common choices for activation functions include the threshold, hyperbolic tangent and the sigmoid function. It is denoted by Φ.

(1) *Threshold Function (Heavyside function).* A neuron employing this type of activation function is normally referred to as a McCulloch-Pitts model. The model has an all-or-none property [23].

$$f(x) = 0, x<0$$
$$f(x) = 1, x>0$$

(2) *Sigmoid function.* This is the most common form of activation function used in artificial neural networks. The equation for the sigmoidal function is:

$$f(x) = \frac{1}{1+e^{-x}}$$

For backpropagation algorithm, the derivative of the function is required to adjust weights of the previous layers. The derivative of the sigmoidal function is:

$$f(x) = \frac{e^{x}}{(1+e^{x})^{2}}$$

A sigmoidal threshold function is only capable of producing positive output ranging from [0, +1]. If negative output is required, then the hyperbolic tangent threshold function should be considered.

(3) *Hyperbolic tangent function.* Unlike the sigmoid function, hyperbolic tangent yields output values in the range [-1, +1]. The equation of hyperbolic tangent is as follow:

$$f(x) = \frac{e^{2x}-1}{e^{2x}+1}$$

The derivative of the function is:

$$f(x) = 1-e^{\frac{e^{2x}-1}{e^{2x}+1}}$$

3.2.2. **Training Paradigms.** There are many ways to train neural networks. Neural network training methods generally fall into the categories of supervised, unsupervised, and various hybrid approaches.

(1) *Supervised Training.* A training method where the network is trained by providing it with input and matching output patterns. These input-output pairs can be provided by an external teacher, or by the system which contains the network (self-supervised) [24].

(2) *Unsupervised Training.* A training method that does not provide the neural network with expected outputs. In unsupervised learning or self-organisation, an outputμ unit is trained to respond to clusters of pattern within the input . The system is supposed to discover statistically salient features of the input population. Unlike the supervised learning paradigm, there is no a priori set of categories into which the patterns are to be classied rather the system must develop its own representation of the input stimuli [24].

3.2.3 **Error Calculation**. Error calculation is an important aspect of any neural network. The goal of all training algorithms is to minimize the rate of error the neural network may commit. There are two values that must be considered in determining the rate of error for supervised training. First, we must calculate the error for each element of the training set as it is processed. Second, we must calculate the average of the errors for all of the elements of the training set across each sample [8].

Basic Output Error Calculation. The output error basically the difference between the actual input of the neuron and the ideal input provided by the data set. This value is rarely used for any purpose other than as a steppingstone in the calculation of the root mean square error for the entire training set.

$$outputerror = ideal - actual$$

Root Mean Square (RMS). The RMS method is used to calculate the rate of error for a

training set based on predefined ideal results. The RMS method is effective in calculating the rate of error regardless of whether the actual results are above or below the ideal output. To apply RMS to the output of a neural network, we consider the equation:

$$rms = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(outputerror_i)^2}$$

3.2.4. **Learning Algorithms.**

Delta Rule or Least Mean Squared Error (LMS). A training technique that adjusts a network's weights based on differences between output and the ideal output. It is derived from units that uses the linear threshold activation function.

$$\Delta W_{ij} = \eta(\delta) X_i$$

where ΔW_{ij} is the weight update, δ as the output error (ideal – actual), η as the learning rate and x_i as the input.

Generalized Delta Rule or Back Propagation (BP). Back propagation is a method for training neural networks. It works by analyzing the output layer and evaluating the contribution to the error of each of the previous layer's neurons. The previous layer is adjusted to attempt to minimize its contribution to the error. This process continues until the program has worked its way back to the input layer.

The delta rule presented above can be generalized to be used in a set of network whose activation functions are non-linear such as the sigmoid and the hyperbolic tangent functions. The

generalized delta rule could be shown in the equation:

$$\Delta W_{jk} = \eta \sum_{a}(\delta_k)\cdot f'\left(\sum_{b} x_i w_i\right)\cdot x_j$$

where ΔW_{jk} is the weight update, δ_k as the output error (ideal – actual) in output layer, η as the learning rate and x_i as the input in the input layer, x_j as the output of the hidden layer, and f' as the derivative of the activation function used.

3.2.5. **Network Architectures.**

Single layer feed forward neural network. The simplest form of a layered network, consisting of an input layer of source nodes that project onto an output layer of neurons. The network is strictly feedforward, no cycles of the information are allowed. Figure 3.1.4. shows an example of this type of network. The designation of single-layer refers to the output layer of neurons, the input layer is not counted since no computation is performed there [23].

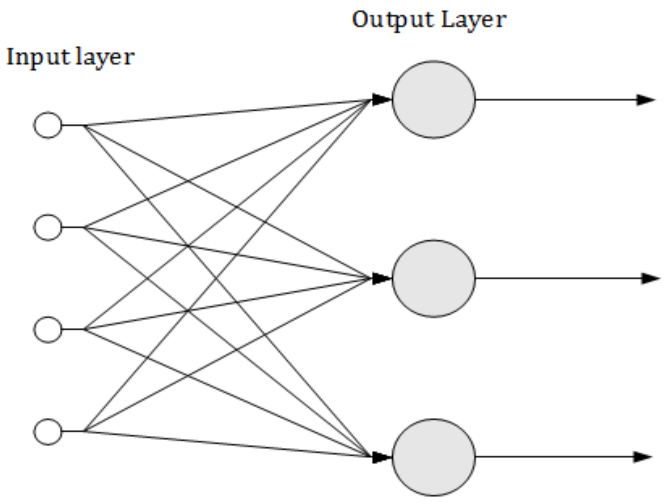


Figure 2. Single Layer Feed forward network.

Multi layer feed forward neural network. This class of feed-forward neural networks

contains one or more hidden layers, whose computation nodes are correspondingly called hidden neurons. The hidden neurons intervene between the input and output layers, enabling the network to extract higher order statistics [23].

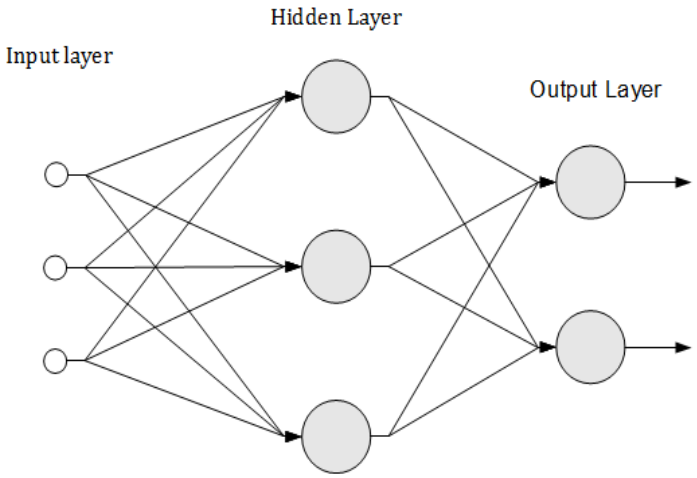


Figure 3. Multi layer feed forward neural network with one hidden layer.

Recurrent neural networks. A recurrent neural network has a similar architecture to that of a multi-layer feed-forward neural network, but contains at least one feedback loop. This could be self-feedback, a situation where the output of a neuron is fed-back into its own input, or the output of a neuron could be feed to the inputs of one or more neurons on the same or preceding layers [23].

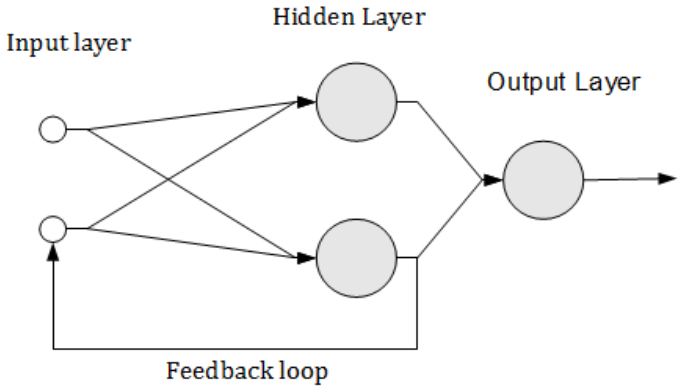


Figure 4. Recurrent Neural Network

Self Organizing Maps (SOM). A type of neural network that is trained using unsupervised learning to produce a low-dimensional (typically two-dimensional), representation of the input space of the training samples, called a map. This map can be used to classify new patterns outside of the training set [8].

3.3. DECISION SUPPORT SYSTEMS (DSS)

The concept of decision support has evolved from two main areas of research: the theoretical studies of organizational decision making done at the Carnegie Institute of Technology during the late 1950s and early 1960s; and the technical work on interactive computer systems, mainly carried out at the Massachusetts Institute of Technology in the 1960s. It is considered that the concept of decision support system became an area of research of its own in the middle of the 1970s, before gaining in intensity during the 1980s [25].

The range of fields using decision support systems had increased as newer DSS frameworks had been introduced. Certain fields include banking and finance, business and management, machine and engine monitoring, forest management, and agriculture production. Medical applications of decision support systems have been intensively used in the past decade. Software companies and medical institutions have exerted much effort to produce viable decision support systems to cover all aspects of clinical tasks.

Clinical decision support systems (CDSS) are active knowledge systems, which use two or

more items of patient data to generate case-specific advice. The main purpose of a CDSS is to assist clinicians at the point of care. CDSS are interactive computer programs designed to assist health professionals with decision-making tasks. The clinician interacts with the software utilizing both the clinician's knowledge and the software to make a better analysis of the patients data than either human or software could make on their own. Typically the system makes suggestions for the clinician to look through and the clinician picks useful information and removes erroneous suggestions [26].

There are two main types of CDSS:

(1) *Knowledge-Based.* Most CDSS consist of three parts: the knowledge base, inference engine, and mechanism to communicate. The knowledge base contains the rules and associations of compiled data which most often take the form of IF-THEN rules. The inference engine combines the rules from the knowledge base with the patient's data. The communication mechanism will allow the system to show the results to the user as well as have input into the system .

(2) *Non knowledge-Based*. Since these kind of systems do not use a knowledge base, they use artificial intelligence and machine learning to find the needed output from the set of input paramaters provided by the user. Nonknowledge-based systems learn from past experiences to find patterns in the clinical data. Typical nonknowledge-based CDSS use genetic algorithms and/or artificial neural networks [26].

IV. DESIGN AND IMPLEMENTATION

A. CONTEXT DIAGRAM

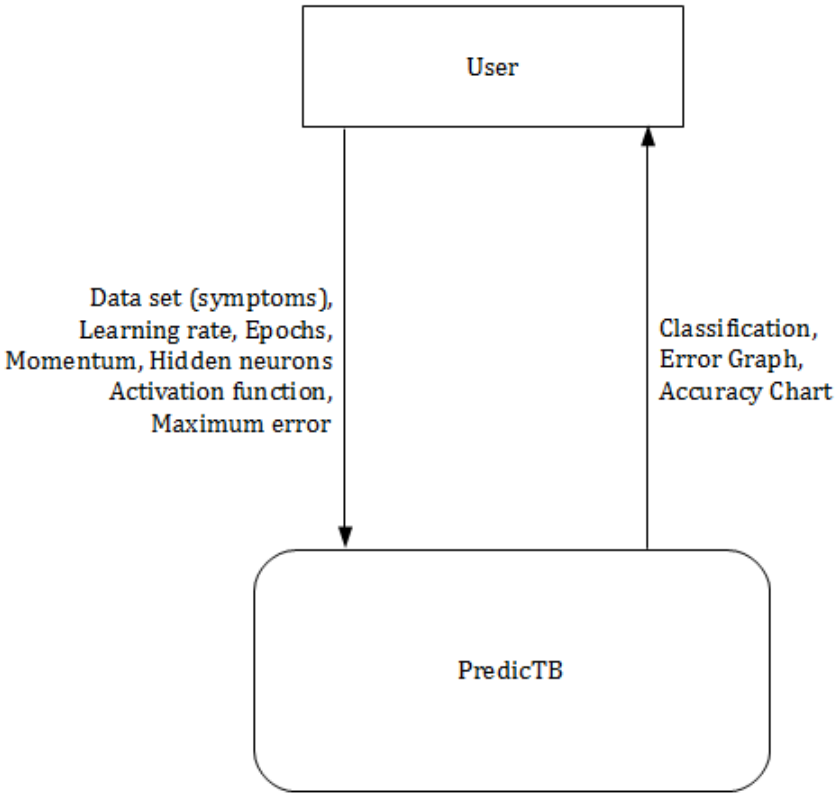


Figure 5. Context Diagram, PredicTB

The context diagram above shows the general interaction of the user to the system. The user needs to input the data set needed for training, and as well as evaluating, the network. Data set contains symptomatics for the diagnosis of tuberculosis. Some symptomatics include fever, coughing, sputum discharge, hemoptysis, among others. The user also needs to input training parameters and other variables like activation function and number of hidden neurons. Output of the system includes the diagnosis of the disease and some charts and graphs for visualizing the data.

B. PROCESS FLOW DIAGRAM

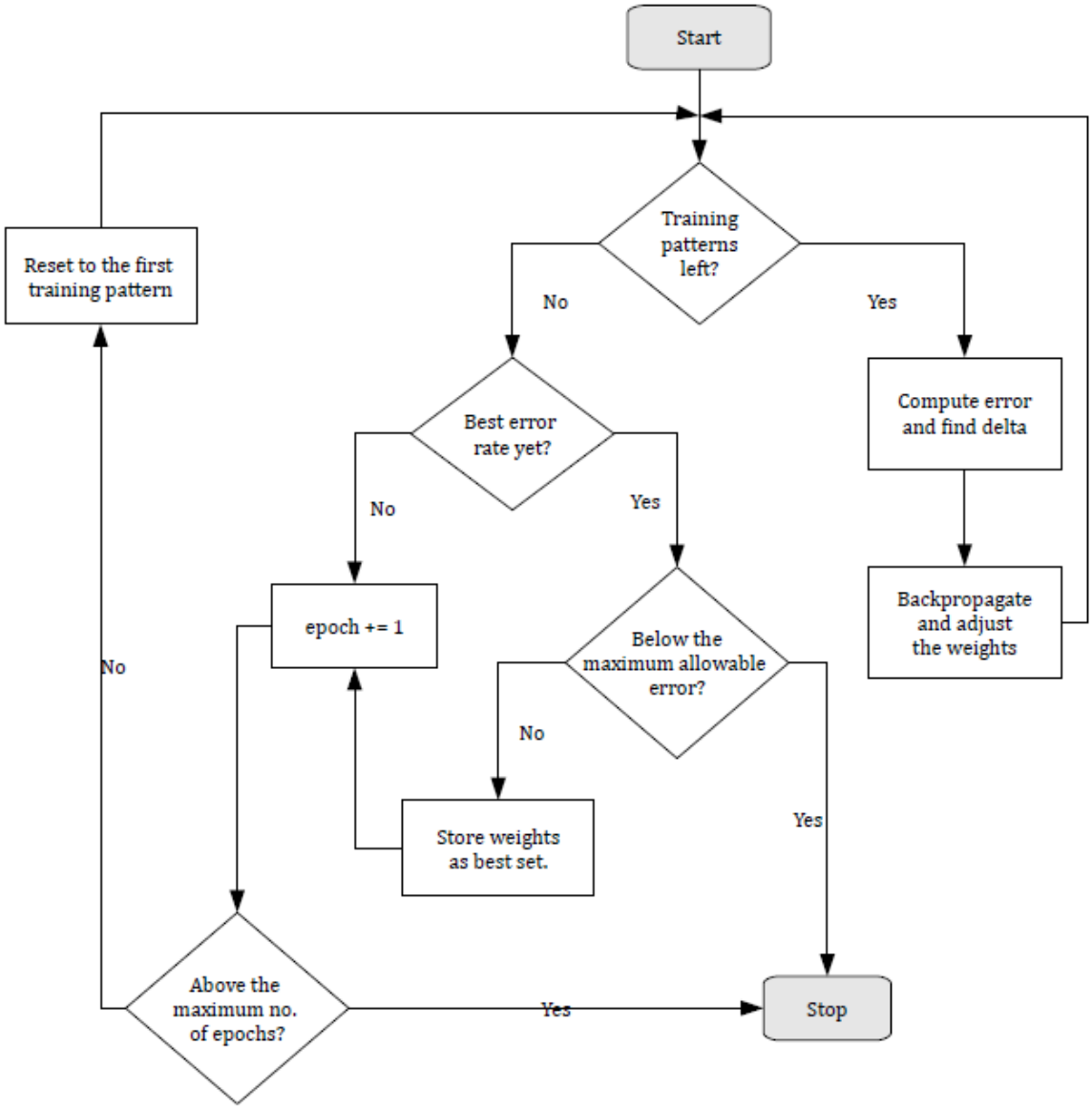


Figure 6. Supervised Training with back propagation algorithm

The above figure shows the flow chart of supervised training with back propagation. The generalized delta rule will be applied to update the weights and thresholds of the network. Convergence is reached if the root mean squared error is below the maximum error or whenever the maximum number of epochs is achieved.

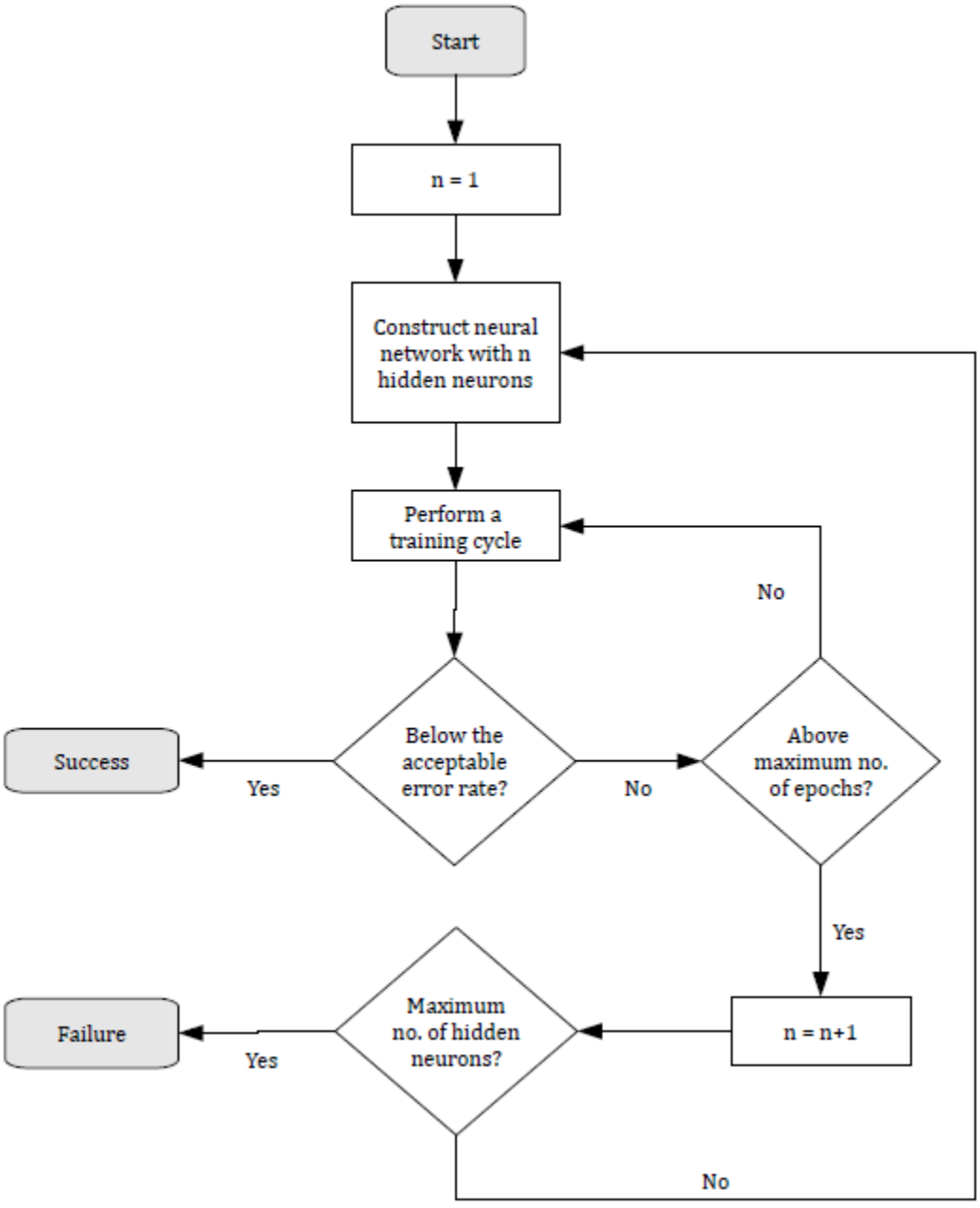


Figure 7. Incremental Pruning

The imcremental pruning algorithm described in figure 7 will be used if the user will not be inputing a predefined number of neurons. Pruning will start with one neuron and eventually increase if no progress is achieved with the current network.

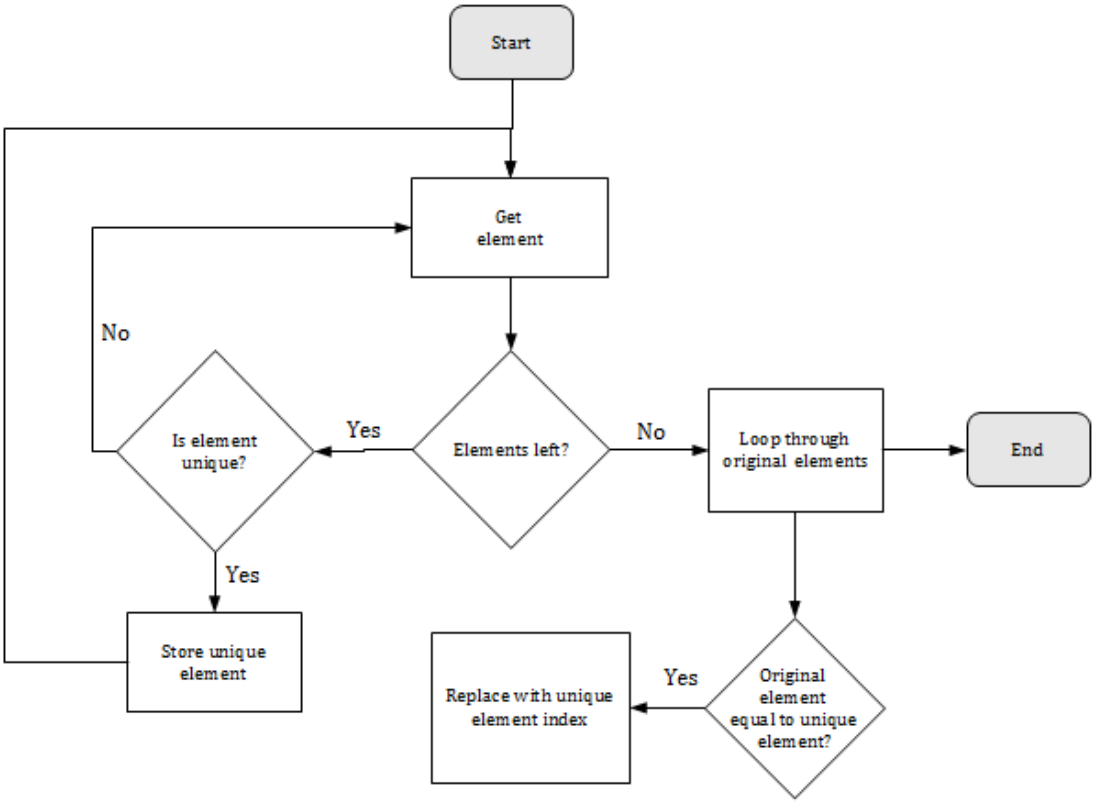


Figure 8. oneOf Mapping

OneOf mapping (figure 8) will be used for nominal-valued columns. We first get the number of unique elements in the original data and store this into an array. We loop through the original data and compare it to the array of unique elements. If equal, we replace the element with the index of the unique element. Once mapped, data can now be normalized using the algorithm described in the figure below.

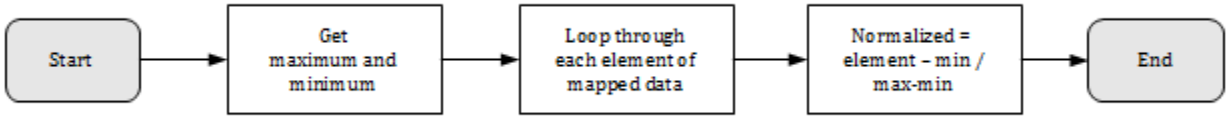


Figure 9. Normalization

C. DATA DLOW DIAGRAM (DFD)

TOP-LEVEL DFD

The system can basically be divided into three main modules: managing the data set, training the network and evaluation or prediction of patient data. Managing the data set includes adding or importing patient data, mapping nominal values and normalization. Training the network includes propagation and backpropagation to achieve final network. The evaluation module can either be patient data prediction or accuracy testing of the created network. The data flow diagram and its corresponding subexplosions are the following:

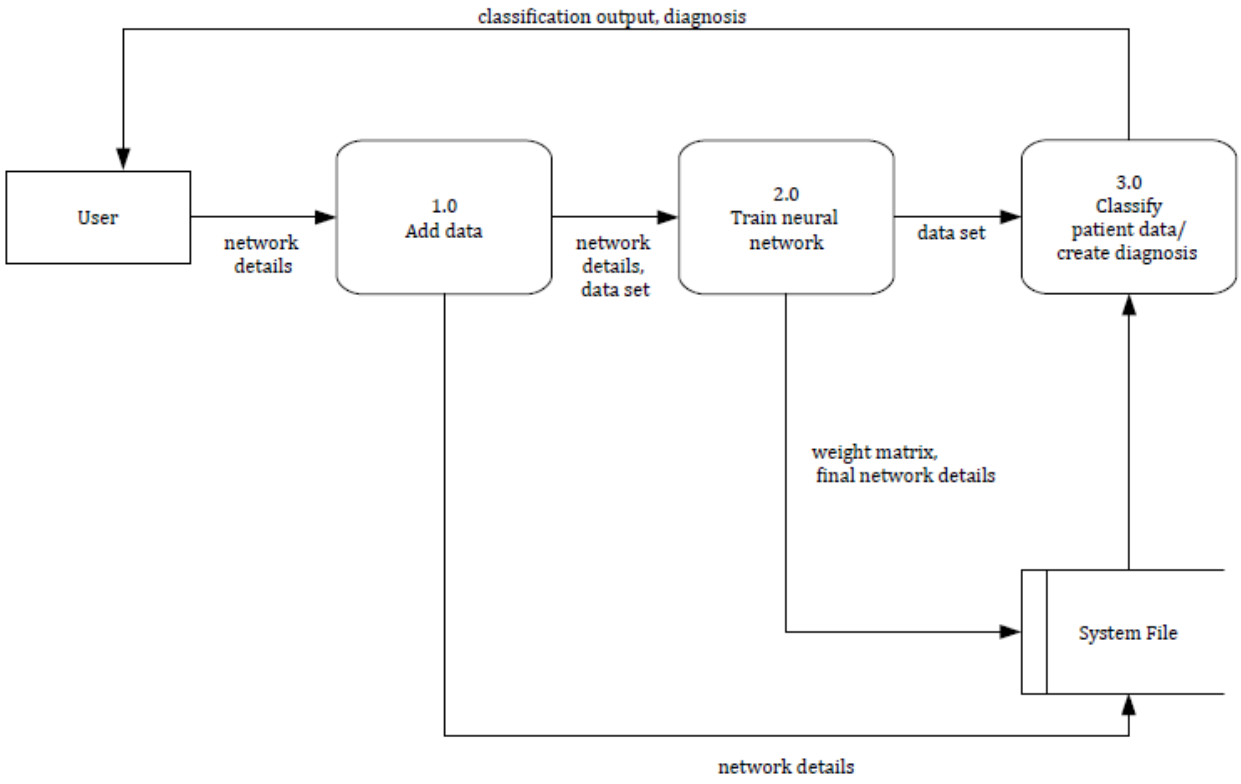
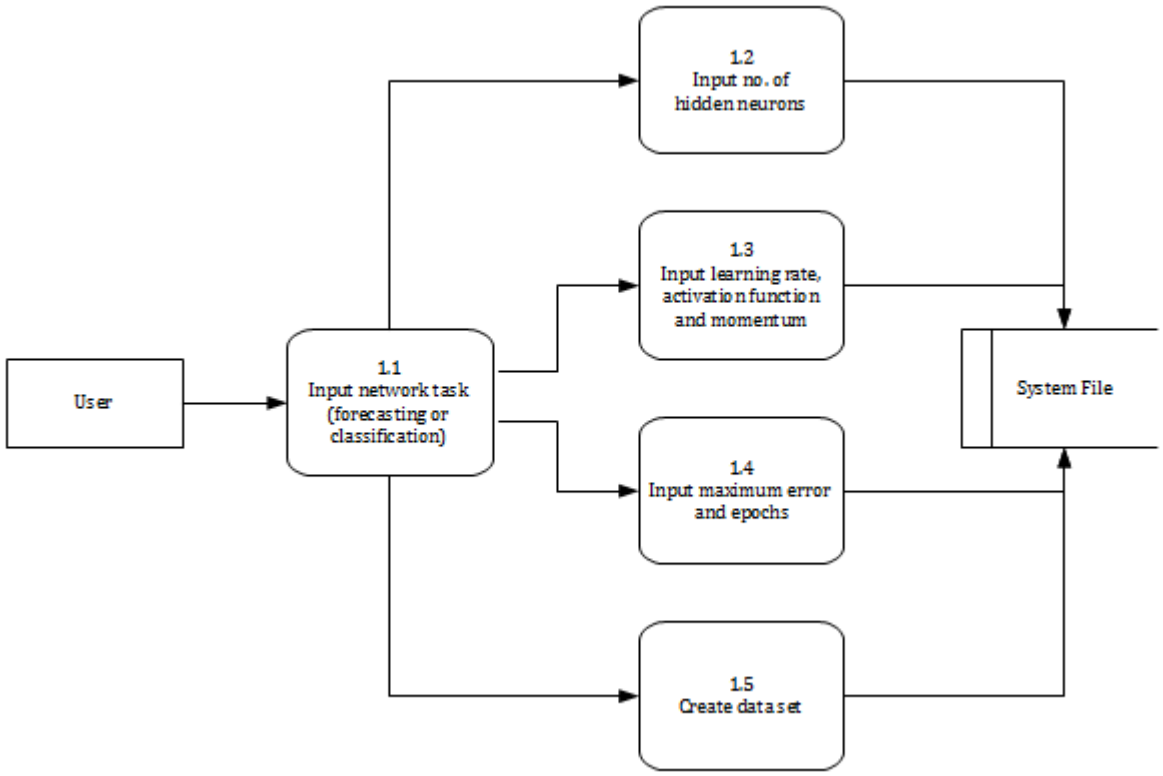


