CLASS: a nondestructive flaw classification system

Loren G. Knutson

Iowa State University

Follow this and additional works at: http://lib.dr.iastate.edu/rtd

Part of the Electrical and Computer Engineering Commons

Recommended Citation

http://lib.dr.iastate.edu/rtd/235

This Thesis is brought to you for free and open access by Iowa State University Digital Repository. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.
CLASS: A nondestructive evaluation flaw classification system.

by

Loren Gene Knutson

A Thesis Submitted to the
Graduate Faculty in Partial Fulfillment of the
Requirement for the Degree of
MASTER OF SCIENCE

Department: Electrical and Computer Engineering
Major: Electrical Engineering

Approved:

Dr. Jennifer L. Davidson
For the Major Department

Dr. Lester W. Schmerr
Dr. John P. Basart
For the Graduate College

Iowa State University
Ames, Iowa
1995
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>NDE FLAW CLASSIFICATION</td>
<td>1</td>
</tr>
<tr>
<td>BACKGROUND</td>
<td>4</td>
</tr>
<tr>
<td>THE CLASS INTERFACE</td>
<td>7</td>
</tr>
<tr>
<td>CLASS - A SUMMARY</td>
<td>29</td>
</tr>
<tr>
<td>EXTENDING CLASS</td>
<td>31</td>
</tr>
<tr>
<td>CONCLUSIONS AND FUTURE WORK</td>
<td>33</td>
</tr>
<tr>
<td>BIBLIOGRAPHY</td>
<td>34</td>
</tr>
<tr>
<td>APPENDIX A - CLASS MANUAL</td>
<td>36</td>
</tr>
<tr>
<td>APPENDIX B - PROCEDURES FOR EXTENDING CLASS</td>
<td>80</td>
</tr>
<tr>
<td>APPENDIX C - LEARNING ALGORITHMS</td>
<td>82</td>
</tr>
<tr>
<td>APPENDIX D - ADDING A LEARNING ALGORITHM TO CLASS</td>
<td>93</td>
</tr>
</tbody>
</table>
ACKNOWLEDGMENTS

I would like to thank Dr. Lester W. Schmerr for his guidance, direction, and support of my research. I would also like to thank the Center for Nondestructive Evaluation as they provided me with a research assistantship for my Master's effort. Finally, I would like to thank my wife Brenda for being supportive through this time and helping me to stay focused on my objectives.
ABSTRACT

One important application of nondestructive evaluation (NDE) methods involves the classification of the types of defects present in materials based on the measured response of those defects to different types of interrogating energy. This classification process typically involves 1) the transformation of the "raw" NDE measurements into domains more suitable for classification purposes, 2) the extraction of features in those domains that are distinguishing characteristics of different flaw types, and 3) the application of a particular classification method for making a decision on the type of flaw present (based on the strength of the features).

Here, we describe the elements of a software system, CLASS, that performs the entire classification process for those NDE methods where the underlying NDE measurements are essentially one-dimensional (such as ultrasonics). This research effort is concerned with the design and implementation of code which comprises the CLASS software package. Some of the important features of CLASS are described, including its extensive use of a user-friendly graphical interface, its use of classification algorithms that are primarily data driven (i.e. they require little user expertise in building the classifiers) and the ability to easily extend CLASS by adding new features and classification methods.
NDE FLAW CLASSIFICATION

Nondestructive evaluation (NDE) methods place various forms of energy into materials and try to examine the materials without harming them or affecting their performance. Examples of NDE methods and the types of energy they use are: Ultrasound - acoustic energy, Eddy currents - electrical energy, and X-rays - penetrating radiation. One important application of these NDE methods is to find, classify and characterize (size) flaws based on their response to the types of energy present. Here we are interested only in the classification process, i.e. distinguishing the types of flaw present.

The steps involved in the NDE flaw classification process are shown in Figure 1. First, the raw signals need to be captured and stored. In ultrasonics, for example, this would be a voltage versus time signal (Figure 2). Such signals may be acceptable as is. However, in some cases, additional pre-processing steps may be needed (such as zeroing baseline noise or aligning signals, etc.) as shown in Figure 1. Second, features must be derived from those signals. In some cases those features can be obtained from the raw signals themselves. However, in many cases it is desirable to first transform the raw signals into other domains so that features more characteristic of the flaw type can be obtained. In ultrasonics, for example, it is often desirable to use features taken from the frequency spectrum (frequency domain) of the signals. Once the data are available in the desired transformed domain(s), then the features can be extracted. Finally, through the use of these features, the flaws are classified into types using a particular classification method. In general, there are more than two classes (flaw types) needed. In the ultrasonic inspection of welds, for example, we are often interested in distinguishing between cracks, porosity, slag inclusions, and other welding imperfections.
Figure 1. NDE data path flow chart for the classification process.

Figure 2. Ultrasonic signal example from the hard alpha inclusion problem.
The system, CLASS, that is described in this work, implements all the elements of Figure 1 in a software package that is both "user-friendly" and extensible. In the following BACKGROUND section we will outline the reasons for building CLASS and the choices made (for both hardware and software) in its implementation. In the section, CLASS INTERFACE we will explain the various parts of CLASS by demonstrating CLASS in action on a particular classification problem - the detection of hard alpha inclusions in titanium. In the section CLASS - A SUMMARY we will give a brief outline of the current contents of CLASS, its capabilities, and its limitations. In the section EXTENDING CLASS we will give a brief overview of how to add transforms, features and learning algorithms to CLASS. Finally, we will give a section outlining CONCLUSIONS AND FUTURE WORK. A users manual for CLASS is also included in an Appendix to give a more detailed description of how CLASS is structured and operates for an interested user.
BACKGROUND

In previous work at the Center for Nondestructive Evaluation (CNDE) sponsored by the NSF Industrial/University program a variety of techniques for flaw classification (mostly for ultrasonics) have been considered. Some examples of these techniques are:

- Expert Systems [20, 21]
- Neural Networks (Probabilistic, Back-Propagation, Hierarchical Nets) [2, 18]
- Statistical Methods (K-Nearest Neighbors) [2]
- Decision Trees [11, 12, 13, 14, 15]

CLASS originated out of a desire to collect some of those techniques into a package usable by workers in the NDE field who may not also be experts on building classifiers. Thus, we concentrated on placing in CLASS techniques that are primarily "data driven", i.e. the classification is built directly from the data with little or no user interaction. Also, we wanted to be able to easily add new features and new classification techniques to the package. No system on the market we are aware of has both of these capabilities and the same variety of classification methods (particularly decision trees and adaptive probabilistic neural nets) available. ICEPAK (by Tektrnd International Inc.), for example, is primarily a signal processing tool. It will pre-process, extraction feature, and classify signals just like CLASS but it does not appear to be easily expanded by the user and does not have the wide range of classifying algorithms like CLASS will have. IUNDE (by Information Research Laboratory Inc.) is another example of software on the market. It is designed to be used for pattern recognition of ultrasonic NDE signals. It provides signal pre-processing, feature extraction, and classification but it too appears to be limited in user expandability and classification routines. There are many examples of signal processing programs on the market today, but these two mentioned are ones that are being applied to NDE techniques. CLASS is developed
for NDE classification and should not be compared to applications meant strictly for signal processing.

An Intel-based personal computer was chosen as the platform for the package because it is the platform primarily used by the Centers sponsors for acquiring and analyzing their experimental data. In building a package like CLASS, one of the difficult decisions to make is in the choice of the language to use since efficiency in development is essential. Visual Basic was chosen in this case because CLASS is primarily a visual interface to data manipulation and learning algorithms and Visual Basic has many built-in interface-building functions that make it ideally suited to this application. In developing CLASS, the data path flow chart of Figure 1 directed the development of the interface. We knew we would probably need to design several demo interfaces before finding one that worked well for the targeted user. This meant using a programming language (environment) that allowed quick implementation of interfaces and gave flexibility so that when the final interface design was selected it could also be used for the actual implementation. This was the primary reason for choosing Visual Basic. Also, Visual Basic does have the ability to interface to programs written in other languages, like C, which may be more appropriate for constructing the features and classifiers themselves. Thus, we did not feel that the choice of Visual Basic was limiting in this aspect. We wanted to allow the user to expand CLASS in the easiest way possible for them. This meant not being restrictive on what language the user could use for expansion. Although most of CLASS will run under Windows 3.1, some learning algorithms need a 32-bit operating environment to work properly. Because of this, it is recommended to operate CLASS under Windows 3.1 (or later version) with the 32-bit extension software added or to use Windows NT 3.1 (or later version). The extension software can be obtained from Microsoft free of charge.
CLASS is primarily setup for analysis of one-dimensional ultrasonic data but could also be used with eddy current data (complex impedance versus location or frequency) with minor adjustments. It is not designed to deal with two-dimensional data in the form of images.
THE CLASS INTERFACE

CLASS is basically a highly visual interface that allows a user to easily "drive" through the classification process. Figure 3 is a screen capture of the CLASS interface window which shows that CLASS is structured primarily as a series of user selectable "folders". The best way to illustrate the content and use of these folders is with a specific example, so we will "walk through" the use of CLASS on a particular case called the hard alpha detection problem.

The Hard Alpha Problem

In 1987 a titanium disk in the engine of a DC-10 exploded, causing severe damage to the hydraulic system of the aircraft that resulted in its ultimate crash at Sioux City, Iowa. The cause of that disk failure was traced to a large undetected crack that originated from a small
brittle region of the titanium (called a hard alpha inclusion) that was present in the original
titanium billet from which the disk was made. Detecting such inclusions is made very difficult
by the fact that the inclusion itself does not differ substantially from the "good" titanium in the
disk. Thus high performance, adaptive classification methods are needed for this problem.
Note that in this case we have only two classes to distinguish, namely flawed (hard alpha
inclusion present) component and unflawed (hard alpha inclusion absent) component.

Dr. Chien-Ping Chiou [8, 9] at the Center for NDE developed a set of simulated
ultrasonic signals to model the hard alpha detection problem by superimposing the known
scattering response of simple weak scattering spheres (using the Born approximation) onto
the measured ultrasonic noise of titanium specimens. Dr. Chiou's data consisted of 100
signals where the flaw was absent and 15 signals with flaws, where both flaw signals of high
(>1) and low (<1) signal-to-noise ratios were simulated.

The features that Dr. Chiou chose to use for detecting the hard alpha inclusion were
statistical features derived directly from the voltage versus time signals so that no
transformations of the data were required in this example. The statistical features employed
were:

- mean
- absolute mean
- variance
- skewness
- kurtosis
- zero crossing

**Using CLASS on the Hard Alpha Problem**

Figure 4 shows CLASS when it is first brought up. CLASS forces the user to first
choose an appropriate data file. No folders or menu options are available until a data file is
chosen. The drive, directory and file boxes can be used to choose the data file. CLASS can be told what file to use by the user double clicking on the file name. The mouse can also be used to drag the file name over to the filename box. For this example, we will select the data file name that has the hard alpha inclusion data. Figure 3 shows CLASS after this data file has been chosen. Notice that CLASS specifically tells the user the location and file name that it currently is using (in this case the path is "c:\class\data" and the file name is "hrdalpha.mdb").

CLASS stores all information for a particular problem into a Microsoft Access standard format database file. What this means to the user is that only one file exists on the disk for each classification problem that the user is dealing with. For CLASS, it means a
performance boost for reading and writing data as well as easily keeping track of all information. How to store all the data CLASS creates was a substantial question from the start. There were really only two "good" alternatives that could be found to answer this question. The first alternative was to put all files into a highly ordered sub directory tree that would keep the multiple files created separated from one another. The files could be saved in DOS, ASCII text format so that programs external to CLASS could easily retrieve or modify data as needed by the user. This technique was found to be extremely slow for data acquisition when moderate size (100-150 samples) training and testing data sets were used. The second alternative, using a standardized database file format, was decided upon to be implemented and is currently being used by CLASS. As mentioned before, all data is stored in a Microsoft Access format database file. This alternative has several advantages over the other technique. It places all information created by CLASS into one file so no information can be accidentally erased or moved. It also speeds up data acquisition considerably for CLASS. Data can still be retrieved and modified by programs other than CLASS. This allows the user to use another program (for example Microsoft Access 2.0) to manipulate the data if a certain technique or operation isn't implemented by CLASS. File size was a consideration, because the database file can grow to be several megabytes in size, but both alternatives create large amounts of stored information on the disk and the database structure stores that information more efficiently than the sub directory tree would have done. The disadvantage to using the database file structure is the programming needed to implement database acquisition, but because Visual Basic was chosen for implementation this task was made easier. Visual Basic has several new built in commands to allow easier access to data stored within database files.

