G e o X p l o r e Documentation

0. What is *GeoXplore*?

GeoXplore is a program that contains 3 different systems of which each can be trained on input data to learn to classify data from a similar source. They are: i) ***Fuzzy Clustering*** of feature vectors into a number of classes; ii) a ***Radial Basis Functional Link Network*** (RBFLN) that is an efficient and accurate new type of neural network that trains on feature vectors for which the class is known; and iii) a ***Probabilistic Neural Network*** (PNN) that trains on feature vectors with known classes to provide the maximum standardized probability density function value of the correct class. After training, any of these systems can process new unknown feature vectors from a similar source as the training vectors and recognize the class to which they belong.

1. Running *GeoXplore*

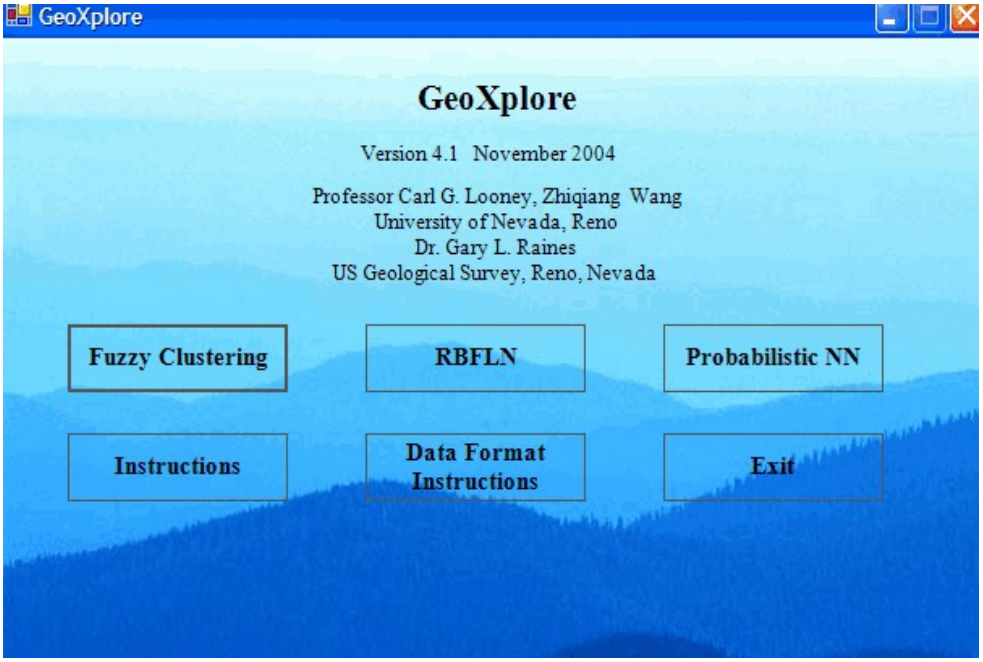
1.1 Starting *GeoXplore*. Here we assume *Geoxplore* is located in the directory *d:\path*. To start the program, either of the following mehods can be used.

Method 1. Bring up a command line window (also called a DOS window) and change to the directory *d:\path*. Then type the underlined part below to start the program.

d:\path> GeoXplore <ENTER>

Method 2. Click ***Start*** at the bottom left of the screen, click on ***Run***, then click on ***Browse***, and then go to the directory (designated by *d:\path* here) and click on ***GeoXplore.exe*** or click on ***GeoXplore*** (if the suffix ***.exe*** does not show). Figure 1 comes up.

Figure 1. The first and main window of *GeoXplore*.



1.2 Choosing a System for Training. The program is now running and the window shown in Figure 1 is visible on the computer screen where the first row of buttons allow selection of one of *fuzzy clustering*, *radial basis functional link net* (RBFLN, which is a neural network), or *probabilistic neural network* (PNN). The 3 different systems are described below. They all give approximately the same results, but one may do better or worse than the others depending on the data and the user given parameters during the training. They can be used to check each other and the best results used.

A. Fuzzy Clustering - this system clusters the input vectors of features (attributes) in an input training file into clusters. The class memberships of the feature vectors need not be known because this clustering decides the classes from the data only (if the classes for the feature vectors are given, they will not be used). Each cluster represents a class. A *center*, or *prototype* vector is computed for each class by a new fuzzy clustering method that is immune to outliers. When an unknown feature vector is put into the system after training (learning), it is assigned to the class to whose prototype it is closest. Goodness of classification values are computed.

B. Radial Basis Functional Link Net (RBFLN) - this special new type of neural network trains a set of parameters so the network yields the correct class number as an output value when a feature vector is input. The training is done on the training files of input vectors on which the fuzzy clustering system trains, but here, unlike fuzzy clustering, the class memberships for the training data must be included in the file and will be used. This system is like a radial basis function neural network except it has extra lines from the input nodes to the output nodes to model the linear parts of an input-output mapping as well as the nonlinear parts. Thus it is more accurate and it trains more quickly.

C. Probabilistic Neural Network (PNN) - This is a modification of the standard probabilistic neural network that builds a Gaussian mixture probability density function for each class, which is a sum of Gaussians that is standardized appropriately to have a volume (N-dimensional for N features) of unity. Our modification reduces the number of Gaussians by eliminating those whose centers are too close to another Gaussian center. Thus it is faster than a standard PNN. This also reduces the extraneous error that builds up when lots of feature vectors in the same class are close together (too many Gaussians).

1.3 The Training (Input) Data File Format. There are two types of input files: 1) *training data* (usually relatively small, up to a few hundred); and 2) *classification data* to be classified as being in the default class or not (may be very large, up to hundreds of thousands of feature vectors). The single format for all input data files is described below with comments after the pound sign (such comments do not appear in the data files and must not be put there). Note that the data are in comma delimited rows with no spaces allowed anywhere (a space will cause a misread and invalidate all computation). There is a carriage return at the end of each row (be sure there is no space before the carriage return). The last column is the fuzzy belief (0 to 1) that the input feature vector is in the default desired class. Here we only designate the degree to which it is thought that an input feature belongs to the single desired (default) class. If it is not in the default class then it is outside that class, so we have a yes or no tempered with a fuzzy belief where 1.0 is absolute *yes* and 0.0 is absolute *no* of membership.

Training File Row Data	**Comment on Training Data**
N	#no. features (components) in input feature vectors (integer)
M	#dummy number, but necessary (integer)
J	#always set to 1 in this version (integer)
Q	#no. feature vectors in this file (integer)
1,c1,d1,x11,x21,...,xN1,D1	#first column is vector no., necessary but not used here (integer)
2,c2,d2,x12,x22,...,xN2,D2	#second col. contains user values unused here, but necessary (integer)
: : :	#third column necessary dummy values (integer or float)
Q,cQ,dQ,x1Q,...,xNQ,DQ	#fourth to fourth+N columns are N feature values (integer or float)
	#last col. is fuzzy belief for default (desired) class (integer or float)

As examples, consider the training data in the two files below.

Training File 1

```
2
18
1
20
1,1,1.1,1,1,0.8
2,1,1.1,2,1,0.8
3,2,1.1,2,5,0.1
4,2,1.1,3,5,0.2
5,3,1.1,5,2,0.1
6,3,1.1,6,2,0.2
7,4,1.1,5,8,0.1
8,4,1.1,6,8,0.1
9,5,1.1,8,5,0.1
10,5,1.1,9,5,0.2
11,1,1.1,1,2,0.8
12,1,1.1,2,2,0.8
13,2,1.1,2,6,0.2
14,2,1.1,3,6,0.3
15,3,1.1,5,3,0.1
16,3,1.1,6,3,0.1
17,4,1.1,5,9,0.2
18,4,1.1,6,9,0.1
19,5,1.1,8,5,0.3
20,5,1.1,9,5,0.2
```

Training File 2

```
2
20
1
32
1,1,2,0.244898,0.902655,0.100000
2,0,4,0.346939,0.902655,0.90000
3,0,8,0.244898,0.893805,0.90000
4,0,9,0.183673,0.893805,0.90000
5,1,11,0.183673,0.920354,0.100000
6,1,19,0.530612,0.893805,0.200000
```