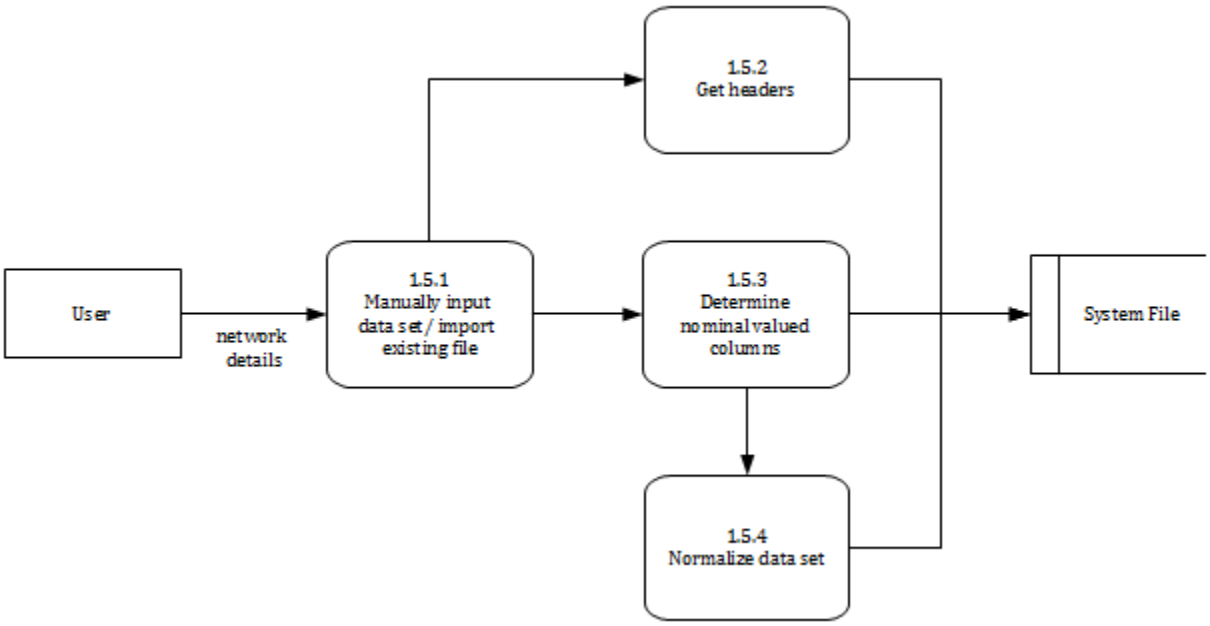
Figure 10. Data Flow Diagram, PredicTB

DFD SUBEXPLOSIONS

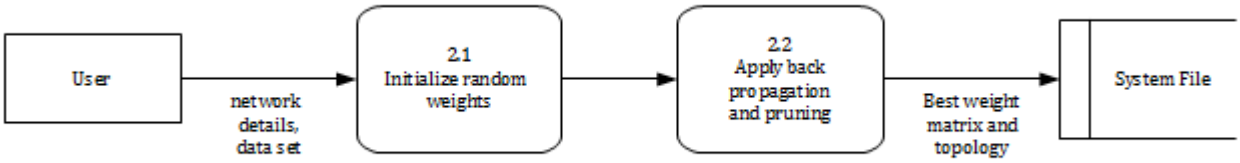
Subexplosion: Level 1 – 1.0 Add data



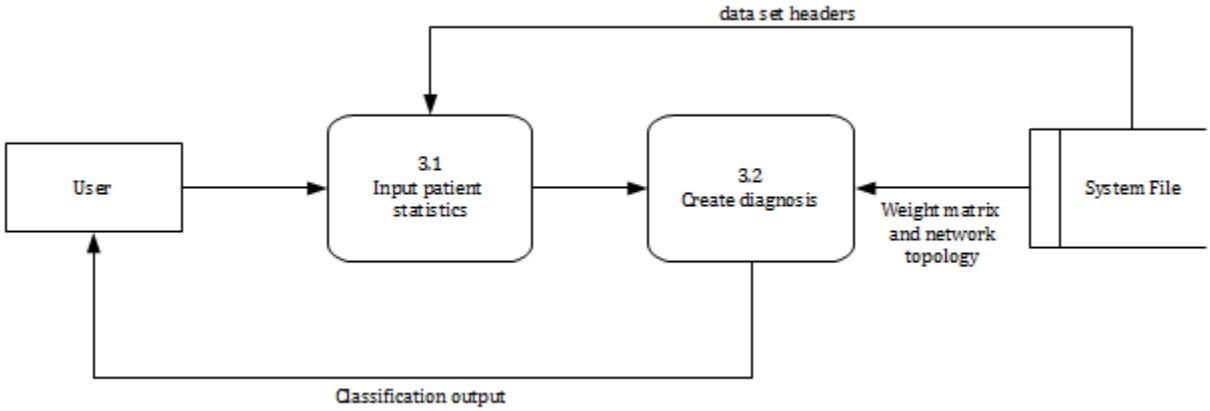
Subexplosion: Level 2 – 1.5 Create data set



Subexplosion: Level 1 – 2.0 Train Neural Network



Subexplosion: Level 1 – 3.0 Classify patient data / create diagnosis



D. DATA SET

Various data sets can be used into the system. Data can vary depending on the available medical records as well as the measures used by the physicians to create a diagnosis for tuberculosis. The data set used for training may or may not include the following TB indicators.

1	Coughing
2	Sputum Discharge
3	Fever
4	Weight Loss
5	Tiredness
6	Chest Pain
7	Hemoptysis
8	Dyspnea
9	Back Pain
10	HIV

V. RESULTS

PredicTB is composed of three tabs: Data Set, Training and Evaluation. A splash screen will be shown once the software is opened. User can create a new project or open an existing one by clicking the File menu bar at the top. Keyboard shortcuts (Ctrl+N or Ctrl+Shift+O) are also available.

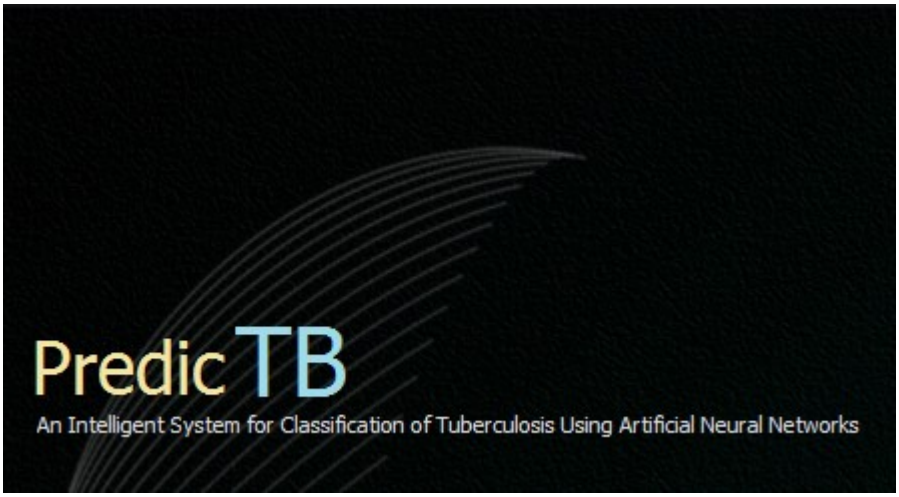


Figure 11. Splash Screen, PredicTB

Once the user created or opened a project file, first task is to create a data to be used in order to train the system. The user can choose between importing an existing file or create a new one using the built-in Excel-like sheet. If the user opts to import file, he can only choose between an Microsoft Excel file (.xls) or a comma-separated value (.csv) file. Other file formats will not be accepted by the system. All cells must contain values, otherwise, creating the mapped data will not be possible.

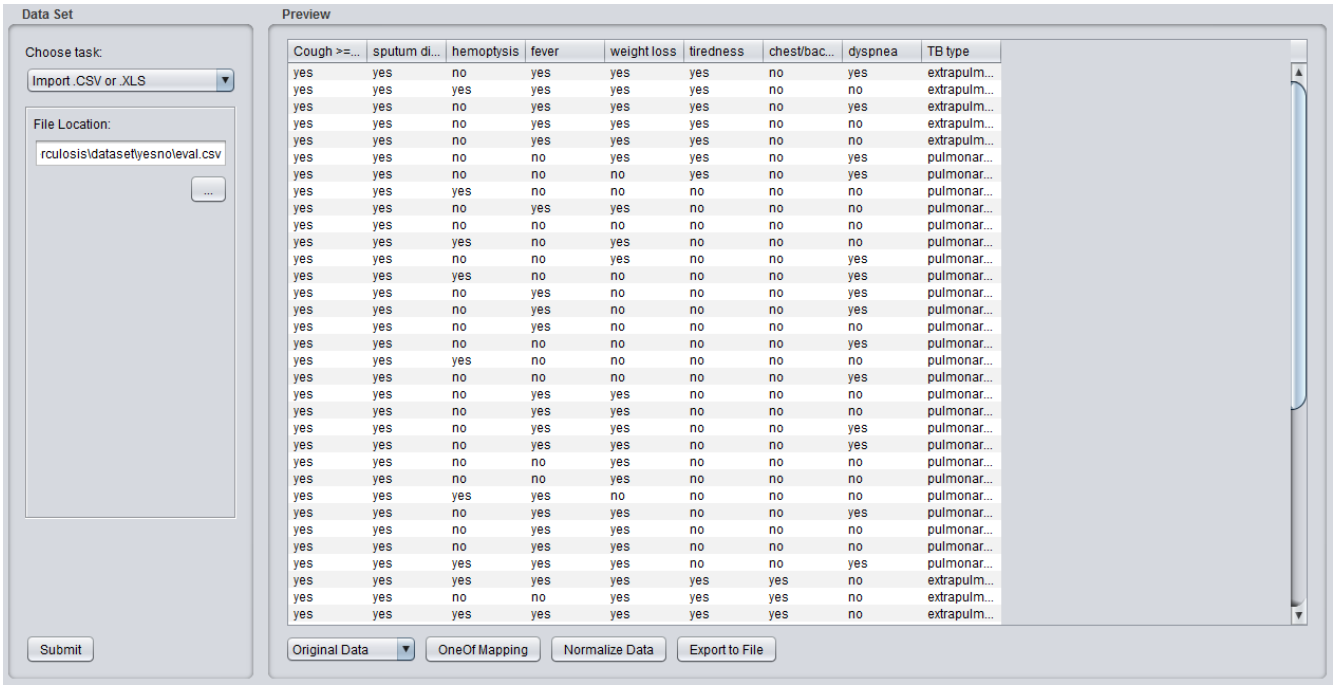


Figure 12. Data Set Tab, PredicTB

If creation of data set is complete, the user must map all nominal-valued columns and convert it to numbers. To do this, the user must click OneOf Mapping Button. Another form will pop-up and the user must check all columns that need to be converted.

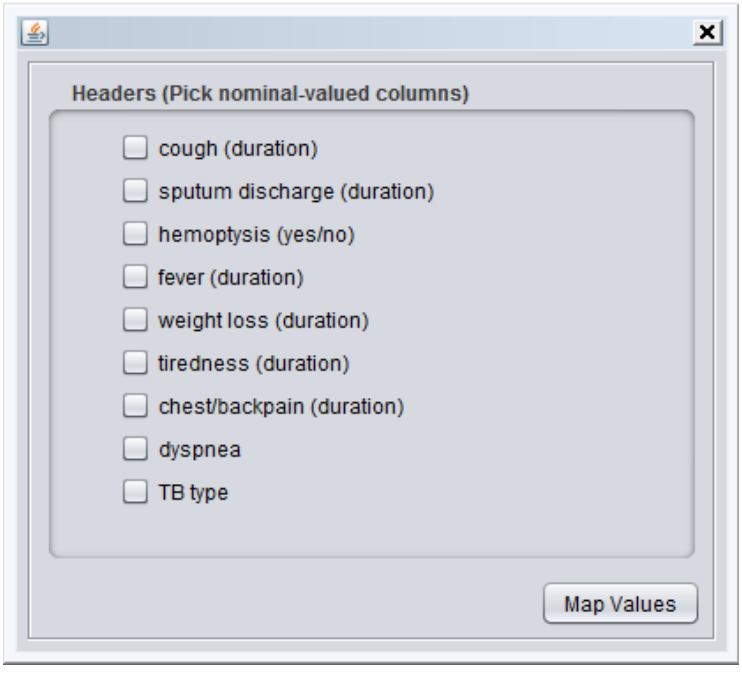


Figure 13. Mapping Form, PredicTB

After OneOf Mapping, normalization should be done to limit the range of the data values to [0, 1]. This makes the data set ready for machine learning. At this stage, the user must input the maximum and minimum values of each ordinal-valued columns. The maximum and minimum values for nominal-valued columns are already computed during the OneOf mapping process. Also, in this stage, the user must choose which among the columns will be predicted and will serve as the output class.

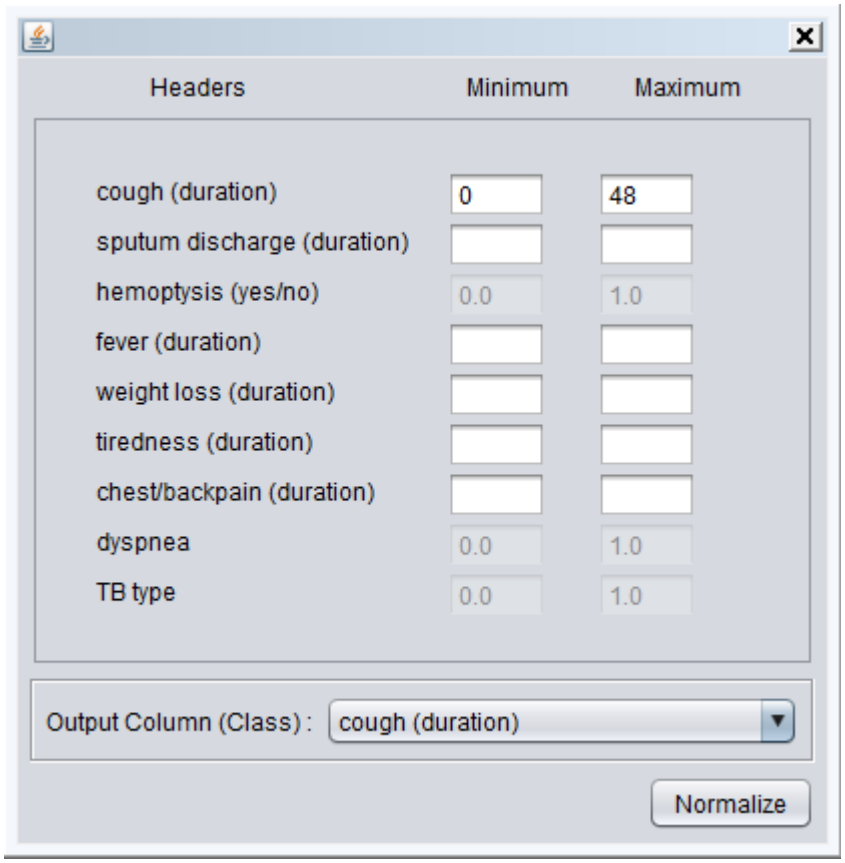


Figure 14. Normalization Form, PredicTB

An 'Export' button is provided beside the 'Normalize' button just in case the user wants to save the data into a specific file. They user may also view the original data, mapped data and normalized data by clicking on the combo box placed under the preview panel for the data set.

This concludes the 'Data set' tab of the system and the user may start moving to the 'Training' tab. On the training tab, the user can find a button for editing the training coefficients and topology that will be used for training the network. Users can use default values for the system or may want to input their own for optimization.

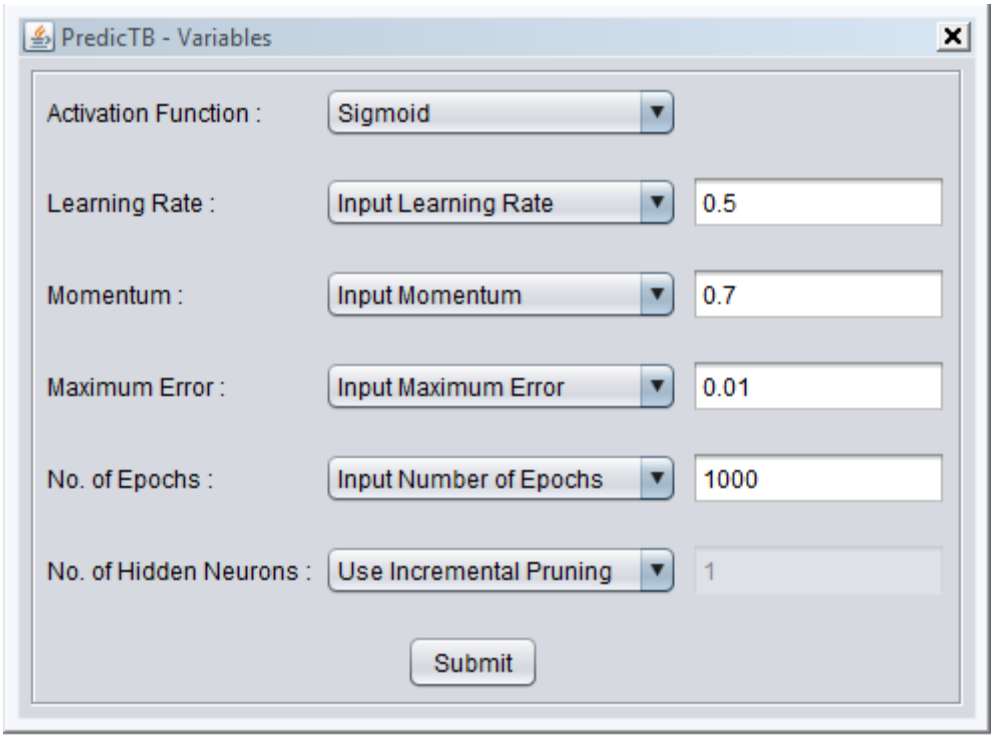


Figure 15. Variable Form, PredicTB

If done, the user can start training the network. The epoch number and current error is shown in a text area. Pausing and stopping the training process is also possible. Pausing saves the last known variables including hidden neuron count (especially for incremental pruning) and the weights and thresholds of each network neuron. Stopping, on the other hand, resets all the values of the neural network. The user may also see the graph of how the error propagates as time increases by clicking the button 'View Error Graph'. The error shown is the total root mean squared error of all the neurons in the network.

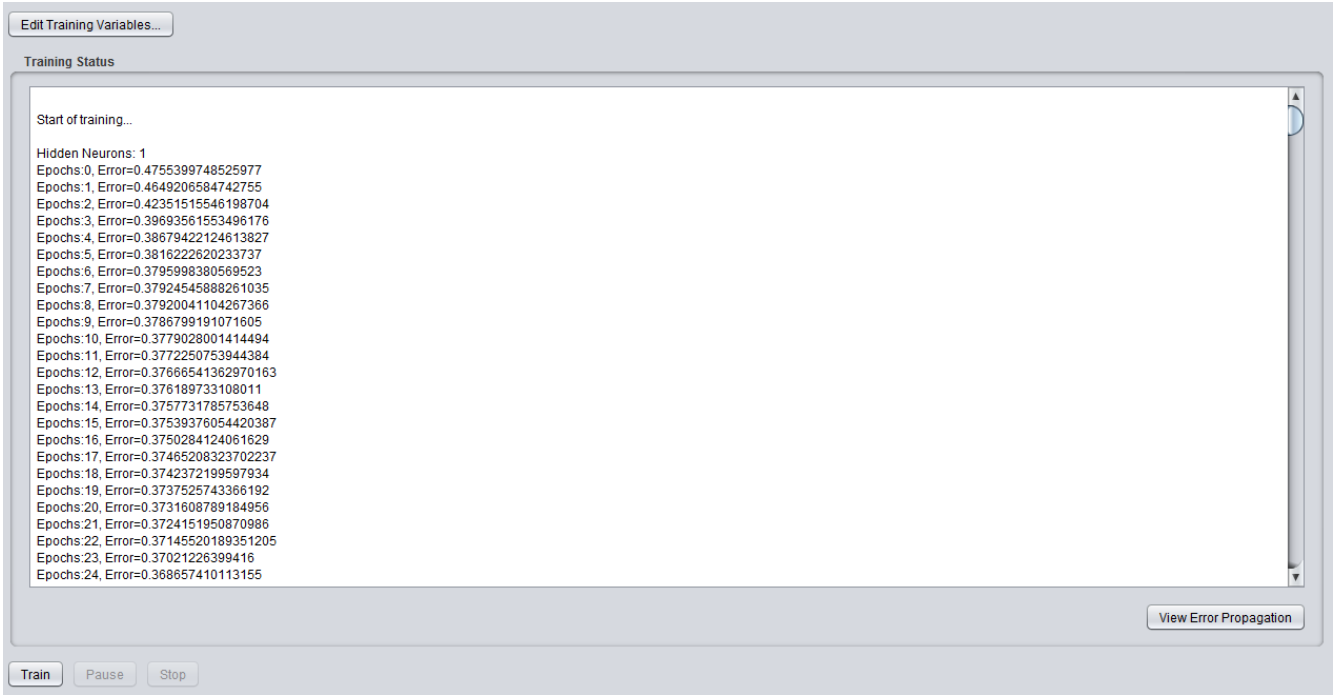


Figure 12. Training Tab, PredicTB

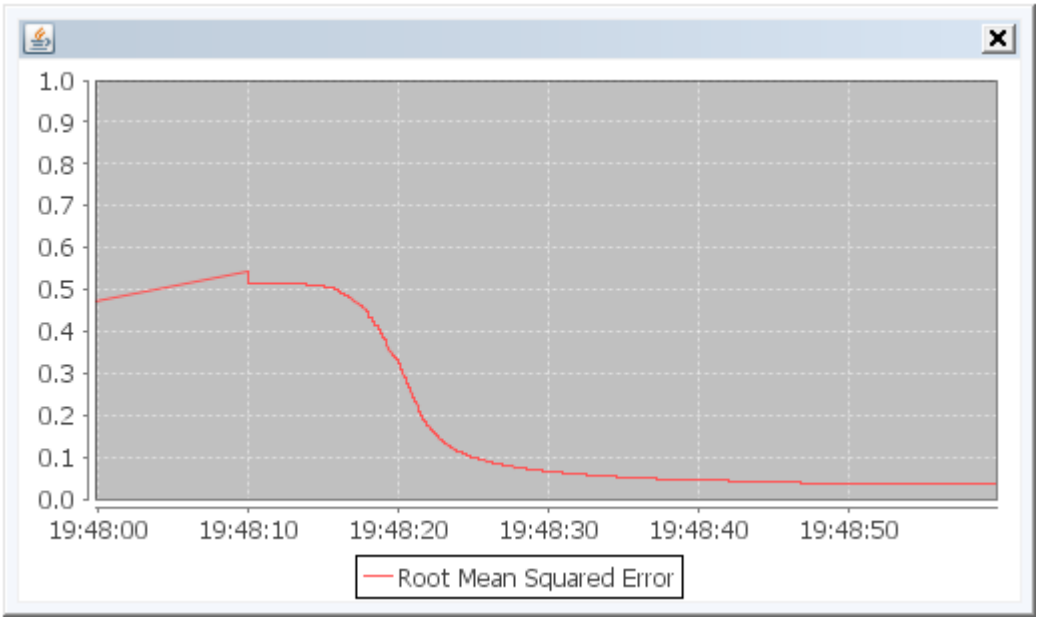


Figure 13. Error Propagation Graph, PredicTB

The user may wait for the convergence of the network before the evaluating it or may start the evaluation once the system is paused. On the 'Evaluation' tab, the user can choose for

accuracy testing or patient classification. Accuracy testing will serve as the beta testing for the network before the actual deployment of the project file. The evaluation file for accuracy testing must have the same format as the original file, also containing the actual values. Upon evaluation, the system checks if the predicted value and the actual value is the same and computes for the percentage of correct classification. Patient classification, on the other hand, is used when the project is already deployed and the user does not really know the exact output of the patient data.

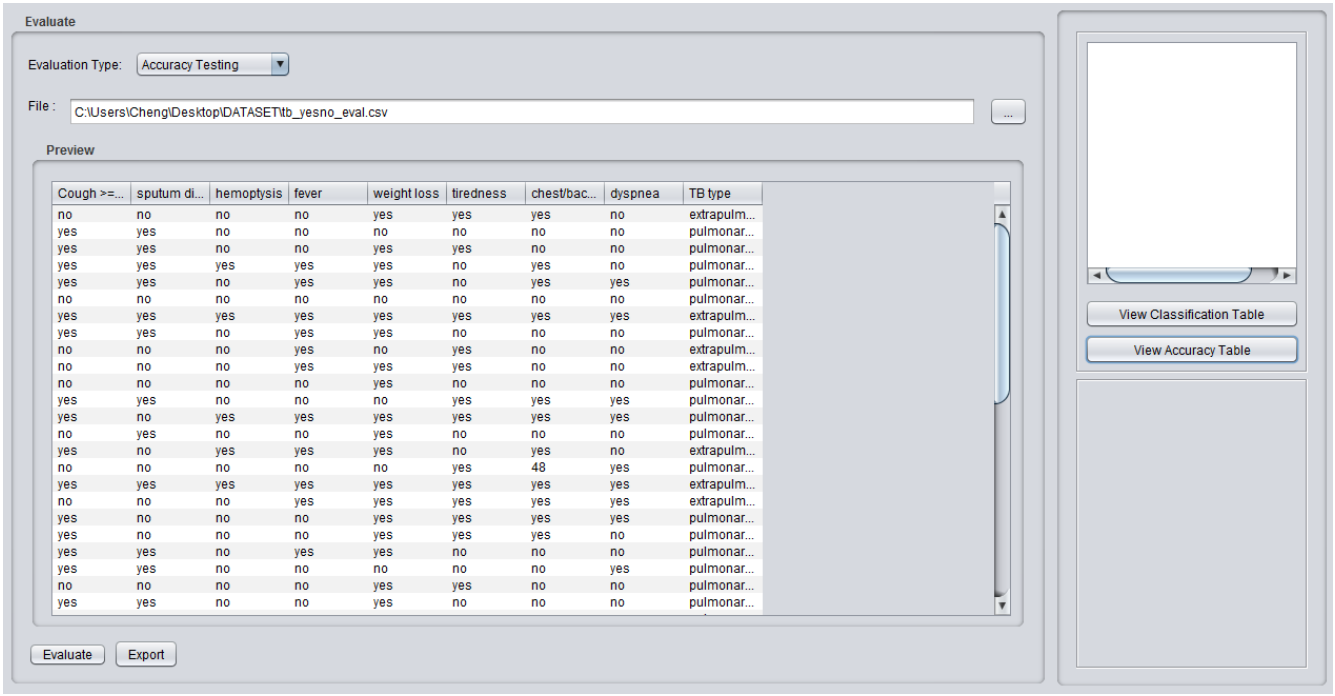


Figure 14. Evaluation Tab, PredicTB

For accuracy testing, an accuracy chart aside from the statistics will be shown by clicking through the 'View Accuracy Table' button. For patient classification, this button will be disabled since no actual output is provided. For both case, a classification table is shown to see how diverse the evaluated data is. The user may wish to evaluate other data by inputting a different file.

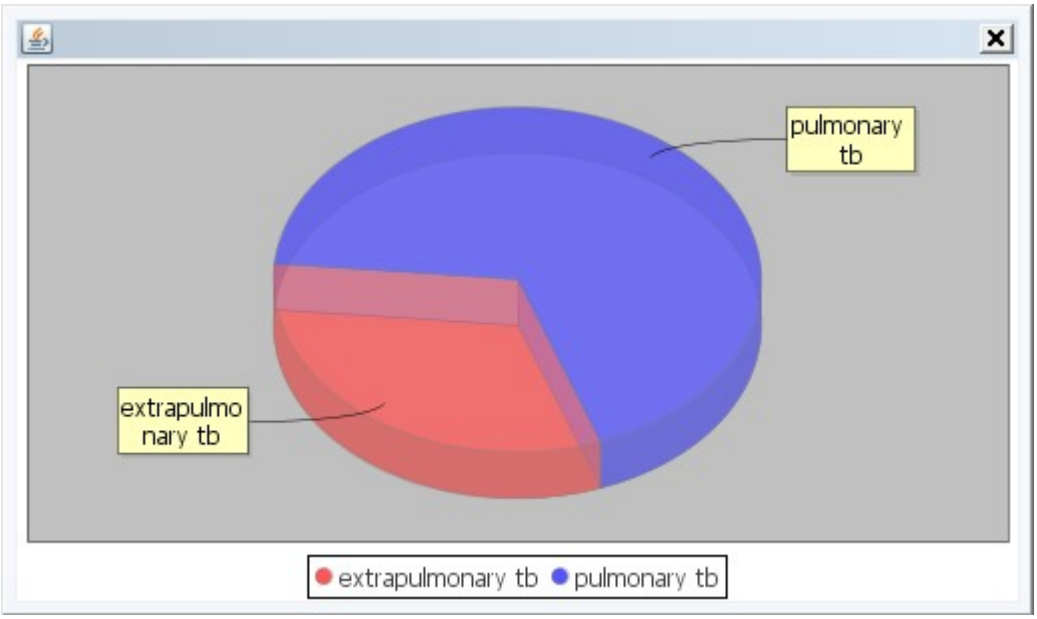


Figure 15. Classification Chart, PredicTB

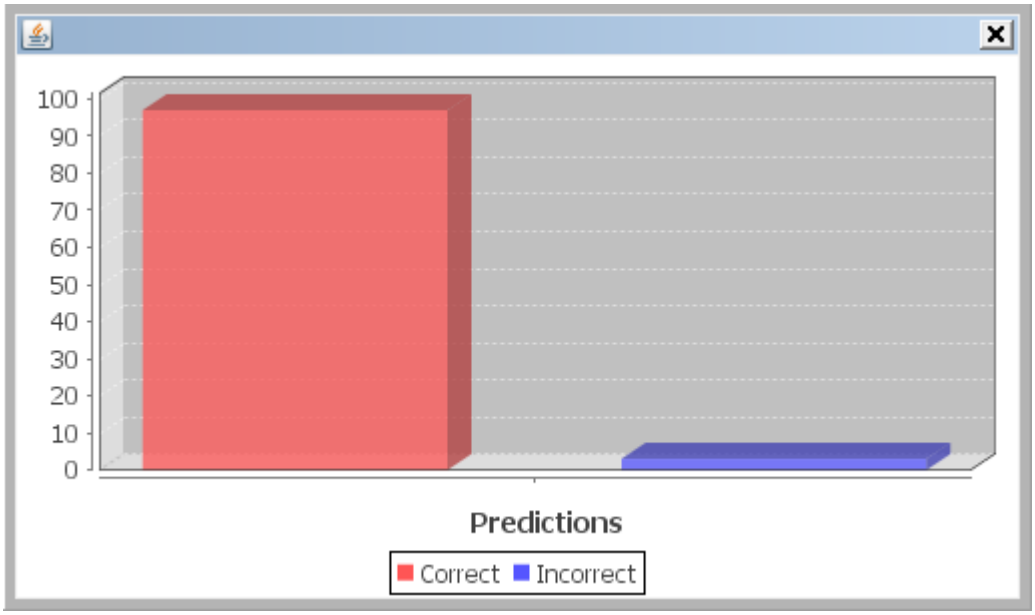


Figure 16. Accuracy Chart, PredicTB

VI. DISCUSSION

The data set used to train and evaluate the system was obtained from the UP Prime TB DOTS Clinic of the Philippine General Hospital. The data set contains symptoms from 174 patients covered by the TB DOTS program. There are eight variables included in the data set, seven are considered as input neurons and one as the output neuron. The input variables consists of coughing, sputum discharge, hemoptysis (coughing up blood), fever, weight loss, chest and/or back pain, and dyspnea (shortness of breath). The output variable is the type of tuberculosis and could be either pulmonary or extrapulmonary.