Within the database all information is separated into tables (Figure 5). Example names of tables that are present within the database file are: Header Info, Training Data, Training
Data - Features, Testing Data, Testing Data - Features, Concepts Learned Results as well as several others. This is a short list of some of the tables that can be found within the database file. The "Training Data" table, for example, contains all the raw data signals for training of the learning algorithm while the other related table "Training Data - Features" holds the features derived from that raw data. The same holds true for the testing data which is found in the "Testing Data" table. Also, there is a table (Header Info) that stores information about the database so that CLASS knows how to handle the different aspects of the database. Other information stored within this file will be: classification results from the chosen learning algorithms, training information for each individual learning algorithm, calculated transformations and user defined options; just to name a few.

![Database Structure Diagram]

Figure 5. A graphical representation of the database used by CLASS.
While creating the database structure a few obstacles needed to be overcome. The most important one was how to overcome a limitation that Visual Basic places on table size. Visual Basic, because of how its database engine is implemented, requires tables within the database to have less than 256 fields (columns) but there is no limit on the length (number of rows) of the table. Thus, the table to store the raw data and all transforms had to be setup in a special way such that this limitation would not be a factor. It was decided that the best way to avoid this problem would be to place all data samples for a particular domain (transform) into a single column (Figure 6). To do this all samples would have to be the same length so that the start of each sample could easily be calculated. Therefore, CLASS requires all data samples to be the same length (i.e. same number of discrete points). Data samples are placed within one column and a particular sample thus can easily be found by simply calculating the position of the sample using equation (1). The manual found in Appendix A describes the database structure in more detail.

\[
row \# = (sample \ length) \times (sample \ # - 1) + 1
\]

The example structure of Figure 6 shows that in the database the frequency domain data is kept separate from the time domain data. The question may arise as to why both data sets are stored when one domain (time domain) can be calculated from the other domain (frequency domain). Although this design does use more hard disk space it also minimizes the amount of time needed for calculating the transformations specified by the user. This is important since average data sets may be large and slow down the classification dramatically if transformations had to be calculated every time the user wanted to change something within the classification process. The decision was also based on the fact that hard disk storage is constantly getting cheaper.
<table>
<thead>
<tr>
<th>ID</th>
<th>Concept</th>
<th>Time Domain</th>
<th>Freq. Domain (Real)</th>
<th>Freq. Domain (Image)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Figure 6](image.png)

Figure 6. A graphical representation of how the "raw" data table is arranged.

Once CLASS has been told what database to use (by double clicking on the database's filename) the classification process can then be started. Note that the folders follow the flowchart of Figure 1 so that by working left to right through the folders, the user can easily choose all options needed for the classification process. The next step in the process is manipulating the data if needed. Figure 7 shows the View/Edit file folder. This folder allows the user to view the data files header information as well as edit the data in two different ways: either a spreadsheet or a graphsheet window.
Figure 7. The view/edit folder found within CLASS.

Figure 8 shows what CLASS' spreadsheet window looks like. This spreadsheet is specialized for tasks that users usually need to do to modify the raw data. The user can modify one cell at a time by highlighting the cell to modify and then start typing the new number wanted. A second way to modify data, and the real power behind CLASS' spreadsheet, is to highlight a range of cells (example: the first 20 columns), and type the number wanted and press the fill range button. This three step process will change all the cells highlighted to the new number entered. A good example of when this option would be used is in the removal of the "initial pulse" that is found in the signal when using an ultrasonic contact transducer. This initial pulse usually occurs during the first part of the signal and should be
removed because it is a large amplitude noise signal caused by the contact between the transducer and the material being probed.

The spreadsheet is organized so that each row is an individual, time series, data sample and each column represents a discrete point in time for the signal. The very first column shows the concept, or classification, of the data sample shown in the row. The numbers found in the individual cells represent the voltage signal for the data sample for the specified time for the column it is found in. Thus, column "T0" is the initial discrete point and column "T1" is the next discrete point found in the time series. Finally, the data samples are loaded into the spreadsheet in exactly the same order they are placed within the data file.

Figure 9 shows CLASS' graphsheet window. This window allows the user to graphically edit the raw data, one signal at a time. There are actually four ways to edit the
data within this window. Editing can be done manually by using the two text boxes just above the left side of the graph window. This is not very time effective, but good to use if only a few data points need to be modified. The other three ways use the mouse to edit the data. They are: 1) the mouse enters both the Y and X axis values desired for modifying the data sample, 2) the mouse enters only the X axis value and the Y value is assumed to equal zero for the data sample modification, and 3) the user chooses a left point and a right point and

![Figure 9. The graphsheet edit window.](image)

CLASS calculates a straight line connecting those two points thus putting a ramp (tapering effect) within the data sample. These different ways allow the user to modify the data so that unwanted noise can be removed or so a ramp function can be put into the sample so that one can generate a smooth transition to a baseline. The graphsheet was created primarily to allow the user to graphically view, print, and copy (to other applications) the data samples included within the database. As this application was being implemented it was seen that it could easily
allow graphical editing of the samples if needed by the user, thus the most popular ways NDE users usually modify data samples was included within this graphsheet. One difficulty that had to be overcome was how video resolution caused a problem with editing of the graph. Data samples can be any length and usually are 512 to 2000 points in length but video resolution is considerably lower than this on most PCs. Because of this, the mouse pointer can only pick specific points within the graph. For example, the mouse pointer might be able to point at the pixel on the screen that represents the discrete point 314 on the data sample and the next pixel to the right on the screen could represent the discrete point 331. The problem to overcome is how can the discrete point 315 to 330 be modified by the user. It was decided that a straight line interpolation between the new inputted values for the discrete points 314 and 331 would be calculated. Thus, all points within the data sample can be modified if needed. To minimize this effect, the video resolution should be made as high as possible and the graph should be made as large as possible on the screen.

Once editing is done, the next step is to choose the domains (i.e. the transformations) needed. Figure 10 shows what the transformations folder looks like for CLASS. Here the user can choose multiple transformations. The transformations chosen to be calculated are then marked by an "X" in each choice box. If transformations have already been calculated (from a previous session) they are marked with a small asterisk, *, just to the right of the "X". Note that CLASS is smart enough not to recalculate transformations that have already been calculated. Figure 10 shows that for this example, only the time domain is needed because all the features that Dr. Chiou chose come from the time domain (i.e. voltage versus time).

With the desired transformations chosen, the user moves over to the next folder which is the next step in the classification process. Figure 11 shows the features/filters folder. Features are calculations that characterize the time sequence data sample signals. If properly chosen, these features can characterize the difference between the concepts to be classified.
This folder is identical to the transformations folder in how it works. The only difference is that this folder also contains two check boxes to the right of the available features menu. These check boxes will allow two options (which are not implemented in CLASS yet) that we expected would be useful. Checking the "Filter out the Outliers" box will use an algorithm that weeds out possible outliers (i.e. data samples whose features are not consistent with other similarly classified samples within the data set) found within the data set by using a three stage clustering algorithm [3]. Choosing the "Dempster-Schafer" box utilizes an algorithm that can calculate a confidence level for how a data sample has been classified [4, 5, 6, 7]. An example would be "Data sample 3 was classified as FLAWED with a HIGH confidence level (0.91 to 0.97) of being correct". Within the features menu, the different features are separated by which transformation each one is used with. Once again, CLASS is smart enough not to recalculate features that have already been calculated. For the hard alpha inclusion problem,
we will choose the same six features used by Dr. Chiou for this problem as shown in Figure 11.

The next step in the process is the Learning algorithm folder (Figure 12). This folder has two menus that work in the same way the transformations and features menus operated. An "X" marks the algorithm that was chosen (with the mouse) to be used in the classification process and an asterisk marks the last algorithm used. Both menus within this folder contain choices of classification methods. They differ only in that they separate the unsupervised and supervised algorithms so the user can easily differentiate between the two types of algorithms. However, the user can only pick one algorithm at a time. Figure 12 shows that the ID3 decision tree algorithm [10] is the selected learning algorithm and the K-nearest neighbors [1] method was the last algorithm used. Once this step is complete, the user can choose the "Train & Test with current options" menu option under the "CLASS" menu item within the

Figure 11. The features/filter folder found within CLASS.
Figure 12. The learning algorithms folder found within CLASS.

main menu bar at the upper left part of the window. This menu option starts the classification process.

Once the classification process is started, the next few steps are controlled by CLASS. Figure 13 shows the transformations window and Figure 14 shows the similar features window. Both windows have a gauge in the upper right corner of the window to tell the user how much has been calculated and how much is left to be done for each table the program has to work with.

Once the transformations and features are calculated and placed in the database tables, CLASS brings up the chosen learning algorithm. For this example, the ID3 Decision Tree algorithm was selected so Figure 15 shows the interface window for that algorithm. This is a
good example of what CLASS' algorithms should look like to a user to maintain a consistent interface. First note that each previous window for CLASS has a standard steel gray interface. The learning algorithms need to keep that appearance as well. Also, all the algorithms should incorporate the "Database Info" window so that the user knows what data file and what tables are going to be used and possibly modified. There should be a drop down box (shown in Figure 15 with the words "Chosen Features" inside it) filled with the selected features, so that the user can recall which features are currently being used. There should be the "Current Processing Information" box with a memo window and gauge so that the user will always know what process is being done and how much is left to do for that process. Finally, there needs to be some version of the four buttons (easily allowing for training/testing, help and quitting the learning algorithm) found at the bottom right corner of the screen. CLASS will maintain the same "look and feel" if all algorithms adhere to these simple interface standards.
Usually the parameters that are specific to the learning algorithm are kept close together. For this example, the decision tree parameters are found grouped together within the box labeled "ID3 Decision Tree Options" in the learning algorithms window. The five parameters control how the decision tree will respond to the data set during training.

The first parameter ("Number of Trees to Generate") limits the number of trees that will be generated. The decision tree algorithm builds trees based on random divisions of the training data set, so a different tree can be generated from the same data set when a different set of divisions within the training set are used. These divisions are based on the principle of minimizing the entropy (or amount of disorder) within the training set as governed by equation (2), where the variable $p_i$ is the probability of class $i$ occurring within the training data.
The second parameter ("Training fraction to use") tells the algorithm what percentage of the training samples will be used for the development (training) of the decision tree. The rest will be used for testing of the decision tree during the training stage. If there are 115 training samples and a value of 0.75 is entered for this parameter then 86 (random) samples will be used for training and 29 (random) samples will be used for testing (during training) of the decision tree. The third parameter ("Window Size to use") dictates how many of the samples to be used for training (governed by the second parameter) will be used for training initially. The ID3 algorithm is a multiple step algorithm where the decision tree is grown from the initial set of randomly picked training samples and tested with the unchosen training samples. Incorrectly classified training samples are added to the initial set of picked training samples.
and then the tree is retrained. This proceeds until all the "testing" training samples are correctly classified or until all possible training points (chosen through the second parameter) are used for training. The fourth parameter ("Maximum Number of Conjuncts") limits the depth of the tree. This parameter says how many levels (length of a branch) a decision tree can have. Finally, the fifth parameter ("Do Pruning") tells the algorithm if it can combine (or remove) leaves of a decision tree if it is statistically satisfactory to do so. Some times branches within a decision tree become specialized and are only valid for a small number of data samples. This allows the algorithm to remove these "specialized" branches so that generalization (as opposed to memorization) of the data space is obtained. Once all parameters are chosen the train/test button is pressed and the classification process becomes automated once again. Finally, when the learning algorithm has been trained and tested, the user is returned to the CLASS work area. Output from the training/testing of the algorithm can then be viewed under the last folder, (Figure 16).