7,0,22,0.244898,0.884956,0.900000
8,0,24,0.387755,0.893805,0.860000
9,1,26,0.0,0.902655,0.100000
10,1,28,0.755102,0.902655,0.24000
11,1,29,0.755102,0.893805,0.100000
12,0,35,0.714286,0.893805,0.93000
13,1,36,0.265306,0.911504,0.16000
14,0,37,0.346939,0.893805,0.89000
15,1,43,0.265306,0.902655,0.17000
16,1,55,0.755102,0.920354,0.11000
17,1,60,0.755102,0.911504,0.10000
18,1,61,0.755102,0.982301,0.21000
19,1,71,0.755102,0.938053,0.10000
20,1,72,0.755102,0.964602,0.18000
21,1,74,0.755102,1.0,0.08000
22,0,83,0.530612,0.884956,0.88000
23,0,87,0.163265,0.902655,0.900000
24,1,90,0.530612,0.955752,0.15000
25,1,92,0.755102,0.955752,0.10000
26,1,108,0.571429,0.920354,0.21000
27,0,118,0.571429,0.884956,0.90000
28,0,128,0.77551,0.893805,0.87000
29,1,137,0.897959,0.893805,0.24000
30,0,154,0.040816,0.893805,0.91000
31,0,159,0.183673,1.0,0.88000
32,0,179,0.040816,0.884956,0.93000

1.4 The Classification (Input) Data File Format. The format of the classification input data files (data of unknown class to be classified) is like the training data file, but the last column here is not known. It is thus a dummy variable here, so no training can be done on these files (only classification with an already trained RBFLN or PNN algorithm, or fuzzy clustering can be done).

Classification File Data	**Comment on Classification Data**
N	#no. feature components in input vector (integer)
M	#dummy number, but necessary (integer)
J	#always set to 1 (integer)
Q	#no. vectors in this file (integer)
1,d1,dd1,x11,x21,...,xN1,D1	#first column is vector number (integer)
2,d2,dd2,x12,x22,...,xN2,D2	#second and third columns are dummies
: : :	#fourth to fourth+N columns are N features (integer or float)
Q,dQ,ddQ,x1Q,x2Q,...,xNQ,DQ	#last column is a dummy (unknown) column (integer or float)

An example of a classification file is given next.

2
22
1
40
1,0,1722.625,0.244898,0.911504,0
2,0,4736.0625,0.244898,0.902655,0

3,0,681.375,0.183673,0.902655,0
4,0,254.0,0.346939,0.902655,0
5,0,114.25,0.387755,0.902655,0
6,0,227.4375,0.020408,0.902655,0
7,0,89.125,0.020408,0.911504,0
8,0,14485.6875,0.244898,0.893805,0
9,0,1926.375,0.183673,0.893805,0
10,0,330.4375,0.183673,0.911504,0
11,0,179.6875,0.183673,0.920354,0
12,0,119.1875,0.183673,0.929204,0
13,0,58.25,0.183673,0.938053,0
14,0,47.5625,0.183673,0.946903,0
15,0,119.1875,0.346939,0.911504,0
16,0,22.125,0.183673,0.955752,0
17,0,648.75,0.244898,0.920354,0
18,0,219.1875,0.530612,0.902655,0
19,0,506.1875,0.530612,0.893805,0
20,0,309.0,0.244898,0.929204,0
21,0,176.0,0.244898,0.938053,0
22,0,1981.0,0.244898,0.884956,0
23,0,207.625,0.387755,0.884956,0
24,0,1185.0625,0.387755,0.893805,0
25,0,1143.75,0.0,0.893805,0
26,0,323.0625,0.0,0.902655,0
27,0,144.4375,0.0,0.911504,0
28,0,1121.3125,0.755102,0.902655,0
29,0,2610.8125,0.755102,0.893805,0
30,0,0.25,0.102041,0.911504,0
31,0,69.0,0.346939,0.920354,0
32,0,17.75,0.183673,0.964602,0
33,0,16.125,0.183673,0.973451,0
34,0,17.1875,0.387755,0.911504,0
35,0,2084.5,0.714286,0.893805,0
36,0,138.5,0.265306,0.911504,0
37,0,1241.8125,0.346939,0.893805,0
38,0,31.875,0.346939,0.929204,0
39,0,5.375,0.183673,0.99115,0
40,0,126.0,0.244898,0.946903,0

After training, the text box on the screen shows the *vector number*, the *input fuzzy belief* (of default class membership), and the *training fuzzy beliefs* of the trained network for each training vector.

1.5. The Fuzzy Clustering Classification (Output) Results Data Files. The fuzzy clustering output differs from the RBFLN and PNN. Below is shown the RBFLN and PNN output format.

Data	**Comment on RBFLN and PNN Results Data**
1,1,0.847	#first column is vector number (integer)
2,1,0.216	#second col. is the default class number that is 1(integer)
:	:
Q,1,0.769	#third col. is belief (fuzzy truth) of membership in default class (float)

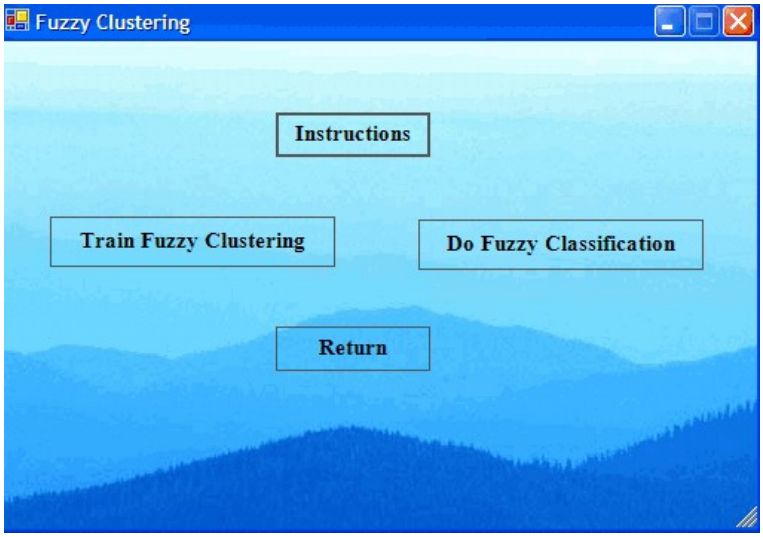
For the Fuzzy Clustering algorithm, the classification output data is written to the output file as shown below.

Data	Comment on Fuzzy Clustering Results Data
1,2,0.1,0.93,0.23,...0.17	#first column is vector number (integer)
2,1,0.88,0.26,0.17,...,0.08	#next column is the class number (default class or not, e.g., 1 is default class, 2 is not default class)
: : :	
Q,2,0.2,0.79,0.01,0.27,...,0.11	#next K columns are fuzzy beliefs for memberships in the K classes, respectively

2. Using the Fuzzy Clustering System

2.1 Training the Fuzzy Clustering System. Here we click the Fuzzy Clustering button in the window of Figure 1. The new window shown below in Figure 2 comes up on the click. The top button gives brief instructions. The first thing to do is to train the fuzzy clustering system on a training (input) data file. We will go through the steps here. Click the ***Train Fuzzy Clustering*** button now to bring up the window of Figure 3.

Figure 2. The first *Fuzzy Clustering* window.

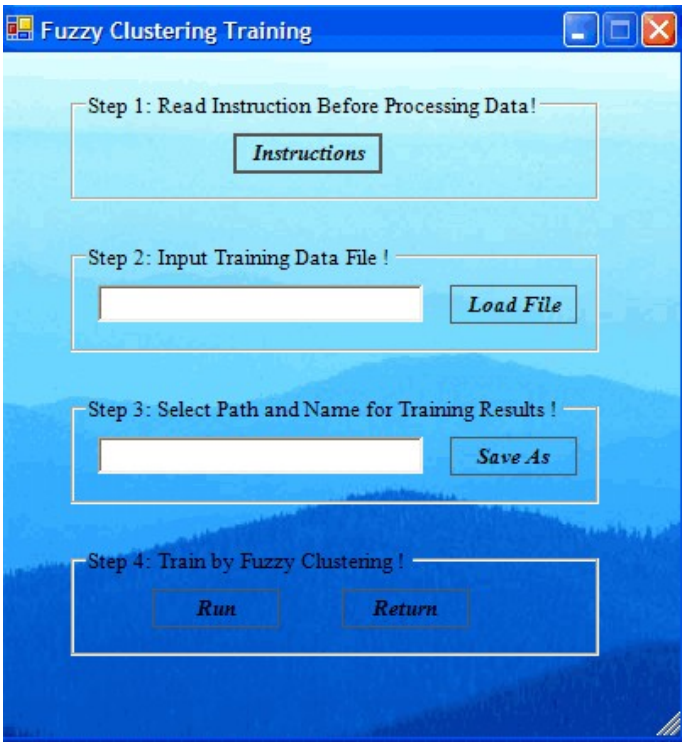


We now click the ***Load File*** button of Figure 3 and select the path and filename of the training file to use to train this system. A window will come up and allow the selection of a data file for training. Its name has the format **.dta* . Either click twice on the file name or click once on it and then click ***Open*** on the lower right. That file will be read and closed.