We first created a sample project which handles both numerical and nominal values. Duration for symptoms like coughing, sputum discharge, fevee, weight loss and chest/back pain are indicated in the data. We limit the data from 0 to 48 weeks depending on the length of time the patient experienced the symptom. On the other hand, symptoms like dyspnea and hemoptysis are classified into either absence or presence on the patients.

The data is divided into two disjoint parts – training set and evaluation set. 117 patients were included in the training data while 57 are included for evaluation. The network is trained with 0.5 learning rate and 0.7 momentum. A maximum error of 0.01 is set to be the convergence factor of the network. We started training the system with one hidden neuron, adding additional neuron once the system has converged to find the optimal network topology. We stop adding the neurons if the number of epochs is greater than the epochs obtained from previous training

process. For one and two hidden neurons, the error propagation graph fluctuates and underfitting and overfitting is observed. The results of the whole training process and the corresponding epochs before convergence is achieved is shown in the table below.

Hidden Neurons	Epochs
3	2426
4	2320
5	2246
6	2194
7	2166
8	2027
9	1627
10	2049

We can see that using 9 hidden neurons converged faster and has the least number of epochs, therefore the optimal network topology for the specific data used. After training, the evaluation data is passed on to the system. Out of the 57 evaluation data, 17 are extrapulmonary and 40 are pulmonary TB patients. Evaluation results show that 48 were correctly mapped to its correct classification while the remaining 9 gave the wrong results. The accuracy rate achieved is 84.21%.

We created another project which handles nominal data only. The duration of the symptoms were classified whether they are greater than or equal to two weeks or not. Since the data was less complex than the first project, we tried training the network with 0.001 as the maximum error. This will result to far greater number of iterations compared to the previous project.

We repeat all the process we did on the first sample project. Again, overfitting and underfitting were observed when only one or two hidden neurons are used. The following results show that using 8 neurons in the hidden layer converged faster with only 12148 iterations.

Hidden Neurons	Epochs
3	20587
4	17678
5	16481
6	13273
7	13018
8	12148
9	12657

Evaluating the system, the system was able to classify 55 patient data correctly and only 2 wrong predictions. This reaches an accuracy rate of 96.49%, around 11 percent higher than the previous project with both numerical and nominal valued data.

VII. CONCLUSION

We have created a system for the classification of tuberculosis that could help healthcare providers diagnose and classify patient data, especially fo rural areas where doctors are not always available. The system is called PredictTB and was implemented by the use of a multilayer feedforward neural network.

PredicTB can create projects files that can be used to provide different classification tasks for tuberculosis. An initial data set is required to train the network until it reaches convergence. Evaluation set is also needed to check how accurate the system can be. Retraining the network system is also possible if the expert is not satisfied with the evaluation results.

PredicTB uses algorithms like backpropagation, incremental pruning and adaptive learning rate in order to reach a sufficiently accurate system. A sample project was created to test how the system performs. The data set was obtained from 174 patients from the UP Prime TB DOTS. The system converged and provided 96.48% accuracy on mapping given input variables to a specific output. From the results of the sample project, we can conclude that artificial neural networks is an accurate and reliable method can be applied on healthcare problems like diagnosis and classification.

VIII. RECOMMENDATION

The system can be either be extended to a new online information system for tuberculosis patients or linked with an ongoing database system like the National TB Registry used by TB Clinics in the country. In this way, manual tabulation of patient data into an Excel or comma-separated value file can be avoided since the database can easily be exported to a desired format. It could also be accessed by patients who does no have time to have themselves checked by a physician. They could know initial diagnosis through logging in to the system and evaluating their own set of symptomatics. Patients undergoing the DOTS program could also use the system and see possible actions and medications they may use without actually consulting their health care provider. Another possible user could be the experts who will be managing the system and training the network. Once in a while, they can retrain the system when sufficient number of patient data was added to their database. Also, aside from the basic classification, we can train the system to provide recommendations to the user for a complete decision support system.

IX. BIBLIOGRAPHY

[1] Imianvan A.A. and Obi J. C. "Fuzzy Cluster Means Expert System for the Diagnosis of Tuberculosis". Global Journal of Computer Science & Technology, Volume 11, Issue Version 1, April 2011.

[2] Wikibooks. "Statistics and Analysis of Tuberculosis". <http://en.wikibooks.org/wiki/Statistics/Analysis_of_Tuberculosis>

[3] Department of Health. "National TB Control Program". <http://www.doh.gov.ph/programs/tb.html>

[4] United States Agency for International Development. "Tuberculosis Profile of Philippines". <http://www.usaid.gov/our_work/global_health/id/tuberculosis/countries/asia/philippines.pdf >

[5] R. Dybowski and V. Gant, Clinical Applications of Artificial Neural Networks, Cambridge University Press, 2007.

[6] Statsoft. "Neural Networks". <http://www.statsoft.com/textbook/neural-networks/ >

[7] R.W. Brause. Medical Analysis and Diagnosis by Neural Networks.

[8] Heaton, Jeff. Introduction to Neural Networks for Java, 2nd Edition. Heaton Research, Inc., 2008.

[9] N.H. Phuong, D.H. Hung, D.T. Tuan, "Designing an Experimental Expert System for Lung Tuberculosis Diagnostics using Fuzzy Set Theory". 1998.

[10] T. Uçar, D. Karahoca, A. Karahoca, "Predicting the Existence of Mycobacterium Tuberculosis Infection by Bayesian Networks and Rough Sets". 2010.

[11] T. Asha, S. Natarajan, K.N.B. Murthl. "Diagnosis of Tuberculosis using Ensemble methods". 2010.

[12] M. Sotaquir´a, L. Rueda and R. Narvaez, "Detection and quantification of bacilli and clusters present in sputum smear samples: a novel algorithm for pulmonary tuberculosis diagnosis". International Conference on Digital Image Processing, 2009.

[13] V. Makkapati, R. Agrawal and R. Acharya, "Segmentation and Classification of Tuberculosis Bacilli from ZN-stained Sputum Smear Images". 5th Annual IEEE Conference on Automation Science and Engineering. Bangalore, India, August 2009

[14] K. Veropoulos, C. Campbell and G. Learmonth, "Image Processing and Neural Computing used

in the Diagnosis of Tuberculosis". 1998.

[15] Y. Benfu, S. Hongmei, S. Ye, L. Xiuhui and Z. Bin, "Study on the artificial neural network in the diagnosis of smear negative pulmonary tuberculosis". World Congress on Computer Science and Information Engineering, 2009.

[16] A.A. El-Solh, C.B. Hsiao, S. Goodnough, J. Serghani, and B.J.B. Grant, "Predicting Active Pulmonary Tuberculosis Using an Artificial Neural Network". CHEST Official journal of the American College of Chest Physicians, 1999.

[17] M.K. Osman, M.Y. Mashor and H. Jaafar, "Detection of Mycobacterium Tuberculosis in Ziehl-Neelsen Stained Tissue Images using Zemike Moments and Hybrid Multilayered Perceptron Network". 2010.

[18] A.A. Bakar and F. Febriyani, "Rough Neural Network Model for Tuberculosis Patient Categorization". Proceedings of the International Conference on Electrical Engineering and Informatics. Institut Teknologi Bandung, Indonesia, June 2007.

[19] S. Moein, S. A. Monadjemi, and P. Moallem. "A Novel Fuzzy-Neural Based Medical Diagnosis System", World Academy of Science, Engineering and Technology 37, 2008.

[20] M. K. Osman, F. Ahmad, Z. Saad, M. Y. Mashor, H. Jaafar, "A Genetic Algorithm-Neural Network Approach for Mycobacterium Tuberculosis Detection in Ziehl-Neelsen Stained Tissue Slide Images". 2010.

[21] Tuberculosis. MedicineNet.
<http://www.medterms.com/script/main/art.asp?articlekey=6304>

[22] Artificial Neural Networks. <http://en.wikipedia.org/wiki/Artificial_neural_network>

[23] Forecasting. Wikipedia. <http://en.wikipedia.org/wiki/Forecasting>

[24] David Veitch. Wavelet Neural Networks and their application in the study of dynamical systems, August 2005.

[25] Keen, P. G. W. (1978). Decision support systems: an organizational perspective. Addison-Wesley Pub. Co.

[26] Decision support systems. February 2009. <http://www.openclinical.org/dss.html>

X. APPENDIX

AboutBox.java

```
package PredicTB;

import org.jdesktop.application.Action;

public class AboutBox extends javax.swing.JDialog {

    public AboutBox(java.awt.Frame parent) {
        super(parent);
        initComponents();
        getRootPane().setDefaultButton(closeButton);
    }

    @Action public void closeAboutBox() {
        dispose();
    }

    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        closeButton = new javax.swing.JButton();
        javax.swing.JLabel appTitleLabel = new javax.swing.JLabel();
        javax.swing.JLabel versionLabel = new javax.swing.JLabel();
        javax.swing.JLabel appVersionLabel = new javax.swing.JLabel();
        javax.swing.JLabel vendorLabel = new javax.swing.JLabel();
        javax.swing.JLabel appVendorLabel = new javax.swing.JLabel();
        javax.swing.JLabel homepageLabel = new javax.swing.JLabel();
        javax.swing.JLabel appHomepageLabel = new javax.swing.JLabel();
        javax.swing.JLabel appDescLabel = new javax.swing.JLabel();
        javax.swing.JLabel imageLabel = new javax.swing.JLabel();

        setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
        org.jdesktop.application.ResourceMap resourceMap =
org.jdesktop.application.Application.getInstance(PredicTB.PredicTBApp.class).getContext().getResourceMap(AboutBox.class);
        setTitle(resourceMap.getString("title"));
        setModal(true);
        setName("aboutBox");
        setResizable(false);

        javax.swing.ActionMap actionMap =
org.jdesktop.application.Application.getInstance(PredicTB.PredicTBApp.class).getContext().getActionMap(AboutBox.class, this);
        closeButton.setAction(actionMap.get("closeAboutBox"));
        closeButton.setName("closeButton");

        appTitleLabel.setFont(appTitleLabel.getFont().deriveFont(appTitleLabel.getFont().getStyle() | java.awt.Font.BOLD,
appTitleLabel.getFont().getSize()+4));
        appTitleLabel.setText(resourceMap.getString("Application.title"));
        appTitleLabel.setName("appTitleLabel");

        versionLabel.setFont(versionLabel.getFont().deriveFont(versionLabel.getFont().getStyle() | java.awt.Font.BOLD));
        versionLabel.setText(resourceMap.getString("versionLabel.text"));
        versionLabel.setName("versionLabel");

        appVersionLabel.setText(resourceMap.getString("Application.version"));
```

```
    appVersionLabel.setName("appVersionLabel");

    vendorLabel.setFont(vendorLabel.getFont().deriveFont(vendorLabel.getFont().getStyle() | java.awt.Font.BOLD));
    vendorLabel.setText(resourceMap.getString("authorLabel.text"));
    vendorLabel.setName("authorLabel");

    appVendorLabel.setText(resourceMap.getString("Application.vendor"));
    appVendorLabel.setName("appAuthorLabel");

    homepageLabel.setFont(homepageLabel.getFont().deriveFont(homepageLabel.getFont().getStyle() | java.awt.Font.BOLD));
    homepageLabel.setText(resourceMap.getString("emailLabel.text"));
    homepageLabel.setName("emailLabel");

    appHomepageLabel.setText(resourceMap.getString("Application.homepage"));
    appHomepageLabel.setName("appEmailLabel");

    appDescLabel.setText(resourceMap.getString("appDescLabel.text"));
    appDescLabel.setName("appDescLabel");

    imageLabel.setIcon(resourceMap.getIcon("imageLabel.icon"));
    imageLabel.setName("imageLabel");

    javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
    getContentPane().setLayout(layout);
    layout.setHorizontalGroup(
      layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGroup(layout.createSequentialGroup()
        .addComponent(imageLabel)
        .addGap(18, 18, 18)
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
          .addGroup(javax.swing.GroupLayout.Alignment.LEADING, layout.createSequentialGroup()
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
              .addComponent(versionLabel)
              .addComponent(vendorLabel)
              .addComponent(homepageLabel))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
              .addComponent(appVersionLabel)
              .addComponent(appVendorLabel)
              .addComponent(appHomepageLabel)))
          .addComponent(appTitleLabel, javax.swing.GroupLayout.Alignment.LEADING)
          .addComponent(appDescLabel, javax.swing.GroupLayout.Alignment.LEADING, javax.swing.GroupLayout.DEFAULT_SIZE,
280, Short.MAX_VALUE)
          .addComponent(closeButton))
        .addContainerGap())
    );
    layout.setVerticalGroup(
      layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addComponent(imageLabel, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
      .addGroup(layout.createSequentialGroup()
        .addContainerGap()
        .addComponent(appTitleLabel)
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addComponent(appDescLabel)
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
          .addComponent(versionLabel)
```

```
                .addComponent(appVersionLabel))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                .addComponent(vendorLabel)
                .addComponent(appVendorLabel))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                .addComponent(homepageLabel)
                .addComponent(appHomepageLabel))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 19, Short.MAX_VALUE)
            .addComponent(closeButton)
            .addContainerGap())
        );

        pack();
    }// </editor-fold>

    // Variables declaration - do not modify
    private javax.swing.JButton closeButton;
    // End of variables declaration

}
```

AccuracyChart.java

```
package PredicTB;

import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.axis.CategoryAxis;
import org.jfree.chart.axis.CategoryLabelPositions;
import org.jfree.chart.plot.CategoryPlot;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.chart.renderer.category.CategoryItemRenderer;
import org.jfree.data.category.CategoryDataset;
import org.jfree.data.category.DefaultCategoryDataset;

public class AccuracyChart extends javax.swing.JDialog{

    private int correct = 0;
    private int incorrect = 0;

    public void update(){
        CategoryDataset dataset = createDataset();
        JFreeChart chart = createChart(dataset);
        ChartPanel chartPanel = new ChartPanel(chart);
        chartPanel.setPreferredSize(new java.awt.Dimension(500, 270));
        this.add(chartPanel);
    }

   private CategoryDataset createDataset() {
        final DefaultCategoryDataset dataset = new DefaultCategoryDataset();
        dataset.addValue(correct, "Correct", "");
        dataset.addValue(incorrect, "Incorrect", "");
        return dataset;
    }
```

```
    private JFreeChart createChart(final CategoryDataset dataset) {

        final JFreeChart chart = ChartFactory.createBarChart3D(
            "",     // chart title
            "Predictions",              // domain axis label
            "",                 // range axis label
            dataset,                  // data
            PlotOrientation.VERTICAL, // orientation
            true,                   // include legend
            true,                   // tooltips
            false                   // urls
        );

        final CategoryPlot plot = chart.getCategoryPlot();
        final CategoryAxis axis = plot.getDomainAxis();
        axis.setCategoryLabelPositions(
            CategoryLabelPositions.createUpRotationLabelPositions(Math.PI / 8.0)
        );

        final CategoryItemRenderer renderer = plot.getRenderer();
        renderer.setItemLabelsVisible(true);

        return chart;

    }

    /**
     * @param correct the correct to set
     */
    public void setCorrect(int correct) {
        this.correct = correct;
    }

    /**
     * @param incorrect the incorrect to set
     */
    public void setIncorrect(int incorrect) {
        this.incorrect = incorrect;
    }

}
```

Chart.java

```
package PredicTB;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import javax.swing.JPanel;

public class Chart extends Thread {

    JPanel panel;
    String type;

    public Chart(JPanel panel, String type) {
        this.panel = panel;
```

```
        this.type = type;
    }

    public void draw() {
        if (type.equals("pie")) {
            Graphics g = panel.getGraphics();
            g.setColor(Color.red);
            Dimension d = panel.getSize();
            g.fillOval(d.width / 4, d.height / 4, d.width/2, d.height/2);
            g.dispose();
        } else {
            Graphics g = panel.getGraphics();
            Dimension d = panel.getSize();
            g.fillRect(d.width / 4, d.height / 4, d.width/2, d.height/2);
            g.dispose();
        }
    }

    @Override
    public void run() {
        draw();
    }
}
```

ClassChart.java

```
package PredicTB;

import java.util.ArrayList;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.PiePlot3D;
import org.jfree.data.general.DefaultPieDataset;
import org.jfree.data.general.PieDataset;
import org.jfree.util.Rotation;

public class ClassChart extends javax.swing.JDialog{

    String[] prediction;

    public ClassChart(String[] prediction) {
        this.prediction = prediction;
        PieDataset dataset = createDataSet();
        JFreeChart chart = createChart(dataset);
        final ChartPanel chartPanel = new ChartPanel(chart);
        chartPanel.setPreferredSize(new java.awt.Dimension(500, 270));
        this.add(chartPanel);
    }

    private PieDataset createDataSet(){
        final DefaultPieDataset result = new DefaultPieDataset();
        ArrayList<String> uniqueValues = new ArrayList<String>();

        for(int i = 0; i < prediction.length; i++){
            boolean hasSame = false;
            for(int j = 0; j < uniqueValues.size(); j++){
                if(prediction[i].equals(uniqueValues.get(j))){
```

```
                hasSame = true;
                break;
            }
          }
          if(hasSame == false){
            uniqueValues.add(prediction[i]);
          }
        }

        int [] count = new int[uniqueValues.size()];
        for(int i = 0; i < count.length; i++){
          for(int j = 0; j < prediction.length; j++){
            if(uniqueValues.get(i).equalsIgnoreCase(prediction[j])){
              count[i] = count[i] + 1;
            }
          }
        }

        for(int i = 0; i < count.length; i++){
          result.setValue(uniqueValues.get(i), count[i]);
        }

        return result;
    }

    private JFreeChart createChart(final PieDataset dataset) {

        final JFreeChart chart = ChartFactory.createPieChart3D("",  dataset, true,  true, false);

        final PiePlot3D plot = (PiePlot3D) chart.getPlot();
        plot.setStartAngle(290);
        plot.setDirection(Rotation.CLOCKWISE);
        plot.setForegroundAlpha(0.5f);
        plot.setNoDataMessage("No data to display");
        return chart;
    }
}
```

DialogBox.java

```
package PredicTB;

import javax.swing.JOptionPane;
import javax.swing.JPanel;

public class DialogBox {

    JPanel mainPanel;

    public DialogBox(JPanel mainpanel){
        this.mainPanel = mainpanel;
    }

     protected int createDialogBox(String str, String type) {
        if (type.equals("yesno")) {
            return JOptionPane.showConfirmDialog(mainPanel, str, "PredicTB", JOptionPane.YES_NO_OPTION);
        } else {
            return JOptionPane.showConfirmDialog(mainPanel, str, "PredicTB", JOptionPane.PLAIN_MESSAGE);
```

```
    }
  }
}
```

ErrorGraph.java

```
package PredicTB;

import java.awt.BorderLayout;
import javax.swing.JPanel;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.axis.ValueAxis;
import org.jfree.chart.plot.XYPlot;
import org.jfree.data.time.Millisecond;
import org.jfree.data.time.TimeSeries;
import org.jfree.data.time.TimeSeriesCollection;
import org.jfree.data.xy.XYDataset;

public class ErrorGraph extends javax.swing.JDialog {

  private TimeSeries series;
  private double lastValue = 1;

  public ErrorGraph() {

    this.series = new TimeSeries("Root Mean Squared Error", Millisecond.class);
    final TimeSeriesCollection dataset = new TimeSeriesCollection(this.getSeries());
    final JFreeChart chart = createChart(dataset);

    final ChartPanel chartPanel = new ChartPanel(chart);
    final JPanel content = new JPanel(new BorderLayout());
    content.add(chartPanel);
    //content.add(button, BorderLayout.SOUTH);
    chartPanel.setPreferredSize(new java.awt.Dimension(500, 270));
    setContentPane(content);
  }

  private JFreeChart createChart(final XYDataset dataset) {
    final JFreeChart result = ChartFactory.createTimeSeriesChart(
      "",
      "",
      "",
      dataset,
      true,
      true,
      false
    );
    final XYPlot plot = result.getXYPlot();
    ValueAxis axis = plot.getDomainAxis();
    axis.setAutoRange(true);
    axis.setFixedAutoRange(60000.0);  // 60 seconds
    axis = plot.getRangeAxis();
    axis.setRange(0.0, 1.0);
    return result;
  }
```

```
    public void update(double error){
        this.setLastValue(error);
        Millisecond now = new Millisecond();
        this.getSeries().addOrUpdate(now, error);
    }

    /**
     * @return the series
     */
    public TimeSeries getSeries() {
        return series;
    }

    /**
     * @param series the series to set
     */
    public void setSeries(TimeSeries series) {
        this.series = series;
    }

    /**
     * @return the lastValue
     */
    public double getLastValue() {
        return lastValue;
    }

    /**
     * @param lastValue the lastValue to set
     */
    public void setLastValue(double lastValue) {
        this.lastValue = lastValue;
    }
}
```

FileFilter.java

```
package PredicTB;

import java.io.File;

public class FileFilter extends javax.swing.filechooser.FileFilter {

    String filtername;

    public FileFilter(String fname){
        this.filtername = fname;
    }

  public boolean accept(File file) {
    String filename = file.getName();
    return filename.endsWith(filtername);
  }

  public String getDescription() {
    return "*" + filtername;
  }
}
```

HandleCSV.java

```
package PredicTB;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.StringTokenizer;
import java.util.logging.Level;
import java.util.logging.Logger;

public class HandleCSV {

   private File csvFile;
   private String[] headers;
   private ArrayList<ArrayList<String>> arr;

   protected String[] getFileHeaders() {
      try {
         BufferedReader InputStream = null;
         InputStream = new BufferedReader(new FileReader(getCsvFile()));
         String line = InputStream.readLine();
         StringTokenizer st = new StringTokenizer(line, ",");
         String[] header = new String[st.countTokens()];
         int i = 0;
         while (st.hasMoreTokens()) {
            header[i] = st.nextElement().toString();
            i++;
         }
         InputStream.close();
         headers = header;

      } catch (IOException ex) {
         Logger.getLogger(HandleCSV.class.getName()).log(Level.SEVERE, null, ex);
      }

      return headers;
   }

   protected ArrayList<ArrayList<String>> getFileData() {
      ArrayList<ArrayList<String>> data = new ArrayList<ArrayList<String>>();

      try {
         BufferedReader InputStream = null;
         InputStream = new BufferedReader(new FileReader(getCsvFile()));
         String line = null;
         int one = 0;
         while ((line = InputStream.readLine()) != null) {
            if (one == 0) {
               one = 1;    //skip lang yung first row since assume na header sya
            } else {
               StringTokenizer st = new StringTokenizer(line, ",");
```

```
            ArrayList<String> content = new ArrayList<String>();
            int i = 0;
            while (st.hasMoreTokens()) {
               content.add(st.nextElement().toString());
               i++;
            }
            data.add(content);
         }
      }

      InputStream.close();

   } catch (IOException ex) {
      Logger.getLogger(HandleCSV.class.getName()).log(Level.SEVERE, null, ex);
   }

   return data;
}

protected void writeFile() {
   try {
      PrintWriter outputStream = new PrintWriter(new FileOutputStream(getCsvFile()));

      String line = "";
      for (int i = 0; i < getHeaders().length; i++) {
         line += getHeaders()[i];
         if (i != getHeaders().length - 1) {
            line += ",";
         }
      }
      outputStream.println(line);
      line = "";

      for (int i = 0; i < getArr().size(); i++) {
         ArrayList<String> temp = new ArrayList<String>();
         temp = getArr().get(i);
         for (int j = 0; j < temp.size(); j++) {
            line += temp.get(j);
            if (j != temp.size() - 1) {
               line += ",";
            }
         }
         outputStream.println(line);
         outputStream.flush();
         line = "";
      }

   } catch (FileNotFoundException ex) {
      Logger.getLogger(HandleCSV.class.getName()).log(Level.SEVERE, null, ex);
   }
}

/**
 * @return the csvFile
 */
public File getCsvFile() {
   return csvFile;
}
```

```
    /**
     * @return the headers
     */
    public String[] getHeaders() {
        return headers;
    }

    /**
     * @param csvFile the csvFile to set
     */
    public void setCsvFile(File csvFile) {
        this.csvFile = csvFile;
    }

    /**
     * @param headers the headers to set
     */
    public void setHeaders(String[] headers) {
        this.headers = headers;
    }

    /**
     * @return the arr
     */
    public ArrayList<ArrayList<String>> getArr() {
        return arr;
    }

    /**
     * @param arr the arr to set
     */
    public void setArr(ArrayList<ArrayList<String>> arr) {
        this.arr = arr;
    }
}
```

HandleExcel.java

```
package PredicTB;

import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.logging.Level;
import java.util.logging.Logger;
import jxl.*;
import jxl.write.Label;
import jxl.write.WritableSheet;
import jxl.write.WritableWorkbook;
import jxl.write.WriteException;

public class HandleExcel {

    private File xlsFile;
    private String[] headers;
    private ArrayList<ArrayList<String>> arr;
```

```
protected String[] getFileHeaders() {
   try {
      Workbook workbook = Workbook.getWorkbook(getXlsFile());
      Sheet sheet = workbook.getSheet(0);
      int colNum = sheet.getColumns();
      setHeaders(new String[colNum]);

      for (int i = 0; i < colNum; i++) {
         getHeaders()[i] = sheet.getCell(i, 0).getContents().toString();
      }

      workbook.close();
   } catch (Exception e) {
      e.printStackTrace();
   }

   return getHeaders();
}

protected ArrayList<ArrayList<String>> getFileData() {
   ArrayList<ArrayList<String>> data = new ArrayList<ArrayList<String>>();

   try {
      Workbook workbook = Workbook.getWorkbook(getXlsFile());
      Sheet sheet = workbook.getSheet(0);
      int colNum = sheet.getColumns();
      int rowNum = sheet.getRows();

      for (int i = 1; i < rowNum; i++) {
         //start with 2nd row since we assume the 1st column is the headers
         ArrayList<String> temp = new ArrayList<String>();
         for (int j = 0; j < colNum; j++) {
            temp.add(sheet.getCell(j, i).getContents().toString());
         }
         data.add(temp);
      }

      workbook.close();
   } catch (Exception e) {
      e.printStackTrace();
   }

   return data;
}

protected void writeFile() {
   try {
      WritableWorkbook workbook = Workbook.createWorkbook(getXlsFile());
      WritableSheet sheet = workbook.createSheet("Sheet 1", 0);

      for (int i = 0; i < getHeaders().length; i++) {
         Label lab = new Label(i, 0, getHeaders()[i]);
         sheet.addCell(lab);
      }

      for (int i = 0; i < getArr().size(); i++) {
         ArrayList<String> temp = new ArrayList<String>();
         temp = getArr().get(i);
```

```
            for (int j = 0; j < temp.size(); j++) {
                Label lab = new Label(j, i+1, temp.get(j));
                sheet.addCell(lab);
            }
        }

        workbook.write();
        workbook.close();

    } catch (WriteException ex) {
        Logger.getLogger(HandleExcel.class.getName()).log(Level.SEVERE, null, ex);
    } catch (IOException ex) {
        Logger.getLogger(HandleExcel.class.getName()).log(Level.SEVERE, null, ex);
    }
}

/**
 * @return the xlsFile
 */
public File getXlsFile() {
    return xlsFile;
}

/**
 * @param xlsFile the xlsFile to set
 */
public void setXlsFile(File xlsFile) {
    this.xlsFile = xlsFile;
}

/**
 * @param headers the headers to set
 */
public void setHeaders(String[] headers) {
    this.headers = headers;
}

/**
 * @return the arr
 */
public ArrayList<ArrayList<String>> getArr() {
    return arr;
}

/**
 * @param arr the arr to set
 */
public void setArr(ArrayList<ArrayList<String>> arr) {
    this.arr = arr;
}

/**
 * @return the headers
 */
public String[] getHeaders() {
    return headers;
}
}
```

MappingForm.java

```
package PredicTB;

import java.awt.Dimension;
import java.util.ArrayList;
import javax.swing.JCheckBox;

public class MappingForm extends javax.swing.JDialog {
    private ArrayList<ArrayList<Double>> mappedDataDouble;
    private //passed from eval
    ArrayList<ArrayList<Double>> mappedDataReturn;

    private ArrayList<ArrayList<String>> mappedData;
    private String[] nominalHeaders;
    private double[] min;
    private double[] max;
    private ArrayList<ArrayList<String>> origData;
    private ArrayList<ArrayList<String>> origEval;
    private String[] headers;
    JCheckBox[] jc;

    /** Creates new form MappingForm */
    public MappingForm(java.awt.Frame parent) {
        initComponents();
    }

    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        jPanel1 = new javax.swing.JPanel();
        jButton1 = new javax.swing.JButton();
        jScrollPane1 = new javax.swing.JScrollPane();
        jPanel2 = new javax.swing.JPanel();

        setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
        setName("Form");

        jPanel1.setBorder(javax.swing.BorderFactory.createEtchedBorder());
        jPanel1.setName("jPanel1");

        org.jdesktop.application.ResourceMap resourceMap =
org.jdesktop.application.Application.getInstance(PredicTB.PredicTBApp.class).getContext().getResourceMap(MappingForm.class)
;
        jButton1.setText(resourceMap.getString("jButton1.text"));
        jButton1.setName("jButton1");
        jButton1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton1ActionPerformed(evt);
            }
        });

        jScrollPane1.setBorder(javax.swing.BorderFactory.createTitledBorder(resourceMap.getString("jScrollPane1.border.title")));
        jScrollPane1.setAutoscrolls(true);
        jScrollPane1.setName("jScrollPane1");
```

```
    jPanel2.setAutoscrolls(true);
    jPanel2.setName("jPanel2");

    javax.swing.GroupLayout jPanel2Layout = new javax.swing.GroupLayout(jPanel2);
    jPanel2.setLayout(jPanel2Layout);
    jPanel2Layout.setHorizontalGroup(
      jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGap(0, 344, Short.MAX_VALUE)
    );
    jPanel2Layout.setVerticalGroup(
      jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGap(0, 197, Short.MAX_VALUE)
    );

    jScrollPane1.setViewportView(jPanel2);

    javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);
    jPanel1.setLayout(jPanel1Layout);
    jPanel1Layout.setHorizontalGroup(
      jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGroup(jPanel1Layout.createSequentialGroup()
        .addContainerGap()
        .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
          .addComponent(jButton1, javax.swing.GroupLayout.Alignment.TRAILING)
          .addComponent(jScrollPane1, javax.swing.GroupLayout.DEFAULT_SIZE, 356, Short.MAX_VALUE))
        .addContainerGap())
    );
    jPanel1Layout.setVerticalGroup(
      jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, jPanel1Layout.createSequentialGroup()
        .addContainerGap()
        .addComponent(jScrollPane1, javax.swing.GroupLayout.DEFAULT_SIZE, 223, Short.MAX_VALUE)
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addComponent(jButton1)
        .addContainerGap())
    );

    javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
    getContentPane().setLayout(layout);
    layout.setHorizontalGroup(
      layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGroup(layout.createSequentialGroup()
        .addContainerGap()
        .addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
        .addContainerGap())
    );
    layout.setVerticalGroup(
      layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGroup(layout.createSequentialGroup()
        .addContainerGap()
        .addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
        .addContainerGap())
    );

    pack();
  }// </editor-fold>
```