Within this last folder the user can choose to view two different types of information. The first button is labeled "View Concepts Learned Results". This button shows how the training data was classified as well as overall percent of correct classification (or probability of detection (POD) in the hard alpha case) and percentage of incorrect classification (probability of false alarm (POF)) for the training data set. The values for POD and POF are calculated by equation (3) and equation (4). Figure 15 shows the window that is brought up when the user clicks on this button.

\[
POD_{Concept(n)} = \frac{\text{# of samples correctly classified as concept}(n)}{\text{total # of samples for concept}(n)}
\]  
\[
POF_{Concept(n)} = \sum_{m=1}^{\text{total # of known concepts}} \frac{\text{# of samples incorrectly classified as concept}(n)}{\text{total # of samples for concept}(m)}
\]  

(3)  

(4)
Within this window (Figure 17) there are two tables. The top table shows the overall classification (POD & POF) for each known concept within the training data set. The bottom table breaks the classification results up into the individual training data samples. A variety of information is found within this second table including: all of the calculated feature values (the window in Figure 17 shows only the values for the "zero crossing" feature known to CLASS as "Transform1-Feature6"), calculated classification, actual classification, whether or not the sample was classified correctly and any notes that the learning algorithm might have made for the data sample (column not shown in Figure 17). The window is scrollable so that the user can view all of information found within this table. CLASS has one nice feature that allows learning algorithms to tell the user about problems it had for a data sample (this output

Figure 16. The display output folder found within CLASS.
found in the "notes" section). CLASS also allows the user to put this information into a formatted page that can be viewed within a window or printed to paper. Figure 18 shows what this looks like on the screen. Notice that this formatted page removes all of the calculated features and only shows the classification results.

Figure 17. The concepts learned results window found within CLASS.
The second button shown in Figure 16 is labeled "View Learning Convergence and Confidence Levels". This button deals with the training aspect of the classification process. If the algorithm is iterative by nature (an example would be the backpropagation neural net) then the user can view how well it converged through the iterations. Since decision trees are not iterative an example is included of what the backpropagation neural net would have produced if it has been chosen as the learning algorithm to use. Figure 19 shows what the window looks like when the second button (from Figure 16) has been clicked.

<table>
<thead>
<tr>
<th>ID</th>
<th>Classified Concept</th>
<th>Actual Concept</th>
<th>Correct Classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Non_Flaw</td>
<td>Non_Flaw</td>
<td>Yes</td>
</tr>
<tr>
<td>2</td>
<td>Flaw</td>
<td>Flaw</td>
<td>Yes</td>
</tr>
<tr>
<td>3</td>
<td>Non_Flaw</td>
<td>Non_Flaw</td>
<td>Yes</td>
</tr>
<tr>
<td>4</td>
<td>Non_Flaw</td>
<td>Flaw</td>
<td>No</td>
</tr>
<tr>
<td>5</td>
<td>Flaw</td>
<td>Non_Flaw</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Flaw</td>
<td>Flaw</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Flaw</td>
<td>Non_Flaw</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Flaw</td>
<td>Flaw</td>
<td></td>
</tr>
</tbody>
</table>

Figure 18. The formatted window of the concepts learned results found within CLASS.
Figure 19. The convergence graph window found within CLASS.
CLASS - A SUMMARY

CLASS is in the preliminary stages of a program's life cycle. Most interface features and options are currently available. There are however some options still to be implemented: 1) marking outliers found within the training set and 2) the confidence level calculations using the Dempster-Schafer knowledge-base expert system.

Currently CLASS supports only time domain signals but two other transforms ("Magnitude-Freq. Domain" and "Phase-Freq. Domain") are being implemented and tested and should be available in the near future. CLASS has only six features available right now because of the transformation limitations. These six features are: 1) Mean (equation (5)), 2) Absolute Mean (equation (6)), 3) Variance (equation (7)), 4) Skewness (equation (8)), 5) Kurtosis (equation (9)), and 6) Number of Zero Crossings (equation (10)).

\[
\text{mean} = \mu = \frac{1}{N} \sum_{i} x_i \quad (5)
\]

\[
\text{absolute mean} = \frac{1}{N} \sum_{i} |x_i| \quad (6)
\]

\[
\text{variance} = \sigma^2 = \frac{1}{N-1} \sum_{i} (x_i - \mu)^2 \quad (7)
\]

\[
\text{skewness} = \frac{1}{N} \sum_{i} \left( \frac{x_i - \mu}{\sigma} \right)^3 \quad (8)
\]

\[
\text{kurtosis} = \frac{1}{N} \sum_{i} \left( \frac{x_i - \mu}{\sigma} \right)^4 - 3 \quad (9)
\]

\[
\text{zero crossing} = \sum_{i} P_i; P_i = 1 \text{ if } x_i * x_{i+1} < 0, P_i = 0 \text{ otherwise} \quad (10)
\]

There are three learning algorithms included with CLASS presently. "K-nearest neighbors" is an unsupervised learning algorithm while the "ID3 Decision Tree" and the "Probabilistic Neural Network" are both supervised learning algorithms. Note that a backpropagation neural network algorithm is planned and should be implemented shortly.
Finally, when it comes to data storage CLASS is very restrictive. All data must be stored within the database file (using the Microsoft Access format standard). Because of this all the data samples must be placed into the database before CLASS can use any of the data information. Note that this process (which has been automated) is described thoroughly within the manual found in APPENDIX A. Because the Access standard is used, third party database software can be used to retrieve any information stored within the database file.
EXTENDING CLASS

CLASS allows the user to easily add transformations, features and learning algorithms without having to recompile the code for CLASS itself. There are actually two different procedures involved. One procedure is for adding transformations and features, the other is for adding learning algorithms. The two tables in Appendix B outline both of these procedures.

Adding a transformation or feature is very easy with CLASS. Code is already written that reads and writes the information from the database file and passes the data to the subroutines that calculate the individual transformations and features. This means the user only has to write the subroutine to calculate the new transformation or feature. The new subroutine is placed within the code for the transformation program or the features program and the program is recompiled. These are small, external programs that CLASS controls, thus they are completely separate from the code used to implement CLASS. This means updating is easier and that CLASS itself does not need to be recompiled. The last step is to add the transformation or feature name to CLASS' initialization file so that CLASS knows the new transformation or feature has been added and so that it knows how to use the new transformation or feature.

Adding a learning algorithm is slightly more complicated from a programmers point of view. This is because each algorithm needs to access not only the data but also prior training results and user defined parameter values. Because of this, the programmer needs to essentially write all the code necessary for a stand alone algorithm. Many different learning algorithm examples are included in the first version of CLASS so the user can use that code as an example of how to read and write data that is stored within a CLASS database file. The learning algorithm needs to be able to read data from the user chosen data sample tables found within the file as well as store training information that pertains to itself within the database.
file. Once the code is written and compiled for the learning algorithm the rest of the process is simple. All that is needed to allow CLASS to use the new algorithm is to include a line within CLASS' initialization file that tells class what the name of the learning algorithm is. By following the rules shown in Table 6 of Appendix B, the user not only tells CLASS the name of the algorithm but also what type of algorithm it is as well as its executable name.
CONCLUSIONS AND FUTURE WORK

The strength of CLASS lies in its "user-friendly" interface and its extendibility. The interface takes the general data flow of a classification problem and implements it graphically within CLASS. Also, new features and classification techniques can be easily added to CLASS. This allows the work developed at the Center for NDE to be easily placed within the package CLASS so that it can be used by workers in the NDE field.

The current limitations of CLASS are two fold. First, CLASS needs to have more features and learning algorithms added to it. As shown previously, CLASS easily allows for these to be added, but it will take time. Second, CLASS is limited to one-dimensional "raw" data signals. Because of what CLASS was designed for and how it was written, it only handles data similar to ultrasonic data (one-dimensional). To be completely flexible, CLASS should be modified to handle eddy current (complex) data. Other aspects that need to be addressed (from a software engineers point of view) is to bring CLASS completely through the alpha and beta testing stages so that it is completely tested. This will not only weed out unknown programming errors but also improve the user interface because of the feedback (from users) involved in the beta testing stage.

To summarize, CLASS is by no means finished. It was designed with expansion and flexibility in mind. Following is a short list of some areas that need to be addressed in the future: 1) add other transformations, features and learning algorithms, 2) use CLASS in conjunction with flaw scattering simulations to test classification algorithms (this is important because it is hard to get many real samples with flaws of different types, locations, etc.), 3) add capabilities for more easily handling of eddy current data, and 4) try to generate classification explanations (rule-based expert systems) directly from classification results of learning algorithms such as decision trees and neural nets.
BIBLIOGRAPHY


17. Specht, D. F. "Enhancements to Probabilistic Neural Networks"; *Proceedings IEEE International Conference on Neural Networks* June 7 - 11, 1992, Baltimore, MD.


APPENDIX A - CLASS MANUAL

Ver 1.0

By: Loren Knutson

Date: April 29, 1995

Introduction

This manual will attempt to address three main issues as well as an in-depth overview of how to use all of CLASS' features. The following is a list of the issues to be addressed.

1. CLASS was written to simplify the flaw classification process when using ultrasonic waves. During development it became extremely flexible, and very large. It has several strong, nice features as well as a few weaknesses, both of which will be addressed in this manual.

2. CLASS gives a unifying interface for Artificial Intelligence algorithms. It has been written so new algorithms can be added and still maintain the same "feel". This manual will discuss, in-depth, how the interface is and should be set up as well as how new algorithms may be added.

3. CLASS gives an easy way to take raw ultrasound data, transform it into different domains if needed and then calculate features based on the domains the data is in. This manual will attempt to discuss how new transforms and features can be added to CLASS so that CLASS can be expanded when ever needed.

CLASS will be broken down into it's main features. There are multiple ways to do some of the same tasks in CLASS. This manual is written so that the first sections discussed are probably the quickest (i.e. easiest) way to accomplish the task discussed.
CLASS terminology

CLASS is set up so that it has separate and very different areas. Within the main window (the window that is visible when CLASS first starts up) there are six folders. These folders work just like the file folders found in a file cabinet. Each one has different information in them. They are set up so that if you follow the folders from left to right and answer all applicable information, you can easily run a complete classification problem. Certain folders have buttons that will open up another window within CLASS. These windows are completely separate ideas or tasks and thus are kept separate from CLASS' main window. For example, CLASS has a spreadsheet window that opens up when you tell CLASS that you want to edit your raw data in the spreadsheet. Notice that this is a completely separate idea from trying to classify data (which is what the main windows main task is).

The Toolbar

The toolbar, show in Figure 20(a), is one of the easiest ways to accomplish a given task. It is definitely the quickest and most straightforward method to use. At certain times (depending where you happen to be within CLASS) a button might be grayed out, indicating that the button has no defined action for that area of CLASS and thus can not be used at that time. Following is a brief description of what each button does, as well as a picture of what the button looks like when it is not grayed.

The "File New" button.

CLASS is designed so that when it first starts up, no data file is open. This forces the user to select the data file that is to be used during the session. No other options are available until this selection is made. This button, see Figure 20(b), will reset CLASS so that it is back to original startup up configuration. It closes all file connections that are open and disables all options.
The "File Open" button.

This button, see Figure 20(c), is similar to the "File New" button but it doesn't close the open files. All it does is give you a quick way of getting back to the data file selection folder.

The "File Save" button.

This button, see Figure 20(d), is not used at this time. In future releases you will be able to save the way CLASS' options have been set and how the windows have been arranged.

The "Print" button.

This button, see Figure 20(e), is used to print out pertinent information. Depending on where you are within CLASS you will print out different information. Some folders don't allow printing because of the type of information they have. Also, different windows within CLASS use this button.

The "Clipboard Cut" button.

This button, see Figure 20(f), is used to cut (remove) the information from CLASS and put it in the Windows clipboard. This button is only valid in a couple of screens and never actually removes the information completely from the data file. Because of this the cut button for CLASS is no different than the copy button. It is included just to keep the Windows interface standards.

The "Clipboard Copy" button.

This button, see Figure 20(g), is used to copy information from CLASS and put it in the Windows clipboard. Depending on where you are within CLASS it will copy different information to the clipboard. Some areas will copy text but other areas will copy a graph to the clipboard. CLASS was designed so that it knows what data should be copied. It is suggested that you experiment with CLASS and see what
information is put into the clipboard depending on which folder you are using or which window is currently active.