Next, click the ***Save As*** button of Figure 3 to provide a filename for the file that will hold the training results, that is, the parameters that were learned during training. In this case, this file holds the centers of the clusters (classes) and the sigma values for the Gaussians centered on these centers. These files have the name format **.cen* to indicate that the cluster centers are

stored here. Now click on Run at the bottom (Fig. 3) and the program runs and shows the window in Figure 4.

Figure 3. The *Fuzzy Clustering Training* window.



Click the ***Eliminate Small Clusters*** button (Fig. 4) next and then click it again. Repeat until the number of clusters does not change on a click. The results of this are shown in Figure 5. We may not know how many clusters K there should be, but we will use the Xie-Bene clustering validity to tell us what value of K gives the best clustering, although this may not be the number K that we want.

Next click the ***Merge and Fuzzy Clustering*** button of Figure 6 to get the window shown in Figure 7. Here a default parameter of 0.4 is shown. This is the proportion of the average distance between cluster centers to use as a threshold for merging two clusters. If the average distance between pairs of cluster centers is a, then if any distance between a pair of cluster centers is less than $0.4a$, the clusters will be merged into one cluster and fuzzy clustering is done again with one less cluster center. First use the default 0.4. If the number of clusters reduces, then repeat with the same parameter. Otherwise, increase the parameter from 0.4 (0.4, 0.5, 0.6,... is usually good). Note the clustering validity value for each value of the number of clusters K. The K value with the lowest clustering validity is usually the best unless there is other information to override it. Stop at K = 2 clusters and examine the clustering validity values, or possibly stop when K is the known value. For our data, when we stop at K = 2, the result is that in Figure 8.

Figure 4. The preliminary results of clustering.

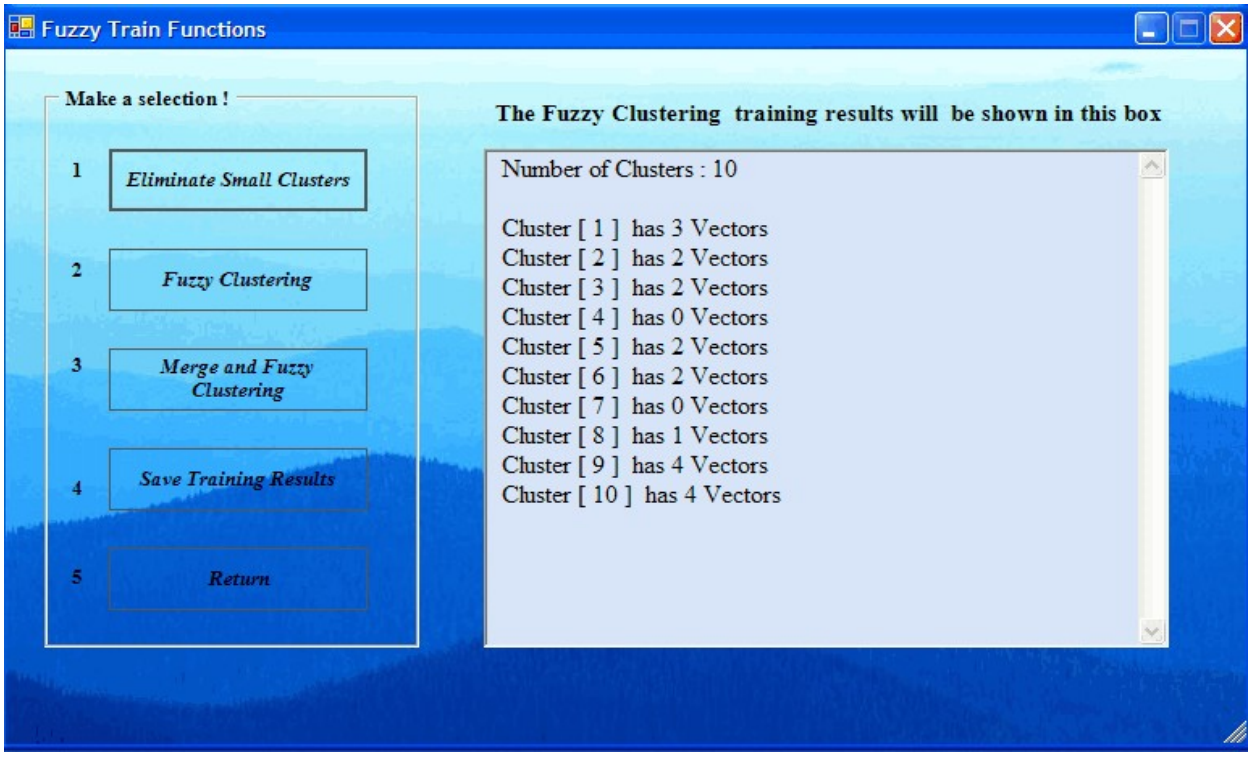
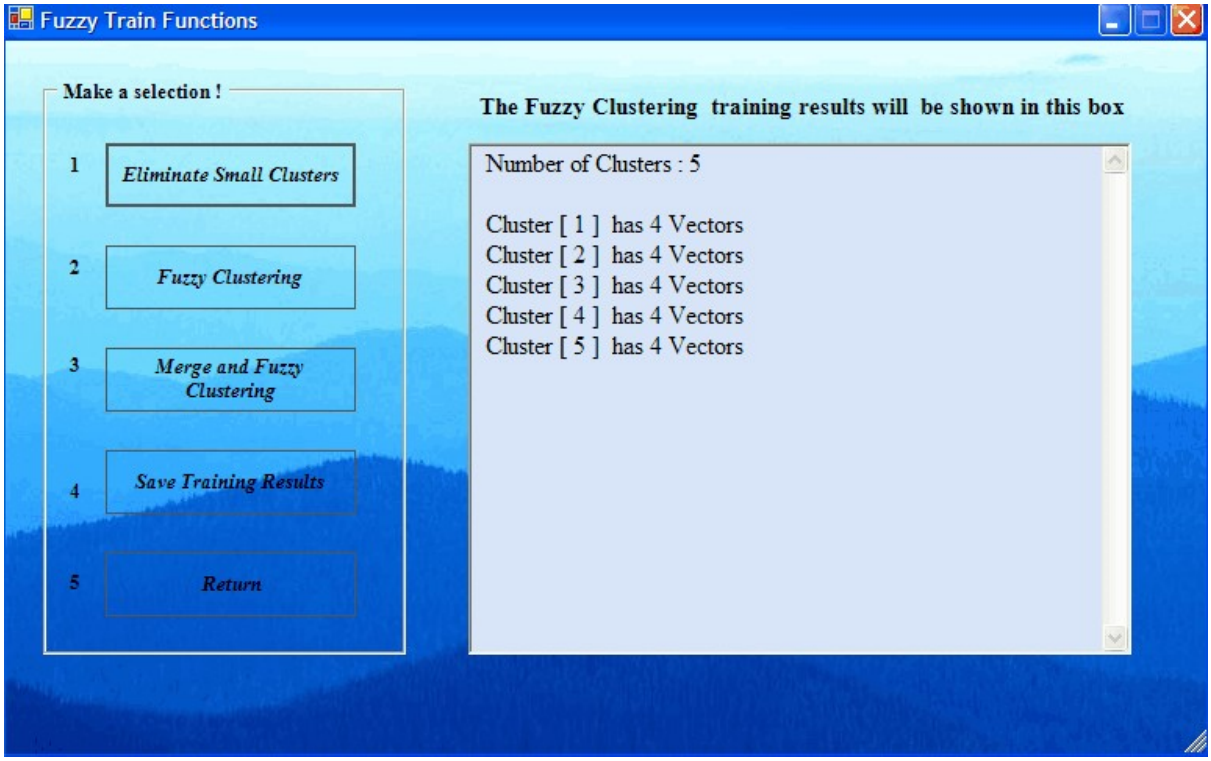


Figure 5. The results of eliminating small clusters.