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {

   setNominalHeaders(new String[headers.length]);
   setMin(new double[headers.length]);
   setMax(new double[headers.length]);
   setMappedData(new ArrayList<ArrayList<String>>());

   for (int i = 0; i < getHeaders().length; i++) {
      if (jc[i].isSelected()) {
         nominalHeaders[i] = headers[i];
      } else {
         nominalHeaders[i] = " ";
      }
   }

   String[][] newOrigData = ArrToString(origData);
   String[][] dataToMap = new String[newOrigData.length][newOrigData[1].length];

   // get the columns that has nominal values and transfer to another array
   for (int i = 0; i < newOrigData.length; i++) {
      for (int j = 0; j < newOrigData[i].length; j++) {
         if (!nominalHeaders[j].trim().isEmpty()) {
            dataToMap[i][j] = newOrigData[i][j];
         } else {
            dataToMap[i][j] = "";
         }
      }
   }

   String[][] mapped = new String[newOrigData.length][newOrigData[1].length];

   // get uniqe elements of that column
   for (int i = 0; i < dataToMap[1].length; i++) {

      if (!nominalHeaders[i].trim().isEmpty()) {

         // do something in here na magoutput sa file para masave
         // yung mapping nung column, the unique element
         // and the index of unique element

         ArrayList<String> uniqueElement = new ArrayList<String>();

         // by the end of this, nakuha mo na yung unique elements
         for (int j = 0; j < dataToMap.length; j++) {
            boolean isUnique = true;
            for (int k = 0; k < uniqueElement.size(); k++) {
               if (dataToMap[j][i].equals(uniqueElement.get(k))) {
                  isUnique = false;
               }
            }
            if (isUnique == true) {
               uniqueElement.add(dataToMap[j][i]);
            }
         }

         // gamitin yung index nung uniqueElement para sa oneOf mapping
         for (int j = 0; j < dataToMap.length; j++) {
```

```
            for (int k = 0; k < uniqueElement.size(); k++) {
               if (dataToMap[j][i].equals(uniqueElement.get(k))) {
                  mapped[j][i] = Integer.toString(k);
               }
            }
         }

         // get the maximum and minimum of the column
         min[i] = 0;
         max[i] = uniqueElement.size() - 1;
      }
   }

   // create the arrayList na mappedData by combining origData and mapped
   for (int i = 0; i < mapped.length; i++) {
      ArrayList<String> temp = new ArrayList<String>();
      for (int j = 0; j < mapped[i].length; j++) {
         if (mapped[i][j] == null) {
            temp.add(newOrigData[i][j]);
         } else {
            temp.add(mapped[i][j]);
         }
      }
      mappedData.add(temp);
   }

   this.setVisible(false);
}

protected void createComponents() {
   jPanel2.removeAll();
   jc = new JCheckBox[getHeaders().length];
   for (int i = 0; i < getHeaders().length; i++) {
      jc[i] = new JCheckBox();
      jc[i].setText(getHeaders()[i]);
      jc[i].setName("CheckBox" + (i + 1));
      jc[i].setLocation(30, i * 25);
      jc[i].setSize(getHeaders()[i].length() * 20, 20);
      jc[i].setVisible(true);
      jPanel2.add(jc[i]);
   }

   if (jc[getHeaders().length - 1].getY() > jPanel2.getPreferredSize().getHeight()) {
      jPanel2.setPreferredSize(new Dimension(300, jc[getHeaders().length - 1].getY() + 20));
   }
}

private String[][] ArrToString(ArrayList<ArrayList<String>> arr) {
   String[][] ret = new String[arr.size()][arr.get(1).size()];

   for (int i = 0; i < ret.length; i++) {
      for (int j = 0; j < ret[i].length; j++) {
         ret[i][j] = arr.get(i).get(j).toString();
      }
   }

   return ret;
}
```

```
// Variables declaration - do not modify
private javax.swing.JButton jButton1;
private javax.swing.JPanel jPanel1;
private javax.swing.JPanel jPanel2;
private javax.swing.JScrollPane jScrollPane1;
// End of variables declaration

/**
 * @return the headers
 */
public String[] getHeaders() {
   return headers;
}

/**
 * @param headers the headers to set
 */
public void setHeaders(String[] headers) {
   this.headers = headers;
}

/**
 * @return the nominalHeaders
 */
public String[] getNominalHeaders() {
   return nominalHeaders;
}

/**
 * @param nominalHeaders the nominalHeaders to set
 */
public void setNominalHeaders(String[] nominalHeaders) {
   this.nominalHeaders = nominalHeaders;
}

/**
 * @return the origData
 */
public ArrayList<ArrayList<String>> getOrigData() {
   return origData;
}

/**
 * @param origData the origData to set
 */
public void setOrigData(ArrayList<ArrayList<String>> origData) {
   this.origData = origData;
}

/**
 * @return the min
 */
public double[] getMin() {
   return min;
}

/**
 * @return the max
```

```
 */
public double[] getMax() {
   return max;
}

/**
 * @return the mappedData
 */
public ArrayList<ArrayList<String>> getMappedData() {
   return mappedData;
}

/**
 * @param mappedData the mappedData to set
 */
public void setMappedData(ArrayList<ArrayList<String>> mappedData) {
   this.mappedData = mappedData;
}

/**
 * @param min the min to set
 */
public void setMin(double[] min) {
   this.min = min;
}

/**
 * @param max the max to set
 */
public void setMax(double[] max) {
   this.max =  max;
}

void map() {
   setMappedDataReturn(new ArrayList<ArrayList<Double>>());
   for(int i=0; i < origEval.size(); i++){
      ArrayList<Double> temp = new ArrayList<Double>();
      for(int j = 0; j < origEval.get(i).size(); j++){
         if(nominalHeaders[j] != null){
            for(int k = 0; k < origEval.size(); k++){
               if(origData.get(k).get(j).equals(origEval.get(i).get(j))){
                  temp.add(getMappedDataDouble().get(k).get(j));
                  k = origEval.size(); //to avoid looping again
               }
            }
         }else{
            temp.add(Double.parseDouble(origEval.get(i).get(j)));
         }
      }
      mappedDataReturn.add(temp);
   }
}

/**
 * @param origEval the origEval to set
 */
public void setOrigEval(ArrayList<ArrayList<String>> origEval) {
   this.origEval = origEval;
```

```
    }

    void setMappedDataDouble(ArrayList<ArrayList<Double>> mappedDataDouble) {
        this.mappedDataDouble = mappedDataDouble;
    }

    ArrayList<ArrayList<Double>> getMappedDataReturn() {
        return mappedDataReturn;
    }

    /**
     * @return the mappedDataDouble
     */
    public ArrayList<ArrayList<Double>> getMappedDataDouble() {
        return mappedDataDouble;
    }

    /**
     * @param mappedDataReturn the mappedDataReturn to set
     */
    public void setMappedDataReturn(ArrayList<ArrayList<Double>> mappedDataReturn) {
        this.mappedDataReturn = mappedDataReturn;
    }

}
```

Network.java

```
package PredicTB;

import java.util.ArrayList;

public class Network {

    private double totalError;
    private String actFunction;
    private double learningRate;
    private double maxError;
    private double momentum;
    private int hiddenNeurons;
    private int inputNeurons;
    private int inputHidden;
    private int totalNeurons;
    private double[] weight;
    private double[] weightDelta;
    private double[] weightDeltaAcc;
    private double[] threshold;
    private double[] thresholdDelta;
    private double[] thresholdDeltaAcc;
    private double[] passValue;
    private double[] error;
    private double[] errorDelta;

    public Network(int inputNeuron, String actFunction, double learningRate,
            double maxError, double momentum, int hiddenNeuron){
        this.actFunction = actFunction;
        this.learningRate = learningRate;
        this.maxError = maxError;
```

```
    this.momentum = momentum;
    this.inputNeurons = inputNeuron;
    this.hiddenNeurons = hiddenNeuron;

    this.inputHidden = inputNeuron + hiddenNeuron;
    this.totalNeurons = inputHidden + 1;

    weight = new double[(inputNeuron * hiddenNeuron) + hiddenNeuron];
    weightDelta = new double[(inputNeuron * hiddenNeuron) + hiddenNeuron];
    weightDeltaAcc = new double[(inputNeuron * hiddenNeuron) + hiddenNeuron];

    threshold = new double[totalNeurons];
    thresholdDelta = new double[totalNeurons];
    thresholdDeltaAcc = new double[totalNeurons];

    passValue = new double[totalNeurons];

    error = new double[totalNeurons];
    errorDelta = new double[totalNeurons];

    randomize();
}

public double propagate(ArrayList<Double> actual){

    // initialize pass value of input layer
    for(int i = 0;  i < actual.size(); i++){
        getPassValue()[i] = actual.get(i);
    }

    // input layer to hidden layer
    int weightindex = 0;
    for(int i = getInputNeurons(); i < getInputHidden(); i++){
        double sum = getThreshold()[i];
        for(int j = 0; j < getInputNeurons(); j++){
            sum += (getPassValue()[j] * getWeight()[weightindex++]);
        }
        getPassValue()[i] = activationFunction(sum);
    }

    //hidden layer to output layer
    double sum = getThreshold()[getInputHidden()];
    for (int i = getInputNeurons(); i < getInputHidden(); i++) {
        sum += (getPassValue()[i] * getWeight()[weightindex++]);
    }

    getPassValue()[inputHidden] = activationFunction(sum);

    return getPassValue()[inputHidden];  //output ng isang line of input
}

public void backpropagate(double ideal){
    // error in the output layer
    for (int i = getInputNeurons(); i < getTotalNeurons(); i++) {
        getError()[i] = 0;
    }

    double err = ideal - getPassValue()[getInputHidden()];
```

```
    getError()[inputHidden] = err;
    getErrorDelta()[inputHidden] = err * activationDerivative(getPassValue()[getInputHidden()]);

    setTotalError(getTotalError() + (err * err)); //error squared

    int connection = getInputNeurons() * getHiddenNeurons();
    // weight delata ng connections sa hidden to output
    for (int i = getInputNeurons(); i < getInputHidden(); i++) {
      getWeightDeltaAcc()[connection] += getErrorDelta()[getInputHidden()] * getPassValue()[i];
      getError()[i] += (getWeight()[connection] * getErrorDelta()[getInputHidden()]);
      connection++;
    }
    getThresholdDeltaAcc()[inputHidden] += getErrorDelta()[getInputHidden()];

    // error delta ng hidden layer
    for (int i = getInputNeurons(); i < getInputHidden(); i++) {
      getErrorDelta()[i] = (getError()[i] * activationDerivative(getPassValue()[i]));
    }

    connection = 0;    //start sa connection ng input to hidden
    for (int i = getInputNeurons(); i < getInputHidden(); i++) {
      for (int j = 0; j < getInputNeurons(); j++) {
        getWeightDeltaAcc()[connection] += getErrorDelta()[i] * getPassValue()[j];
        getError()[j] += (getWeight()[connection] * getErrorDelta()[i]);
        connection++;
      }
      getThresholdDeltaAcc()[i] += getErrorDelta()[i];
    }
  }

  public void train(){

    // update weights
    for (int i = 0; i < getWeight().length; i++) {
      getWeightDelta()[i] = (getLearningRate() * getWeightDeltaAcc()[i]) + (getMomentum() * getWeightDelta()[i]);
      getWeight()[i] += getWeightDelta()[i];
      getWeightDeltaAcc()[i] = 0;
    }

    // update thresholds
    for (int i = getInputHidden(); i < getTotalNeurons(); i++) {
      getThresholdDelta()[i] = (getLearningRate() * getThresholdDeltaAcc()[i]) + (getMomentum() * getThresholdDelta()[i]);
      getThreshold()[i] += getThresholdDelta()[i];
      getThresholdDeltaAcc()[i] = 0;
    }
  }

  public void randomize(){
    // create randomized matrices for the threshold and weights

    for (int i = 0; i < getTotalNeurons(); i++) {
      getThreshold()[i] = 0.5 - (Math.random());
      getThresholdDelta()[i] = 0;
      getThresholdDeltaAcc()[i] = 0;
    }
    for (int i = 0; i < getWeight().length; i++) {
      getWeight()[i] = 0.5 - (Math.random());
      getWeightDelta()[i] = 0;
```

```
      getWeightDeltaAcc()[i] = 0;
    }
  }

  public double activationFunction(double val){
    if(getActFunction().equals("tanh")){
      return Math.tanh(val);
    }else{  //sigmoid
      return 1.0 / (1 + Math.exp(-1.0 * val));
    }
  }

  private double activationDerivative(double val) {
    if(getActFunction().equals("tanh")){
      return 1 - val * val;
    }else{  //sigmoid
      return val * (1 - val);
    }
  }

  /**
   * @return the RMS
   */
  public double getRMS(int len) {
    double err = Math.sqrt(getTotalError() / len);
    totalError = 0;
    return err;
  }

  public void reset(){
    totalError = 0;
    actFunction = null;
    learningRate = 0;
    maxError = 0;
    momentum = 0;
    hiddenNeurons = 0;
    inputNeurons = 0;
    inputHidden = 0;
    totalNeurons = 0;
    weight = null;
    weightDelta = null;
    weightDeltaAcc = null;
    threshold = null;
    thresholdDelta = null;
    thresholdDeltaAcc = null;
    passValue = null;
    error = null;
    errorDelta = null;
  }

  /**
   * @return the totalError
   */
  public double getTotalError() {
    return totalError;
  }

  /**
```

```
 * @param totalError the totalError to set
 */
public void setTotalError(double totalError) {
   this.totalError = totalError;
}

/**
 * @return the actFunction
 */
public String getActFunction() {
   return actFunction;
}

/**
 * @param actFunction the actFunction to set
 */
public void setActFunction(String actFunction) {
   this.actFunction = actFunction;
}

/**
 * @return the learningRate
 */
public double getLearningRate() {
   return learningRate;
}

/**
 * @param learningRate the learningRate to set
 */
public void setLearningRate(double learningRate) {
   this.learningRate = learningRate;
}

/**
 * @return the maxError
 */
public double getMaxError() {
   return maxError;
}

/**
 * @param maxError the maxError to set
 */
public void setMaxError(double maxError) {
   this.maxError = maxError;
}

/**
 * @return the momentum
 */
public double getMomentum() {
   return momentum;
}

/**
 * @param momentum the momentum to set
 */
```

```
public void setMomentum(double momentum) {
   this.momentum = momentum;
}

/**
 * @return the hiddenNeurons
 */
public int getHiddenNeurons() {
   return hiddenNeurons;
}

/**
 * @param hiddenNeurons the hiddenNeurons to set
 */
public void setHiddenNeurons(int hiddenNeurons) {
   this.hiddenNeurons = hiddenNeurons;
}

/**
 * @return the inputNeurons
 */
public int getInputNeurons() {
   return inputNeurons;
}

/**
 * @param inputNeurons the inputNeurons to set
 */
public void setInputNeurons(int inputNeurons) {
   this.inputNeurons = inputNeurons;
}

/**
 * @return the inputHidden
 */
public int getInputHidden() {
   return inputHidden;
}

/**
 * @param inputHidden the inputHidden to set
 */
public void setInputHidden(int inputHidden) {
   this.inputHidden = inputHidden;
}

/**
 * @return the totalNeurons
 */
public int getTotalNeurons() {
   return totalNeurons;
}

/**
 * @param totalNeurons the totalNeurons to set
 */
public void setTotalNeurons(int totalNeurons) {
   this.totalNeurons = totalNeurons;
```

```
}

/**
 * @return the weight
 */
public double[] getWeight() {
   return weight;
}

/**
 * @param weight the weight to set
 */
public void setWeight(double[] weight) {
   this.weight = weight;
}

/**
 * @return the weightDelta
 */
public double[] getWeightDelta() {
   return weightDelta;
}

/**
 * @param weightDelta the weightDelta to set
 */
public void setWeightDelta(double[] weightDelta) {
   this.weightDelta = weightDelta;
}

/**
 * @return the weightDeltaAcc
 */
public double[] getWeightDeltaAcc() {
   return weightDeltaAcc;
}

/**
 * @param weightDeltaAcc the weightDeltaAcc to set
 */
public void setWeightDeltaAcc(double[] weightDeltaAcc) {
   this.weightDeltaAcc = weightDeltaAcc;
}

/**
 * @return the threshold
 */
public double[] getThreshold() {
   return threshold;
}

/**
 * @param threshold the threshold to set
 */
public void setThreshold(double[] threshold) {
   this.threshold = threshold;
}
```

```
/**
 * @return the thresholdDelta
 */
public double[] getThresholdDelta() {
   return thresholdDelta;
}

/**
 * @param thresholdDelta the thresholdDelta to set
 */
public void setThresholdDelta(double[] thresholdDelta) {
   this.thresholdDelta = thresholdDelta;
}

/**
 * @return the thresholdDeltaAcc
 */
public double[] getThresholdDeltaAcc() {
   return thresholdDeltaAcc;
}

/**
 * @param thresholdDeltaAcc the thresholdDeltaAcc to set
 */
public void setThresholdDeltaAcc(double[] thresholdDeltaAcc) {
   this.thresholdDeltaAcc = thresholdDeltaAcc;
}

/**
 * @return the passValue
 */
public double[] getPassValue() {
   return passValue;
}

/**
 * @param passValue the passValue to set
 */
public void setPassValue(double[] passValue) {
   this.passValue = passValue;
}

/**
 * @return the error
 */
public double[] getError() {
   return error;
}

/**
 * @param error the error to set
 */
public void setError(double[] error) {
   this.error = error;
}

/**
 * @return the errorDelta
```

```
     */
    public double[] getErrorDelta() {
        return errorDelta;
    }

    /**
     * @param errorDelta the errorDelta to set
     */
    public void setErrorDelta(double[] errorDelta) {
        this.errorDelta = errorDelta;
    }
}
```

Normalization.java

```
package PredicTB;

import java.util.ArrayList;

public class Normalization {
    private ArrayList<ArrayList<Double>> mappedArray;
    private ArrayList<ArrayList<Double>> normalizedArray;
    private double[] max;
    private double[] min;

    protected void normalize(){
        normalizedArray = new ArrayList<ArrayList<Double>>();

        for (int i = 0; i < getMappedArray().size(); i++) {
            ArrayList<Double> temp = new ArrayList<Double>();
            temp = mappedArray.get(i);

            ArrayList<Double> temp1 = new ArrayList<Double>();

            for (int j = 0; j < temp.size(); j++) {
                temp1.add(normalizeItem(temp.get(j), max[j], min[j]));
            }
            normalizedArray.add(temp1);
        }
    }

    protected double [] denormalize(double [] result, int outputCol){
        double [] newresult = new double[result.length];

        for (int i = 0; i < result.length; i++) {
            newresult[i] = denormalizeItem(result[i], getMax()[outputCol], getMin()[outputCol]);
        }

        return newresult;
    }

    protected double denormalizeItem(double value, double max, double min){
      return ((min - max) * value) / (-1);
    }

    protected double normalizeItem(double value, double max, double min){
        return (value - min) / (max - min);
    }
```

```
    /**
     * @return the mappedArray
     */
    public ArrayList<ArrayList<Double>> getMappedArray() {
        return mappedArray;
    }

    /**
     * @param mappedArray the mappedArray to set
     */
    public void setMappedArray(ArrayList<ArrayList<Double>> mappedArray) {
        this.mappedArray = mappedArray;
    }

    /**
     * @return the normalizedArray
     */
    public ArrayList<ArrayList<Double>> getNormalizedArray() {
        return normalizedArray;
    }

    /**
     * @param normalizedArray the normalizedArray to set
     */
    public void setNormalizedArray(ArrayList<ArrayList<Double>> normalizedArray) {
        this.normalizedArray = normalizedArray;
    }

    /**
     * @return the max
     */
    public double[] getMax() {
        return max;
    }

    /**
     * @param max the max to set
     */
    public void setMax(double[] max) {
        this.max = max;
    }

    /**
     * @return the min
     */
    public double[] getMin() {
        return min;
    }

    /**
     * @param min the min to set
     */
    public void setMin(double[] min) {
        this.min = min;
    }
}
```

NormalizeForm.java

```
package PredicTB;

import java.awt.Dimension;
import java.util.ArrayList;
import javax.swing.JLabel;
import javax.swing.JTextField;

public class NormalizeForm extends javax.swing.JDialog {

    private ArrayList<Double> idealArray;
    private ArrayList<ArrayList<Double>> actualArray;
    private int outputCol = 0;

    private String[] headers;
    private String[] nominalHeaders;
    private ArrayList<ArrayList<Double>> mappedData;
    private ArrayList<ArrayList<Double>> normalizeData;
    private double[] min;
    private double[] max;
    JTextField[] jTmin;
    JTextField[] jTmax;
    JLabel[] jlHeader;

    DialogBox dbox;

    public NormalizeForm(java.awt.Frame parent) {
        initComponents();
    }

    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        jScrollPane1 = new javax.swing.JScrollPane();
        jPanel2 = new javax.swing.JPanel();
        jPanel1 = new javax.swing.JPanel();
        jLabel1 = new javax.swing.JLabel();
        jComboBox1 = new javax.swing.JComboBox();
        jButton1 = new javax.swing.JButton();
        jLabel2 = new javax.swing.JLabel();
        jLabel3 = new javax.swing.JLabel();
        jLabel4 = new javax.swing.JLabel();

        setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
        setName("Form");

        jScrollPane1.setName("jScrollPane1");

        jPanel2.setName("jPanel2");
        jPanel2.setPreferredSize(new java.awt.Dimension(350, 100));

        javax.swing.GroupLayout jPanel2Layout = new javax.swing.GroupLayout(jPanel2);
        jPanel2.setLayout(jPanel2Layout);
        jPanel2Layout.setHorizontalGroup(
            jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 376, Short.MAX_VALUE)
```

```
        );
        jPanel2Layout.setVerticalGroup(
            jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 186, Short.MAX_VALUE)
        );

        jScrollPane1.setViewportView(jPanel2);

        jPanel1.setBorder(javax.swing.BorderFactory.createEtchedBorder());
        jPanel1.setName("jPanel1");

        org.jdesktop.application.ResourceMap resourceMap =
org.jdesktop.application.Application.getInstance(PredicTB.PredicTBApp.class).getContext().getResourceMap(NormalizeForm.clas
s);
        jLabel1.setText(resourceMap.getString("jLabel1.text"));
        jLabel1.setName("jLabel1");

        jComboBox1.setName("jComboBox1");

        javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);
        jPanel1.setLayout(jPanel1Layout);
        jPanel1Layout.setHorizontalGroup(
            jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jLabel1)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(jComboBox1, 0, 237, Short.MAX_VALUE)
                .addContainerGap())
        );
        jPanel1Layout.setVerticalGroup(
            jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addContainerGap()
                .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(jLabel1)
                    .addComponent(jComboBox1, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
        );

        jButton1.setText(resourceMap.getString("jButton1.text"));
        jButton1.setName("jButton1");
        jButton1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton1ActionPerformed(evt);
            }
        });

        jLabel2.setText(resourceMap.getString("jLabel2.text"));
        jLabel2.setName("jLabel2");

        jLabel3.setText(resourceMap.getString("jLabel3.text"));
        jLabel3.setName("jLabel3");

        jLabel4.setText(resourceMap.getString("jLabel4.text"));
        jLabel4.setName("jLabel4");
```

```
        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
                .addContainerGap()
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
                    .addComponent(jScrollPane1, javax.swing.GroupLayout.Alignment.LEADING, javax.swing.GroupLayout.DEFAULT_SIZE,
380, Short.MAX_VALUE)
                    .addComponent(jButton1)
                    .addComponent(jPanel1, javax.swing.GroupLayout.Alignment.LEADING, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                    .addGroup(javax.swing.GroupLayout.Alignment.LEADING, layout.createSequentialGroup()
                        .addGap(60, 60, 60)
                        .addComponent(jLabel2)
                        .addGap(110, 110, 110)
                        .addComponent(jLabel3)
                        .addGap(33, 33, 33)
                        .addComponent(jLabel4)))
                .addContainerGap())
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
                .addContainerGap()
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(jLabel2)
                    .addComponent(jLabel3)
                    .addComponent(jLabel4))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(jScrollPane1, javax.swing.GroupLayout.DEFAULT_SIZE, 190, Short.MAX_VALUE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(jButton1)
                .addContainerGap())
        );

        pack();
    }// </editor-fold>

    public void reset(){
        min = null;
        max = null;
        normalizeData = null;
        mappedData = null;
        nominalHeaders = null;
        headers = null;
        actualArray = null;
        idealArray = null;
    }
    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
      //loop through min and max, no blanks should exist
        try {
            for (int i = 0; i < headers.length; i++) {
                min[i] = Double.parseDouble(jTmin[i].getText());
                max[i] = Double.parseDouble(jTmax[i].getText());
```

```
        }

        Normalization norm = new Normalization();
        norm.setMax(max);
        norm.setMin(min);
        norm.setMappedArray(mappedData);
        normalizeData = new ArrayList<ArrayList<Double>>();
        norm.setNormalizedArray(normalizeData);
        norm.normalize();
        normalizeData = norm.getNormalizedArray();

        setOutputCol(jComboBox1.getSelectedIndex());

        this.setVisible(false);
      } catch (Exception ex) {
        dbox = new DialogBox(jPanel2);
        dbox.createDialogBox("Incorrect or insufficient input.", "");
      }
    }

    // Variables declaration - do not modify
    private javax.swing.JButton jButton1;
    private javax.swing.JComboBox jComboBox1;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JPanel jPanel2;
    private javax.swing.JScrollPane jScrollPane1;
    // End of variables declaration

    void createComponents(){
      jPanel2.removeAll();
      jComboBox1.removeAllItems();

      jlHeader = new JLabel[headers.length];
      jTmax = new JTextField[headers.length];
      jTmin = new JTextField[headers.length];

      for (int i = 0; i < headers.length; i++) {
        jlHeader[i] = new JLabel();
        jTmax[i] = new JTextField();
        jTmin[i] = new JTextField();

        jComboBox1.addItem(headers[i]);

        jlHeader[i].setText(headers[i]);
        jlHeader[i].setName(headers[i]);
        jlHeader[i].setLocation(30, (i+1) * 25);
        jlHeader[i].setSize(170, 20);
        jlHeader[i].setVisible(true);

        if(!nominalHeaders[i].trim().isEmpty()){
          jTmax[i].setEnabled(false);
          jTmin[i].setEnabled(false);
        }
```

```
      if(max[i] != 0){
         jTmax[i].setText(Double.toString(max[i]));
         jTmin[i].setText(Double.toString(min[i]));
      }

      jTmax[i].setName("max" + (i + 1));
      jTmax[i].setLocation(280, (i+1) * 25);
      jTmax[i].setSize(50, 24);

      jTmin[i].setName("min" + (i + 1));
      jTmin[i].setLocation(205, (i+1) * 25);
      jTmin[i].setSize(50, 24);

      jPanel2.add(jlHeader[i]);
      jPanel2.add(jTmax[i]);
      jPanel2.add(jTmin[i]);
   }

    if (jlHeader[headers.length - 1].getY() > jPanel2.getPreferredSize().getHeight()) {
      jPanel2.setPreferredSize(new Dimension(350, jlHeader[headers.length - 1].getY() + 40));
   }
}

/**
 * @param headers the headers to set
 */
public void setHeaders(String[] headers) {
   this.headers = headers;
}

/**
 * @return the nominalHeaders
 */
public String[] getNominalHeaders() {
   return nominalHeaders;
}

/**
 * @param nominalHeaders the nominalHeaders to set
 */
public void setNominalHeaders(String[] nominalHeaders) {
   this.nominalHeaders = nominalHeaders;
}

/**
 * @return the mappedData
 */
public ArrayList<ArrayList<Double>> getMappedData() {
   return mappedData;
}

/**
 * @param mappedData the mappedData to set
 */
public void setMappedData(ArrayList<ArrayList<Double>> mappedData) {
   this.mappedData = mappedData;
}
```

```
/**
 * @return the min
 */
public double[] getMin() {
   return min;
}

/**
 * @param min the min to set
 */
public void setMin(double[] min) {
   this.min = min;
}

/**
 * @return the max
 */
public double[] getMax() {
   return max;
}

/**
 * @param max the max to set
 */
public void setMax(double[] max) {
   this.max = max;
}

/**
 * @return the normalizeData
 */
public ArrayList<ArrayList<Double>> getNormalizeData() {
   return normalizeData;
}

/**
 * @param normalizeData the normalizeData to set
 */
public void setNormalizeData(ArrayList<ArrayList<Double>> normalizeData) {
   this.normalizeData = normalizeData;
}

/**
 * @return the idealArray
 */
public ArrayList<Double> getIdealArray() {
   idealArray = new ArrayList<Double>();

   for(int i = 0; i < normalizeData.size(); i++){
      ArrayList<Double> temp = new ArrayList<Double>();
      temp = normalizeData.get(i);
      idealArray.add(temp.get(getOutputCol()));
   }
   return idealArray;
}

/**
 * @return the actualArray
```

```
     */
    public ArrayList<ArrayList<Double>> getActualArray() {
        actualArray = new ArrayList<ArrayList<Double>>();

        for(int i = 0; i < normalizeData.size(); i++){
            ArrayList<Double> temp = new ArrayList<Double>();

            for(int j = 0; j < normalizeData.get(i).size(); j++){
                if(j != getOutputCol()){
                    temp.add(normalizeData.get(i).get(j));
                }
            }

            actualArray.add(temp);
        }
        return actualArray;
    }

    /**
     * @return the outputCol
     */
    public int getOutputCol() {
        return outputCol;
    }

    /**
     * @param outputCol the outputCol to set
     */
    public void setOutputCol(int outputCol) {
        this.outputCol = outputCol;
    }
}
```

PredictTBApp.java

```
package PredicTB;

import org.jdesktop.application.Application;
import org.jdesktop.application.SingleFrameApplication;

/**
 * The main class of the application.
 */
public class PredicTBApp extends SingleFrameApplication {

    /**
     * At startup create and show the main frame of the application.
     */
    @Override protected void startup() {
        show(new PredicTBView(this));
    }

    /**
     * This method is to initialize the specified window by injecting resources.
     * Windows shown in our application come fully initialized from the GUI
     * builder, so this additional configuration is not needed.
     */
    @Override protected void configureWindow(java.awt.Window root) {
```

```
    }

    /**
     * A convenient static getter for the application instance.
     * @return the instance of DesktopApplication1
     */
    public static PredicTBApp getApplication() {
        return Application.getInstance(PredicTBApp.class);
    }

    /**
     * Main method launching the application.
     */
    public static void main(String[] args) {
        launch(PredicTBApp.class, args);
    }
}
```

PredicTBView.java

```
package PredicTB;

import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JTable;
import org.jdesktop.application.ResourceMap;
import org.jdesktop.application.SingleFrameApplication;
import org.jdesktop.application.FrameView;
import org.jdesktop.application.TaskMonitor;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.io.File;
import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.StringTokenizer;
import javax.swing.Timer;
import javax.swing.Icon;
import javax.swing.JDialog;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.table.DefaultTableModel;
import org.jfree.ui.RefineryUtilities;

/**
 * The application's main frame.
 */
public class PredicTBView extends FrameView implements Runnable {