**The "Clipboard Paste" button.**

This button, see Figure 20(h), is used to paste information from the Windows clipboard and put it in CLASS. This button is only valid in a couple of screens and never actually adds the information completely to the data file. Because of this the paste button for CLASS is very limited in function. It is included just to keep the Windows interface standards.

**The "HELP" buttons.**

These buttons, see Figure 20(i), are not fully operational at this time. In future releases you will be able to obtain full help on CLASS and all its options just by pressing the button with the question mark only on it. Also, the context help button, which looks like the button shown in Figure 20(j), will allow you to click on any button or text area that CLASS has and you will receive a full explanation of how that area is used. This will be brought through to all learning algorithms also, thus shortening the learning curve for a new classification technique.

![CLASS toolbar and its buttons](image-url)
How to use the Folders

Raw Data Location: The only place to start.

CLASS forces you to choose the database that you want to work with. There are no options available until a database is chosen. The File, Path and Drive List boxes work just like all other MS Windows file boxes. The Raw Data Location folder is shown in the Figure 21. Notice how the other five folders are not selectable because no database file name has been selected.

![Figure 21. Main window for CLASS.](image)

To select the database follow the following simple steps:

1. Choose the drive that the data file is located on. Let's assume you choose drive C:

2. Choose the directory the file is found in. To do this first double click on the root, shown in Figure 22(a), directory. Now, let's assume your data is found in c:\class\data. You now would double click on the class, shown in
Figure 22(b), subdirectory. Finally, you would double click on the data, shown in Figure 22(c), subdirectory that is found underneath the class subdirectory.

3. Choose the file that you want to work with. To do this you can either double click on the file's name or you can drag the file's name over to the box marked "Current Raw Data File Name". The box will change to the color red when you can "drop" the file name. Either way, the "Current Raw Data File Name" and the "Current Raw Data Path" boxes should be filled with the appropriate information when you have done this last step properly.

![Directory icons used within CLASS.](image)

Notice how the Path List box allows a visual interpretation of the directory tree so that is easy to find your way to the path you need. Also notice that there are two types of folders: the open folder, shown in Figure 22(d), and the closed folder, shown in Figure 22(e). The open folder shows you that the directory has been selected (double clicked) and is now showing all directories underneath that directory. It also is showing all files in the File List box that meet the file type ".*mdb".

**View / Edit File: When the data isn't good enough.**

This folder is the next logical step in classification. Within this folder you can view the header information of the selected data file. Figure 23 shows what the folder looks like.
The header information describes the type of data found within the files as well as when it was created, update dates when data has been changed, the transforms and features that have been calculated, the size and number of samples, the concepts within the file, the tables found within the file and the last algorithm performed on the data. Figure 24 shows an example of all of the text found within the header information section.

Figure 23. View/edit folder for CLASS.

With this folder you have two choices for editing the data. You can use either the spreadsheet by pressing the spreadsheet, see Figure 25(a), button or the graphsheet by pressing the graphsheet, see Figure 25(b), button. The spreadsheet is slow in loading but it allows you to easily edit all data samples within a data table at the same time. An example of what it used for would be to quickly zero out the first 100 columns of all data samples. The graphsheet is used to edit one sample at a time. This is used to easily zero out a user specified section, put in a ramp within a user defined section, or the user can literally "draw" what he or
she wants the modified signal to look like. Each of these two editing capabilities will be discussed in more detail later on in this manual.

**Transformations:** Sometimes, you need to view it differently.

The transformations folder allows you to select what domains are needed to derive the appropriate features. For example, a feature in the Magnitude-Frequency domain might be more appropriate for a particular classification problem than a feature in the original Time domain. Figure 26 shows how the folder looks.

This folder can drastically increase the size of your data file. If you have large amounts of disk space then you could effectively check all the possible options the very first run so that all domains are available. But then your data file will increase in size quickly because the majority of the file size is for saving the different data domains. Note that once a
domain is calculated it will not be recalculated in following classification runs. Thus if possible it would be quicker to calculate all domains the first time if you have the disk space.

The transformations selection menu is easy to use. A check box with a large X (see Figure 27(a)) means that the transformation is selected and will be calculated when the classification process is initiated. An unchecked box (see Figure 27(b)) means the transformation will not be calculated during the next classification run. Note that if there is an asterisk, *, to the right of the box it implies that the transformation has already been calculated and is present within the datafile. If all checked boxes and all asterisks are paired then no transformation will be calculated because the step is unnecessary.

![Figure 26. Transformations folder for CLASS.](image)

Figure 26. Transformations folder for CLASS.

![Figure 27. Icons used to show chosen and unchosen items within CLASS.](image)

Figure 27. Icons used to show chosen and unchosen items within CLASS.
**Features / Filters:** What exactly are the data samples trying to tell us.

Features are how we tell learning algorithms what makes one concept different from another. All data spaces have good data features that separate the different concepts from each other but there are also bad features that obscure the separation. The trick is finding the good features and avoiding the bad features. CLASS allows you to quickly choose a set of features and then test to see how they work for classification. Figure 28 shows what the folder looks like.

![Figure 28. Features/filters folder within CLASS.](image)

Within the Features/Filters folder there are two separate and unique sections. The feature section is a list of available features that CLASS knows how to calculate. These features are divided up into sections separated by the domain (or transform) they are associated with. The second section is the filter section. Under this section are two choices. The first choice allows you the user to try to get rid of any outliers that are in the data. The
second choice allows you the user to try to get an estimate on how well each concept is 
learned with a particular learning algorithm.

The Feature Section

The feature selection menu is similar to the transformations selection menu. The large 
X (see Figure 27(a)) means that the feature is selected and will be calculated when the 
classification process is initiated. An unchecked box (see Figure 27(b)) means the feature will 
not be calculated during the next classification process. Again, if there is an asterisk, *, to the 
left of this area it implies that the feature has already been calculated and is present within the 
datafile. If all checked boxes and all asterisks are paired then no features will be calculated 
because the step is unnecessary.

The Filter Section

CLASS currently does not have either filter implemented. For now only the general 
concept of how each "filter" works will be discussed. For a more in-depth look at how they 
will work please refer to the articles referenced in the BIBLIOGRAPHY [3, 4, 5, 6, 7].

Filter Out the Outliers Option

First, it must be made clear what an outlier really is. This can be show easily using a 
two feature data space. An example is shown in Figure 29. In this example, the data visually 
clusters into two distinct groups except for a few points. These "straying" points are what is 
known as outliers. In this example it is easy to see what points are the outliers. Most 
classifications are not so simple. If there are more than three features it is not possible to 
simply graph the result to find the outliers. The outlier algorithm within CLASS is different 
from most outlier algorithms [3]. First, it does not need the data to be linearly separable. 
Second, it is an iterative procedure that uses a three stage of clustering
algorithm. Thus it can be thought of as using the pattern of how the data is arranged in the
data space to find the outliers. This option is not implemented currently but will be added
shortly.

The Dempster-Schafer Option

This option is also not implemented at this time but will be shortly. The following
paragraph gives a general description of how the algorithm will work.

This operation is derived from the Dempster-Schafer theory of belief functions [4, 5,
6, 7]. It gives the user a probability range of how confident the user can be about the
classification. This option, when chosen, is the last stage to run in the classification process.
After the learning algorithm is trained, this algorithm is essentially trained so that it "knows"
how accurately each concept is classified. During the testing (or classification of unknowns)
stage this algorithm calculates how "well" it believes the concept was classified. For more
information on this process, please see articles 4-7 in the bibliography.

The output from this stage is then put into the datafile. Two examples of this
output are as follows:
• This data sample was classified as CRACK and has a HIGH confidence level (0.91 to 0.97) of being correct.

• This data sample was classified as CRACK and has a LOW confidence level (0.11 to 0.23) of being correct.

This algorithm can also give a "combining" effect for classification and will be included in the future. This output will give the user a better insight into how the data sample should be classified. An example of this type of output is as follows:

• This data sample was classified as CRACK and has a MEDIUM confidence level (0.52 to 0.61) of being correct. A HIGH COMBINED confidence level (0.88 to 0.94) was calculated for this being a combination of both CRACK and POROSITY.

**Learning Algorithms: Teaching an old computer new tricks.**

The learning algorithms folder is where you tell CLASS how you want it to "think". (This is meant as loosely as possible because computers really can't think.) Each algorithm has a different approach to how it works, so each will give different results at the end of the classification process. Because of how CLASS was designed, it is easy to try several different algorithms to see which one does a better job of classification. A sample of how the folder appears is in Figure 30.

The learning algorithms selection menu is similar to the transformations selection menu and the features selection menu. A check box with an X (see Figure 27(a)) means that the learning algorithm is selected and will be used when the classification process is initiated. An unchecked box (see Figure 27(b)) means the learning algorithm will not be used during the next classification process. Again, if there is an asterisk, *, to the left of this area it implies
that the learning algorithm was the last one used. The asterisk is there to indicate the nature of the last attempt to do classification.

Notice that the selection menu is divided up into two areas. These areas are "Supervised" and "Unsupervised" learning algorithms. Although not all artificial intelligence algorithms can easily be divided up into these two categories, they do generally exhibit characteristics of at least one of them. When an algorithm is added to CLASS, it must be placed in one of these two categories. Certain characteristics have to be looked at to establish which section it should go into.

What is an Unsupervised Learning Algorithm?

To summarize what an unsupervised learning algorithm is we can look at their general characteristics. Unsupervised learning algorithms usually use the training data, in some capacity, directly for classification. There is usually no interactive procedure that takes place. Finally, no attempt to generalize the data space of the training data is done. In other words, decision surfaces are usually not found by the learning algorithm.
A good, simple example is the K-Nearest Neighbors (KNN) algorithm [2]. The algorithm does exactly what the name implies. If K = 3 then the algorithm will search the entire training set and try to find the 3 nearest points to the testing point. After it finds these "nearest neighbors" it then takes a poll of them and finds out which concept has the majority vote from all "nearest neighbors". The testing point will then be classified as the majority concept.

Notice from the above example how the characteristics of the KNN algorithm matches the characteristics of the unsupervised learning algorithm. First, the training data is used directly for classification. Second, no iterations take place. Finally, no generalization of the data space is ever attempted. The entire training data set is used for every testing data sample.

What is a Supervised Learning Algorithm?

Supervised learning algorithms can also be summarized by a few general characteristics. The algorithm is usually based on an iterative procedure. Through iteration, a generalization of the data space is attempted to be "learned". This generalization (not the training data) is then used for classification of testing data samples.

One well known example of a supervised learning algorithm is the Backpropagation Neural Network (BpNN) algorithm. This algorithm, through an iterative procedure, tries to find a generalized set of decision boundaries that are multi-dimensional surfaces separating the training data's data space into regions. These surfaces are stored within the weights of the network, $W_{ij}$ and $P_{ij}$, as shown in Figure 31.

While the algorithm is within it's learning phase, the weights are constantly being changed and updated. It is only after calculating that the decision surfaces are separating the training data within a specified error that the iterative procedure is finished. During the testing stage, the final weights that were achieved are then used for classification of the testing
data samples. No training data is used for classification. Please note that this is a very
simplified definition of how the BpNN works. Readers that are more interested in the internal
calculations and advancements of the BpNN should find a technical article that describes the
algorithm more precisely. The book written by Dayhoff [2] is a good example.

Figure 31. Backpropagation neural network structure.

Notice from the above example how the characteristics of the BpNN algorithm
matches the characteristics of the supervised learning algorithm. First, the training data is
used for generalization of the data space through the calculation of decision boundaries. It is
not used directly for classification. Second, it is an iterative procedure and only stops when
the generalization is calculated to be "close enough".

**Display Output: Let's see the results.**

The last folder (Figure 32) is where CLASS stores the final results of the classification
process. Here is where the user can find not only how the data has been classified, but also
well how the algorithm "learned" the training data (if it is an iterative procedure). Other
output that can be found is the Probability of Detection (POD) and the Probability of False
Alarms (POF) for the training data. These two values are pertinent only to the training and testing stages. They give the user another way to evaluate how a learning algorithm is doing. They are not valid for classification in general, thus they are not calculated for data sets that have unknown data. Finally, algorithms that have the capability of giving notes (or error messages) to the user also put the information here for the user to find.

There are two buttons to choose from within this folder. The button on the left is used to find out information about classification of the testing (or unknown) data. The button on the right shows information about the convergence of the learning algorithm when applicable.