Now click the ***Fuzzy Clustering*** button (Fig. 5) to get the results in the window of Figure 6. Notice the clustering validity number. The smaller this number, the better the clustering goodness. While it is usually a good way of determining the best number K of classes, it does not always provide the best classification number, so if the we know how many clusters there should be, then we should use that number for K. For example, if a test is either benign or malignant, then the correct number of clusters should be K = 2. However, if some data does not fall into either, then there may be K = 3 classes: benign, malignant and indeterminate. Classification is an intuitive art as well as a science.

Figure 6. The results of fuzzy clustering with clustering validity.

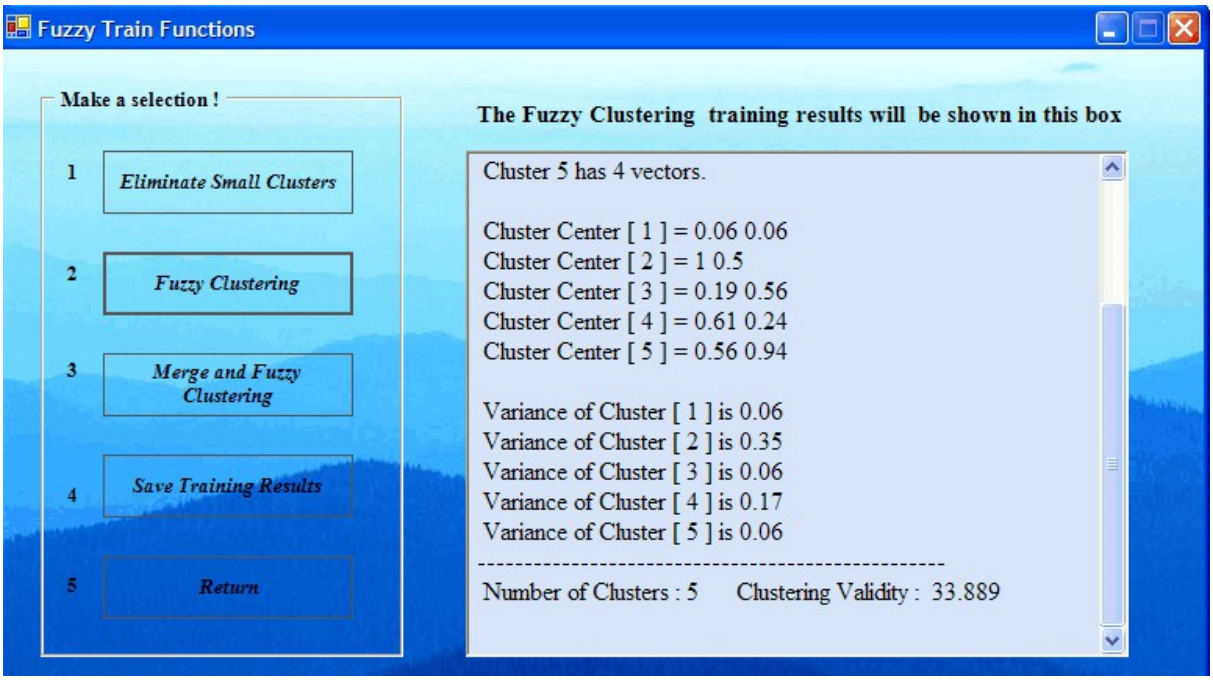


Figure 7. The fuzzy merging parameter.

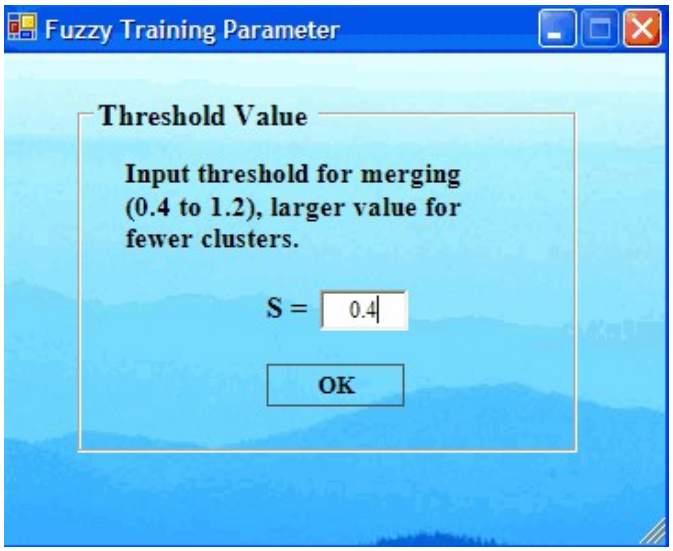


Figure 8. The final results of merging and fuzzy clustering.

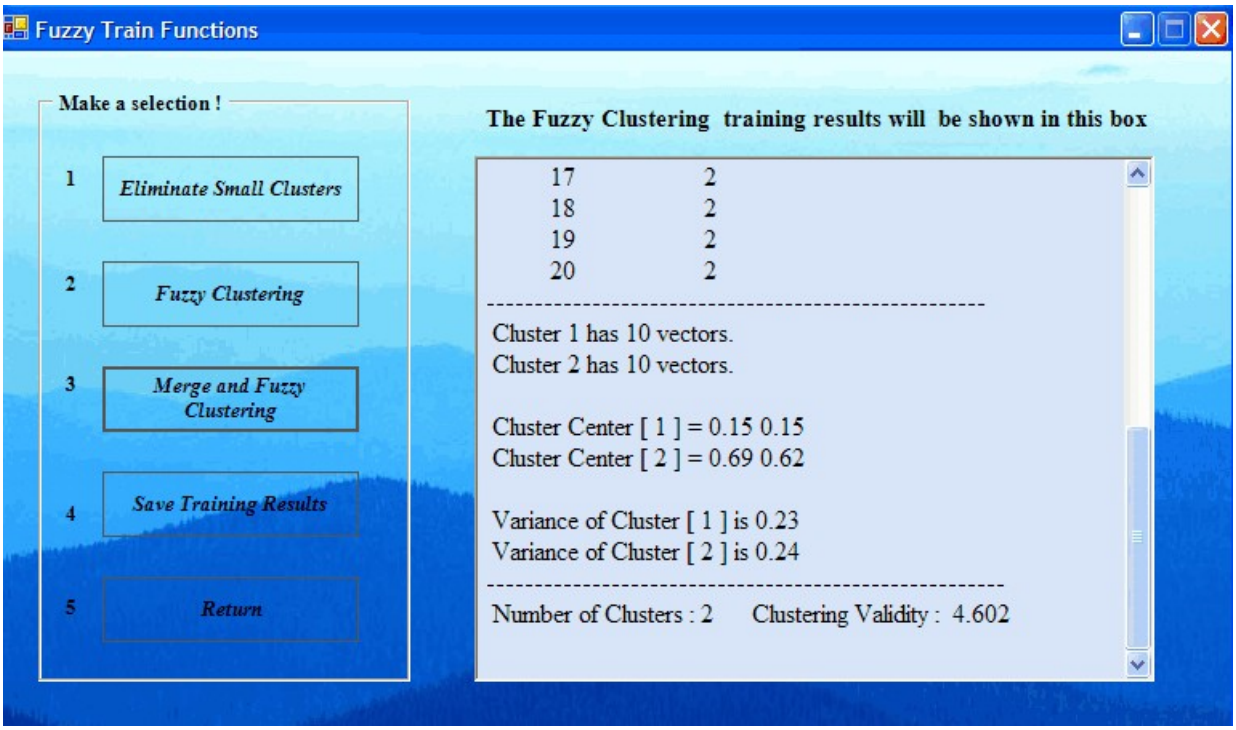


Figure 8 shows the results and the clustering validity is lower than for K = 3 or 4, so 2 is the correct number of clusters for this synthetic data (see Figure 6 where the clustering validity is 33.889 for 5 clusters). After returning and training again with K = 5, we click on the ***Save Training Results*** button in Figure 8 to save the training parameters in the file (**.cen*) whose name we provided before. We use these parameters next when we classify unknown feature data.