    /* VARIABLES */

    protected Thread worker = null;
    protected Network network;
    protected Pruning prune;

    File trainingFile;
```

```
File evaluationFile;

ProjectFile project;
DialogBox dbox;
Variables variable;
VariableForm vf;
MappingForm mf;
NormalizeForm nf;
ErrorGraph errgraph;
AccuracyChart accChart;
ClassChart classchart;

String[] headers;
String[] nominalHeaders;
double[] maximum;
double[] minimum;

ArrayList<ArrayList<String>> origData = new ArrayList<ArrayList<String>>();
ArrayList<ArrayList<Double>> mappedData = new ArrayList<ArrayList<Double>>();
ArrayList<ArrayList<Double>> normalizedData = new ArrayList<ArrayList<Double>>();

ArrayList<ArrayList<String>> origEval = new ArrayList<ArrayList<String>>();
ArrayList<ArrayList<Double>> mappedEval = new ArrayList<ArrayList<Double>>();
ArrayList<ArrayList<Double>> normalizedEval = new ArrayList<ArrayList<Double>>();

private boolean continueTrain = false;
private boolean threadStop = false;

DefaultTableModel model = new DefaultTableModel();
DefaultTableModel model1 = new DefaultTableModel();

int wrong;
int right;
String[] prediction;

/* FUNCTIONS */
public PredicTBView(SingleFrameApplication app) {
  super(app);

  initComponents();

  jLabel3.setText("Welcome to PredicTB!");
  ClassButton.setEnabled(false);
  AccuracyButton.setEnabled(false);

  ImportFileTextField.setVisible(false);
  dbox = new DialogBox(mainPanel);
  variable = new Variables();

  vf = new VariableForm(jFrame1, variable);
  vf.setLocationRelativeTo(mainPanel);
  vf.setVisible(false);

  mf = new MappingForm(jFrame1);
  mf.setLocationRelativeTo(mainPanel);
  mf.setVisible(false);

  nf = new NormalizeForm(jFrame1);
```

```
nf.setLocationRelativeTo(mainPanel);
nf.setVisible(false);

PauseButton.setEnabled(false);
StopButton.setEnabled(false);

errgraph = new ErrorGraph();
errgraph.pack();
RefineryUtilities.centerFrameOnScreen(errgraph);

DataSetTable.setModel(model);
EvalTable.setModel(model1);

// status bar initialization - message timeout, idle icon and busy animation, etc
ResourceMap resourceMap = getResourceMap();
int messageTimeout = resourceMap.getInteger("StatusBar.messageTimeout");
messageTimer = new Timer(messageTimeout, new ActionListener() {

   public void actionPerformed(ActionEvent e) {
      statusMessageLabel.setText("");
   }
});
messageTimer.setRepeats(false);
int busyAnimationRate = resourceMap.getInteger("StatusBar.busyAnimationRate");
for (int i = 0; i < busyIcons.length; i++) {
   busyIcons[i] = resourceMap.getIcon("StatusBar.busyIcons[" + i + "]");
}
busyIconTimer = new Timer(busyAnimationRate, new ActionListener() {

   public void actionPerformed(ActionEvent e) {
      busyIconIndex = (busyIconIndex + 1) % busyIcons.length;
      statusAnimationLabel.setIcon(busyIcons[busyIconIndex]);
   }
});
idleIcon = resourceMap.getIcon("StatusBar.idleIcon");
statusAnimationLabel.setIcon(idleIcon);
progressBar.setVisible(false);

// connecting action tasks to status bar via TaskMonitor
TaskMonitor taskMonitor = new TaskMonitor(getApplication().getContext());
taskMonitor.addPropertyChangeListener(new java.beans.PropertyChangeListener() {

   public void propertyChange(java.beans.PropertyChangeEvent evt) {
      String propertyName = evt.getPropertyName();
      if ("started".equals(propertyName)) {
         if (!busyIconTimer.isRunning()) {
            statusAnimationLabel.setIcon(busyIcons[0]);
            busyIconIndex = 0;
            busyIconTimer.start();
         }
         progressBar.setVisible(true);
         progressBar.setIndeterminate(true);
      } else if ("done".equals(propertyName)) {
         busyIconTimer.stop();
         statusAnimationLabel.setIcon(idleIcon);
         progressBar.setVisible(false);
         progressBar.setValue(0);
      } else if ("message".equals(propertyName)) {
```

```
            String text = (String) (evt.getNewValue());
            statusMessageLabel.setText((text == null) ? "" : text);
            messageTimer.restart();
        } else if ("progress".equals(propertyName)) {
            int value = (Integer) (evt.getNewValue());
            progressBar.setVisible(true);
            progressBar.setIndeterminate(false);
            progressBar.setValue(value);
        }
    }
  });
}

/* show about box */
public void showAboutBox() {
  if (aboutBox == null) {
    JFrame mainFrame = PredicTBApp.getApplication().getMainFrame();
    aboutBox = new AboutBox(mainFrame);
    aboutBox.setLocationRelativeTo(mainFrame);
  }
  PredicTBApp.getApplication().show(aboutBox);
}

@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents() {

  mainPanel = new javax.swing.JPanel();
  mainTabPane = new javax.swing.JTabbedPane();
  DataSetTab = new javax.swing.JPanel();
  DataSetPanel = new javax.swing.JPanel();
  ManualImportComboBox = new javax.swing.JComboBox();
  ChooseTaskLabel = new javax.swing.JLabel();
  ChooseTaskLayeredPane = new javax.swing.JLayeredPane();
  InputHeaderPanel = new javax.swing.JPanel();
  HeadersLabel = new javax.swing.JLabel();
  jScrollPane2 = new javax.swing.JScrollPane();
  HeadersEditorPane = new javax.swing.JEditorPane();
  jCheckBox1 = new javax.swing.JCheckBox();
  jCheckBox2 = new javax.swing.JCheckBox();
  jCheckBox3 = new javax.swing.JCheckBox();
  jCheckBox4 = new javax.swing.JCheckBox();
  jCheckBox5 = new javax.swing.JCheckBox();
  jCheckBox6 = new javax.swing.JCheckBox();
  jCheckBox7 = new javax.swing.JCheckBox();
  jCheckBox8 = new javax.swing.JCheckBox();
  jCheckBox9 = new javax.swing.JCheckBox();
  InputFilePanel = new javax.swing.JPanel();
  ImportFileLabel = new javax.swing.JLabel();
  ImportFileTextField = new javax.swing.JTextField();
  ImportFileButton = new javax.swing.JButton();
  DataSetSubmitButton = new javax.swing.JButton();
  PreviewPanel = new javax.swing.JPanel();
  DataSetScrollPane = new javax.swing.JScrollPane();
  DataSetTable = new javax.swing.JTable();
  NormalizeDataButton = new javax.swing.JButton();
  ExportFileButton = new javax.swing.JButton();
  oneOfMappingButton = new javax.swing.JButton();
```

```
DataSetComboBox = new javax.swing.JComboBox();
TrainingTab = new javax.swing.JPanel();
TrainingStatusPanel = new javax.swing.JPanel();
TrainingStatusScrollPane = new javax.swing.JScrollPane();
TraningStatusArea = new javax.swing.JTextArea();
ErrorGraphButton = new javax.swing.JButton();
TrainButton = new javax.swing.JButton();
PauseButton = new javax.swing.JButton();
StopButton = new javax.swing.JButton();
TrainingVariableButton = new javax.swing.JButton();
EvaluationTab = new javax.swing.JPanel();
jPanel6 = new javax.swing.JPanel();
EvalPreviewPanel = new javax.swing.JPanel();
jScrollPane3 = new javax.swing.JScrollPane();
EvalTable = new javax.swing.JTable();
EvaluateButton = new javax.swing.JButton();
EvalTextBox = new javax.swing.JTextField();
EvalOpenFileButton = new javax.swing.JButton();
EvalFileLabel = new javax.swing.JLabel();
EvalExportButton = new javax.swing.JButton();
EvalTypeLabel = new javax.swing.JLabel();
EvalTypeComboBox = new javax.swing.JComboBox();
jPanel1 = new javax.swing.JPanel();
jPanel2 = new javax.swing.JPanel();
jScrollPane5 = new javax.swing.JScrollPane();
StatTextArea = new javax.swing.JTextArea();
AccuracyButton = new javax.swing.JButton();
ClassButton = new javax.swing.JButton();
jPanel3 = new javax.swing.JPanel();
menuBar = new javax.swing.JMenuBar();
javax.swing.JMenu fileMenu = new javax.swing.JMenu();
NewProject = new javax.swing.JMenuItem();
OpenProject = new javax.swing.JMenuItem();
jSeparator1 = new javax.swing.JSeparator();
SaveProject = new javax.swing.JMenuItem();
SaveAsProject = new javax.swing.JMenuItem();
jSeparator3 = new javax.swing.JSeparator();
javax.swing.JMenuItem exitMenuItem = new javax.swing.JMenuItem();
javax.swing.JMenu helpMenu = new javax.swing.JMenu();
AboutMenuItem = new javax.swing.JMenuItem();
jSeparator5 = new javax.swing.JSeparator();
HelpMenuItem = new javax.swing.JMenuItem();
statusPanel = new javax.swing.JPanel();
javax.swing.JSeparator statusPanelSeparator = new javax.swing.JSeparator();
statusMessageLabel = new javax.swing.JLabel();
statusAnimationLabel = new javax.swing.JLabel();
progressBar = new javax.swing.JProgressBar();
jLabel3 = new javax.swing.JLabel();
jFrame1 = new javax.swing.JFrame();

mainPanel.setMinimumSize(new java.awt.Dimension(800, 500));
mainPanel.setName("mainPanel");
mainPanel.setPreferredSize(new java.awt.Dimension(800, 500));

mainTabPane.setTabLayoutPolicy(javax.swing.JTabbedPane.SCROLL_TAB_LAYOUT);
mainTabPane.setAutoscrolls(true);
mainTabPane.setMinimumSize(new java.awt.Dimension(600, 500));
mainTabPane.setName("mainTabPane");
```

```
        mainTabPane.setOpaque(true);

        DataSetTab.setAutoscrolls(true);
        org.jdesktop.application.ResourceMap resourceMap =
org.jdesktop.application.Application.getInstance(PredicTB.PredicTBApp.class).getContext().getResourceMap(PredicTBView.class)
;
        DataSetTab.setFont(resourceMap.getFont("DataSetTab.font"));
        DataSetTab.setName("DataSetTab");

        DataSetPanel.setBorder(javax.swing.BorderFactory.createTitledBorder(resourceMap.getString("DataSetPanel.border.title")));
        DataSetPanel.setAutoscrolls(true);
        DataSetPanel.setFont(resourceMap.getFont("DataSetPanel.font"));
        DataSetPanel.setName("DataSetPanel");

        ManualImportComboBox.setModel(new javax.swing.DefaultComboBoxModel(new String[] { "Input Manually", "Import .CSV
or .XLS" }));
        ManualImportComboBox.setName("ManualImportComboBox");
        ManualImportComboBox.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                ManualImportComboBoxActionPerformed(evt);
            }
        });

        ChooseTaskLabel.setText(resourceMap.getString("ChooseTaskLabel.text"));
        ChooseTaskLabel.setName("ChooseTaskLabel");

        ChooseTaskLayeredPane.setName("ChooseTaskLayeredPane");

        InputHeaderPanel.setBorder(javax.swing.BorderFactory.createEtchedBorder());
        InputHeaderPanel.setName("InputHeaderPanel");

        HeadersLabel.setText(resourceMap.getString("HeadersLabel.text"));
        HeadersLabel.setName("HeadersLabel");

        jScrollPane2.setName("jScrollPane2");

        HeadersEditorPane.setName("HeadersEditorPane");
        jScrollPane2.setViewportView(HeadersEditorPane);

        jCheckBox1.setText(resourceMap.getString("jCheckBox1.text"));
        jCheckBox1.setName("jCheckBox1");

        jCheckBox2.setText(resourceMap.getString("jCheckBox2.text"));
        jCheckBox2.setName("jCheckBox2");

        jCheckBox3.setText(resourceMap.getString("jCheckBox3.text"));
        jCheckBox3.setName("jCheckBox3");

        jCheckBox4.setText(resourceMap.getString("jCheckBox4.text"));
        jCheckBox4.setName("jCheckBox4");

        jCheckBox5.setText(resourceMap.getString("jCheckBox5.text"));
        jCheckBox5.setName("jCheckBox5");

        jCheckBox6.setText(resourceMap.getString("jCheckBox6.text"));
        jCheckBox6.setName("jCheckBox6");

        jCheckBox7.setText(resourceMap.getString("jCheckBox7.text"));
```

```
jCheckBox7.setName("jCheckBox7");

jCheckBox8.setText(resourceMap.getString("jCheckBox8.text"));
jCheckBox8.setName("jCheckBox8");

jCheckBox9.setText(resourceMap.getString("jCheckBox9.text"));
jCheckBox9.setName("jCheckBox9");

javax.swing.GroupLayout InputHeaderPanelLayout = new javax.swing.GroupLayout(InputHeaderPanel);
InputHeaderPanel.setLayout(InputHeaderPanelLayout);
InputHeaderPanelLayout.setHorizontalGroup(
   InputHeaderPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
   .addGroup(InputHeaderPanelLayout.createSequentialGroup()
      .addContainerGap()
      .addGroup(InputHeaderPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
         .addComponent(jScrollPane2, javax.swing.GroupLayout.DEFAULT_SIZE, 176, Short.MAX_VALUE)
         .addComponent(jCheckBox9)
         .addComponent(jCheckBox8)
         .addComponent(jCheckBox7)
         .addComponent(jCheckBox6)
         .addComponent(jCheckBox5)
         .addComponent(jCheckBox4)
         .addComponent(jCheckBox3)
         .addComponent(jCheckBox2)
         .addComponent(jCheckBox1)
         .addComponent(HeadersLabel))
      .addContainerGap())
);
InputHeaderPanelLayout.setVerticalGroup(
   InputHeaderPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
   .addGroup(InputHeaderPanelLayout.createSequentialGroup()
      .addContainerGap()
      .addComponent(jCheckBox1)
      .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
      .addComponent(jCheckBox2)
      .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
      .addComponent(jCheckBox3)
      .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
      .addComponent(jCheckBox4)
      .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
      .addComponent(jCheckBox5)
      .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
      .addComponent(jCheckBox6)
      .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
      .addComponent(jCheckBox7)
      .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
      .addComponent(jCheckBox8)
      .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
      .addComponent(jCheckBox9)
      .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
      .addComponent(HeadersLabel)
      .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
      .addComponent(jScrollPane2, javax.swing.GroupLayout.DEFAULT_SIZE, 134, Short.MAX_VALUE)
      .addContainerGap())
);

InputHeaderPanel.setBounds(0, 0, 200, 390);
ChooseTaskLayeredPane.add(InputHeaderPanel, javax.swing.JLayeredPane.DEFAULT_LAYER);
```

```
    InputFilePanel.setBorder(javax.swing.BorderFactory.createEtchedBorder());
    InputFilePanel.setName("InputFilePanel");

    ImportFileLabel.setText(resourceMap.getString("ImportFileLabel.text"));
    ImportFileLabel.setName("ImportFileLabel");

    ImportFileTextField.setText(resourceMap.getString("ImportFileTextField.text"));
    ImportFileTextField.setName("ImportFileTextField");

    ImportFileButton.setText(resourceMap.getString("ImportFileButton.text"));
    ImportFileButton.setName("ImportFileButton");
    ImportFileButton.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
        ImportFileButtonActionPerformed(evt);
      }
    });

    javax.swing.GroupLayout InputFilePanelLayout = new javax.swing.GroupLayout(InputFilePanel);
    InputFilePanel.setLayout(InputFilePanelLayout);
    InputFilePanelLayout.setHorizontalGroup(
      InputFilePanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGroup(InputFilePanelLayout.createSequentialGroup()
        .addContainerGap()
        .addGroup(InputFilePanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
          .addComponent(ImportFileTextField, javax.swing.GroupLayout.DEFAULT_SIZE, 176, Short.MAX_VALUE)
          .addComponent(ImportFileLabel)
          .addComponent(ImportFileButton, javax.swing.GroupLayout.Alignment.TRAILING))
        .addContainerGap())
    );
    InputFilePanelLayout.setVerticalGroup(
      InputFilePanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGroup(InputFilePanelLayout.createSequentialGroup()
        .addContainerGap()
        .addComponent(ImportFileLabel)
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addComponent(ImportFileTextField, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addComponent(ImportFileButton)
        .addContainerGap(306, Short.MAX_VALUE))
    );

    InputFilePanel.setBounds(0, 0, 200, 390);
    ChooseTaskLayeredPane.add(InputFilePanel, javax.swing.JLayeredPane.DEFAULT_LAYER);

    DataSetSubmitButton.setText(resourceMap.getString("DataSetSubmitButton.text"));
    DataSetSubmitButton.setName("DataSetSubmitButton");
    DataSetSubmitButton.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
        DataSetSubmitButtonActionPerformed(evt);
      }
    });

    javax.swing.GroupLayout DataSetPanelLayout = new javax.swing.GroupLayout(DataSetPanel);
    DataSetPanel.setLayout(DataSetPanelLayout);
    DataSetPanelLayout.setHorizontalGroup(
      DataSetPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
```

```
            .addGroup(DataSetPanelLayout.createSequentialGroup()
                .addContainerGap()
                .addGroup(DataSetPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(ChooseTaskLayeredPane, javax.swing.GroupLayout.DEFAULT_SIZE, 200, Short.MAX_VALUE)
                    .addComponent(ChooseTaskLabel)
                    .addComponent(ManualImportComboBox, 0, 200, Short.MAX_VALUE)
                    .addComponent(DataSetSubmitButton))
                .addContainerGap())
        );
        DataSetPanelLayout.setVerticalGroup(
            DataSetPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(DataSetPanelLayout.createSequentialGroup()
                .addContainerGap()
                .addComponent(ChooseTaskLabel)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(ManualImportComboBox, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                .addComponent(ChooseTaskLayeredPane, javax.swing.GroupLayout.DEFAULT_SIZE, 398, Short.MAX_VALUE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(DataSetSubmitButton)
                .addContainerGap())
        );

        PreviewPanel.setBorder(javax.swing.BorderFactory.createTitledBorder(resourceMap.getString("PreviewPanel.border.title")))
;
        PreviewPanel.setName("PreviewPanel");

        DataSetScrollPane.setBackground(resourceMap.getColor("DataSetScrollPane.background"));
        DataSetScrollPane.setName("DataSetScrollPane");

        DataSetTable.setModel(new javax.swing.table.DefaultTableModel(
            new Object [][] {

            },
            new String [] {

            }
        ));
        DataSetTable.setAutoResizeMode(javax.swing.JTable.AUTO_RESIZE_OFF);
        DataSetTable.setName("DataSetTable");
        DataSetTable.setRowSelectionAllowed(false);
        DataSetTable.getTableHeader().setResizingAllowed(false);
        DataSetTable.getTableHeader().setReorderingAllowed(false);
        DataSetTable.addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyPressed(java.awt.event.KeyEvent evt) {
                DataSetTableKeyPressed(evt);
            }
        });
        DataSetScrollPane.setViewportView(DataSetTable);

        NormalizeDataButton.setText(resourceMap.getString("NormalizeDataButton.text"));
        NormalizeDataButton.setName("NormalizeDataButton");
        NormalizeDataButton.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                NormalizeDataButtonActionPerformed(evt);
            }
        });
```

```
        ExportFileButton.setText(resourceMap.getString("ExportFileButton.text"));
        ExportFileButton.setName("ExportFileButton");
        ExportFileButton.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                ExportFileButtonActionPerformed(evt);
            }
        });

        oneOfMappingButton.setText(resourceMap.getString("oneOfMappingButton.text"));
        oneOfMappingButton.setName("oneOfMappingButton");
        oneOfMappingButton.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                oneOfMappingButtonActionPerformed(evt);
            }
        });

        DataSetComboBox.setModel(new javax.swing.DefaultComboBoxModel(new String[] { "Original Data", "Mapped Data",
"Normalized Data" }));
        DataSetComboBox.setName("DataSetComboBox");
        DataSetComboBox.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                DataSetComboBoxActionPerformed(evt);
            }
        });

        javax.swing.GroupLayout PreviewPanelLayout = new javax.swing.GroupLayout(PreviewPanel);
        PreviewPanel.setLayout(PreviewPanelLayout);
        PreviewPanelLayout.setHorizontalGroup(
            PreviewPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, PreviewPanelLayout.createSequentialGroup()
                .addContainerGap()
                .addGroup(PreviewPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
                    .addComponent(DataSetScrollPane, javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE, 526, Short.MAX_VALUE)
                    .addGroup(javax.swing.GroupLayout.Alignment.LEADING, PreviewPanelLayout.createSequentialGroup()
                        .addComponent(DataSetComboBox, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(oneOfMappingButton)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(NormalizeDataButton)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(ExportFileButton)))
                .addContainerGap())
        );
        PreviewPanelLayout.setVerticalGroup(
            PreviewPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, PreviewPanelLayout.createSequentialGroup()
                .addComponent(DataSetScrollPane, javax.swing.GroupLayout.DEFAULT_SIZE, 462, Short.MAX_VALUE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addGroup(PreviewPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(oneOfMappingButton)
                    .addComponent(NormalizeDataButton)
                    .addComponent(ExportFileButton)
                    .addComponent(DataSetComboBox, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
                .addContainerGap())
```

```
    );

    javax.swing.GroupLayout DataSetTabLayout = new javax.swing.GroupLayout(DataSetTab);
    DataSetTab.setLayout(DataSetTabLayout);
    DataSetTabLayout.setHorizontalGroup(
      DataSetTabLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGroup(DataSetTabLayout.createSequentialGroup()
        .addContainerGap()
        .addComponent(DataSetPanel, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addComponent(PreviewPanel, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
        .addContainerGap())
    );
    DataSetTabLayout.setVerticalGroup(
      DataSetTabLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGroup(DataSetTabLayout.createSequentialGroup()
        .addContainerGap()
        .addGroup(DataSetTabLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
          .addComponent(PreviewPanel, javax.swing.GroupLayout.Alignment.TRAILING,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
          .addComponent(DataSetPanel, javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
        .addContainerGap())
    );

    mainTabPane.addTab(resourceMap.getString("DataSetTab.TabConstraints.tabTitle"), DataSetTab);

    TrainingTab.setName("TrainingTab");

    TrainingStatusPanel.setBorder(javax.swing.BorderFactory.createTitledBorder(resourceMap.getString("'TrainingStatusPanel.b
order.title")));
    TrainingStatusPanel.setName("'TrainingStatusPanel");

    TrainingStatusScrollPane.setName("'TrainingStatusScrollPane");

    TraningStatusArea.setEditable(false);
    TraningStatusArea.setLineWrap(true);
    TraningStatusArea.setRows(5);
    TraningStatusArea.setWrapStyleWord(true);
    TraningStatusArea.setName("TraningStatusArea");
    TrainingStatusScrollPane.setViewportView(TraningStatusArea);

    ErrorGraphButton.setText(resourceMap.getString("ErrorGraphButton.text"));
    ErrorGraphButton.setName("ErrorGraphButton");
    ErrorGraphButton.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
        ErrorGraphButtonActionPerformed(evt);
      }
    });

    javax.swing.GroupLayout TrainingStatusPanelLayout = new javax.swing.GroupLayout(TrainingStatusPanel);
    TrainingStatusPanel.setLayout(TrainingStatusPanelLayout);
    TrainingStatusPanelLayout.setHorizontalGroup(
      TrainingStatusPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, TrainingStatusPanelLayout.createSequentialGroup()
        .addContainerGap()
```

```
            .addGroup(TrainingStatusPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
               .addComponent(TrainingStatusScrollPane, javax.swing.GroupLayout.Alignment.LEADING)
               .addComponent(ErrorGraphButton))
            .addContainerGap())
      );
      TrainingStatusPanelLayout.setVerticalGroup(
         TrainingStatusPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
         .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, TrainingStatusPanelLayout.createSequentialGroup()
            .addComponent(TrainingStatusScrollPane, javax.swing.GroupLayout.DEFAULT_SIZE, 394, Short.MAX_VALUE)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
            .addComponent(ErrorGraphButton)
            .addContainerGap())
      );

      TrainButton.setText(resourceMap.getString("TrainButton.text"));
      TrainButton.setName("TrainButton");
      TrainButton.addActionListener(new java.awt.event.ActionListener() {
         public void actionPerformed(java.awt.event.ActionEvent evt) {
            TrainButtonActionPerformed(evt);
         }
      });

      PauseButton.setText(resourceMap.getString("PauseButton.text"));
      PauseButton.setName("PauseButton");
      PauseButton.addActionListener(new java.awt.event.ActionListener() {
         public void actionPerformed(java.awt.event.ActionEvent evt) {
            PauseButtonActionPerformed(evt);
         }
      });

      StopButton.setText(resourceMap.getString("StopButton.text"));
      StopButton.setName("StopButton");
      StopButton.addActionListener(new java.awt.event.ActionListener() {
         public void actionPerformed(java.awt.event.ActionEvent evt) {
            StopButtonActionPerformed(evt);
         }
      });

      TrainingVariableButton.setText(resourceMap.getString("TrainingVariableButton.text"));
      TrainingVariableButton.setName("TrainingVariableButton");
      TrainingVariableButton.addActionListener(new java.awt.event.ActionListener() {
         public void actionPerformed(java.awt.event.ActionEvent evt) {
            TrainingVariableButtonActionPerformed(evt);
         }
      });

      javax.swing.GroupLayout TrainingTabLayout = new javax.swing.GroupLayout(TrainingTab);
      TrainingTab.setLayout(TrainingTabLayout);
      TrainingTabLayout.setHorizontalGroup(
         TrainingTabLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
         .addGroup(TrainingTabLayout.createSequentialGroup()
            .addContainerGap()
            .addGroup(TrainingTabLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
               .addComponent(TrainingStatusPanel, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
               .addGroup(TrainingTabLayout.createSequentialGroup()
                  .addComponent(TrainButton)
                  .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
```

```
                .addComponent(PauseButton)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(StopButton))
            .addComponent(TrainingVariableButton))
        .addContainerGap())
    );
    TrainingTabLayout.setVerticalGroup(
      TrainingTabLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, TrainingTabLayout.createSequentialGroup()
        .addContainerGap()
        .addComponent(TrainingVariableButton)
        .addGap(11, 11, 11)
        .addComponent(TrainingStatusPanel, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addGroup(TrainingTabLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
          .addComponent(TrainButton)
          .addComponent(PauseButton)
          .addComponent(StopButton))
        .addContainerGap())
    );

    mainTabPane.addTab(resourceMap.getString("TrainingTab.TabConstraints.tabTitle"), TrainingTab);

    EvaluationTab.setName("EvaluationTab");

    jPanel6.setBorder(javax.swing.BorderFactory.createTitledBorder(resourceMap.getString("jPanel6.border.title")));
    jPanel6.setName("jPanel6");

    EvalPreviewPanel.setBorder(javax.swing.BorderFactory.createTitledBorder(resourceMap.getString("EvalPreviewPanel.border
.title")));
    EvalPreviewPanel.setName("EvalPreviewPanel");

    jScrollPane3.setName("jScrollPane3");

    EvalTable.setModel(new javax.swing.table.DefaultTableModel(
      new Object [][] {

      },
      new String [] {

      }
    ));
    EvalTable.setAutoResizeMode(javax.swing.JTable.AUTO_RESIZE_OFF);
    EvalTable.setName("EvalTable");
    EvalTable.setRowSelectionAllowed(false);
    jScrollPane3.setViewportView(EvalTable);

    javax.swing.GroupLayout EvalPreviewPanelLayout = new javax.swing.GroupLayout(EvalPreviewPanel);
    EvalPreviewPanel.setLayout(EvalPreviewPanelLayout);
    EvalPreviewPanelLayout.setHorizontalGroup(
      EvalPreviewPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGroup(EvalPreviewPanelLayout.createSequentialGroup()
        .addContainerGap()
        .addComponent(jScrollPane3, javax.swing.GroupLayout.DEFAULT_SIZE, 484, Short.MAX_VALUE))
    );
    EvalPreviewPanelLayout.setVerticalGroup(
      EvalPreviewPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
```

```
        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, EvalPreviewPanelLayout.createSequentialGroup()
            .addContainerGap()
            .addComponent(jScrollPane3, javax.swing.GroupLayout.DEFAULT_SIZE, 340, Short.MAX_VALUE))
    );

    EvaluateButton.setText(resourceMap.getString("EvaluateButton.text"));
    EvaluateButton.setName("EvaluateButton");
    EvaluateButton.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            EvaluateButtonActionPerformed(evt);
        }
    });

    EvalTextBox.setText(resourceMap.getString("EvalTextBox.text"));
    EvalTextBox.setName("EvalTextBox");

    EvalOpenFileButton.setText(resourceMap.getString("EvalOpenFileButton.text"));
    EvalOpenFileButton.setName("EvalOpenFileButton");
    EvalOpenFileButton.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            EvalOpenFileButtonActionPerformed(evt);
        }
    });

    EvalFileLabel.setText(resourceMap.getString("EvalFileLabel.text"));
    EvalFileLabel.setName("EvalFileLabel");

    EvalExportButton.setText(resourceMap.getString("EvalExportButton.text"));
    EvalExportButton.setName("EvalExportButton");
    EvalExportButton.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            EvalExportButtonActionPerformed(evt);
        }
    });

    EvalTypeLabel.setText(resourceMap.getString("EvalTypeLabel.text"));
    EvalTypeLabel.setName("EvalTypeLabel");

    EvalTypeComboBox.setModel(new javax.swing.DefaultComboBoxModel(new String[] { "Patient Classification", "Accuracy
Testing" }));
    EvalTypeComboBox.setName("EvalTypeComboBox");
    EvalTypeComboBox.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            EvalTypeComboBoxActionPerformed(evt);
        }
    });

    javax.swing.GroupLayout jPanel6Layout = new javax.swing.GroupLayout(jPanel6);
    jPanel6.setLayout(jPanel6Layout);
    jPanel6Layout.setHorizontalGroup(
        jPanel6Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(jPanel6Layout.createSequentialGroup()
            .addContainerGap()
            .addGroup(jPanel6Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addComponent(EvalPreviewPanel, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                .addGroup(jPanel6Layout.createSequentialGroup()
                    .addComponent(EvaluateButton)
```

```
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(EvalExportButton))
            .addGroup(jPanel6Layout.createSequentialGroup()
                .addComponent(EvalTypeLabel)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                .addComponent(EvalTypeComboBox, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
            .addGroup(jPanel6Layout.createSequentialGroup()
                .addComponent(EvalFileLabel)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                .addComponent(EvalTextBox, javax.swing.GroupLayout.DEFAULT_SIZE, 416, Short.MAX_VALUE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                .addComponent(EvalOpenFileButton)))
        .addContainerGap())
    );
    jPanel6Layout.setVerticalGroup(
      jPanel6Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, jPanel6Layout.createSequentialGroup()
        .addContainerGap()
        .addGroup(jPanel6Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
          .addComponent(EvalTypeLabel)
          .addComponent(EvalTypeComboBox, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
        .addGap(18, 18, 18)
        .addGroup(jPanel6Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
          .addComponent(EvalFileLabel)
          .addGroup(jPanel6Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(EvalTextBox, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(EvalOpenFileButton)))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
        .addComponent(EvalPreviewPanel, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addGroup(jPanel6Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
          .addComponent(EvaluateButton, javax.swing.GroupLayout.PREFERRED_SIZE, 23,
javax.swing.GroupLayout.PREFERRED_SIZE)
          .addComponent(EvalExportButton))
        .addContainerGap())
    );

    jPanel1.setBorder(javax.swing.BorderFactory.createTitledBorder(""));
    jPanel1.setName("jPanel1");

    jPanel2.setBorder(javax.swing.BorderFactory.createEtchedBorder());
    jPanel2.setName("jPanel2");

    jScrollPane5.setName("jScrollPane5");