View Concepts Learned Results

This button gives, when clicked, brings up a window with all the information about how the test data was classified. Figure 33 shows how the Concepts Learned Results window is divided. Notice that there are two tables within the window. The top table shows the POD

![Figure 32. Display output results folder within CLASS.](image-url)
and POF for each concept within the testing data. These values for POD and POF are calculated as shown in equation (11) and equation (12) respectively.

If the POD is high and the POF is low, for all concepts, then the learning algorithm has done a good job training itself. This table gives a good "overall" feeling of how well the training part of the classification process has done.

![Concepts Learned Results Window](image)

Figure 33. Concepts learned results window.

The second table lists all known information about the training data. It lists everything from the values of the different features to how the data sample was classified. The last three columns are a summary of how data samples were classified. The column labeled "Classified Concept" shows how the data sample was classified by the learning algorithm. The next
column, labeled "Actual Concept" shows what the sample is supposed to be classified as. The last column, labeled "Correct Classification", can be used for quick reference to see if the learning algorithm correctly classified the data sample. A "YES" in this column tells you that the data sample was classified correctly.

\[
POD_{\text{Concept}(n)} = \frac{\text{# of samples correctly classified as concept}(n)}{\text{total # of samples for concept}(n)}
\]

\[
POF_{\text{Concept}(n)} = \sum_{m=1}^{\text{total # of known concepts}} \frac{\text{# of samples incorrectly classified as concept}(n)}{\text{total # of samples for concept}(m)}
\]

Another feature of CLASS is the button in the bottom left corner of the window labeled "Create Summary Report". This button generates a formatted document that can be sent to a window and to the printer. This document summarizes all the information found in the second table. The document can also be sent to a comma separated values (CSV) file so that it can be easily loaded into a spreadsheet program.

The image shown in Figure 34 is an example of what the window looks like. The five left buttons at the top of the window act just like VCR buttons. Following is a quick summary of the functions for all seven buttons.

- The first button, see Figure 35(a), brings you back to the first page of the formatted output.
- The second button, see Figure 35(b), brings you to the previous page.
- The third button, see Figure 35(c), brings you to the next page.
- The fourth button, see Figure 35(d), brings you to the last page of the formatted output.
Table 1 - Class: Output Results

<table>
<thead>
<tr>
<th>ID</th>
<th>Classified Concept</th>
<th>Actual Concept</th>
<th>Correct Classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Non_Flaw</td>
<td>Non_Flaw</td>
<td>Yes</td>
</tr>
<tr>
<td>2</td>
<td>Flaw</td>
<td>Flaw</td>
<td>Yes</td>
</tr>
<tr>
<td>3</td>
<td>Non_Flaw</td>
<td>Non_Flaw</td>
<td>Yes</td>
</tr>
<tr>
<td>4</td>
<td>Non_Flaw</td>
<td>Flaw</td>
<td>No</td>
</tr>
<tr>
<td>5</td>
<td>Flaw</td>
<td>Non_Flaw</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Flaw</td>
<td>Flaw</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Flaw</td>
<td>Non_Flaw</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Flaw</td>
<td>Flaw</td>
<td></td>
</tr>
</tbody>
</table>

Figure 34. The formatted window of the concepts learned results found within CLASS.

- The fifth button, see Figure 35(e), is a stop button and stops loading data into memory when it is pressed. This can be useful if there are several test data samples and your interested only in the first few.
- The sixth button, see Figure 35(f), switches the window view between "full page view" and normal view.
- The seventh button, see Figure 35(g), prints the formatted document to the standard printer in Windows.

Figure 35. Buttons found within formatted concepts learned results window.
View Learning Convergence and Confidence Levels

This button will show the convergence of the learning algorithm. This is only valid for learning algorithm that are iterative procedures. Thus, a Decision Tree or K-Nearest Neighbors algorithm would not have a convergence graph. CLASS is smart enough to know when an algorithm has a convergence graph or not. If the algorithm doesn't have a convergence graph, then CLASS opens a "Not Available" window (shown in Figure 36) on the screen.

Figure 36. Window shown by CLASS if no convergence graph available.

Learning algorithms that are iterative will have a convergence graph. CLASS allows the learning algorithm to calculate the convergence graph. This allows the implementor of the learning algorithm to decide what would be the best way to calculate how accurately the algorithm has been trained up to the iteration it is on. For some algorithms, for example, the mean squared error can be used to show the convergence path. Since each learning algorithm could be different, the help information could be consulted as long as such information is provided with each algorithm. The author of the module needs to include all aspects of the learning algorithm, including how the convergence graph is calculated. The window shown in
Figure 37 is an example of what this graph will look like if the algorithm supports a convergence graph where the y-axis is the amount of error and the x-axis is the iteration number. In this example the convergence of the algorithm is well behaved. It quickly approaches zero error.

![Convergence & Confidence Results](image)

**Figure 37.** Example convergence graph.

However, not all classification runs will be so nice. The help information provided with each learning algorithm could provide more information on what to look for in this graph. Topics that need to be addressed in these sections by the author include:

- Oscillation within the convergence rate
- Divergence rather than convergence
- Other topics that are related to the specific learning algorithm

**What's in the Menus?**

Certain aspects of CLASS are only accessible from the menu bar as shown in Figure 38. The following part of the manual will go step by step through each menu selection and
describe what the selection will do. Notice that some menu selections are identical to a toolbar button that was discussed earlier.

**The "File" Menu.**

**The "File New" Menu Command**

This menu command, see Figure 39 is exactly like the "File New" toolbar button. CLASS is designed so that when it first starts up, no data file is open. This forces the user to select the data file that is to be used during the session. No other options are available until this selection is made. This button will reset CLASS so that it is back to its original startup up configuration. It closes all file connections that are open and disables all options.

**The "File Open" Menu Command**

This menu command, see Figure 39 is exactly like the "File Open" toolbar button. This button is similar to the "File New" button but it doesn't close the open files. All it does is give you a quick way of getting back to the data file selection folder.

**The "File Printer Setup" Menu Command**

This command, see Figure 39, has no equal anywhere else within CLASS. This option brings up the setup window for the printer. This window allows you to select which printer you want to use for printer output. Using this option allows you to direct the output for the "Print" toolbar button.
The "File Exit" Menu Command

This command is the same as the big red stop sign (Figure 40) that you can find on CLASS' main folder. Selecting this option properly closes the connection to the database file as well as exiting the application.

The "CLASS" Menu.

The "Train & Test" Menu Command

This menu option, see Figure 41, starts the classification process. Once all options have been chosen, clicking on this option will start the training and testing for the current database file. CLASS will perform the following steps:

1. Calculate transforms, if needed, for both the training and testing data tables.
2. Calculate features, if needed, for both the training and testing data tables.
3. Run the chosen learning algorithm for training.
4. Run the chosen learning algorithm for testing.
5. Show classification results from the testing data.
The training and testing data are found in two specific tables that must be available within the database file. The training data must be located in the table "Training Data" and the testing data must be located in the table "Testing Data".

**The "Re-Test" Menu Command**

This menu option allows the user the ability to set up an alternative testing data table. Let's assume data was gathered several months back and the learning algorithm was trained and tested on that data. New data could be gathered today and placed in an alternative data table. By using this option, the user can specify which data table has the new testing data. CLASS then performs the following steps:

1. Calculate transforms, if needed, for the specified table.
2. Calculate features, if needed, for the specified table.
3. Run the chosen learning algorithm for testing using the specified table.
4. Show classification results from the specified tables.

This allows the user to verify that the data that trained the algorithm in the past is still valid for the data being obtained in the present. It is always a good idea to check the reliability of an algorithm from time to time.

**The "Classify Unknowns" Menu Command**

This menu option allows the user to classify a set of unknown data samples. Once a learning algorithm has been trained, this option can be used to classify unknowns. By using this option, the user can specify which data table has the new testing data. CLASS then performs the following steps:

1. Calculate transforms, if needed, for the specified table.
2. Calculate features, if needed, for the specified table.
3. Run the chosen learning algorithm for classifying unknowns using the specified table.

4. Show classification results from the specified tables.

One feature CLASS will have implemented in the future is the ability to run the chosen algorithm through training if needed. Right now CLASS will not automatically do this for the user.

**The "Window" Menu.**

**The "Cascade Windows" Menu Command**

This option, see Figure 42, will arrange all open windows, that belong to CLASS, in a cascading order. It will arrange the windows, on top but slightly offset from each other, in the upper left hand corner of the CLASS work area. The current window will be the top window when this is completed.

**The "Tile Windows" Menu Commands**

These two options, see Figure 42, arrange the windows, that belong to CLASS, in a tiled order. The horizontal option places the windows side by side, where the vertical option places the windows below each other. Most of the time, especially when there are three windows or more visible, these two options will appear to do the same task. The only difference is the order in which the windows are placed.

Please note that these two menu options do not usually change the size of the windows. Most of CLASS' windows are fixed in size and do not resize. Please keep this in mind when using these two options.

**The "Arrange Icons" Menu Command**

This option, see Figure 42, will arrange the icons that are found on CLASS' work area. Any of CLASS' windows that have been minimized will be an icon within the work area. If
for some reason they become moved or can't be found, using this option will place them (side by side) in the lower left corner of the work area.

The Numbered Menu Command

Each window that is opened in CLASS is given a number by CLASS and then has it's name and number placed within this menu. When the desired menu option is chosen, the window corresponding to the menu option selected will be brought to the top and made the current window. This is the easiest way to switch between windows when there are multiple windows floating around within the CLASS work area.

The "Help" Menu.

The "Contents" Menu Command

This menu option, see Figure 43, allows the user to call up CLASS' on-line help file. The table of contents page is automatically shown. The on-line help file is essentially this manual with "hot links" and "search keys" added in so that the user can look up how to use a feature easily. This option is the same as the toolbar button with a question mark, see Figure 44(a), on it.

The "Search" Menu Command

This menu option, see Figure 43, allows the user to call up CLASS' on-line help file with the search dialog box automatically open. This bypasses the steps needed to open the search box after the help window is shown.
The "Context-Sensitive" Menu Command

This menu option, see Figure 43, turns on the context sensitive help cursor which looks like, see Figure 44(b). Once this cursor is visible, the user can click on virtually any location on the CLASS window and get an immediate help window explaining what the option or area is used for and how to use it. This option is the same as the toolbar button with the arrow and question mark, see Figure 44(c).

![Figure 43. Help menu items.](image1)

![Figure 44. Toolbar help buttons and mouse icon.](image2)

The "How to Use Help" Menu Command

This menu option, see Figure 43, opens up a help window that explains how to use the windows help system. This help comes with the Windows operating system and most Windows programs allow an easy access menu link to this help window.

The "About CLASS" Menu Command

This menu option, see Figure 43, opens up a small dialog window showing information about CLASS (version number and last major modification date) and the environment that it is running in. Figure 45 is an example of what the window should look like.
Note that under the horizontal line is some very helpful information about the environment that CLASS is running in. It tells the user what mode Windows is running in. The example window shows 386 enhanced mode. It tells the user if a math co-processor is present (the example window shown states that the math co-processor is present). The two most important pieces of information the window tells the user is how much free memory is available and how much system resources is available. If CLASS seems to act strange or is very slow check these two values. If the free memory is low (less than one megabyte of memory) or if the system resources is low (less than 30% free) then the environment is causing the problems. You should at your earliest convenience close all applications and restart Windows.

CLASS' Spreadsheet

CLASS' spreadsheet is a specialized spreadsheet version. It's strength is filling in ranges with a user specified number. Figure 46 shows what the spreadsheet window looks like.

Figure 45. About CLASS window.
The main item in the window is the grid of cells that comprise the spreadsheet. This grid works just like any other spreadsheet program on the market. To edit one cell all you need to do is click on the cell you want to edit and start typing the number you want to place in the cell. If for some reason you want to start with the number that is already found in the cell, use the "/" key to place the number in the edit box (Figure 47(a)). The button with an x on it (Figure 47(b)) will clear the edit box and not change the cell that is currently being edited. The "Enter" key or the button with a check mark (Figure 47(c)) on it will commit the changes to the current cell.