2.2 Classifying Unknown Data with the Trained Fuzzy Clustering System. Now we click ***Return*** at the bottom of Figure 8 and then click another ***Return*** to return to the window of Figure 2. This time we click the ***Do Fuzzy Classification*** button to bring up the window shown in Figure 9. Here we first load the file of training parameters that we saved as a **.cen* file. Then we load in the data file (**.dta*) of unknown feature vectors to be recognized as belonging to a particular class. The last file we must select is the file for saving the results of the classification. This file must have the name of the form **.fuz* and can be sent to the printer or examined by an editor or use of *Microsoft Excel* spreadsheet (the input data files can also be viewed or entered by use of *MS Excel*. The overall goodness of classification numbers are also shown in Figure 10.

We click ***Run*** at the bottom of Figure 9 to let the fuzzy clustering system recognize the input feature data by assigning it to the appropriate class. Figure 10 shows the results of this click. The results file (**.fuz*) is shown next in Figure 11 where in each row the first number is the vector number, the next is the winning class number, the third is the fuzzy belief of classification in Class 1, and the fourth is the fuzzy belief of classification in Class 2.

Figure 9. Loading the training parameters and the data for classification.

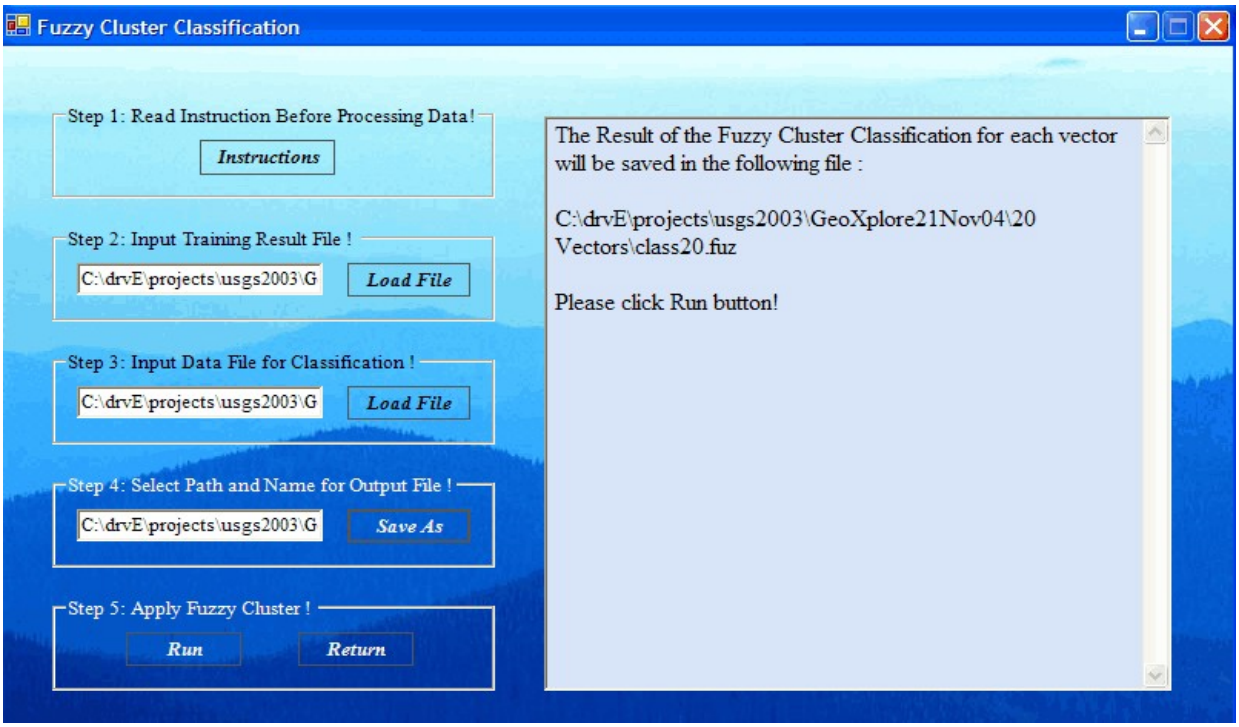
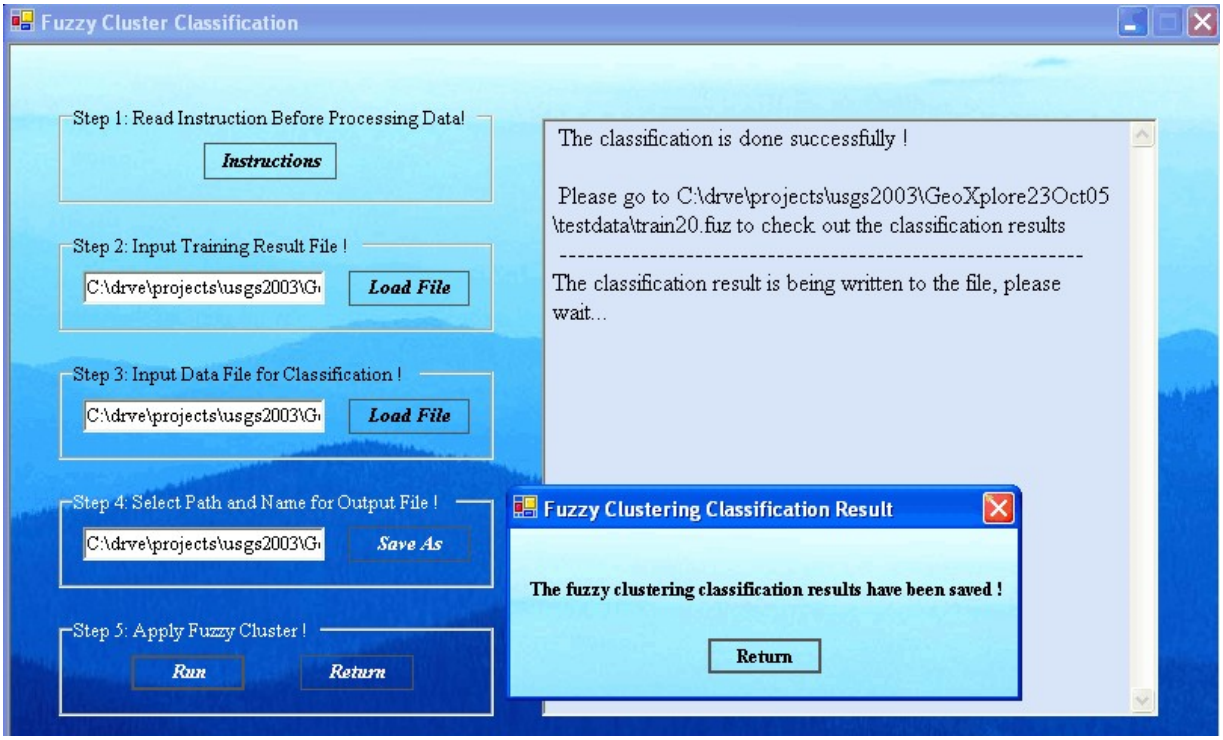


Figure 10. The results of running the fuzzy clustering classification function.