    StatTextArea.setColumns(20);
    StatTextArea.setRows(5);
    StatTextArea.setName("StatTextArea");
    jScrollPane5.setViewportView(StatTextArea);

    AccuracyButton.setText(resourceMap.getString("AccuracyButton.text"));
    AccuracyButton.setName("AccuracyButton");
    AccuracyButton.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
```

```
            AccuracyButtonActionPerformed(evt);
        }
    });

    ClassButton.setText(resourceMap.getString("ClassButton.text"));
    ClassButton.setName("ClassButton");
    ClassButton.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            ClassButtonActionPerformed(evt);
        }
    });

    javax.swing.GroupLayout jPanel2Layout = new javax.swing.GroupLayout(jPanel2);
    jPanel2.setLayout(jPanel2Layout);
    jPanel2Layout.setHorizontalGroup(
        jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, jPanel2Layout.createSequentialGroup()
            .addContainerGap()
            .addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
                .addComponent(jScrollPane5, javax.swing.GroupLayout.Alignment.LEADING, javax.swing.GroupLayout.DEFAULT_SIZE,
196, Short.MAX_VALUE)
                .addComponent(AccuracyButton, javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE, 196, Short.MAX_VALUE)
                .addComponent(ClassButton, javax.swing.GroupLayout.DEFAULT_SIZE, 196, Short.MAX_VALUE))
            .addContainerGap())
    );
    jPanel2Layout.setVerticalGroup(
        jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, jPanel2Layout.createSequentialGroup()
            .addContainerGap()
            .addComponent(jScrollPane5, javax.swing.GroupLayout.DEFAULT_SIZE, 234, Short.MAX_VALUE)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
            .addComponent(ClassButton)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(AccuracyButton)
            .addContainerGap())
    );

    jPanel3.setBorder(javax.swing.BorderFactory.createEtchedBorder());
    jPanel3.setName("jPanel3");

    javax.swing.GroupLayout jPanel3Layout = new javax.swing.GroupLayout(jPanel3);
    jPanel3.setLayout(jPanel3Layout);
    jPanel3Layout.setHorizontalGroup(
        jPanel3Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGap(0, 216, Short.MAX_VALUE)
    );
    jPanel3Layout.setVerticalGroup(
        jPanel3Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGap(0, 161, Short.MAX_VALUE)
    );

    javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);
    jPanel1.setLayout(jPanel1Layout);
    jPanel1Layout.setHorizontalGroup(
        jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, jPanel1Layout.createSequentialGroup()
            .addContainerGap()
```

```
            .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
                .addComponent(jPanel3, javax.swing.GroupLayout.Alignment.LEADING, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .addComponent(jPanel2, javax.swing.GroupLayout.Alignment.LEADING, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
            .addContainerGap())
        );
        jPanel1Layout.setVerticalGroup(
            jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jPanel2, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(jPanel3, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                .addContainerGap())
        );

        javax.swing.GroupLayout EvaluationTabLayout = new javax.swing.GroupLayout(EvaluationTab);
        EvaluationTab.setLayout(EvaluationTabLayout);
        EvaluationTabLayout.setHorizontalGroup(
            EvaluationTabLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, EvaluationTabLayout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jPanel6, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap())
        );
        EvaluationTabLayout.setVerticalGroup(
            EvaluationTabLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(EvaluationTabLayout.createSequentialGroup()
                .addContainerGap()
                .addGroup(EvaluationTabLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(jPanel6, javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                    .addComponent(jPanel1, javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
                .addContainerGap())
        );

        mainTabPane.addTab(resourceMap.getString("EvaluationTab.TabConstraints.tabTitle"), EvaluationTab);

        javax.swing.GroupLayout mainPanelLayout = new javax.swing.GroupLayout(mainPanel);
        mainPanel.setLayout(mainPanelLayout);
        mainPanelLayout.setHorizontalGroup(
            mainPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, mainPanelLayout.createSequentialGroup()
                .addContainerGap()
                .addComponent(mainTabPane, javax.swing.GroupLayout.DEFAULT_SIZE, 821, Short.MAX_VALUE)
                .addContainerGap())
        );
        mainPanelLayout.setVerticalGroup(
            mainPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(mainPanelLayout.createSequentialGroup()
```

```
            .addContainerGap()
            .addComponent(mainTabPane, javax.swing.GroupLayout.DEFAULT_SIZE, 575, Short.MAX_VALUE)
            .addContainerGap())
        );

        mainTabPane.getAccessibleContext().setAccessibleName(resourceMap.getString("jTabbedPane1.AccessibleContext.accessible
Name"));

        menuBar.setName("menuBar");

        fileMenu.setText(resourceMap.getString("fileMenu.text"));
        fileMenu.setName("fileMenu");

        NewProject.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_N,
java.awt.event.InputEvent.CTRL_MASK));
        NewProject.setText(resourceMap.getString("NewProject.text"));
        NewProject.setName("NewProject");
        NewProject.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                NewProjectActionPerformed(evt);
            }
        });
        fileMenu.add(NewProject);

        OpenProject.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_O,
java.awt.event.InputEvent.SHIFT_MASK | java.awt.event.InputEvent.CTRL_MASK));
        OpenProject.setLabel(resourceMap.getString("OpenProject.label"));
        OpenProject.setName("OpenProject");
        OpenProject.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                OpenProjectActionPerformed(evt);
            }
        });
        fileMenu.add(OpenProject);

        jSeparator1.setName("jSeparator1");
        fileMenu.add(jSeparator1);

        SaveProject.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_S,
java.awt.event.InputEvent.CTRL_MASK));
        SaveProject.setEnabled(false);
        SaveProject.setLabel(resourceMap.getString("SaveProject.label"));
        SaveProject.setName("SaveProject");
        SaveProject.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                SaveProjectActionPerformed(evt);
            }
        });
        fileMenu.add(SaveProject);

        SaveAsProject.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_S,
java.awt.event.InputEvent.SHIFT_MASK | java.awt.event.InputEvent.CTRL_MASK));
        SaveAsProject.setEnabled(false);
        SaveAsProject.setLabel(resourceMap.getString("SaveAsProject.label"));
        SaveAsProject.setName("SaveAsProject");
        SaveAsProject.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                SaveAsProjectActionPerformed(evt);
```

```
        }
    });
    fileMenu.add(SaveAsProject);

    jSeparator3.setName("jSeparator3");
    fileMenu.add(jSeparator3);

    javax.swing.ActionMap actionMap =
org.jdesktop.application.Application.getInstance(PredicTB.PredicTBApp.class).getContext().getActionMap(PredicTBView.class,
this);
    exitMenuItem.setAction(actionMap.get("quit"));
    exitMenuItem.setName("exitMenuItem");
    fileMenu.add(exitMenuItem);

    menuBar.add(fileMenu);

    helpMenu.setText(resourceMap.getString("helpMenu.text"));
    helpMenu.setName("helpMenu");
    helpMenu.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            helpMenuActionPerformed(evt);
        }
    });

    AboutMenuItem.setText(resourceMap.getString("AboutMenuItem.text"));
    AboutMenuItem.setName("AboutMenuItem");
    AboutMenuItem.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            AboutMenuItemActionPerformed(evt);
        }
    });
    helpMenu.add(AboutMenuItem);

    jSeparator5.setName("jSeparator5");
    helpMenu.add(jSeparator5);

    HelpMenuItem.setText(resourceMap.getString("HelpMenuItem.text"));
    HelpMenuItem.setName("HelpMenuItem");
    helpMenu.add(HelpMenuItem);

    menuBar.add(helpMenu);

    statusPanel.setName("statusPanel");

    statusPanelSeparator.setName("statusPanelSeparator");

    statusMessageLabel.setName("statusMessageLabel");

    statusAnimationLabel.setHorizontalAlignment(javax.swing.SwingConstants.LEFT);
    statusAnimationLabel.setName("statusAnimationLabel");

    progressBar.setName("progressBar");

    jLabel3.setText(resourceMap.getString("jLabel3.text"));
    jLabel3.setName("jLabel3");

    javax.swing.GroupLayout statusPanelLayout = new javax.swing.GroupLayout(statusPanel);
    statusPanel.setLayout(statusPanelLayout);
```

```
        statusPanelLayout.setHorizontalGroup(
            statusPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(statusPanelSeparator, javax.swing.GroupLayout.DEFAULT_SIZE, 841, Short.MAX_VALUE)
            .addGroup(statusPanelLayout.createSequentialGroup()
                .addContainerGap()
                .addComponent(statusMessageLabel)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(jLabel3, javax.swing.GroupLayout.DEFAULT_SIZE, 643, Short.MAX_VALUE)
                .addGap(18, 18, 18)
                .addComponent(progressBar, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(statusAnimationLabel)
                .addContainerGap())
        );
        statusPanelLayout.setVerticalGroup(
            statusPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(statusPanelLayout.createSequentialGroup()
                .addComponent(statusPanelSeparator, javax.swing.GroupLayout.PREFERRED_SIZE, 2,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addGroup(statusPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, statusPanelLayout.createSequentialGroup()
                        .addGroup(statusPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                            .addComponent(statusMessageLabel)
                            .addComponent(statusAnimationLabel)
                            .addComponent(progressBar, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
                        .addGap(3, 3, 3))
                    .addGroup(statusPanelLayout.createSequentialGroup()
                        .addComponent(jLabel3, javax.swing.GroupLayout.DEFAULT_SIZE, 14, Short.MAX_VALUE)
                        .addContainerGap())))
        );

        jFrame1.setName("jFrame1");

        javax.swing.GroupLayout jFrame1Layout = new javax.swing.GroupLayout(jFrame1.getContentPane());
        jFrame1.getContentPane().setLayout(jFrame1Layout);
        jFrame1Layout.setHorizontalGroup(
            jFrame1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 400, Short.MAX_VALUE)
        );
        jFrame1Layout.setVerticalGroup(
            jFrame1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 300, Short.MAX_VALUE)
        );

        setComponent(mainPanel);
        setMenuBar(menuBar);
        setStatusBar(statusPanel);
    }// </editor-fold>

    private void dataSetSubmit() {
        model.setColumnCount(0);
        model1.setColumnCount(0);

        if (ManualImportComboBox.getSelectedIndex() == 0) {
```

```
int i = 0;
if(jCheckBox1.isSelected()){
   i++;
}
if(jCheckBox2.isSelected()){
   i++;
}
if(jCheckBox3.isSelected()){
   i++;
}
if(jCheckBox4.isSelected()){
   i++;
}
if(jCheckBox5.isSelected()){
   i++;
}
if(jCheckBox6.isSelected()){
   i++;
}
if(jCheckBox7.isSelected()){
   i++;
}
if(jCheckBox8.isSelected()){
   i++;
}
if(jCheckBox9.isSelected()){
   i++;
}

StringTokenizer st = new StringTokenizer(HeadersEditorPane.getText(), "\n");
headers = new String[(i + st.countTokens())];

i = 0;
if(jCheckBox1.isSelected()){
   headers[i] = "Coughing";
   i++;
}
if(jCheckBox2.isSelected()){
   headers[i] = "Sputum Discharge";
   i++;
}
if(jCheckBox3.isSelected()){
   headers[i] = "Hemoptysis";
   i++;
}
if(jCheckBox4.isSelected()){
   headers[i] = "Fever";
   i++;
}
if(jCheckBox5.isSelected()){
   headers[i] = "Weight Loss";
   i++;
}
if(jCheckBox6.isSelected()){
   headers[i] = "Chest Pain";
   i++;
}
if(jCheckBox7.isSelected()){
```

```
            headers[i] = "Back Pain";
            i++;
        }
        if(jCheckBox8.isSelected()){
            headers[i] = "Dyspnea";
            i++;
        }
        if(jCheckBox9.isSelected()){
            headers[i] = "HIV Positive";
            i++;
        }

        while (st.hasMoreTokens()) {
            headers[i] = st.nextElement().toString();
            i++;
        }

        for(int j = 0; j < headers.length; j++){
            model.addColumn(headers[j]);
            model1.addColumn(headers[j]);
        }

        model.setRowCount(1);
        model1.setRowCount(1);
    } else {
        if (trainingFile == null) {
            //no training file
        } else if (trainingFile.getName().toLowerCase().toString().contains(".csv")) {
            //csv file

            HandleCSV hc = new HandleCSV();
            hc.setCsvFile(trainingFile);
            headers = hc.getFileHeaders();
            for (int i = 0; i < headers.length; i++) {
                model.addColumn(headers[i]);
                model1.addColumn(headers[i]);
            }
            origData = hc.getFileData();
        } else {
            //Excel File

            HandleExcel he = new HandleExcel();
            he.setXlsFile(trainingFile);
            headers = he.getFileHeaders();
            for (int i = 0; i < headers.length; i++) {
                model.addColumn(headers[i]);
                model1.addColumn(headers[i]);
            }
            origData = he.getFileData();
        }
        model.setRowCount(origData.size());
        model1.setRowCount(1);
        convertArrayListToTable(origData, DataSetTable);

        mf.setHeaders(headers);
        mf.setOrigData(origData);
        mf.createComponents();
```

```
      ArrayList<ArrayList<String>> m = null;
      ArrayList<ArrayList<Double>> n = null;
      mf.setMappedData(m);
      nf.setNormalizeData(n);
    }
}

private void helpMenuActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
}

private void AboutMenuItemActionPerformed(java.awt.event.ActionEvent evt) {
    showAboutBox();
}

/** TRAINING PHASE **/
private void evaluationSubmit() {
    try {
      if (evaluationFile == null) {
        //no training file
      } else if (evaluationFile.getName().toLowerCase().toString().contains(".csv")) {
        //csv file

        HandleCSV hc = new HandleCSV();
        hc.setCsvFile(evaluationFile);
        origEval = hc.getFileData();
      } else {
        //Excel File

        HandleExcel he = new HandleExcel();
        he.setXlsFile(evaluationFile);
        origEval = he.getFileData();
      }

      model1.setRowCount(0);
      model1.setRowCount(origEval.size());

      for (int i = 0; i < origEval.size(); i++) {
        ArrayList<String> temp = new ArrayList<String>();
        temp = origEval.get(i);
        for (int j = 0; j < temp.size(); j++) {
          if(j != nf.getOutputCol()){
            EvalTable.setValueAt(temp.get(j), i, j);
          }else{
            EvalTable.setValueAt("", i, j);
          }
        }
      }
    } catch (Exception e) {
      model1.setRowCount(1);
      dbox.createDialogBox("Error importing file!", "");
    }
}

private void SaveProjectActionPerformed(java.awt.event.ActionEvent evt) {
    if (project == null) {
      String name = JOptionPane.showInputDialog("Project name:");
      if (name != null) {
```

```
            try {
                name += ".ptb";
                File file = new File(name);
                if (file.exists()) {
                    if (dbox.createDialogBox("Project already exists. Overwrite?", "yesno") == 0) {
                        project = new ProjectFile(file);
                        saveFile();
                        project.writeFile();
                        SaveAsProject.setEnabled(true);
                        SaveProject.setEnabled(true);
                        jLabel3.setText("PredicTB - " + file.getCanonicalPath().toString());
                    }
                } else {
                    project = new ProjectFile(file);
                    saveFile();
                    project.writeFile();
                    SaveAsProject.setEnabled(true);
                    SaveProject.setEnabled(true);
                    jLabel3.setText("PredicTB - " + file.getCanonicalPath().toString());
                }
            } catch (IOException ex) {
                Logger.getLogger(PredicTBView.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
    } else {
        saveFile();
        project.writeFile();
    }
}

private void SaveAsProjectActionPerformed(java.awt.event.ActionEvent evt) {
    JFileChooser fileChooser = new JFileChooser(new File("."));
    fileChooser.showSaveDialog(null);

    if (fileChooser.getSelectedFile() != null) {
        project.setProjectFile(fileChooser.getSelectedFile());
        saveFile();
        project.writeFile();
    }
}

private void saveFile() {
    project.setContinueTraining(continueTrain);
    project.setOutputCol(nf.getOutputCol());
    project.setEpochs(variable.getEpochs());
    project.setActivationFunction(variable.getActivationFunction());
    project.setLearningRate(variable.getLearningRate());
    project.setMaxError(variable.getMaxError());
    project.setMomentum(variable.getMomentum());
    project.setNoOfHiddenNeurons(variable.getNoOfHiddenNeurons());

    project.setHeaders(this.headers);
    project.setNominalHeaders(this.nominalHeaders);
    project.setMinimum(this.minimum);
    project.setMaximum(this.maximum);

    project.setOrigData(this.origData);
    project.setMappedData(this.mappedData);
```

```
    this.normalizedData = nf.getNormalizeData();
    project.setNormalizedData(this.normalizedData);

    project.setOrigEval(this.origEval);
    project.setMappedEval(this.mappedEval);
    project.setNormalizedEval(this.normalizedEval);

    try{
        project.setWeight(network.getWeight());
        project.setWeightDelta(network.getWeightDelta());
        project.setWeightDeltaAcc(network.getWeightDeltaAcc());
        project.setThreshold(network.getThreshold());
        project.setThresholdDelta(network.getThresholdDelta());
        project.setThresholdDeltaAcc(network.getThresholdDeltaAcc());
    }catch(Exception e){

    }

    dbox.createDialogBox("Successfully saved to project file!", "");
}

private void NewProjectActionPerformed(java.awt.event.ActionEvent evt) {
    String name = JOptionPane.showInputDialog("Project name:");
    if(name != null){
        try {
            name += ".ptb";
            File file = new File(name);
            if(file.exists()){
                if (dbox.createDialogBox("Project already exists. Overwrite?", "yesno") == 0) {
                    project = new ProjectFile(file);
                    project.writeFile();
                    SaveAsProject.setEnabled(true);
                    SaveProject.setEnabled(true);
                    jLabel3.setText("PredicTB - " + file.getCanonicalPath().toString());
                }
            }else{
                project = new ProjectFile(file);
                project.writeFile();
                SaveAsProject.setEnabled(true);
                SaveProject.setEnabled(true);
                jLabel3.setText("PredicTB - " + file.getCanonicalPath().toString());
            }
        } catch (IOException ex) {
            Logger.getLogger(PredicTBView.class.getName()).log(Level.SEVERE, null, ex);
        }
    }else{

    }
}

private void OpenProjectActionPerformed(java.awt.event.ActionEvent evt) {
    JFileChooser fileChooser = new JFileChooser(new File("."));
    fileChooser.setFileFilter(new FileFilter(".ptb"));

    fileChooser.showOpenDialog(null);
    File projectFile = null;
```

```
boolean opened = true;
try {
   projectFile = fileChooser.getSelectedFile();
} catch (Exception e) {
   e.printStackTrace();
}

if (projectFile == null) {
   //canceled the file chooser
} else if (!projectFile.getName().toString().contains(".ptb")) {
   //file chosen is not xls or csv
   dbox.createDialogBox("Invalid project file!", "");
} else {
   //accept file
   SaveAsProject.setEnabled(true);
   SaveProject.setEnabled(true);
   mainTabPane.setEnabledAt(1, true);
   mainTabPane.setEnabledAt(2, true);
   mainTabPane.setEnabledAt(3, true);
   TraningStatusArea.setText("");
   EvalTextBox.setText("");

   try {
      jLabel3.setText("PredicTB - " + projectFile.getCanonicalPath().toString());
   } catch (IOException ex) {
      Logger.getLogger(PredicTBView.class.getName()).log(Level.SEVERE, null, ex);
   }

   project = new ProjectFile(projectFile);
   project.readFile();

   try {

      this.continueTrain = project.isContinueTraining();

      variable.setEpochs(project.getEpochs());
      variable.setActivationFunction(project.getActivationFunction());
      variable.setLearningRate(project.getLearningRate());
      variable.setMaxError(project.getMaxError());
      variable.setMomentum(project.getMomentum());
      variable.setNoOfHiddenNeurons(project.getNoOfHiddenNeurons());

      if (variable.getActivationFunction().equalsIgnoreCase("tanh")) {
         vf.ActivationComboBox.setSelectedIndex(1);
      } else {
         vf.ActivationComboBox.setSelectedIndex(0);
      }

      vf.EpochComboBox.setSelectedIndex(1);
      vf.LearningComboBox.setSelectedIndex(1);
      vf.ErrorComboBox.setSelectedIndex(1);
      vf.MomentumComboBox.setSelectedIndex(1);

      vf.EpochTextField.setText(Integer.toString(project.getEpochs()));
      vf.LearningTextField.setText(Double.toString(project.getLearningRate()));
      vf.ErrorTextField.setText(Double.toString(project.getMaxError()));
      vf.MomentumTextField.setText(Double.toString(project.getMomentum()));
```

```
if (project.getNoOfHiddenNeurons() == 0) {
   vf.HiddenComboBox.setSelectedIndex(0);
} else {
   vf.HiddenTextField.setText(Integer.toString(project.getNoOfHiddenNeurons()));
}

this.headers = project.getHeaders();
this.nominalHeaders = project.getNominalHeaders();
this.minimum = project.getMinimum();
this.maximum = project.getMaximum();
this.origData = project.getOrigData();

mf.setMappedData(ArrDoubleToString(project.getMappedData()));
mf.setNominalHeaders(project.getNominalHeaders());
mf.setOrigData(project.getOrigData());
mf.setMin(project.getMinimum());
mf.setMax(project.getMaximum());
this.mappedData = project.getMappedData();

nf.setNormalizeData(project.getNormalizedData());
nf.setNominalHeaders(project.getNominalHeaders());
nf.setHeaders(project.getHeaders());
nf.setOutputCol(project.getOutputCol());
nf.setMin(project.getMinimum());
nf.setMax(project.getMaximum());
nf.setMappedData(project.getMappedData());
this.normalizedData = project.getNormalizedData();

try {
   network = new Network(normalizedData.get(0).size() - 1,
         project.getActivationFunction(),
         project.getLearningRate(),
         project.getMaxError(),
         project.getMomentum(),
         project.getNoOfHiddenNeurons());

   network.setWeight(project.getWeight());
   network.setWeightDelta(project.getWeightDelta());
   network.setWeightDeltaAcc(project.getWeightDeltaAcc());
   network.setThreshold(project.getThreshold());
   network.setThresholdDelta(project.getThresholdDelta());
   network.setThresholdDeltaAcc(project.getThresholdDeltaAcc());
} catch (Exception e) {
}

model.setColumnCount(0);
model.setRowCount(0);
model1.setRowCount(0);
model1.setColumnCount(0);

if (origData != null) {
   for (int i = 0; i < headers.length; i++) {
      model.addColumn(headers[i]);
      model1.addColumn(headers[i]);
   }
   model.setRowCount(origData.size());
   model1.setRowCount(1);
   convertArrayListToTable(origData, DataSetTable);
```

```
            }

        } catch (Exception e) {
            opened = false;
        }

        if (opened == true) {
            dbox.createDialogBox("Successfully opened project file!", "");
        }
    }
}

    private void EvalOpenFileButtonActionPerformed(java.awt.event.ActionEvent evt) {
        JFileChooser fileChooser = new JFileChooser(new File("."));
        fileChooser.setFileFilter(new FileFilter(".csv"));
        fileChooser.addChoosableFileFilter(new FileFilter(".xls"));

        fileChooser.showOpenDialog(null);

        try {
            evaluationFile = fileChooser.getSelectedFile();
        } catch (Exception e) {
            e.printStackTrace();
            evaluationFile = null;
        }

        if (evaluationFile == null) {
            //canceled the file chooser
        } else if (!evaluationFile.getName().toString().contains(".csv") && !evaluationFile.getName().toString().contains(".xls")) {
            //file chosen is not xls or csv
            dbox.createDialogBox("Error reading file. Only .XLS or .CSV file can be imported!", "");
            evaluationFile = null;
        } else {
            //accept file
            EvalTextBox.setText(evaluationFile.toString());
            if (EvalTable.getRowCount() > 1) {
                if (dbox.createDialogBox("This will overwrite current data set. Continue?", "yesno") == 0) {
                    evaluationSubmit();
                }
            } else {
                evaluationSubmit();
            }
        }
}

    private void EvaluateButtonActionPerformed(java.awt.event.ActionEvent evt) {

        try{
            // get mapped
            MappingForm mform = new MappingForm(jFrame1);
            mform.setNominalHeaders(nominalHeaders);
            mform.setHeaders(headers);
            mform.setOrigData(origData);
            mform.setOrigEval(origEval);
            mform.setMappedDataDouble(mappedData);
            mform.map();
            mappedEval = mform.getMappedDataReturn();
```

```
// normalize
Normalization norm = new Normalization();
norm.setMax(maximum);
norm.setMin(minimum);
norm.setMappedArray(mappedEval);
norm.setNormalizedArray(normalizedEval);
norm.normalize();
normalizedEval = norm.getNormalizedArray();


ArrayList<ArrayList<Double>> actualEval;
NormalizeForm nform = new NormalizeForm(jFrame1);

if(EvalTypeComboBox.getSelectedIndex() == 1){
   // cut to ideal and actual
   nform.setOutputCol(nf.getOutputCol());
   nform.setNormalizeData(normalizedEval);
   actualEval = nform.getActualArray();
}else{
   actualEval = normalizedEval;
}

Normalization norm1 = new Normalization();
double[] max = new double[normalizedEval.get(0).size()];
double[] min = new double[normalizedEval.get(0).size()];

//range ng tanh is -1 to 1 so normalize again
if (variable.getActivationFunction().equals("tanh")) {
   for (int i = 0; i < max.length; i++) {
      max[i] = 1;
      min[i] = -1;
   }

   norm1.setMax(max);
   norm1.setMin(min);
   norm1.setMappedArray(normalizedEval);
   norm1.normalize();
   ArrayList<ArrayList<Double>> newNormalized = norm1.getNormalizedArray();

   if(EvalTypeComboBox.getSelectedIndex() == 1){
      NormalizeForm nform1 = new NormalizeForm(jFrame1);
      nform1.setVisible(false);
      nform1.setNormalizeData(newNormalized);
      nform1.setOutputCol(nf.getOutputCol());
      actualEval = nform.getActualArray();
   }else{
      actualEval = newNormalized;
   }
}

double[] result = new double[actualEval.size()];
for (int i = 0; i < actualEval.size(); i++) {
   result[i] = network.propagate(actualEval.get(i));
}

//DENORMALIZE
```

```
// since tanh, put the range back to [0,1] like sig
if(network.getActFunction().equals("tanh")){
   for(int i = 0; i < min.length; i++){
      min[i] = 0;
      max[i] = 1;
   }
   norm1.setMin(min);
   norm1.setMax(max);
   result = norm1.denormalize(result, nf.getOutputCol());
}

//put it back to range of map data
result = norm.denormalize(result, nf.getOutputCol());
for(int i = 0; i < result.length; i++){
   result[i] = Math.round(result[i]);
}

prediction = new String[result.length];
for(int i = 0; i < result.length; i++){
   for(int j = 0; j < mappedData.size(); j++){
      if(result[i] == mappedData.get(j).get(nf.getOutputCol())){
         prediction[i] = origData.get(j).get(nf.getOutputCol());
         break;
      }
   }
}

for (int i = 0; i < normalizedEval.size(); i++){
   EvalTable.setValueAt(prediction[i], i, nf.getOutputCol());
}

try{
   StatTextArea.setText("");
   StatTextArea.append("\nNumber of evaluated data:" + prediction.length);
   StatTextArea.append("\n");
   wrong = 0;
   right = 0;
   if (EvalTypeComboBox.getSelectedIndex() == 1) {
      for (int i = 0; i < prediction.length; i++) {
         if (!prediction[i].equals(origEval.get(i).get(nf.getOutputCol()))) {
            //do something here na maghighlight nung cell na mali
            wrong++;
         } else {
            //tama
            right++;
         }
      }
      StatTextArea.append("\nCorrect predictions: " + right);
      StatTextArea.append("\nWrong predictions: " + wrong);
      StatTextArea.append("\n");
      StatTextArea.append("\nAccuracy Rate: " + roundTwoDecimals(((double)right/(double)(right+wrong))*100) + "%");
      ClassButton.setEnabled(true);
      AccuracyButton.setEnabled(true);
   } else {
      ClassButton.setEnabled(true);
      AccuracyButton.setEnabled(false);
   }
}catch(Exception ex){
```

```
            StatTextArea.setText("");
            ClassButton.setEnabled(false);
            AccuracyButton.setEnabled(false);
            EvalTypeComboBox.setSelectedIndex(0);
            dbox.createDialogBox("Accuracy testing cannot be done if output column is not provided!", "");
        }

    } catch (Exception e) {
        dbox.createDialogBox("An error was found. Please check if the network has already been trained or if there are empty cells
in the data!", "");
    }
}

  double roundTwoDecimals(double d) {
    DecimalFormat twoDForm = new DecimalFormat("#.##");
    return Double.valueOf(twoDForm.format(d));
  }

  private void TrainingVariableButtonActionPerformed(java.awt.event.ActionEvent evt) {
    vf.setVisible(true);
    vf.requestFocusInWindow();
}

  private void StopButtonActionPerformed(java.awt.event.ActionEvent evt) {

    try{
      if (dbox.createDialogBox("Stop training?", "yesno") == 0) {
        TrainButton.setEnabled(true);
        PauseButton.setEnabled(false);
        StopButton.setEnabled(false);
        if (continueTrain == true) {
          //nakapause nung nastop
          continueTrain = false;
        }
        threadStop = true;
        prune.threadStop = true;
        network.reset();
        network = null;
        worker = null;
      }
    }catch(Exception e){

    }
}

  private void PauseButtonActionPerformed(java.awt.event.ActionEvent evt) {
    try{
      TrainButton.setEnabled(true);
      PauseButton.setEnabled(false);
      StopButton.setEnabled(true);

      continueTrain = true;
      threadStop = true;
      prune.threadStop = true;
      worker = null;
    }catch(Exception e){

    }
```

```
}

    private void TrainButtonActionPerformed(java.awt.event.ActionEvent evt) {
        try {
            origEval = null;
            mappedEval = null;
            normalizedEval = null;
            model1.setColumnCount(0);
            for(int i = 0; i < headers.length; i++){
                model1.addColumn(headers[i]);
            }
            model1.setRowCount(1);
            EvalTextBox.setText("");
            evaluationFile = null;