One nice specialized feature that CLASS' spreadsheet has is the fill range button (Figure 47(d)). This will fill any selected range with a user defined number. To use this button, first select the desired range using the mouse. You can do this by clicking the same way as any other spreadsheet program. You can click on a row or column heading to highlight the row or column. You can click and drag an area within the grid. Once the desired area is selected, type the number you want to put in the range and press the fill range button. Note that pressing enter will not fill the range with the number.

Figure 46. CLASS spreadsheet window.
The two drop down boxes (Figure 48) are for switching between transforms and data tables. Click on the down arrow to get the list of available data tables or transforms. Then just click on the name of the table or transform you want to edit. Note that once CLASS has calculated a transform it will not calculate it again if it finds it present in the datafile. This allows you to edit a transformation without fear of losing your modifications the next time you do a classification run.

![Image of drop down list boxes](image)

Figure 48. Drop down list boxes to switch between different data sets.

The toolbar on the side of the grid gives the user five helpful quick functions.

- The first button (Figure 49(a)) saves the updates to disk permanently.
  Once pressed no undo is possible.

- The second button (Figure 49(b)) is a form of an undo. It reloads the data from the file. If you have made a change that you don't want to keep then pressing this button will undo that change.
- The third button (Figure 49(c)) copies the current row (the row where the dotted outlined cell is) to the clipboard. This makes it easy to copy a data sample to another application.
- The fourth button (Figure 49(d)) switches to CLASS' graphsheet window. It will close the spreadsheet window before switching. If you have any unsaved changes, CLASS will ask you if you want to save them before switching.
- This fifth button (Figure 49(e)) closes the spreadsheet window. If you have any unsaved changes, CLASS will ask you if you want to save them before closing the window.

Finally, if for some reason the grid doesn't have both vertical and horizontal scroll bars, you can press the redraw screen (Figure 49(f)) to update the grid on the screen. This is useful sometimes when the window has been resized smaller and the grid does not update properly.

![Image](c) (b) (c)

![Image](d) (e) (f)

Figure 49. Shortcut buttons for spreadsheet.

**CLASS' Graphsheet**

CLASS' graphsheet is a specialized graphical editing application. It's strength is the ability to graphically edit a data sample using the computer's mouse. Figure 50 shows what the graphsheet window looks like.
Notice that graphsheet only works with one data sample at a time and that the sample is drawn on a X-Y chart within the window. To switch between samples, use the scroll bar to the immediate left of the graph window.

You can edit the sample in several different ways. The first way is to manually enter the changes you want through the keyboard. You can do this by using the "Enter data point value" button, see Figure 51, and the two text boxes to its immediate right. By typing the new value desired and the X location for the point, you can enter the new value by pressing the "Enter data point value" button.

You can graphically edit the data sample in one of three following ways:

1. The mouse pointer enters the actual Y value it's pointing at for the X location it's pointing at.
2. The mouse pointer enters a zero for the Y value at the X location it's pointing at.
3. A straight line is calculated connecting the two X-Y data pairs selected by the mouse pointer.

Figure 50. Graphsheet window.
Using the option buttons within the area to the left of the X-Y chart, you can choose between one of these three ways to way the sample. CLASS defaults to the first option as shown in the Figure 52.

The first two options do exactly what they sound like. The only difference between the two is the value they place in the sample for the Y value. To change one point only, you can point the mouse cursor on the chart for the data sample you want to change and press the left mouse button. If you are using the first option, the Y value that the mouse pointer is pointing at will be placed in the sample. If you are using the second option, a zero will be placed in the sample. You can also click on a starting location and drag the cursor of the data sample to an ending location to change a range of values within the sample. Note that the values for each point dragged over is changed using the same logic as described above. The third option is slightly different. By clicking on two points within the graph you can draw a straight line between those two points. The Y values for the end points are taken from the data sample and not the mouse cursor.

One limitation that CLASS has is that neighboring pixels on the screen probably do not represent neighboring points on the graph. An example of this would be one pixel might
represent X location 512 and the pixel just to the right of it represents X location 531. Thus there is real data that falls in the 513 to 530 range, but it can't be accessed because of the video limitations. CLASS addresses this problem by interpolating a straight line between the two points 512 and 531. Note that you can limit this resolution problem by using the maximum video resolution your card can produce along with making the graph window as large as possible.

The two drop down boxes, see Figure 48, work the same way as they do in the spreadsheet window. They are how you can switch between transforms and data tables. Click on the down arrow to get the list of available data tables or transforms. Then just click on the name of the table or transform you want to edit. Note that once CLASS has calculated a transform it will not calculate it again if it finds it present in the datafile. This allows you to edit a transformation without fear of losing your modifications the next time you do a classification run.

The toolbar on the side of the grid gives the user five helpful quick functions.

- The first button (Figure 53(a)) saves the updates to disk permanently. Once pressed no undo is possible.
- The second button (Figure 53(b)) is a form of an undo. It reloads the data from the file. If you have made a change that you don't want to keep then pressing this button will undo that change.
- The third button (Figure 53(c)) copies the current graph to the clipboard. This makes it easy to copy a data sample to another application.
- The fourth button (Figure 53(d)) switches to CLASS' spreadsheet window. It will close the graphsheet window before switching. If you have any unsaved changes, CLASS will ask you if you want to save them before switching.
• The fifth button (Figure 53(e)) closes the graphsheet window. If you have any unsaved changes, CLASS will ask you if you want to save them before closing the window.

Finally, if for some reason the graph doesn't appear to have resized itself to fit properly within the window, you can press the redraw screen (Figure 53(f)) to update the grid on the screen. This is useful sometimes when the window has been resized smaller and the graph does not update properly.

The Database File Structure

CLASS uses a dynamic but simple file structure for storing all of its information. All data is stored in one database file. The database is in Microsoft Access format. Data is separated within the database by using tables.

![Database Table Icons](c), (b), (c)

![Redraw Screen Icon](d), (e), (f)

Figure 53. Shortcut buttons for graphsheet window.

Only three data tables are required within the database. The first required table is named "Header Info". This table stores all the information pertinent to the rest of the data within the database file. The other two required tables are named "Testing Data" and "Training Data". These tables hold the raw data for testing and training of the learning algorithms. CLASS will have a runtime error if any of these three files are not present.
Other tables that are found within the database are calculated features from the raw data, user specified parameters for each individual learning algorithm, classification results, and learning algorithm convergence data when applicable. When features are calculated from raw data a new table is created. The naming of this new table follows an exact naming convention. For instance, if CLASS calculated the features for the table "Training Data" the new table would be named "Training Data - Features".

The procedure for creating these tables is simple. CLASS comes with a utility that reads in your raw data and creates a brand new database file. The utility is named datamakr and will step you through the creating of a new database. The main window is shown in Figure 54.

Note that there are two buttons at the top left hand side of the window. The first one creates an empty database file. Datamakr will step you through the process by asking questions about the data that will be placed within the database. What you will need to know before creating the database is:

1. The number of databases that will be added (remember the training and testing data tables are mandatory).
2. What concepts will be used for classification.
3. The maximum possible length for the raw data samples.
4. The number of data samples within each table.
5. The names of the data tables that you want to use (only for the tables other than the training and testing data tables).

Future releases of datamakr will allow you to append new tables into the database at later times, but for now you can only use datamakr to create and add data for the first time. If you need to add a new table later on, you will need to use a third party database software to
do it manually. All you need to do is add the new table and place it's name in the "Header Info" table.

**Expanding CLASS' capabilities**

Adding transformations, features, and learning algorithms to CLASS is very easy. All that is needed is a fundamental understanding of how to read and write Visual Basic code. Learning algorithms can be written in any language, whereas transformations and features must use the Visual Basic compiler. Some understanding of the Windows environment and how to connect programs together is helpful, but goes beyond the scope of this manual.

![Database Maker](image)

**Figure 54. Datamakers main window.**

**Adding transformations and features to CLASS**

Let's discuss how to add a transformation or feature calculation routine. The same five step process is used to add either a transformation or a feature. For simplicity the procedure found in Appendix B - Table 5 refers only to adding a feature, but "transformation"
can be substituted for any place "feature" is seen. The note contained within Table 5 also explains the minor naming convention differences between transformations and features.

Note that the programs "CLASS: Features" and "CLASS: Transformations" are found within the files features.exe and transfrm.exe. These programs are small in code length and have all data access routines already implemented. The only code that must be written by the user is the subroutine itself for the transform or feature calculation. The data from the database file will be passed to your subroutine through a data array. That's all there is to adding a transformation or feature to CLASS. CLASS will then know how to use the new transformation or feature the next time it is executed.

Adding Learning Algorithms to CLASS

New let's discuss how to add a learning algorithm routine. This process is similar to adding transformations and features to CLASS. The main difference, see Appendix B - Table 6, is that the programmer must write all of the program code for the learning algorithm. Because learning algorithms are so different, there is no easy way to supply routines for data access. CLASS does come with example learning algorithms that you can use as models for your programs.

Once the code for the learning algorithm is done, adding it to CLASS is easy. As you can see, the steps to tell CLASS how to access the algorithm is similar to adding transformations and features. That's all there is to adding a learning algorithm to CLASS. CLASS will now know how to use the new algorithm the next time it is executed.

How the Learning algorithm communicates with CLASS.

Following is a short list of all the items your learning algorithm will need to know now to do to communicate with CLASS properly. One item that is not mentioned is that your learning algorithm should store its internal data (such as: user defined parameters and its trained state) within the database as well. You can choose any format you desire as long as it
is named the same as your learning algorithms executable name. For instance, if your algorithm will be compiled to the name LAS-9999.exe then you need to use the table name "LAS-9999".

**Command Line arguments**

CLASS tells the learning algorithm what data file to use and what needs to be done through command line arguments. Following is an example of what CLASS would pass to a learning algorithm as command line arguments.

```
/Ic:\class\data\hrdalpha.mdb /TTraining Data, Testing Data
```

There are three switches that your learning algorithm should be able to deal with. They are as follows:

- **/I** The data file that should be used.
- **/T** The data tables to be used for training and testing of the algorithm.
  Multiple tables are listed.
- **/U** The data tables to be used for classification of unknown data tables.
  Note that the table "Training Data" should be assumed to contain the training data if needed during classification. Multiple tables can be listed.

Please note what CLASS passes as a data table name. It will pass to your learning algorithm the table containing the raw data, but your learning algorithm needs the table containing the features. This means that you must append " - Features" to the end of the data table names.

**Getting the user defined options**

The table "User Defined Options" can be used to find out what the user has chosen for features and transformations, if needed. Table 1 shows the table structure.
If (for example) multiple features have been chosen, they will all be listed under the features field within multiple rows. This applies also to transformations. Most of the time your learning algorithm will not need the other two fields.

Table 1. User Defined Options Table Format

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
<th>Field Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformations</td>
<td>Text</td>
<td>255</td>
</tr>
<tr>
<td>Features</td>
<td>Text</td>
<td>255</td>
</tr>
<tr>
<td>Filters</td>
<td>Text</td>
<td>255</td>
</tr>
<tr>
<td>Learning Algorithm</td>
<td>Text</td>
<td>255</td>
</tr>
</tbody>
</table>

**Inputting data features**

Data is brought in for training, testing and classification through tables as well. All the data tables containing features will have format shown in Table 2.

**Outputting classification results**

Classification output should go to the table "Output Results". Please see the KNN code for an example of how to write the data output to this table. Table 3 shows the table structure.

**Outputting convergence results**

Your algorithm must modify this table whether your algorithm is iterative or not. Table 4 shows the table structure.

If your algorithm is not iterative, then you must place the value "Not Available" in the very first row of the Title field. If your algorithm is iterative, then place a title, incorporating the name of your algorithm, in the first row of the Title field. Then for each iteration place the value that represents the amount of error, for your algorithm, into the Value field. An example of what this value might be is mean square error.
### Table 2. Data Table Format

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
<th>Field Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Long Integer</td>
<td></td>
</tr>
<tr>
<td>Concept</td>
<td>Text</td>
<td>255</td>
</tr>
<tr>
<td>(All the features names)</td>
<td>double</td>
<td></td>
</tr>
<tr>
<td>Each feature gets its own column of data.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table 3. Classification Results Table Format

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
<th>Field Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Long Integer</td>
<td></td>
</tr>
<tr>
<td>Notes</td>
<td>Text</td>
<td>255</td>
</tr>
<tr>
<td>(All the features names)</td>
<td>double</td>
<td></td>
</tr>
<tr>
<td>Each feature gets its own column of output.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Classified Concept</td>
<td>Text</td>
<td>50</td>
</tr>
<tr>
<td>Actual Concept</td>
<td>Text</td>
<td>50</td>
</tr>
<tr>
<td>Correct Classification</td>
<td>Boolean</td>
<td></td>
</tr>
</tbody>
</table>

### Table 4. Convergence Results Table Format

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
<th>Field Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>Text</td>
<td>255</td>
</tr>
<tr>
<td>iteration</td>
<td>Long Integer</td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>Double</td>
<td></td>
</tr>
</tbody>
</table>
How the Learning algorithm should look.