The classification results of fuzzy clustering (in a file with the **.fuz* name) are shown below.

```
1,1,0.527,0.171
2,1,0.657,0.208
3,2,0.594,0.727
4,2,0.703,0.820
5,1,0.980,0.363
6,1,0.991,0.324
7,2,0.451,0.928
8,2,0.457,0.828
9,1,0.744,0.464
10,1,0.644,0.327
11,1,0.556,0.262
12,1,0.693,0.320
13,2,0.509,0.817
14,2,0.602,0.922
15,1,0.981,0.516
16,1,0.992,0.461
17,2,0.331,0.825
18,2,0.334,0.736
19,1,0.744,0.464
20,1,0.644,0.327
16,2,0.476,0.964,0.646
17,3,0.087,0.201,0.654
18,3,0.065,0.194,0.732
19,3,0.147,0.572,0.907
20,3,0.085,0.425,0.811
```

Now we click ***Return*** in Figure 10 and click ***Return*** again as necessary to get back to the window of Figure 2, and click ***Return*** again to go back to the window of Figure 1. From there we can select ***RBFLN*** or ***PNN*** to train and classify with one of those systems.

2.3 When Fuzzy Clustering is a Good Method. When the classes form groups that fit inside circles of a fixed radius, then fuzzy clustering is an excellent method that is somewhat robust by being immune to outliers.

However, in some cases the classes may be elongated in shape or a shape that curves like a crescent or other irregular object. In this case it is better to cluster the feature vectors into more smaller circular clusters of which each will be a subclass of a class of feature vectors. In such cases when there is one class of interest versus all the others but the class of interest does not fit into a circle, then multiple subclasses of the default class are needed - it may be simpler in this case to use the RBFLN or the PNN.

3. The Radial Basis Functional Link Net (RBFLN)

3.1 Training the the RBFLN System. When GeoXplore is run as described in Section 1.1, the window in Figure 1 comes up. We click on the ***RBFLN*** button in that figure to get the window shown in Figure 11.

Figure 11. The first *RBFLN* window.

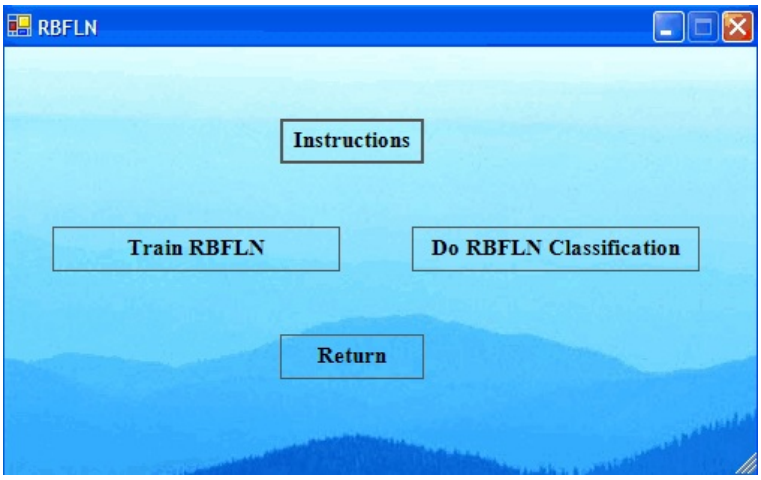
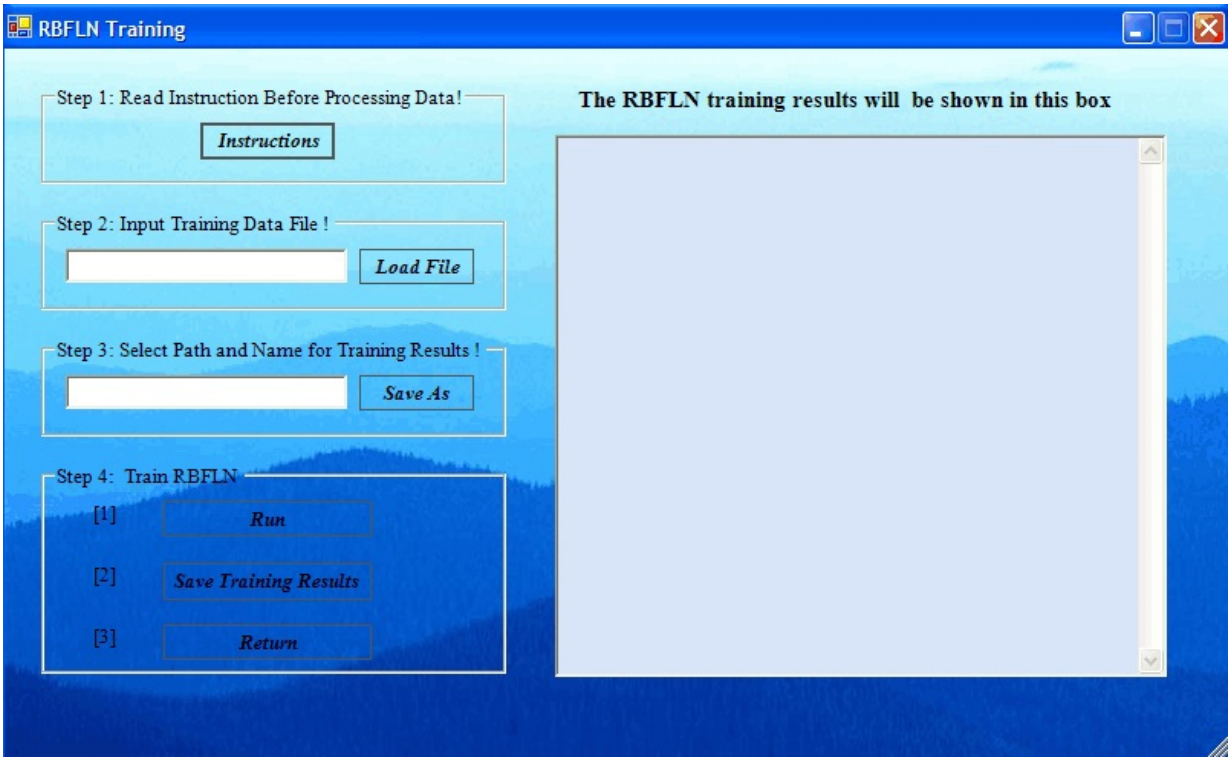


Figure 12. The *Train RBFLN* window.

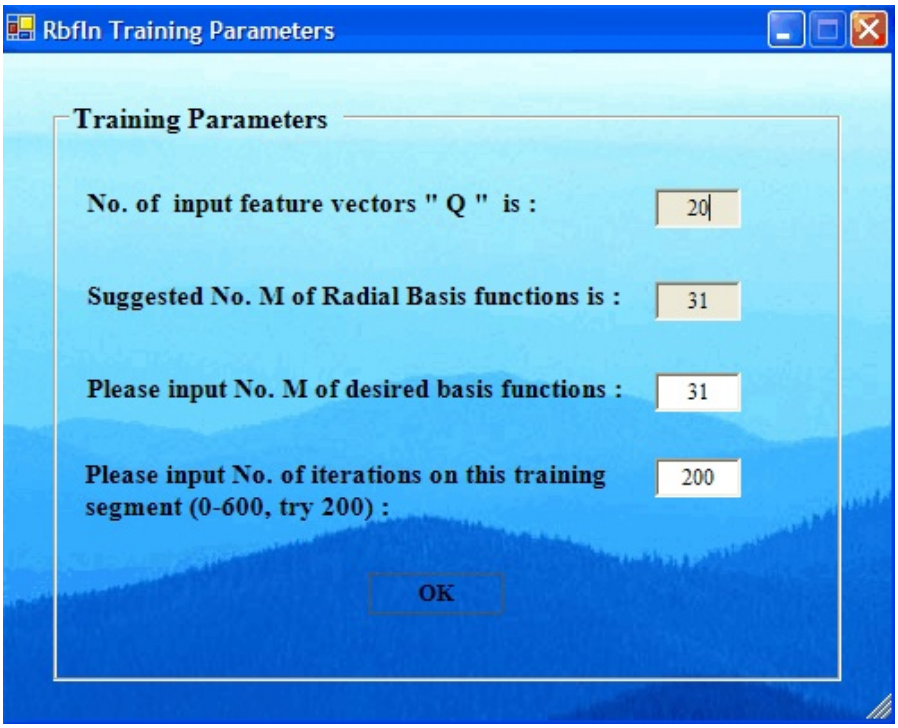


We now click the ***Train RBFLN*** button of Figure 11 to bring up the window in Figure 12. Next, we click the ***Load File*** button and browse to select the path and filename of the training file to use to train this system. A window will come up and allow the selection of a data file for training whose name has the name format **.dta*. Either click twice on the file name or click once on it and then click ***Open*** on the lower right. That file will be read in and closed.