            TrainButton.setEnabled(false);
            PauseButton.setEnabled(true);
            StopButton.setEnabled(true);
            if (worker != null) {
                worker = null;
            }
            worker = new Thread(this);
            threadStop = false;
            worker.setPriority(Thread.MIN_PRIORITY);
            worker.start();
        } catch (Exception e) {
        }
    }

    private void ErrorGraphButtonActionPerformed(java.awt.event.ActionEvent evt) {
        errgraph.setVisible(true);
}

    private void DataSetComboBoxActionPerformed(java.awt.event.ActionEvent evt) {
        if (DataSetComboBox.getSelectedIndex() == 1) {
            if (mf.getMappedData() == null) {
                DataSetComboBox.setSelectedIndex(0);
                DataSetComboBox.hidePopup();
                dbox.createDialogBox("No mapped data yet. Press oneOf mapping button.", "");
            } else {
                try {
                    mappedData = ArrStringToDouble(mf.getMappedData());
                    convertArrayListToTable(mf.getMappedData(), DataSetTable);
                } catch (Exception ex) {
                    DataSetComboBox.setSelectedIndex(0);
                    DataSetComboBox.setPopupVisible(false);
                    if (dbox.createDialogBox("Not all nominal-valued columns are mapped.", "") == 0) {
                        mf.setHeaders(headers);
                        mf.setOrigData(origData);
                        mf.createComponents();
                        mf.setVisible(true);
                        mf.requestFocusInWindow();
                    }
                }
            }
        } else if (DataSetComboBox.getSelectedIndex() == 2) {
            if (nf.getNormalizeData() == null) {
                DataSetComboBox.setSelectedIndex(0);
```

```
            DataSetComboBox.hidePopup();
            dbox.createDialogBox("No normalized data yet. Press normalize data button.", "");
        } else {
            try {
                normalizedData = nf.getNormalizeData();
                convertArrayListToTable(ArrDoubleToString(normalizedData), DataSetTable);
            } catch (Exception ex) {
                DataSetComboBox.setSelectedIndex(0);
                DataSetComboBox.setPopupVisible(false);
            }
        }
    } else {
        try{
            convertArrayListToTable(origData, DataSetTable);
        }catch(Exception e){

        }
    }
}

    private void oneOfMappingButtonActionPerformed(java.awt.event.ActionEvent evt) {

     try{
        origData.clear();

        for (int i = 0; i < DataSetTable.getRowCount(); i++) {
            ArrayList<String> temp = new ArrayList<String>();
            boolean add = true;
            for (int j = 0; j < DataSetTable.getColumnCount(); j++) {
                String content = DataSetTable.getModel().getValueAt(i, j).toString();
                if(content.equals(null) || content.isEmpty()){
                    add = false;
                    break;
                }
                temp.add(content);
            }
            if(add == true){
                origData.add(temp);
            }
        }

        nf.reset();
        mf.setHeaders(headers);
        mf.setOrigData(origData);
        mf.createComponents();
        mf.setVisible(true);
        mf.requestFocusInWindow();

     }catch(Exception e){
         dbox.createDialogBox("Empty cells are not allowed!", "");
     }
}

    private void ExportFileButtonActionPerformed(java.awt.event.ActionEvent evt) {

     ArrayList<ArrayList<String>> temp;
     if (DataSetComboBox.getSelectedIndex() == 0) {
        temp = origData;
```

```
            DataSetTable.setEnabled(true);
        } else if (DataSetComboBox.getSelectedIndex() == 1) {
            temp = ArrDoubleToString(mappedData);
            DataSetTable.setEnabled(false);
        } else {
            temp = ArrDoubleToString(normalizedData);
            DataSetTable.setEnabled(false);
        }

        if (DataSetTable.getColumnCount() != 0) {
            JFileChooser fileChooser = new JFileChooser(new File("."));
            fileChooser.showSaveDialog(null);

            if (fileChooser.getSelectedFile() != null) {
                if (fileChooser.getSelectedFile().toString().contains(".csv")) {
                    HandleCSV hc = new HandleCSV();
                    hc.setArr(temp);
                    hc.setHeaders(headers);
                    hc.setCsvFile(fileChooser.getSelectedFile());
                    hc.writeFile();
                    dbox.createDialogBox("Succesfully exported to " + fileChooser.getName(fileChooser.getSelectedFile()) + "!", "");
                } else if (fileChooser.getSelectedFile().toString().contains(".xls")) {
                    HandleExcel hc = new HandleExcel();
                    hc.setArr(temp);
                    hc.setHeaders(headers);
                    hc.setXlsFile(fileChooser.getSelectedFile());
                    hc.writeFile();
                    dbox.createDialogBox("Succesfully exported to " + fileChooser.getName(fileChooser.getSelectedFile()) + "!", "");
                } else {
                    dbox.createDialogBox("Cannot export to a file other than .CSV or .XLS", "");
                }
            }
        } else {
            dbox.createDialogBox("Nothing to export. Create a data set.", "");
        }
    }

    private void NormalizeDataButtonActionPerformed(java.awt.event.ActionEvent evt) {
        try {
            mappedData = ArrStringToDouble(mf.getMappedData());
            maximum = mf.getMax();
            minimum = mf.getMin();
            nominalHeaders = mf.getNominalHeaders();
            nf.setHeaders(headers);
            nf.setNominalHeaders(nominalHeaders);
            nf.setMappedData(mappedData);
            nf.setMax(maximum);
            nf.setMin(minimum);
            nf.createComponents();
            nf.setVisible(true);
            nf.requestFocusInWindow();
        } catch (Exception ex) {
            if (mf.getMappedData() == null) {
                dbox.createDialogBox("Mapped data is needed for normalizing.", "");
            } else {
                if (dbox.createDialogBox("Not all nominal-valued columns are mapped.", "") == 0) {
                    mf.setHeaders(headers);
                    mf.setOrigData(origData);
```

```
                mf.createComponents();
                mf.setVisible(true);
                mf.requestFocusInWindow();
            }
        }
    }
}

    private void DataSetSubmitButtonActionPerformed(java.awt.event.ActionEvent evt) {
        if (!ImportFileTextField.getText().toString().equals("") || ManualImportComboBox.getSelectedIndex() == 0) {
            trainingFile = new File(ImportFileTextField.getText().toString());
            if (DataSetTable.getColumnCount() != 0) {
                if (dbox.createDialogBox("This will overwrite current data set. Continue?", "yesno") == 0) {
                    dataSetSubmit();
                }
            } else {
                dataSetSubmit();
            }
        }
        DataSetComboBox.setSelectedIndex(0);
}

    private void ImportFileButtonActionPerformed(java.awt.event.ActionEvent evt) {
        JFileChooser fileChooser = new JFileChooser(new File("."));
        fileChooser.setFileFilter(new FileFilter(".csv"));
        fileChooser.addChoosableFileFilter(new FileFilter(".xls"));

        fileChooser.showOpenDialog(null);

        try {
            trainingFile = fileChooser.getSelectedFile();
        } catch (Exception e) {
            e.printStackTrace();
            trainingFile = null;
        }

        if (trainingFile == null) {
            //canceled the file chooser
        } else if (!trainingFile.getName().toString().contains(".csv") && !trainingFile.getName().toString().contains(".xls")) {
            //file chosen is not xls or csv
            dbox.createDialogBox("Only .XLS or .CSV file can be imported!", "");
            trainingFile = null;
        } else {
            //accept file
            ImportFileTextField.setText(trainingFile.toString());
        }
}

    private void ManualImportComboBoxActionPerformed(java.awt.event.ActionEvent evt) {
        if (ManualImportComboBox.getSelectedIndex() == 1) {
            InputHeaderPanel.setVisible(false);
            InputFilePanel.setVisible(true);
            ImportFileButton.setVisible(true);
            ImportFileLabel.setVisible(true);
            ImportFileTextField.setVisible(true);
        } else {
            InputHeaderPanel.setVisible(true);
            InputFilePanel.setVisible(false);
```

```
        ImportFileButton.setVisible(false);
        ImportFileLabel.setVisible(false);
        ImportFileTextField.setVisible(false);
        HeadersEditorPane.setText("");
    }
}

    private void DataSetTableKeyPressed(java.awt.event.KeyEvent evt) {
        if (evt.getKeyCode() == KeyEvent.VK_ENTER && DataSetTable.getEditingRow() == DataSetTable.getRowCount()-1) {
            model.addRow(new Object[]{});
        }
    }

    private void EvalExportButtonActionPerformed(java.awt.event.ActionEvent evt) {
        ArrayList<ArrayList<String>> temp = new ArrayList<ArrayList<String>>();
        convertTableToArrayList(temp, EvalTable);
        if (EvalTable.getColumnCount() != 0) {
            JFileChooser fileChooser = new JFileChooser(new File("."));
            fileChooser.showSaveDialog(null);

            if (fileChooser.getSelectedFile() != null) {
                if (fileChooser.getSelectedFile().toString().contains(".csv")) {
                    HandleCSV hc = new HandleCSV();
                    hc.setArr(temp);
                    hc.setHeaders(headers);
                    hc.setCsvFile(fileChooser.getSelectedFile());
                    hc.writeFile();
                    dbox.createDialogBox("Succesfully exported to " + fileChooser.getName(fileChooser.getSelectedFile()) + "!", "");
                } else if (fileChooser.getSelectedFile().toString().contains(".xls")) {
                    HandleExcel hc = new HandleExcel();
                    hc.setArr(temp);
                    hc.setHeaders(headers);
                    hc.setXlsFile(fileChooser.getSelectedFile());
                    hc.writeFile();
                    dbox.createDialogBox("Succesfully exported to " + fileChooser.getName(fileChooser.getSelectedFile()) + "!", "");
                } else {
                    dbox.createDialogBox("Cannot export to a file other than .CSV or .XLS", "");
                }
            }
        } else {
            dbox.createDialogBox("Nothing to export. Create a data set.", "");
        }
    }

    private void EvalTypeComboBoxActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }

    private void ClassButtonActionPerformed(java.awt.event.ActionEvent evt) {
        classchart = new ClassChart(prediction);
        classchart.pack();
        RefineryUtilities.centerFrameOnScreen(classchart);
        classchart.setVisible(true);
    }

    private void AccuracyButtonActionPerformed(java.awt.event.ActionEvent evt) {
         accChart = new AccuracyChart();
         double rightPerc = (double)right/(double)(right+wrong);
```

```
    double wrongPerc = (double)wrong/(double)(right+wrong);
    double hundred = 100;
    int rightPerc1 = (int)(rightPerc*hundred);
    int wrongPerc1 = (int)(wrongPerc*hundred);
    accChart.setIncorrect(wrongPerc1);
    accChart.setCorrect(rightPerc1);
    accChart.update();
    accChart.pack();
    RefineryUtilities.centerFrameOnScreen(accChart);
    accChart.setVisible(true);
}

// Variables declaration - do not modify
private javax.swing.JMenuItem AboutMenuItem;
private javax.swing.JButton AccuracyButton;
private javax.swing.JLabel ChooseTaskLabel;
private javax.swing.JLayeredPane ChooseTaskLayeredPane;
private javax.swing.JButton ClassButton;
private javax.swing.JComboBox DataSetComboBox;
private javax.swing.JPanel DataSetPanel;
private javax.swing.JScrollPane DataSetScrollPane;
private javax.swing.JButton DataSetSubmitButton;
private javax.swing.JPanel DataSetTab;
private javax.swing.JTable DataSetTable;
private javax.swing.JButton ErrorGraphButton;
private javax.swing.JButton EvalExportButton;
private javax.swing.JLabel EvalFileLabel;
private javax.swing.JButton EvalOpenFileButton;
private javax.swing.JPanel EvalPreviewPanel;
private javax.swing.JTable EvalTable;
private javax.swing.JTextField EvalTextBox;
private javax.swing.JComboBox EvalTypeComboBox;
private javax.swing.JLabel EvalTypeLabel;
private javax.swing.JButton EvaluateButton;
private javax.swing.JPanel EvaluationTab;
private javax.swing.JButton ExportFileButton;
private javax.swing.JEditorPane HeadersEditorPane;
private javax.swing.JLabel HeadersLabel;
private javax.swing.JMenuItem HelpMenuItem;
private javax.swing.JButton ImportFileButton;
private javax.swing.JLabel ImportFileLabel;
private javax.swing.JTextField ImportFileTextField;
private javax.swing.JPanel InputFilePanel;
private javax.swing.JPanel InputHeaderPanel;
private javax.swing.JComboBox ManualImportComboBox;
private javax.swing.JMenuItem NewProject;
private javax.swing.JButton NormalizeDataButton;
private javax.swing.JMenuItem OpenProject;
private javax.swing.JButton PauseButton;
private javax.swing.JPanel PreviewPanel;
private javax.swing.JMenuItem SaveAsProject;
private javax.swing.JMenuItem SaveProject;
private javax.swing.JTextArea StatTextArea;
private javax.swing.JButton StopButton;
private javax.swing.JButton TrainButton;
private javax.swing.JPanel TrainingStatusPanel;
private javax.swing.JScrollPane TrainingStatusScrollPane;
private javax.swing.JPanel TrainingTab;
```

```
private javax.swing.JButton TrainingVariableButton;
private javax.swing.JTextArea TraningStatusArea;
private javax.swing.JCheckBox jCheckBox1;
private javax.swing.JCheckBox jCheckBox2;
private javax.swing.JCheckBox jCheckBox3;
private javax.swing.JCheckBox jCheckBox4;
private javax.swing.JCheckBox jCheckBox5;
private javax.swing.JCheckBox jCheckBox6;
private javax.swing.JCheckBox jCheckBox7;
private javax.swing.JCheckBox jCheckBox8;
private javax.swing.JCheckBox jCheckBox9;
private javax.swing.JFrame jFrame1;
private javax.swing.JLabel jLabel3;
private javax.swing.JPanel jPanel1;
private javax.swing.JPanel jPanel2;
private javax.swing.JPanel jPanel3;
private javax.swing.JPanel jPanel6;
private javax.swing.JScrollPane jScrollPane2;
private javax.swing.JScrollPane jScrollPane3;
private javax.swing.JScrollPane jScrollPane5;
private javax.swing.JSeparator jSeparator1;
private javax.swing.JSeparator jSeparator3;
private javax.swing.JSeparator jSeparator5;
private javax.swing.JPanel mainPanel;
private javax.swing.JTabbedPane mainTabPane;
private javax.swing.JMenuBar menuBar;
private javax.swing.JButton oneOfMappingButton;
private javax.swing.JProgressBar progressBar;
private javax.swing.JLabel statusAnimationLabel;
private javax.swing.JLabel statusMessageLabel;
private javax.swing.JPanel statusPanel;
// End of variables declaration
private final Timer messageTimer;
private final Timer busyIconTimer;
private final Icon idleIcon;
private final Icon[] busyIcons = new Icon[15];
private int busyIconIndex = 0;
private JDialog aboutBox;

// Method Helpers
private void printArrayList(ArrayList<ArrayList<String>> arr) {
   for (int i = 0; i < arr.size(); i++) {
      ArrayList<String> temp = new ArrayList<String>();
      temp = arr.get(i);
      for (int j = 0; j < temp.size(); j++) {
         System.out.print(temp.get(j) + " ");
      }
      System.out.println();
   }
}

private void printArrayListDouble(ArrayList<ArrayList<Double>> arr) {
   for (int i = 0; i < arr.size(); i++) {
      ArrayList<Double> temp = new ArrayList<Double>();
      temp = arr.get(i);
      for (int j = 0; j < temp.size(); j++) {
         System.out.print(temp.get(j) + " ");
      }
```

```
            System.out.println();
        }
    }

    private void convertArrayListToTable(ArrayList<ArrayList<String>> arr, JTable table) {
        for (int i = 0; i < arr.size(); i++) {
            ArrayList<String> temp = new ArrayList<String>();
            temp = arr.get(i);
            for (int j = 0; j < temp.size(); j++) {
                table.setValueAt(temp.get(j), i, j);
            }
        }
    }

    private void convertTableToArrayList(ArrayList<ArrayList<String>> arr, JTable table) {
        for (int i = 0; i < table.getRowCount(); i++) {
            ArrayList<String> temp = new ArrayList<String>();
            for (int j = 0; j < table.getColumnCount(); j++) {
                temp.add(table.getValueAt(i, j).toString());
            }
            arr.add(temp);
        }
    }

    private ArrayList<ArrayList<String>> ArrDoubleToString(ArrayList<ArrayList<Double>> arr) {
        ArrayList<ArrayList<String>> ret = new ArrayList<ArrayList<String>>();

        for (int i = 0; i < arr.size(); i++) {
            ArrayList<Double> temp = new ArrayList<Double>();
            temp = arr.get(i);

            ArrayList<String> temp1 = new ArrayList<String>();
            for (int j = 0; j < temp.size(); j++) {
                temp1.add(Double.toString(temp.get(j)));
            }

            ret.add(temp1);
        }

        return ret;
    }

    private ArrayList<ArrayList<Double>> ArrStringToDouble(ArrayList<ArrayList<String>> arr) {
        ArrayList<ArrayList<Double>> ret = new ArrayList<ArrayList<Double>>();

        for (int i = 0; i < arr.size(); i++) {
            ArrayList<String> temp = new ArrayList<String>();
            temp = arr.get(i);

            ArrayList<Double> temp1 = new ArrayList<Double>();
            for (int j = 0; j < temp.size(); j++) {
                temp1.add(Double.parseDouble(temp.get(j)));
            }

            ret.add(temp1);
        }

        return ret;
```

```
}

public void run() {
   //USED FOR TRAINING
   threadStop = false;

   if (continueTrain == true) {
      // nakapause lang
      boolean success = true;
       if (nf.getNormalizeData() != null) { //dapat may normalized na

         TraningStatusArea.append("\nContinue training...\n");
         TraningStatusArea.setCaretPosition(TraningStatusArea.getText().length() - 1);
         ArrayList<ArrayList<Double>> actual = nf.getActualArray();
         ArrayList<Double> ideal = nf.getIdealArray();

         //range ng tanh is -1 to 1 so normalize again
         if(variable.getActivationFunction().equals("tanh")){
            Normalization norm = new Normalization();
            double [] max = new double[nf.getNormalizeData().get(0).size()];
            double [] min = new double[nf.getNormalizeData().get(0).size()];
            for(int i = 0; i < max.length; i++){
               max[i] = 1;
               min[i] = -1;
            }
            norm.setMax(max);
            norm.setMin(min);
            norm.setMappedArray(nf.getNormalizeData());
            norm.normalize();

            ArrayList<ArrayList<Double>> newNormalized = norm.getNormalizedArray();

            NormalizeForm nform = new NormalizeForm(jFrame1);
            nform.setVisible(false);
            nform.setNormalizeData(newNormalized);
            nform.setOutputCol(nf.getOutputCol());
            actual = nform.getActualArray();
            ideal = nform.getIdealArray();
         }

         if (vf.HiddenComboBox.getSelectedIndex() == 0) {
            //apply pruning
            prune = new Pruning(ideal,
                  actual, network.getActFunction(),
                  variable.getEpochs(), network.getLearningRate(),
                  network.getMaxError(), network.getMomentum());

            prune.threadStop = false;

            int hidden = variable.getNoOfHiddenNeurons();
            while (!prune.isSuccess()) {
               prune.start(hidden);
               variable.setNoOfHiddenNeurons(hidden);
               prune.prune(TraningStatusArea, errgraph);
               hidden++;
               if(threadStop == true){
                  success = false;
                  break;
```

```
            }
          }

          if(success == true){
            TraningStatusArea.append("\n\nFinal hidden neurons: " + (hidden-1) + ", Final Error = " + prune.getError());
            TraningStatusArea.setCaretPosition(TraningStatusArea.getText().length() - 1);
          }
          network = prune.getNetwork();
        } else {
          double error = 1;
          while (error > variable.getMaxError()) {
            boolean stop = false;

            for (int ctr = 0; ctr < variable.getEpochs(); ctr++) {
              for (int i = 0; i < actual.size(); i++) {
                network.propagate(actual.get(i));
                //jTextArea1.append("Actual: " + result + ", Ideal: " + ideal.get(i) + "\n");
                network.backpropagate(ideal.get(i));
                network.train();
              }

              error = network.getRMS(actual.size());
              errgraph.update(error);
              TraningStatusArea.append("\nEpoch " + (ctr + 1) + ", Error: " + error + "");
              TraningStatusArea.setCaretPosition(TraningStatusArea.getText().length() - 1);

              if (error < variable.getMaxError()) {
                break;
              }

              if(threadStop == true){
                success = false;
                stop = true;
                break;
              }
            }

            if(stop == true){
              break;
            }
          }
        }
        TraningStatusArea.append("\n...End of training\n");
        TraningStatusArea.setCaretPosition(TraningStatusArea.getText().length() - 1);
        if(success == true){
          dbox.createDialogBox("Training successful!", "");
          variable.setNoOfHiddenNeurons(network.getHiddenNeurons());
        }
      } else {
        dbox.createDialogBox("Normalized data is needed for training!", "");
      }
      TrainButton.setEnabled(true);
      PauseButton.setEnabled(false);
      StopButton.setEnabled(false);

    } else {
      // start training
       boolean success = true;
```

```
if (nf.getNormalizeData() != null) { //dapat may normalized na

   TraningStatusArea.setText("");
   TraningStatusArea.append("\nStart of training...\n");
   TraningStatusArea.setCaretPosition(TraningStatusArea.getText().length() - 1);
   ArrayList<ArrayList<Double>> actual = nf.getActualArray();
   ArrayList<Double> ideal = nf.getIdealArray();

   //range ng tanh is -1 to 1 so normalize again
   if(variable.getActivationFunction().equals("tanh")){
      Normalization norm = new Normalization();
      double [] max = new double[nf.getNormalizeData().get(0).size()];
      double [] min = new double[nf.getNormalizeData().get(0).size()];
      for(int i = 0; i < max.length; i++){
         max[i] = 1;
         min[i] = -1;
      }
      norm.setMax(max);
      norm.setMin(min);
      norm.setMappedArray(nf.getNormalizeData());
      norm.normalize();

      ArrayList<ArrayList<Double>> newNormalized = norm.getNormalizedArray();

      NormalizeForm nform = new NormalizeForm(jFrame1);
      nform.setVisible(false);
      nform.setNormalizeData(newNormalized);
      nform.setOutputCol(nf.getOutputCol());
      actual = nform.getActualArray();
      ideal = nform.getIdealArray();
   }

   if (variable.getNoOfHiddenNeurons() == 1) {
      //apply pruning
      prune = new Pruning(ideal,
            actual, variable.getActivationFunction(),
            variable.getEpochs(), variable.getLearningRate(),
            variable.getMaxError(), variable.getMomentum());

      prune.threadStop = false;

      int hidden = variable.getNoOfHiddenNeurons();
      while (!prune.isSuccess()) {
         prune.start(hidden);
         variable.setNoOfHiddenNeurons(hidden);
         prune.prune(TraningStatusArea, errgraph);
         hidden++;
         if(threadStop == true){
            success = false;
            break;
         }
      }

      if(success == true){
         TraningStatusArea.append("\n\nFinal hidden neurons: " + (hidden-1) + ", Final Error = " + prune.getError());
         TraningStatusArea.setCaretPosition(TraningStatusArea.getText().length() - 1);
      }
```

```
            network = prune.getNetwork();
          } else {
            network = new Network(actual.get(0).size(),
                variable.getActivationFunction(),
                variable.getLearningRate(), variable.getMaxError(),
                variable.getMomentum(), variable.getNoOfHiddenNeurons());
            double error = 1;
            while (error > variable.getMaxError()) {
              boolean stop = false;

              for (int ctr = 0; ctr < variable.getEpochs(); ctr++) {
                for (int i = 0; i < actual.size(); i++) {
                  network.propagate(actual.get(i));
                  //jTextArea1.append("Actual: " + result + ", Ideal: " + ideal.get(i) + "\n");
                  network.backpropagate(ideal.get(i));
                  network.train();
                }

                error = network.getRMS(actual.size());
                errgraph.update(error);
                TraningStatusArea.append("\nEpoch " + (ctr + 1) + ", Error: " + error + "");
                TraningStatusArea.setCaretPosition(TraningStatusArea.getText().length() - 1);

                if (error < variable.getMaxError()) {
                  break;
                }

                if(threadStop == true){
                  success = false;
                  stop = true;
                  break;
                }
              }

              if(stop == true){
                break;
              }
            }
          }
          TraningStatusArea.append("\n...End of training\n");
          TraningStatusArea.setCaretPosition(TraningStatusArea.getText().length() - 1);
          if(success == true){
            dbox.createDialogBox("Training successful!", "");
            variable.setNoOfHiddenNeurons(network.getHiddenNeurons());
          }
        } else {
          dbox.createDialogBox("Normalized data is needed for training!", "");
        }
        TrainButton.setEnabled(true);
        PauseButton.setEnabled(false);
        StopButton.setEnabled(false);
      }
    }
}
```

ProjectFile.java

```
package PredicTB;
```

```
import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.ArrayList;
import java.util.StringTokenizer;

public class ProjectFile {

   private boolean continueTraining = false;
   private int outputCol;

   private int epochs = 1000;
   private double maxError = 0.01;
   private double momentum = 0.7;
   private double learningRate = 0.5;
   private String activationFunction = "sig";
   private int noOfHiddenNeurons = 0;

   private String[] headers;
   private String[] nominalHeaders;
   private double[] maximum;
   private double[] minimum;

   private ArrayList<ArrayList<String>> origData = new ArrayList<ArrayList<String>>();
   private ArrayList<ArrayList<Double>> mappedData = new ArrayList<ArrayList<Double>>();
   private ArrayList<ArrayList<Double>> normalizedData = new ArrayList<ArrayList<Double>>();
   private ArrayList<ArrayList<String>> origEval = new ArrayList<ArrayList<String>>();
   private ArrayList<ArrayList<Double>> mappedEval = new ArrayList<ArrayList<Double>>();
   private ArrayList<ArrayList<Double>> normalizedEval = new ArrayList<ArrayList<Double>>();

   private double[] weight;
   private double[] weightDelta;
   private double[] weightDeltaAcc;
   private double[] threshold;
   private double[] thresholdDelta;
   private double[] thresholdDeltaAcc;

   private File projectFile;

   public ProjectFile(File projectFile){
      this.projectFile = projectFile;
   }

   public void readFile(){
       try {

         // BufferedReader d = new BufferedReader(new InputStreamReader(in));

          DataInputStream dis = new DataInputStream(new FileInputStream(projectFile));

          String x = null;
          x = dis.readUTF();
          this.continueTraining = Boolean.parseBoolean(x);
          x = dis.readUTF();
```

```
        this.outputCol = Integer.parseInt(x);
        x = dis.readUTF();
        this.activationFunction = x;
        x = dis.readUTF();
        this.epochs = Integer.parseInt(x);
        x = dis.readUTF();
        this.maxError = Double.parseDouble(x);
        x = dis.readUTF();
        this.momentum = Double.parseDouble(x);
        x = dis.readUTF();
        this.learningRate = Double.parseDouble(x);
        x = dis.readUTF();
        this.noOfHiddenNeurons = Integer.parseInt(x);
        x = dis.readUTF();
        this.headers = parseString(x);
        x = dis.readUTF();
        this.nominalHeaders = parseString(x);
        x = dis.readUTF();
        this.maximum = parseDouble(x);
        x = dis.readUTF();
        this.minimum = parseDouble(x);
        x = dis.readUTF();
        this.weight = parseDouble(x);
        x = dis.readUTF();
        this.weightDelta = parseDouble(x);
        x = dis.readUTF();
        this.weightDeltaAcc = parseDouble(x);
        x = dis.readUTF();
        this.threshold = parseDouble(x);
        x = dis.readUTF();
        this.thresholdDelta = parseDouble(x);
        x = dis.readUTF();
        this.thresholdDeltaAcc = parseDouble(x);
        x = dis.readUTF();
        this.origData = parseString1(x);
        x = dis.readUTF();
        this.mappedData = parseDouble1(x);
        x = dis.readUTF();
        this.normalizedData = parseDouble1(x);
        x = dis.readUTF();
        /*
        this.origEval = parseString1(x);
        x = dis.readUTF();
        this.mappedEval = parseDouble1(x);
        x = dis.readUTF();
        this.normalizedEval = parseDouble1(x);
        */
      } catch (Exception ex) {

      }
    }

    public void writeFile(){
      try {

        DataOutputStream dos = new DataOutputStream(new FileOutputStream(projectFile));

        String str = null;
```

```
            dos.writeUTF("" + continueTraining);
            dos.writeUTF("" + outputCol);
            dos.writeUTF("" + activationFunction);
            dos.writeUTF("" + epochs);
            dos.writeUTF("" + maxError);
            dos.writeUTF("" + momentum);
            dos.writeUTF("" + learningRate);
            dos.writeUTF("" + noOfHiddenNeurons);

            str = returnString(headers);
            dos.writeUTF(str);
            str = returnString(nominalHeaders);
            dos.writeUTF(str);
            str = returnDouble(maximum);
            dos.writeUTF(str);
            str = returnDouble(minimum);
            dos.writeUTF(str);
            str = returnDouble(weight);
            dos.writeUTF(str);
            str = returnDouble(weightDelta);
            dos.writeUTF(str);
            str = returnDouble(weightDeltaAcc);
            dos.writeUTF(str);
            str = returnDouble(threshold);
            dos.writeUTF(str);
            str = returnDouble(thresholdDelta);
            dos.writeUTF(str);
            str = returnDouble(thresholdDeltaAcc);
            dos.writeUTF(str);
            str = returnString(origData);
            dos.writeUTF(str);
            str = returnDouble(mappedData);
            dos.writeUTF(str);
            str = returnDouble(normalizedData);
            dos.writeUTF(str);
            /*
            str = returnString(origEval);
            dos.writeUTF(str);
            str = returnDouble(mappedEval);
            dos.writeUTF(str);
            str = returnDouble(normalizedEval);
            dos.writeUTF(str);
            */
            dos.close();

        } catch (Exception ex) {
            System.out.println(ex);
        }
    }

    protected double[] parseDouble(String data){
        double[] temp;

        if(!data.isEmpty()){
            StringTokenizer st = new StringTokenizer(data, ",");
            int i = 0;
            temp = new double[st.countTokens()];
```

```
      while(st.hasMoreTokens()){
        temp[i++] = Double.parseDouble(st.nextToken());
      }
    }else{
      temp = null;
    }

    return temp;
  }

  protected String[] parseString(String data){
    String[] temp;

    if(!data.isEmpty()){
      StringTokenizer st = new StringTokenizer(data, ",");
      int i = 0;
      temp = new String[st.countTokens()];
      while(st.hasMoreTokens()){
        temp[i++] = st.nextToken();
      }
    }else{
      temp = null;
    }

    return temp;
  }

  protected ArrayList<ArrayList<Double>> parseDouble1(String data){
    ArrayList<ArrayList<Double>> temp = new ArrayList<ArrayList<Double>>();

    if (!data.isEmpty()) {
      StringTokenizer st = new StringTokenizer(data, ";");
      while (st.hasMoreTokens()) {
        ArrayList<Double> temp1 = new ArrayList<Double>();
        StringTokenizer st1 = new StringTokenizer(st.nextToken(), ",");
        while (st1.hasMoreTokens()) {
          temp1.add(Double.parseDouble(st1.nextToken()));
        }
        temp.add(temp1);
      }
    } else {
      temp = null;
    }

    return temp;
  }

  protected ArrayList<ArrayList<String>> parseString1(String data){
    ArrayList<ArrayList<String>> temp = new ArrayList<ArrayList<String>>();

    if(!data.isEmpty()){
      StringTokenizer st = new StringTokenizer(data, ";");
      while (st.hasMoreTokens()) {
        ArrayList<String> temp1 = new ArrayList<String>();
        StringTokenizer st1 = new StringTokenizer(st.nextToken(), ",");
        while (st1.hasMoreTokens()) {
          temp1.add(st1.nextToken());
        }
```