There are certain visual characteristics that all learning algorithms should maintain. Figure 55 is an example of what the interface should look like.

Some of the key features that should be present are as follows:

- The window should have the gray, 3D etched look to it.
- There should be a database info area. Telling the user what database and what tables are being used.
- There should be a drop down list box (box in Figure 55 with "Chosen Features" in it) filled in with the chosen features.

![Figure 55. K-nearest neighbors main window.](image)
• A current process information area (with the percent done gauge) should be present.
• There should be a "Context-sensitive help" button and a regular help button. Allowing the user quick and easy help for the learning algorithm.
• There should be a "Train / Test" button to start the training and testing of the algorithm.
• There should be a "STOP" button to exit the program.
• Finally, there should be a "CLASS" icon in the lower right corner of the etched area on the learning algorithm window. This icon is there shows to the user that this program is part of the CLASS classification system and helps keep the uniform look and feel that CLASS should have.

By following these guidelines, a uniform appearance is maintained between learning algorithms. The layout of the window does not matter, other than it should not be cluttered and it should be easy to use. User defined parameters should be easily seen while allowing the user to have a clear understanding of what the parameters do. The other learning algorithms that are released with CLASS show good examples of how the different features can be placed within the learning algorithms window. APPENDIX C has an overview of the learning algorithms currently released with CLASS.
Table 5. Adding a Transformation or Feature

<table>
<thead>
<tr>
<th>Steps needed To add a new feature named: NewFeatureName to CLASS</th>
<th>CLASS.INI</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Create the subroutine using the following calling convention: Function Transform1 NewFeatureName (DS() As Double, N) As Double</td>
<td>[CLASS]</td>
</tr>
<tr>
<td>2. Add it to “CLASS: Features” and tell the program how to call the subroutine: Function CalculateThisFeature (F$) As Double Dim Feature As Double 'Calculate the feature Select Case F$ 'All these features are for Transform1. . . Case “Transform1-Feature7”: Feature = Transform1 NewFeatureName (DataSample(), UBound(DataSample, 1)) End Select CalculateThisFeature = Feature End Function</td>
<td>[Transformations] TotalNumber=1 Transform1=Time Domain</td>
</tr>
<tr>
<td>3. Recompile the program “CLASS: Features”.</td>
<td>[Features, Transform1] TotalNumber=6 Feature1=Signal Mean Feature2=Signal Absolute Mean Feature3=Signal Variance Feature4=Signal Skewness Feature5=Signal Kurtosis Feature6=Number of Zero Crossings</td>
</tr>
<tr>
<td>4. In CLASS.INI change the line under [Features, Transform1] to: TotalNumber=7</td>
<td>[Learning Algorithms, Unsupervised] TotalNumber=1 Algorithm1=K-Nearest Neighbors</td>
</tr>
<tr>
<td>5. In CLASS.INI add the following line under [Features, Transform1]: Feature7=NewFeatureName</td>
<td>[Learning Algorithms, Supervised] TotalNumber=2 Algorithm1=Probabilistic Neural Network Algorithm2=ID3 Decision Tree (IDEAS)</td>
</tr>
</tbody>
</table>

The next time CLASS runs it will now know how to use the new feature.

Note: The above steps are identical for adding a new Transformation. The only differences are: [Features, Transform1] -> [Transformations], “CLASS: Features” -> “CLASS: Transformations”
### Table 6. Adding a Learning Algorithm

<table>
<thead>
<tr>
<th>Steps needed To add a new algorithm named: NewAlgorithmName to CLASS</th>
<th>CLASS.INI</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Create the algorithm using any language that compiles a windows program.</td>
<td>[CLASS]</td>
</tr>
</tbody>
</table>
| 2. Compile the program using the following naming convention:  
   If Unsupervised Algorithm then:  
   \[ \text{LAU-####} \]  
   If Supervised Algorithm then:  
   \[ \text{LAS-####} \]  
   Where #### is the next number for that type of Learning Algorithm.  
   Example: LAS-0003 is the next name to use for a supervised learning algorithm in the CLASS.INI file to the right. | [Transformations]  
   TotalNumber=1  
   Transform1=Time Domain |
| If the new Algorithm is Unsupervised then:  
3. In CLASS.INI change the line under [Learning Algorithms, Unsupervised] to:  
\[ \text{TotalNumber=2, (if we were using the CLASS.INI file to the right.)} \]  
else if it's Supervised then:  
3. In CLASS.INI change the line under [Learning Algorithms, Supervised] to:  
\[ \text{TotalNumber=3, (if we were using the CLASS.INI file to the right.)} \]  
4. In CLASS.INI add the following line under [Learning Algorithms, *****]:  
\[ \text{Algorithm?=NewAlgorithmName} \]  
Where ***** is Unsupervised or Supervised and ? is the next number in the sequence for that type of Learning Algorithm. | [Learning Algorithms, Unsupervised]  
TotalNumber=1  
Algorithm1=K-Nearest Neighbors  
[Learning Algorithms, Supervised]  
TotalNumber=2  
Algorithm1=Probabilistic Neural Network  
Algorithm2=ID3 Decision Tree (IDEAS) |
APPENDIX C - LEARNING ALGORITHMS

K-Nearest Neighbors (KNN)

This algorithm is the only unsupervised learning algorithm currently implemented in CLASS. It is a standard KNN routine [2] with a slight modification that allows the user to weigh more heavily those concepts that are considered to be more important. Figure 56 shows the window for the KNN classification algorithm. Note that this is not an iterative procedure so no convergence graph is created by the algorithm.

Essentially there are two parameters that the user needs to fill in. The first parameter is the "Number of Nearest Neighbors" found in the upper right corner of the window. This

![Figure 56. K-nearest neighbors main window.](image-url)
routine allows anywhere from 1 to 15 nearest neighbors and defaults to 3 if the algorithm has never been run before on the database. Both odd and even values are allowed, but because of ties possibly occurring it is suggested to always use odd values for this parameter. The user can either type the value into the box or use the spinner button (showing both up and down arrows) to the right of the input box to input this value. If a tie does occur during classification, this algorithm will tell the user by placing a message into the note section of the output results table.

The second parameter is the "Apriori Probabilities" section and can actually consist of several values the user needs to input. There are three choices to choose from: 1) have the probabilities come from the training data, 2) have equal probabilities, or 3) have user defined probabilities. The first option will calculate what the weights (probabilities of occurrence) or importance of a concept should be from the training data. This is calculated by equation (13) for all known concepts "i". If the training set actually represents how often a concept usually occurs then this is a good option to pick. The second option tells the algorithm that all concepts have equal importance and thus should have equal weights (probabilities of occurrence) calculated by equation (14). One thing that should be mentioned is that this algorithm only keeps 4 decimal places for the probabilities, thus there is a chance that the probabilities calculated for either of the first two choices will not add up to 1. Because of this, the algorithm assumes that the first concept is more important than the rest and adds the needed amount to this probability such that all of the probabilities equal 1. The button found below the spreadsheet (labeled "Get Data Info") allows the user to see what values will be used when either of the first two choices are picked. This button calculates the values of the probabilities and puts them into the spreadsheet for the user. The third and final choice is for the user to assign the probabilities for the concepts. This is done by selecting (clicking with the mouse) the desired concept (row) within the small spreadsheet and then typing the value
desired. The two buttons to the left of the edit box work the same way as they do in CLASS' spreadsheet. The button with the X works like an undo button and ignores what was typed. The button with the check mark works like the "Enter" key on the keyboard and enters the value typed into the spreadsheet. Values can be typed in as either probabilities or percents. This means that the user can type either 45 or 0.45 into the edit box and the algorithm will place it into the spreadsheet as 0.45. The two boxes just above the spreadsheet (labeled "Probs. Equal" and "Amount Left") show the user what all the probabilities in the spreadsheet add up to and how far that total is from 1. These numbers are also color coded as follows: Black - occurs when both values are equal to 1, Red - occurs when the value is greater than 1, and Yellow - occurs when the value is less than 1.

\[
P_i = \frac{\text{# of samples in concept } i}{\text{total # of samples in training set}} \tag{13}
\]

\[
P = \frac{1}{\text{# of known concepts}} \tag{14}
\]

The four buttons in the lower right corner allows the user to control the learning algorithm. The button with an arrow and question mark will give the user the ability to click on any item found within the window and get help information back for how that item is used. The button with a question mark will bring up the full help information that is present for this learning algorithm. The "Train / Test" button starts the training and testing process of the learning algorithm using the parameters as defined by the user. Finally, the "Stop" button exits the learning algorithm and returns the user to CLASS.

The rest of the window is used as information to the user. The "Database Info" area tells the user what database and tables are being used. The "This is the list of calculated features" area shows the user all chosen features that are being used. Finally, the "Current Processing Information" area gives feed back to the user about what process is currently being done and the percentage (shown graphically on the gauge) completed for that process.
Probabilistic Neural Network (PNN)

This algorithm is one of two supervised learning algorithms currently implemented in CLASS. It is a combination of PNN ideas compiled from several sources [16, 17, 18]. Figure 57 shows the window for the PNN classification algorithm. Please note that is an example of how a multi-document interface approach can be used to implement a learning algorithm under CLASS. This should be used when there is a considerable number of options made available to the user. Options should be grouped together and placed within their own individual window as shown by this example. Also, options in one window should not rely on how options in another window is filled in. This minimizes the confusion for the user on how parameters are related to one another. Note that the PNN algorithm is not an iterative procedure so no convergence graph is created except for the gradient learning option which modifies the PNN algorithm in such a way as to make it an iterative procedure. When the
The gradient learning option is chosen the PNN algorithm outputs a convergence graph using mean square error of the classification results of the training data as the value for this graph.

Essentially there are four sets of parameters that the user needs to answer. The four sets of parameters are 1) concept weights, 2) shaping parameter, 3) how to use the training data, and 4) what calculation to use within the pattern units (Figure 58).

The weight parameters work like the weights in the KNN algorithm. There are two sets of spreadsheet cells that have values within them. The top spreadsheet is a set of values that are user defined. This is done by selecting (clicking with the mouse) the desired concept (row) within the small spreadsheet and then typing the value desired. The two buttons to the
left of the edit box work the same way as they do in CLASS' spreadsheet. The button with the X works like an undo button and ignores what was typed. The button with the check mark works like the "Enter" key on the keyboard and enters the value typed into the spreadsheet. These values are not restricted to constraints associated with probabilities (i.e. the numbers shown in Figure 57 show that Flaw is twice as important as Non_Flaw). The lower spreadsheet values are calculated from the training data by equation (15) and are the apriori probabilities of each known concept "i" within the data set. The check box found immediately below the associated spreadsheet tells the algorithm whether (checked) or not (unchecked) the values should be used within the calculations.