Next, click the ***Save As*** button to provide a filename for the file that will hold the training results, that is, the neural network *parameters* that were learned during training. In this case, this file holds: i) the fuzzy centers and the sigma values for the Gaussian radial basis functions; 2) the weights on the lines from the Gaussians to the output nodes; and 3) the weights on the extra lines from the input nodes to the output nodes that bypass the Gaussians at the middle layer of neurodes. These files have the name format **.par* to indicate that the parameters for the trained network are stored here.

We are now ready to train, so click the ***Run*** button in the bottom section of the window of Figure 12. The window of Figure 13 comes up with the run variable values shown. The top two parameters are for the users information and can not be changed. The third value can be changed, but it should not vary too much from the suggested value given in the second value. For a larger number M of radial basis functions, the neural network will slower but more accurate up to a point. However, an M value that is too large will have extraneous error building up. For important training, the user may need to experiment by using a smaller M and a larger M and noting the overall measures of training goodness, where a higher belief between 0 and 1 is better (1.0 is perfect).

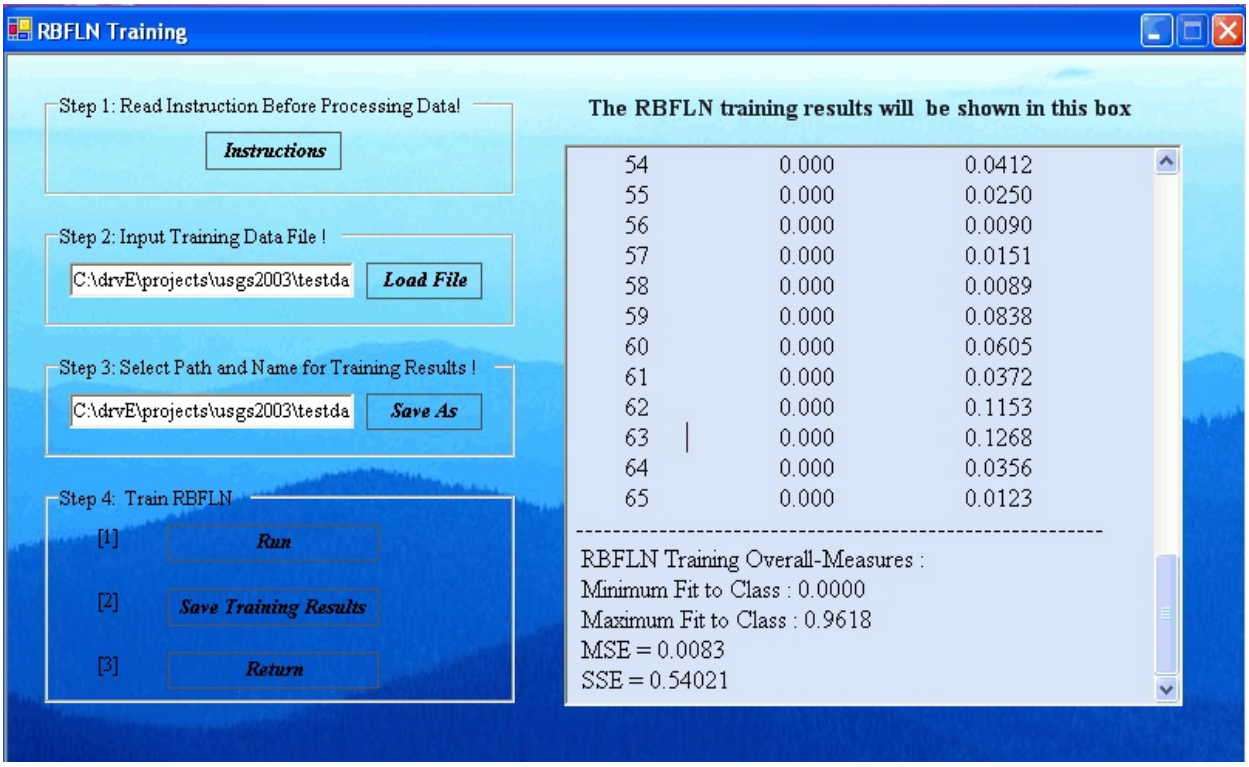
Figure 13. The run parameter window for RBFLN.



The number of iterations also affects the quality of training. If the training data set is small, then 100 iterations may be sufficient. The default of 200 is a good trade-off between under training and over training (where the noise is learned), but for larger training data sets 400 to 600 iterations are better. The user can use, say 100, then 200 and then 400 and examine the results.

Click the ***OK*** button in Figure 13 next for the training with the number of iterations and the number of radial basis functions selected. This will run the training and bring up the results shown in Figure 14. The minimum and maximum fit to class show the fits for training vectors not in the class and in the class, respectively. The mean-square and sum-square errors are also given of the fit to the default and other classes.

Figure 14. The results of training the RBFLN.

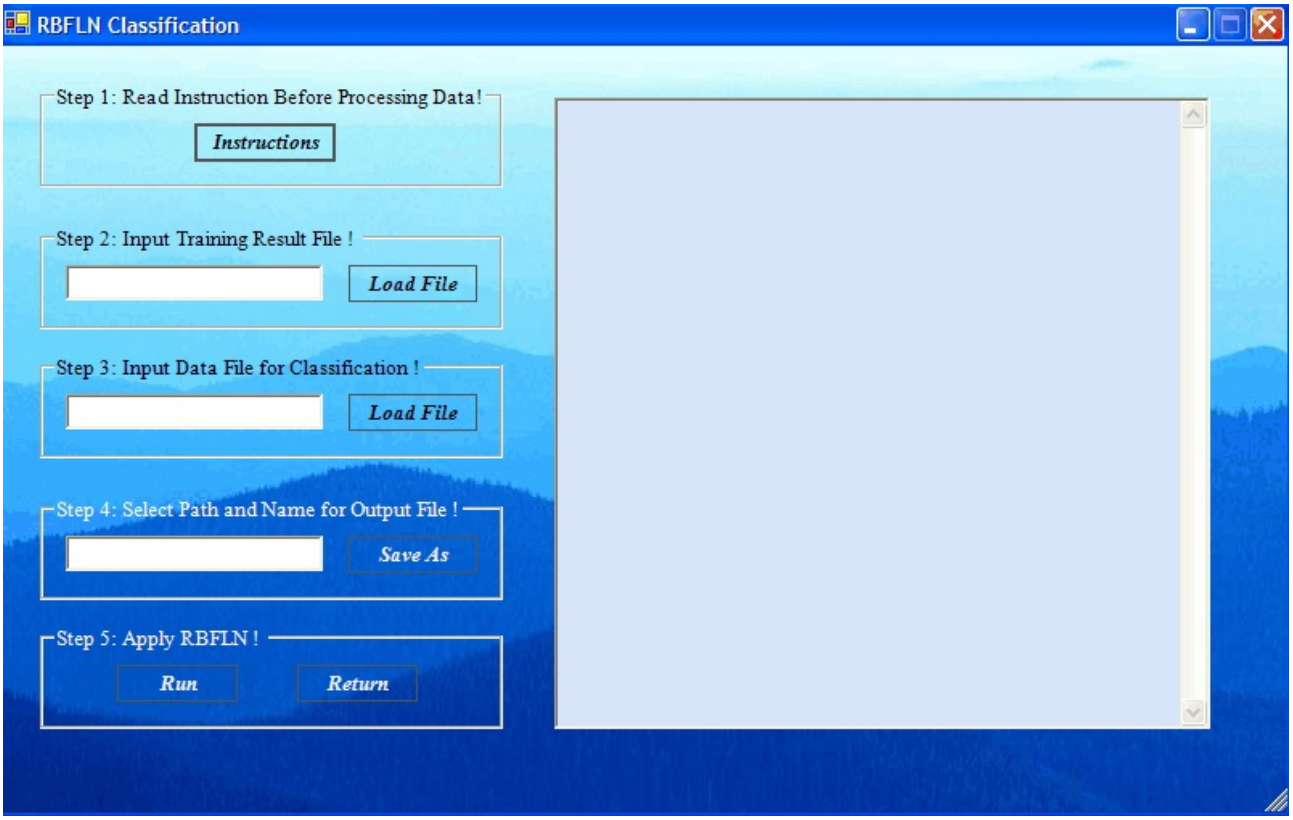


The training may be repeated multiple times with different run parameters (Figure 13) until the overall measures of goodness of training are near optimal. At this point the user should save the training parameters by clicking on the ***Save Training Results*** button in Figure 14. At this point, the user should click the ***Return*** buttons to go back to the window shown in Figure 11. Unknown feature vector data from similar sources can now be classified by use of the the trained RBFLN.