```
            temp.add(temp1);
        }
    } else {
        temp = null;
    }

    return temp;
}


protected String returnDouble(double[] data){
    String str = "";
        if(data != null){
            for (int i = 0; i < data.length; i++) {
                str += Double.toString(data[i]);
                if (i != data.length - 1) {
                    str += ",";
                }
            }
        }
    return str;
}

protected String returnString(String[] data){
    String str = "";
        if(data != null){
            for (int i = 0; i < data.length; i++) {
                str += data[i];
                if (i != data.length - 1) {
                    str += ",";
                }
            }
        }
    return str;
}

protected String returnDouble(ArrayList<ArrayList<Double>> data){
    String str = "";
    if(data != null){
            for (int i = 0; i < data.size(); i++) {
                for(int j = 0; j < data.get(i).size(); j++){
                    str += Double.toString(data.get(i).get(j));
                    if (j != data.get(i).size() - 1) {
                      str += ",";
                    }
                }
                if (i != origData.size() - 1) {
                     str += ";";
                }
            }
        }
    return str;
}

private String returnString(ArrayList<ArrayList<String>> data){
    String str = "";
    if(data != null){
            for (int i = 0; i < data.size(); i++) {
```

```
            for(int j = 0; j < data.get(i).size(); j++){
               str += data.get(i).get(j);
               if (j != data.get(i).size() - 1) {
                 str += ",";
               }
            }
            if (i != origData.size() - 1) {
                str += ";";
            }
         }
      }
   return str;
}

/**
 * @return the continueTraining
 */
public boolean isContinueTraining() {
   return continueTraining;
}

/**
 * @return the epochs
 */
public int getEpochs() {
   return epochs;
}

/**
 * @return the maxError
 */
public double getMaxError() {
   return maxError;
}

/**
 * @return the momentum
 */
public double getMomentum() {
   return momentum;
}

/**
 * @return the learningRate
 */
public double getLearningRate() {
   return learningRate;
}

/**
 * @return the activationFunction
 */
public String getActivationFunction() {
   return activationFunction;
}

/**
 * @return the noOfHiddenNeurons
```

```
 */
public int getNoOfHiddenNeurons() {
   return noOfHiddenNeurons;
}

/**
 * @return the headers
 */
public String[] getHeaders() {
   return headers;
}

/**
 * @return the nominalHeaders
 */
public String[] getNominalHeaders() {
   return nominalHeaders;
}

/**
 * @return the maximum
 */
public double[] getMaximum() {
   return maximum;
}

/**
 * @return the minimum
 */
public double[] getMinimum() {
   return minimum;
}

/**
 * @return the origData
 */
public ArrayList<ArrayList<String>> getOrigData() {
   return origData;
}

/**
 * @return the mappedData
 */
public ArrayList<ArrayList<Double>> getMappedData() {
   return mappedData;
}

/**
 * @return the normalizedData
 */
public ArrayList<ArrayList<Double>> getNormalizedData() {
   return normalizedData;
}

/**
 * @return the origEval
 */
public ArrayList<ArrayList<String>> getOrigEval() {
```

```
    return origEval;
}

/**
 * @return the mappedEval
 */
public ArrayList<ArrayList<Double>> getMappedEval() {
    return mappedEval;
}

/**
 * @return the normalizedEval
 */
public ArrayList<ArrayList<Double>> getNormalizedEval() {
    return normalizedEval;
}

/**
 * @return the weight
 */
public double[] getWeight() {
    return weight;
}

/**
 * @return the weightDelta
 */
public double[] getWeightDelta() {
    return weightDelta;
}

/**
 * @return the weightDeltaAcc
 */
public double[] getWeightDeltaAcc() {
    return weightDeltaAcc;
}

/**
 * @return the threshold
 */
public double[] getThreshold() {
    return threshold;
}

/**
 * @return the thresholdDelta
 */
public double[] getThresholdDelta() {
    return thresholdDelta;
}

/**
 * @return the thresholdDeltaAcc
 */
public double[] getThresholdDeltaAcc() {
    return thresholdDeltaAcc;
}
```

```
/**
 * @return the projectFile
 */
public File getProjectFile() {
   return projectFile;
}

/**
 * @param continueTraining the continueTraining to set
 */
public void setContinueTraining(boolean continueTraining) {
   this.continueTraining = continueTraining;
}

/**
 * @param epochs the epochs to set
 */
public void setEpochs(int epochs) {
   this.epochs = epochs;
}

/**
 * @param maxError the maxError to set
 */
public void setMaxError(double maxError) {
   this.maxError = maxError;
}

/**
 * @param momentum the momentum to set
 */
public void setMomentum(double momentum) {
   this.momentum = momentum;
}

/**
 * @param learningRate the learningRate to set
 */
public void setLearningRate(double learningRate) {
   this.learningRate = learningRate;
}

/**
 * @param activationFunction the activationFunction to set
 */
public void setActivationFunction(String activationFunction) {
   this.activationFunction = activationFunction;
}

/**
 * @param noOfHiddenNeurons the noOfHiddenNeurons to set
 */
public void setNoOfHiddenNeurons(int noOfHiddenNeurons) {
   this.noOfHiddenNeurons = noOfHiddenNeurons;
}

/**
```

```
 * @param headers the headers to set
 */
public void setHeaders(String[] headers) {
   this.headers = headers;
}

/**
 * @param nominalHeaders the nominalHeaders to set
 */
public void setNominalHeaders(String[] nominalHeaders) {
   this.nominalHeaders = nominalHeaders;
}

/**
 * @param maximum the maximum to set
 */
public void setMaximum(double[] maximum) {
   this.maximum = maximum;
}

/**
 * @param minimum the minimum to set
 */
public void setMinimum(double[] minimum) {
   this.minimum = minimum;
}

/**
 * @param origData the origData to set
 */
public void setOrigData(ArrayList<ArrayList<String>> origData) {
   this.origData = origData;
}

/**
 * @param mappedData the mappedData to set
 */
public void setMappedData(ArrayList<ArrayList<Double>> mappedData) {
   this.mappedData = mappedData;
}

/**
 * @param normalizedData the normalizedData to set
 */
public void setNormalizedData(ArrayList<ArrayList<Double>> normalizedData) {
   this.normalizedData = normalizedData;
}

/**
 * @param origEval the origEval to set
 */
public void setOrigEval(ArrayList<ArrayList<String>> origEval) {
   this.origEval = origEval;
}

/**
 * @param mappedEval the mappedEval to set
 */
```

```
public void setMappedEval(ArrayList<ArrayList<Double>> mappedEval) {
   this.mappedEval = mappedEval;
}

/**
 * @param normalizedEval the normalizedEval to set
 */
public void setNormalizedEval(ArrayList<ArrayList<Double>> normalizedEval) {
   this.normalizedEval = normalizedEval;
}

/**
 * @param weight the weight to set
 */
public void setWeight(double[] weight) {
   this.weight = weight;
}

/**
 * @param weightDelta the weightDelta to set
 */
public void setWeightDelta(double[] weightDelta) {
   this.weightDelta = weightDelta;
}

/**
 * @param weightDeltaAcc the weightDeltaAcc to set
 */
public void setWeightDeltaAcc(double[] weightDeltaAcc) {
   this.weightDeltaAcc = weightDeltaAcc;
}

/**
 * @param threshold the threshold to set
 */
public void setThreshold(double[] threshold) {
   this.threshold = threshold;
}

/**
 * @param thresholdDelta the thresholdDelta to set
 */
public void setThresholdDelta(double[] thresholdDelta) {
   this.thresholdDelta = thresholdDelta;
}

/**
 * @param thresholdDeltaAcc the thresholdDeltaAcc to set
 */
public void setThresholdDeltaAcc(double[] thresholdDeltaAcc) {
   this.thresholdDeltaAcc = thresholdDeltaAcc;
}

/**
 * @param projectFile the projectFile to set
 */
public void setProjectFile(File projectFile) {
   this.projectFile = projectFile;
```

```
    }

    /**
     * @return the outputCol
     */
    public int getOutputCol() {
        return outputCol;
    }

    /**
     * @param outputCol the outputCol to set
     */
    public void setOutputCol(int outputCol) {
        this.outputCol = outputCol;
    }
}
```

Pruning.java

```
package PredicTB;

import java.util.ArrayList;
import javax.swing.JTextArea;

public class Pruning {
    private Network network;
    private ArrayList<Double> ideal;
    private ArrayList<ArrayList<Double>> actual;
    private String actFunction;
    private int epochs;
    private double learningRate;
    private double maxError;
    private double momentum;
    private int neurons;
    private int cycles;
    private boolean success;
    private double error;

    protected boolean threadStop = false;

    public Pruning(ArrayList<Double> ideal, ArrayList<ArrayList<Double>> actual,
            String actFunction, int epochs, double learningRate,
            double maxError, double momentum){
        this.ideal = ideal;
        this.actual = actual;
        this.actFunction = actFunction;
        this.epochs = epochs;
        this.learningRate = learningRate;
        this.maxError = maxError;
        this.momentum = momentum;
    }

    public void start(int hidden){
        neurons = hidden;
        success = false;
        error = 1;
        network = new Network(actual.get(0).size(),
                this.actFunction, this.learningRate,
```

```
        this.maxError, this.momentum,
        this.neurons);
}

public void prune(JTextArea TraningStatusArea, ErrorGraph errgraph){
   TraningStatusArea.append("\nHidden Neurons: " + neurons);
   TraningStatusArea.setCaretPosition(TraningStatusArea.getText().length() - 1);
   if(success == true){
      return;
   }else{
      double lastError = 1;
      boolean getOut = false;
      while (error > maxError) {
         for (int i = 0; i < actual.size(); i++) {
            network.propagate(actual.get(i));
            network.backpropagate(ideal.get(i));
            network.train();
         }

         error = network.getRMS(actual.size());
         errgraph.update(error);
         TraningStatusArea.append("\nEpochs:" + cycles + ", Error=" + error);
         TraningStatusArea.setCaretPosition(TraningStatusArea.getText().length() - 1);

         if(threadStop == true){
            return;
         }

         if((cycles % epochs) == 0){
            if((lastError-error) < 0.01){ //stagnant na, no progress
               getOut = true;
               momentum -= 0.01;
               learningRate -= 0.01;
            }else{
               lastError = error;
            }
         }

         if(getOut == true){
            break;
         }
         cycles++;
      }
      if(getOut == false){
         success = true;
      }
   }
}

public void increment(){
   boolean stopTrain = false;
}

/**
 * @return the network
 */
public Network getNetwork() {
   return network;
```

```
}

/**
 * @param network the network to set
 */
public void setNetwork(Network network) {
   this.network = network;
}

/**
 * @return the ideal
 */
public ArrayList<Double> getIdeal() {
   return ideal;
}

/**
 * @param ideal the ideal to set
 */
public void setIdeal(ArrayList<Double> ideal) {
   this.ideal = ideal;
}

/**
 * @return the actual
 */
public ArrayList<ArrayList<Double>> getActual() {
   return actual;
}

/**
 * @param actual the actual to set
 */
public void setActual(ArrayList<ArrayList<Double>> actual) {
   this.actual = actual;
}

/**
 * @return the actFunction
 */
public String getActFunction() {
   return actFunction;
}

/**
 * @param actFunction the actFunction to set
 */
public void setActFunction(String actFunction) {
   this.actFunction = actFunction;
}

/**
 * @return the epochs
 */
public int getEpochs() {
   return epochs;
}
```

```
/**
 * @param epochs the epochs to set
 */
public void setEpochs(int epochs) {
   this.epochs = epochs;
}

/**
 * @return the learningRate
 */
public double getLearningRate() {
   return learningRate;
}

/**
 * @param learningRate the learningRate to set
 */
public void setLearningRate(double learningRate) {
   this.learningRate = learningRate;
}

/**
 * @return the maxError
 */
public double getMaxError() {
   return maxError;
}

/**
 * @param maxError the maxError to set
 */
public void setMaxError(double maxError) {
   this.maxError = maxError;
}

/**
 * @return the momentum
 */
public double getMomentum() {
   return momentum;
}

/**
 * @param momentum the momentum to set
 */
public void setMomentum(double momentum) {
   this.momentum = momentum;
}

/**
 * @return the neurons
 */
public int getNeurons() {
   return neurons;
}

/**
 * @param neurons the neurons to set
```

```
    */
    public void setNeurons(int neurons) {
        this.neurons = neurons;
    }

    /**
     * @return the cycles
     */
    public int getCycles() {
        return cycles;
    }

    /**
     * @param cycles the cycles to set
     */
    public void setCycles(int cycles) {
        this.cycles = cycles;
    }

    /**
     * @return the success
     */
    public boolean isSuccess() {
        return success;
    }

    /**
     * @param success the success to set
     */
    public void setSuccess(boolean success) {
        this.success = success;
    }

    /**
     * @return the error
     */
    public double getError() {
        return error;
    }

    /**
     * @param error the error to set
     */
    public void setError(double error) {
        this.error = error;
    }
}
```

VariableForm.java

```
package PredicTB;

public class VariableForm extends javax.swing.JDialog {

    Variables variable;
    DialogBox dbox;

    /** Creates new form VariableForm */
```

```
    public VariableForm(java.awt.Frame parent, Variables var) {
        this.variable = var;
        initComponents();
        LearningTextField.setText("0.5");
        MomentumTextField.setText("0.7");
        ErrorTextField.setText("0.01");
        EpochTextField.setText("1000");
        HiddenTextField.setText("1");
    }

    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        jPanel1 = new javax.swing.JPanel();
        ActivationLabel = new javax.swing.JLabel();
        ActivationComboBox = new javax.swing.JComboBox();
        LearningLabel = new javax.swing.JLabel();
        LearningComboBox = new javax.swing.JComboBox();
        MomentumLabel = new javax.swing.JLabel();
        MomentumComboBox = new javax.swing.JComboBox();
        LearningTextField = new javax.swing.JTextField();
        MomentumTextField = new javax.swing.JTextField();
        ErrorLabel = new javax.swing.JLabel();
        EpochLabel = new javax.swing.JLabel();
        HiddenLabel = new javax.swing.JLabel();
        HiddenComboBox = new javax.swing.JComboBox();
        EpochComboBox = new javax.swing.JComboBox();
        ErrorComboBox = new javax.swing.JComboBox();
        ErrorTextField = new javax.swing.JTextField();
        EpochTextField = new javax.swing.JTextField();
        HiddenTextField = new javax.swing.JTextField();
        SubmitButton = new javax.swing.JButton();

        setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
        org.jdesktop.application.ResourceMap resourceMap =
org.jdesktop.application.Application.getInstance(PredicTB.PredicTBApp.class).getContext().getResourceMap(VariableForm.class);
        setTitle(resourceMap.getString("Form.title"));
        setAlwaysOnTop(true);
        setName("Form");

        jPanel1.setBorder(javax.swing.BorderFactory.createEtchedBorder());
        jPanel1.setName("jPanel1");

        ActivationLabel.setText(resourceMap.getString("ActivationLabel.text"));
        ActivationLabel.setName("ActivationLabel");

        ActivationComboBox.setModel(new javax.swing.DefaultComboBoxModel(new String[] { "Sigmoid", "Hyperbolic Tangent" }));
        ActivationComboBox.setName("ActivationComboBox");

        LearningLabel.setText(resourceMap.getString("LearningLabel.text"));
        LearningLabel.setName("LearningLabel");

        LearningComboBox.setModel(new javax.swing.DefaultComboBoxModel(new String[] { "Use Default Value", "Input Learning
Rate" }));
        LearningComboBox.setName("LearningComboBox");
        LearningComboBox.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
```

```
            LearningComboBoxActionPerformed(evt);
        }
    });

    MomentumLabel.setText(resourceMap.getString("MomentumLabel.text"));
    MomentumLabel.setName("MomentumLabel");

    MomentumComboBox.setModel(new javax.swing.DefaultComboBoxModel(new String[] { "Use Default Value", "Input
Momentum" }));
    MomentumComboBox.setName("MomentumComboBox");
    MomentumComboBox.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            MomentumComboBoxActionPerformed(evt);
        }
    });

    LearningTextField.setEnabled(false);
    LearningTextField.setName("LearningTextField");

    MomentumTextField.setEnabled(false);
    MomentumTextField.setName("MomentumTextField");

    ErrorLabel.setText(resourceMap.getString("ErrorLabel.text"));
    ErrorLabel.setName("ErrorLabel");

    EpochLabel.setText(resourceMap.getString("EpochLabel.text"));
    EpochLabel.setName("EpochLabel");

    HiddenLabel.setText(resourceMap.getString("HiddenLabel.text"));
    HiddenLabel.setName("HiddenLabel");

    HiddenComboBox.setModel(new javax.swing.DefaultComboBoxModel(new String[] { "Use Incremental Pruning", "Input
Hidden Neurons" }));
    HiddenComboBox.setName("HiddenComboBox");
    HiddenComboBox.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            HiddenComboBoxActionPerformed(evt);
        }
    });

    EpochComboBox.setModel(new javax.swing.DefaultComboBoxModel(new String[] { "Use Default Value", "Input Number of
Epochs" }));
    EpochComboBox.setName("EpochComboBox");
    EpochComboBox.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            EpochComboBoxActionPerformed(evt);
        }
    });

    ErrorComboBox.setModel(new javax.swing.DefaultComboBoxModel(new String[] { "Use Default Value", "Input Maximum
Error" }));
    ErrorComboBox.setName("ErrorComboBox");
    ErrorComboBox.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            ErrorComboBoxActionPerformed(evt);
        }
    });
```

```
    ErrorTextField.setText(resourceMap.getString("ErrorTextField.text"));
    ErrorTextField.setEnabled(false);
    ErrorTextField.setName("ErrorTextField");

    EpochTextField.setText(resourceMap.getString("EpochTextField.text"));
    EpochTextField.setEnabled(false);
    EpochTextField.setName("EpochTextField");

    HiddenTextField.setText(resourceMap.getString("HiddenTextField.text"));
    HiddenTextField.setEnabled(false);
    HiddenTextField.setName("HiddenTextField");

    SubmitButton.setText(resourceMap.getString("SubmitButton.text"));
    SubmitButton.setName("SubmitButton");
    SubmitButton.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
        SubmitButtonActionPerformed(evt);
      }
    });

    javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);
    jPanel1.setLayout(jPanel1Layout);
    jPanel1Layout.setHorizontalGroup(
      jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGroup(jPanel1Layout.createSequentialGroup()
        .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
          .addGroup(jPanel1Layout.createSequentialGroup()
            .addContainerGap()
            .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
              .addComponent(HiddenLabel)
              .addComponent(EpochLabel)
              .addComponent(ErrorLabel)
              .addComponent(LearningLabel)
              .addComponent(MomentumLabel)
              .addComponent(ActivationLabel))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)
              .addComponent(HiddenComboBox, 0, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
              .addComponent(EpochComboBox, 0, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
              .addComponent(ErrorComboBox, 0, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
              .addComponent(MomentumComboBox, 0, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
              .addComponent(LearningComboBox, 0, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
              .addComponent(ActivationComboBox, 0, 177, Short.MAX_VALUE))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
              .addComponent(MomentumTextField, javax.swing.GroupLayout.Alignment.TRAILING,
javax.swing.GroupLayout.DEFAULT_SIZE, 128, Short.MAX_VALUE)
              .addComponent(LearningTextField, javax.swing.GroupLayout.Alignment.TRAILING,
javax.swing.GroupLayout.DEFAULT_SIZE, 128, Short.MAX_VALUE)
              .addComponent(ErrorTextField, javax.swing.GroupLayout.DEFAULT_SIZE, 128, Short.MAX_VALUE)
              .addComponent(EpochTextField, javax.swing.GroupLayout.DEFAULT_SIZE, 128, Short.MAX_VALUE)
              .addComponent(HiddenTextField, javax.swing.GroupLayout.DEFAULT_SIZE, 128, Short.MAX_VALUE)))
          .addGroup(jPanel1Layout.createSequentialGroup()
            .addGap(185, 185, 185)
            .addComponent(SubmitButton)))
        .addContainerGap())
    );
    jPanel1Layout.setVerticalGroup(
```

```
        jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(jPanel1Layout.createSequentialGroup()
            .addContainerGap()
            .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                .addComponent(ActivationComboBox, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addComponent(ActivationLabel))
            .addGap(18, 18, 18)
            .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                .addComponent(LearningLabel)
                .addComponent(LearningComboBox, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addComponent(LearningTextField, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
            .addGap(18, 18, 18)
            .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                .addComponent(MomentumLabel)
                .addComponent(MomentumTextField, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addComponent(MomentumComboBox, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
            .addGap(18, 18, 18)
            .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                .addComponent(ErrorLabel)
                .addComponent(ErrorComboBox, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addComponent(ErrorTextField, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
            .addGap(18, 18, 18)
            .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                .addComponent(EpochLabel)
                .addComponent(EpochComboBox, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addComponent(EpochTextField, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
            .addGap(18, 18, 18)
            .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                .addComponent(HiddenLabel)
                .addComponent(HiddenComboBox, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addComponent(HiddenTextField, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
            .addGap(18, 18, 18)
            .addComponent(SubmitButton)
            .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
    );

    javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
    getContentPane().setLayout(layout);
    layout.setHorizontalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(layout.createSequentialGroup()
            .addContainerGap()
            .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
    );
    layout.setVerticalGroup(
```

```
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(layout.createSequentialGroup()
            .addContainerGap()
            .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
      );

      pack();
    }// </editor-fold>

    private void LearningComboBoxActionPerformed(java.awt.event.ActionEvent evt) {
      if (LearningComboBox.getSelectedIndex() == 0) {
        LearningTextField.setEnabled(false);
        LearningTextField.setText("0.5");
      } else {
        LearningTextField.setEnabled(true);
      }
    }

    private void MomentumComboBoxActionPerformed(java.awt.event.ActionEvent evt) {
      if (MomentumComboBox.getSelectedIndex() == 0) {
        MomentumTextField.setEnabled(false);
        MomentumTextField.setText("0.7");
      } else {
        MomentumTextField.setEnabled(true);
      }
    }

    private void ErrorComboBoxActionPerformed(java.awt.event.ActionEvent evt) {
      if (ErrorComboBox.getSelectedIndex() == 0) {
        ErrorTextField.setEnabled(false);
        ErrorTextField.setText("0.01");
      } else {
        ErrorTextField.setEnabled(true);
      }
    }

    private void EpochComboBoxActionPerformed(java.awt.event.ActionEvent evt) {
      if (EpochComboBox.getSelectedIndex() == 0) {
        EpochTextField.setEnabled(false);
        EpochTextField.setText("1000");
      } else {
        EpochTextField.setEnabled(true);
      }
    }

    private void HiddenComboBoxActionPerformed(java.awt.event.ActionEvent evt) {
      if (HiddenComboBox.getSelectedIndex() == 0) {
        HiddenTextField.setEnabled(false);
      } else {
        HiddenTextField.setEnabled(true);
      }
    }

    private void SubmitButtonActionPerformed(java.awt.event.ActionEvent evt) {
      dbox = new DialogBox(jPanel1);
      boolean exCatched = false;
```

```
        if (ActivationComboBox.getSelectedIndex() == 0) {
            variable.setActivationFunction("sig");
        } else {
            variable.setActivationFunction("tanh");
        }

        try {
            if (MomentumTextField.isEnabled() == true) {
                variable.setMomentum(Double.parseDouble(MomentumTextField.getText().toString()));
            }
            if (LearningTextField.isEnabled() == true) {
                variable.setLearningRate(Double.parseDouble(LearningTextField.getText().toString()));
            }
            if (EpochTextField.isEnabled() == true) {
                variable.setEpochs(Integer.parseInt(EpochTextField.getText().toString()));
            }
            if (ErrorTextField.isEnabled() == true) {
                variable.setMaxError(Double.parseDouble(ErrorTextField.getText().toString()));
            }
            if (HiddenTextField.isEnabled() == true) {
                variable.setNoOfHiddenNeurons(Integer.parseInt(HiddenTextField.getText().toString()));
            }
        } catch (NumberFormatException ex) {
            dbox.createDialogBox("Incorrect input!", "");
            exCatched = true;
        }

        if(exCatched == false){
            this.setVisible(false);
        }
    }
    // Variables declaration - do not modify
    protected javax.swing.JComboBox ActivationComboBox;
    private javax.swing.JLabel ActivationLabel;
    protected javax.swing.JComboBox EpochComboBox;
    private javax.swing.JLabel EpochLabel;
    protected javax.swing.JTextField EpochTextField;
    protected javax.swing.JComboBox ErrorComboBox;
    private javax.swing.JLabel ErrorLabel;
    protected javax.swing.JTextField ErrorTextField;
    protected javax.swing.JComboBox HiddenComboBox;
    private javax.swing.JLabel HiddenLabel;
    protected javax.swing.JTextField HiddenTextField;
    protected javax.swing.JComboBox LearningComboBox;
    private javax.swing.JLabel LearningLabel;
    protected javax.swing.JTextField LearningTextField;
    protected javax.swing.JComboBox MomentumComboBox;
    private javax.swing.JLabel MomentumLabel;
    protected javax.swing.JTextField MomentumTextField;
    private javax.swing.JButton SubmitButton;
    private javax.swing.JPanel jPanel1;
    // End of variables declaration
}
```

Variables.java

```
package PredicTB;
```

```
public class Variables {

   private int epochs = 1000;
   private double maxError = 0.1;
   private double momentum = 0.7;
   private double learningRate = 0.5;
   private String activationFunction = "sig";
   private int noOfHiddenNeurons = 1;

   /**
    * @return the epochs
    */
   public int getEpochs() {
      return epochs;
   }

   /**
    * @param epochs the epochs to set
    */
   public void setEpochs(int epochs) {
      this.epochs = epochs;
   }

   /**
    * @return the maxError
    */
   public double getMaxError() {
      return maxError;
   }

   /**
    * @param maxError the maxError to set
    */
   public void setMaxError(double maxError) {
      this.maxError = maxError;
   }

   /**
    * @return the momentum
    */
   public double getMomentum() {
      return momentum;
   }

   /**
    * @param momentum the momentum to set
    */
   public void setMomentum(double momentum) {
      this.momentum = momentum;
   }

   /**
    * @return the learningRate
    */
   public double getLearningRate() {
      return learningRate;
   }
```

```
    /**
     * @param learningRate the learningRate to set
     */
    public void setLearningRate(double learningRate) {
        this.learningRate = learningRate;
    }

    /**
     * @return the activationFunction
     */
    public String getActivationFunction() {
        return activationFunction;
    }

    /**
     * @param activationFunction the activationFunction to set
     */
    public void setActivationFunction(String activationFunction) {
        this.activationFunction = activationFunction;
    }

    /**
     * @return the noOfHiddenNeurons
     */
    public int getNoOfHiddenNeurons() {
        return noOfHiddenNeurons;
    }

    /**
     * @param noOfHiddenNeurons the noOfHiddenNeurons to set
     */
    public void setNoOfHiddenNeurons(int noOfHiddenNeurons) {
        this.noOfHiddenNeurons = noOfHiddenNeurons;
    }
}
```

XI. ACKNOWLEDGMENT

First of all, I would like to thank Ms. Avegail D. Carpio, M.S. for her patience and steadfast encouragement as my dissertation adviser. Thank you for the input and wise words that helped me complete this special project and sorry for being such a lousy advisee.

I am also grateful to all the people who helped me gather the data set that I needed for testing the system: Dr. Ester C. Santos-Bitanga, Deputy Director for Health Operations of the Philippine General Hospital; Dr. Tammy Dela Rosa, Coordinator for Research, and the employees of the Expanded Hospital Research Office of UP PGH; and Ms. Grace Rondilla, Head Nurse of the UP Prime TB DOTS Clinic.

I would also like to express my gratitude to all my professors for giving me a hard time. For giving extremely hard exams, unjust deadlines, complex machine problems, and challenging projects. I surely appreciate everyone even the professor who gave me a taste of my one and only tres. The fact that they managed to give time and effort for teaching a half-assed student like me is surely something to be thanked for. A special mention should be given to Mong for being the best profossor ever. Fun and learning does not usually go to with the same sentence but you always manages to combine it, plus, it is never easy to find a teacher that can hang out with his students even outside the campus.

I am also very thankful to my friends for just being themselves. They were there when I needed them and even when I didn't. They pretended to listen even though they're not interested, made time to chat even in our busiest days. They cheered me up and made me laugh when I'm awfully stressed from hell a.k.a academics. It's been a great pleasure to have people like them.

To Siramix for three years of fun and enjoyment. They adopted me when I was facing my socially awkward and lonely college days. Kordie, for all the small talks and being the gossip capital of the group. I know we've had our little misunderstanding back then but we both sorted it

out. I will surely remember all the 'trips' we've done together. Jallen, for being the serious adviser and mini-google of the group. Thanks for the weird infos, for your grammar nazi-ness and for the intellectual point of view on certain things. Ruth, for being the group clown, center of attention and the original PI. Sometimes, thinking about how fucked up your life is cheers everyone up. Nikko, for being the semi-sponge. You are a perfect complement to a group - keeping yourself quiet when others are talking and butting into conversations when everyone has nothing to say. Lastly, Naji, for being the bad influence and being my original college drinking and party buddy. Thanks to your fashion, lifestyle and entertainment insights. Also, thanks to your spontaneity, the group has experienced a lot of things together.

To Comsci Batch 2008. Mami JJ, Chester, Aura, Trixe, Mik, Nard, Pat, Renz, Renzy, Lalay, Bev, Dan, Tina, Ana, Eulah, Vienna, Chessell, JM, Migo, Neil, Fiona, Axel, Ian, Patrick and Louie. You guys are not only good classmates, but also very good friends. Thanks for all the lecture notes, sample exams, review sessions and encouraging words especially during SP days. I will surely remember all the tambay moments, block outings, pusoy and king-five-ten games and drinking sessions we've been through. I may be a batch ahead from you, but indeeed, you are my real blockmates.

Comsci Batch 07, I am also thankful for just having you guys around especially on the earlier part of my comsci days. We must agree that you may not be the most welcoming batch, but in way, you played a great part in my college life especially because of the following people: Eunice, Josha, Dan, Ish, Jlo, Carmel, Alvin, Aids, Marian, Ejae and Kawaii. Of course, I would like to give a special thanks to Isay for being Isay. Thank you for the sample codes, notes, tips, pressure, gc-ness, or whatever things you did to help me. We may not have a lot of things in common, but I consider you to be one of the best people I met throughout my college life.

To Uyayis, Orcom Block 8 Batch 2007, and the rest of my friends, thanks for all the experiences you've given me – random conversations, online chitchats, tambay moments and even brief hallway bumps and nods. College life is not fun without the people around you and lucky for me, I never run out of those people because of you.

Lastly, I would like to thank my family for giving me a huge headache. For always asking me how my studies go and when would I be able to propose and finish my thesis. These small acts pushed me to my limits. To ate Donna, for I know you had a lots of hard work to support my studies, always there to provide monetary support and whatever things I needed. To Buang, for allowing me to use your laptop and internet access. Munyi, for just being there whenever I wanted someone to talk to. To my nieces and nephews for being a very good stress relief. And of course, Mama for all the great love and support.

"Some people come into our lives and quickly go. Some stay for awhile and leave footprints on our hearts. And we are never, ever the same."