The second window allows the user to choose how to handle the values for the PNN's sigma parameters. The values for the shaping parameters (sigmas) can be either user inputted or calculated by the algorithm. If user specified is chosen, the value found just below the "User specified Sigma" option button is used for sigma. This value can be directly edited by the user or can be calculated from the training set, using equation (16), where N is the number of features and M is the number of training samples, by pressing the "Calculate Possible Optimal Sigma" button.

\[
P_i = \frac{\text{# of samples in concept } i}{\text{total # of samples in training set}}
\]

\[
\sigma_{opt} = \left( \frac{4}{N+2} \right)^{\frac{1}{N+4}} \left( \frac{1}{M^{1/2}} \right)
\]

If the "Calculate Sigma" option is chosen then there are two possible ways to have the algorithm calculate the sigmas for training. Choosing the covariance matrix option tells the algorithm to calculate the covariance matrix for the training data and use the diagonal of the matrix as the sigmas for training. The gradient learning option starts out the same as the covariance matrix option, but then becomes an iterative procedure that uses a gradient descent
method that gradually changes the sigmas in order to minimize the training errors[16]. This is
done in a three step process. Essentially for every iteration the hold one out training rule is
used. From this, it is known which training samples had their concepts misclassified. Once
this is completed (given a data sample of concept i was classified as concept j), the likelihood
ratio (LR) is calculated for each known concept. This is done using equation (17), where
P[xli] and P[xlj] are directly taken from the outputs of the pattern units found within the PNN
architecture.

\[ LR = \frac{P[xli]}{P[xlj]} \]  

(17)

Once these likelihood ratios are calculated, the task of maximizing P as described in
equation (18) is done using a modified version of Brent's Method [19]. Because of the way
this algorithm is set up, P is a function of the sigmas (smoothing parameters) for the PNN
algorithm.

\[ P = \sum \sum \frac{\text{mean log } LR \text{ for misclassified patterns}}{\text{mean log } LR \text{ for correctly classified patterns}} \]  

(18)

The third and last window is dedicated to the pattern unit and how they operate. First,
the user can tell the algorithm whether to use all of the training data in the training process or
to use cluster centers from the training data. When implemented, the multiple clustering
algorithms can be chosen from ranging from standard statistically based clustering algorithms
to fuzzy logic clustering algorithms. The other set of options is for specifying how the pattern
units will calculate the "distance" between the data sample to be classified and each data
sample within the training set. The Euclidean distance option uses equation (19), where "i"
corresponds to the training data sample ("T" which is a multidimensional vector), and "p" is
the number of features for the data sample to be classified ("X" which is a multidimensional
vector). The "city block" distance option uses equation (20), with the variables representing the same items as listed for the Euclidean distance equation. Both of these two options don't need the data samples to be normalized. The third choice is how the standard PNN algorithm usually does this part of the calculations and does require the data samples to be normalized. The standard PNN calculation used equation (21), with the variables representing the same items as listed for the Euclidean distance equation. Notice that this is just the dot product of two vectors.

\[
d_i = \sum_{j=1}^{p} (X_j - T_j)^2
\]

\[
d_i = \sum_{j=1}^{p} |X_j - T_j|
\]

\[
d_i = X \cdot T_i
\]

The four buttons in the lower left corner allows the user to control the learning algorithm. The button with an arrow and question mark will give the user the ability to click on any item found within the window and get help information back for how that item is used. The button with a question mark will bring up the full help information that is present for this learning algorithm. The "Train / Test" button starts the training and testing process of the learning algorithm using the parameters as defined by the user. Finally, the "Stop" button exits the learning algorithm and returns the user to CLASS.

The rest of the window is used as information to the user. The "Database Info" area tells the user what database and tables are being used. The "This is the list of calculated features" area shows the user all chosen features that are being used. Finally, the "Current Processing Information" area gives feed back to the user about what process is currently being done and the percentage (shown graphically on the gauge) completed for that process.
**ID3 Decision Tree**

This algorithm is one of two supervised learning algorithms currently implemented in CLASS. It is based on the ID3 decision tree algorithm that was implemented by Dr. Bob Forouraghi at CNDE [11, 12, 13, 14, 15]. Figure 59 shows the window for the decision tree classification algorithm. Note that the decision tree algorithm is not an iterative procedure so no convergence graph is created.

There are five parameters that the user needs to answer and they all can be found within the "ID3 Decision Tree Options" area. The five parameters control how the decision tree will respond to the data set during training. The first parameter ("Number of Trees to
Generate") limits the number of trees that will be generated. The decision tree algorithm builds trees based on random divisions of the training data set, so a different tree can be generated from the same data set when a different set of divisions within the training set are used. These divisions are based on the principle of minimizing the entropy (or amount of disorder) within the training set as governed by equation (22), where the variable $p_i$ is the probability of class $i$ occurring within the training data.

$$\text{Entropy} = \sum_i p_i \log p_i$$  \hspace{1cm} (22)

The second parameter ("Training fraction to use") tells the algorithm what percentage of the training samples will be used for the development (training) of the decision tree. The rest will be used for testing of the decision tree during the training stage. If there are 115 training samples and a value of 0.75 is entered for this parameter then 86 (random) samples will be used for training and 29 (random) samples will be used for testing (during training) of the decision tree.

The third parameter ("Window Size to use") dictates how many of the samples to be used for training (governed by the second parameter) will be used for training initially. The ID3 algorithm is a multiple step algorithm where the decision tree is grown from the initial set of randomly picked training samples and tested with the unchosen training samples. Incorrectly classified training samples are added to the initial set of picked training samples and then the tree is retrained. This proceeds until all the "testing" training samples are correctly classified or until all possible training points (chosen through the second parameter) are used for training.

The fourth parameter ("Maximum Number of Conjuncts") limits the depth of the tree. This parameter says how many levels (length of a branch) a decision tree can have. This
parameter is where the user tries to weight the importance of bias vs. variance. If the decision tree is allowed to grow to several layers (example: 30), then there it will have a low variance for classification according to the training set but it will also be biased (or essentially hard-wired) toward the training data set. Because of this it must be kept in mind that to minimize variance, bias toward the training set will go up.

Finally, the fifth parameter ("Do Pruning") tells the algorithm if it can combine (or remove) leaves of a decision tree if it is statistically satisfactory to do so. Some times branches within a decision tree become specialized and are only valid for a small number of data samples. This allows the algorithm to remove these "specialized" branches so that generalization (as opposed to memorization) of the data space is obtained.

The four buttons in the lower left corner allows the user to control the learning algorithm. The button with an arrow and question mark will give the user the ability to click on any item found within the window and get help information back for how that item is used. The button with a question mark will bring up the full help information that is present for this learning algorithm. The "Train / Test" button starts the training and testing process of the learning algorithm using the parameters as defined by the user. Finally, the "Stop" button exits the learning algorithm and returns the user to CLASS.

The rest of the window is used as information to the user. The "Database Info" area tells the user what database and tables are being used. The "This is the list of calculated features" area shows the user all chosen features that are being used. Finally, the "Current Processing Information" area gives feed back to the user about what process is currently being done and the percentage (shown graphically on the gauge) completed for that process.
APPENDIX D - ADDING A LEARNING ALGORITHM TO CLASS

In this section, I will overview how to add a learning algorithm to CLASS and give examples of how some parts of the learning algorithm can be implemented. Only certain (core) sections of code will be discussed. Several subroutines that are essential to a finished learning algorithm will not be mentioned within this section due to the wide variance of how one learning algorithm would be implemented over another algorithm.

To begin discussion of how a learning algorithm would be added to CLASS, let's assume that all code has been written for the learning algorithm and is ready to be compiled.

For CLASS to be able to communicate to the learning algorithm, the algorithm must be able to understand the command line being passed to it. An example of BASIC code that does this is shown in Figure 60 as two subroutines.

Notice how through these two subroutines all parameters (/I, /T, and /U) as described in the Command Line Arguments section of the manual are handled and set up global variables (DataFileName$, TableNameList$, and UnknownTableNameList$ respectively) according to the parameters given them. This code is essential to allowing the learning algorithm to understand what CLASS is telling it to do.

Now that all code is written, including the code shown, we are ready to compile it to an executable file so CLASS can use it when needed. The first decision to make is to decide whether it is a "Supervised" or "Unsupervised" algorithm (as described in the manual). Let's assume we are compiling a backpropagation neural network algorithm, thus it is a supervised algorithm. Because of this, the executable filename must start with the three letters "LAS" which stands for "Learning Algorithm Supervised" (accordingly the unsupervised algorithms start with "LAU"). We then must check the supervised learning algorithms section of the CLASS.INI file and see which algorithm number our new algorithm will be given. Figure 61 shows that under this section, our new algorithm will be number 3 so we must fill out the
name so that it has a total of eight letters with the last four digits being the value of the number we just found, thus our learning algorithm’s executable name will be "LAS-0003.EXE". For example, if the next algorithm number was to be 134 then the new name would be "LAS-0134.EXE".

```vb
Sub GetTableNameList ()
    If InStr(Command$, "/T") Then
        TFlag = True
    Else ' This means the /U option was used for a table of
        TFlag = False ' unknown variables
    End If
    'Find the start and end of the needed information.
    If TFlag Then
        SStart = InStr(1, Command$, "/T", 1) + 2
        SEnd = InStr(SStart, Command$, "/": If SEnd = 0 Then SEnd = Len(Command$)
    Else
        SStart = InStr(1, Command$, "/U", 1) + 2
        SEnd = InStr(SStart, Command$, "/": If SEnd = 0 Then SEnd = Len(Command$)
    End If
    'Store the information where it needs to go.
    If TFlag Then
        TableNameList$ = Mid$(Command$, SStart, SEnd - SStart + 1)
        UnknownTableNameList$ = ""
    Else
        UnknownTableNameList$ = Mid$(Command$, SStart, SEnd - SStart + 1)
        TableNameList$ = ""
    End If
End Sub

Sub GetInputDataFileName ()
    'Find the start and end of the needed information.
    SStart = InStr(1, Command$, "/I", 1) + 2
    SEnd = InStr(SStart, Command$, "/": If SEnd = 0 Then SEnd = Len(Command$)
    'Put the information where it needs to go.
    DataFileName$ = UCase$(Mid$(Command$, SStart, SEnd - SStart))
End Sub
```

Figure 60. Code to understand command line parameters sent be CLASS.
Now we know how what filename to use so we will compile the program to the specified name. This executable file is then placed in the sub directory where all of the CLASS executables reside (suggested to be "c:\class"). The final step to add the algorithm to CLASS is to modify the initialization file and add the new algorithms full descriptive name to the initialization file. The only modifications that need to be made to the file are in the area that is shown in Figure 62. This is the same area as shown in Figure 61 with the changed or added values bolded.

This is all that is needed for adding the new algorithm to CLASS. Two areas that are important is how to read and write information to and from the database. This is best shown in BASIC code once again. Because of space limitations, only the reading of testing data (Figure 63) and the creation of a data table (Figure 64) will be shown.
Sub Test()
Dim T As Table
' Set up variables and arrays if needed.

' Open the table.
Set T = DataFile.OpenTable("Testing Data - Features")
'Test the data.
T.MoveNext
For I = 1 To NumberOfSamples
    ' Load concept for sample.
    TestingSampleConcept(I, I) = T.Fields("Concept")
    ' Load features for sample.
    For J = 1 To UBound(FeatureList$, 1)
        TestingSampleFeatures(I, J) = T.Fields(J + 1)
    Next J
    ' Classify this newly loaded sample
    TestClassify I
    ' Move down 1 row if not at the EOF
    If Not T.EOF Then T.MoveNext
Next I
' Close the table.
T.Close
End Sub

Figure 63. Example code to read data from database file.
Sub CreateConceptsLearnedResultsTable (NN$)
    Dim Td As New TableDef, Fld() As New Field
    Dim T As Table
    Dim Idx As New Index
    Dim I As Integer

    ReDim Fld(1 To 4)

    Td.Name = NN$ ' Set the table name.

    ' Create Fields.
    For I = 1 To 4 ' Set properties for fields.
        Fld(I).Name = Choose(I, "ID", "Concept", "POD", "POF")
        Fld(I).Type = Choose(I, DB_LONG, DB_TEXT, DB_DOUBLE, DB_DOUBLE)
        Fld(I).Size = Choose(I, DB_LONG_SIZE, 50, DB_DOUBLE_SIZE,
                           DB_DOUBLE_SIZE)
        If I = 1 Then Fld(I).Attributes = DB_AUTOINCRFIELD
    Next I

    ' Add all Fields to the TableDef
    For I = 1 To 4
        Td.Fields.Append Fld(I)
    Next I

    Idx.Name = "PrimaryKey" ' Set properties in new Index.
    Idx.Unique = True
    Idx.Primary = True
    Idx.Fields = "ID"
    Td.Indexes.Append Idx ' Add Index to TableDef.

    ' Create Table by Adding TableDef to Database.
    DataFile.TableDefs.Append Td
End Sub

Figure 64. Example code to create a table for the database.