2.2 Classifying Unknown Data with the Trained RBFLN System. To start the classifying process, click the ***Do RBFLN Classification*** button of Figure 11. This will bring up the RBFLN classification window as shown in Figure 15. Now we click the top ***Load File*** button and select the file of parameters that was saved during the training, which has a name of the form **.par*. Next we click

the lower ***Load File*** button and select the file of data to be classified whose name has the form **.dta*. Now we must provide a file name for the file where the results of the classification will be stored in a file with name format **.rbn*, so click the ***Save As*** button and provide a file name. Then click the ***Run*** button.

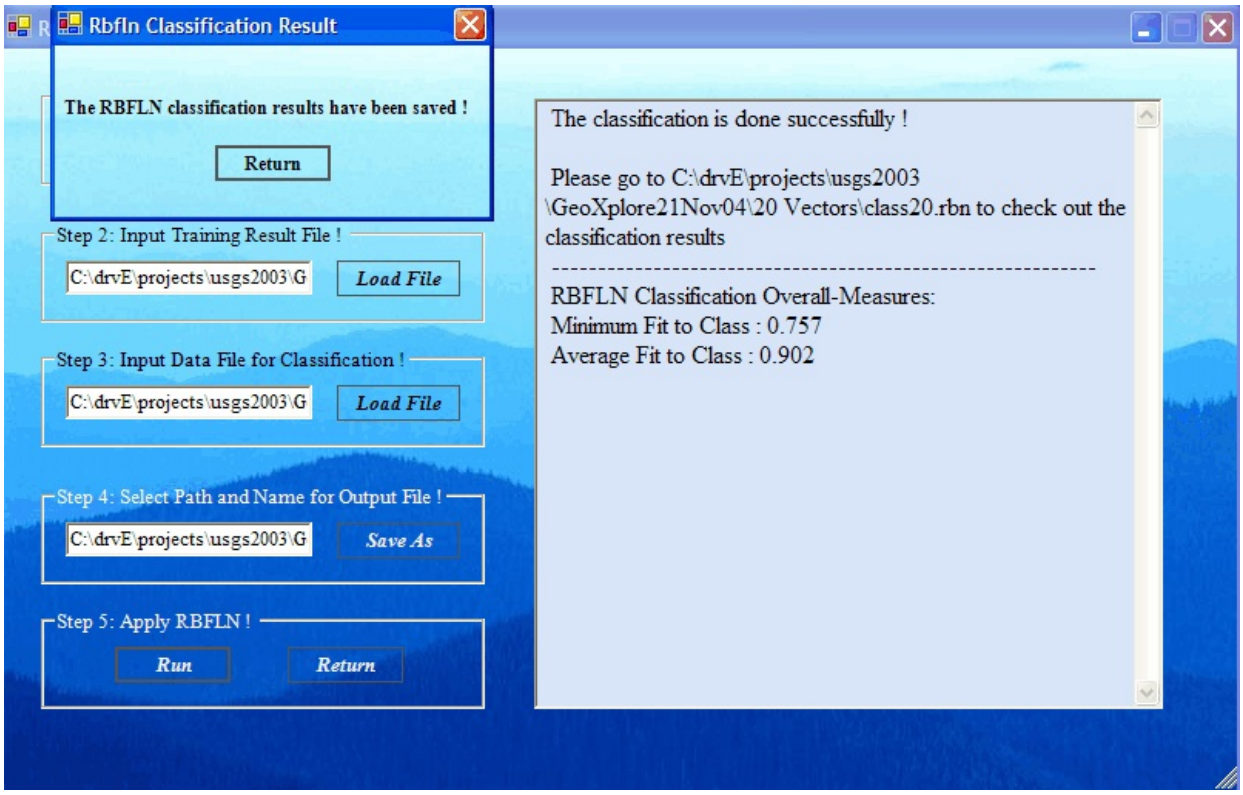
Figure 15. Loading the training parameters and data for RBFLN classification.



The RBFLN network executes with the loaded training parameters and processes the classification input data into a class for each input unknown feature vector. The results are written to the file with name format **.rbn* selected above and can be viewed or printed out with an editor program. The window now appears like the window shown in Figure 16.

The goodness of fit (classification) numbers are shown at the bottom of the textbox in the form of a minimum goodness of fit and an average goodness of fit, both of which are fuzzy beliefs in the classification of this data with the training parameters loaded previously. The classification results file will have the same format as in Section 1.5.

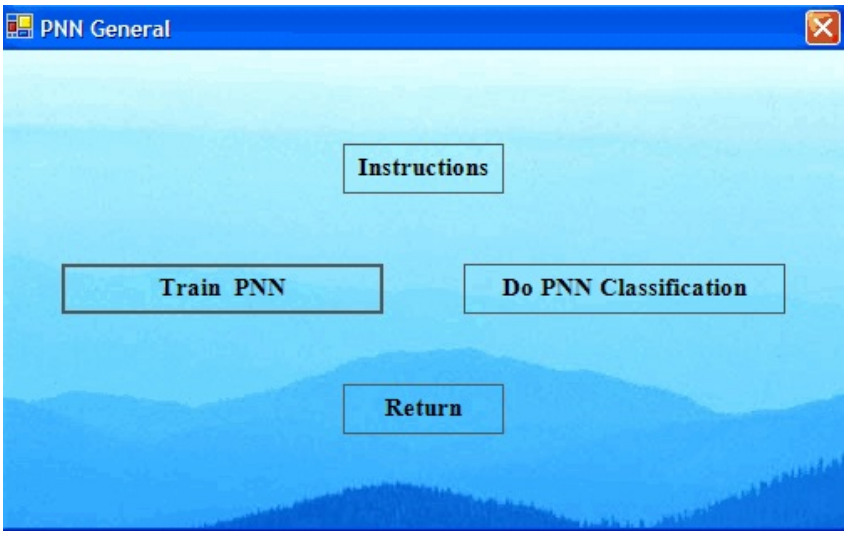
Figure 16. The results of running the RBFLN classification function.



4. The Probabilistic Neural Network (PNN)

4.1 Training the the PNN System. When GeoXplore is run as described in Section 1.1, the window in Figure 1 comes up. We click on the ***Probabilistic NN*** button in that figure to obtain the window shown in Figure 17 below that is similar to the window for the RBFLN shown in Figure 11.

Figure 17. The first *PNN* window.



We now click the ***Train PNN*** button to bring up the window in Figure 18. Next, we click the ***Load File*** button and select the path and filename of the training file to use to train this system. A window will come up and allow browsing for the selection of a data file for training. The file name usually has the format **.dta* . Either click twice on the file name or click once on it and then click ***Open*** on the lower right of the browsing window. That file will be read in and closed.

Next, click the ***Save As*** button to provide a filename for the file that will hold the training parameters, that is, the parameters that were learned during training. In this case, this file holds the fuzzy centers and the sigma values for the mixed Gaussian probability density functions. These files have the name format **.prb* to indicate that the parameters for the PNN network are stored here.

We are now ready to train, so click the ***Run*** button at the bottom of the window. The window of Figure 19 will then come up with a proportion parameter as shown. This parameter is the proportion of the average distance between training feature vectors to use as a threshold. If any vector is closer to another than this threshold, it is eliminated as a center of Gaussians to be summed to construct a probability density function (*pdf*) for a class. A larger threshold eliminates more vectors and increases computational speed, but it is a trade-off between speed and accuracy up to a point. A small proportion for a large number of training vectors uses too many Gaussians and causes extraneous error to build up during computation of the *pdfs*. The user should experiment with this. The choices suggested are quite good usually.

Figure 18. The *Train PNN* window.

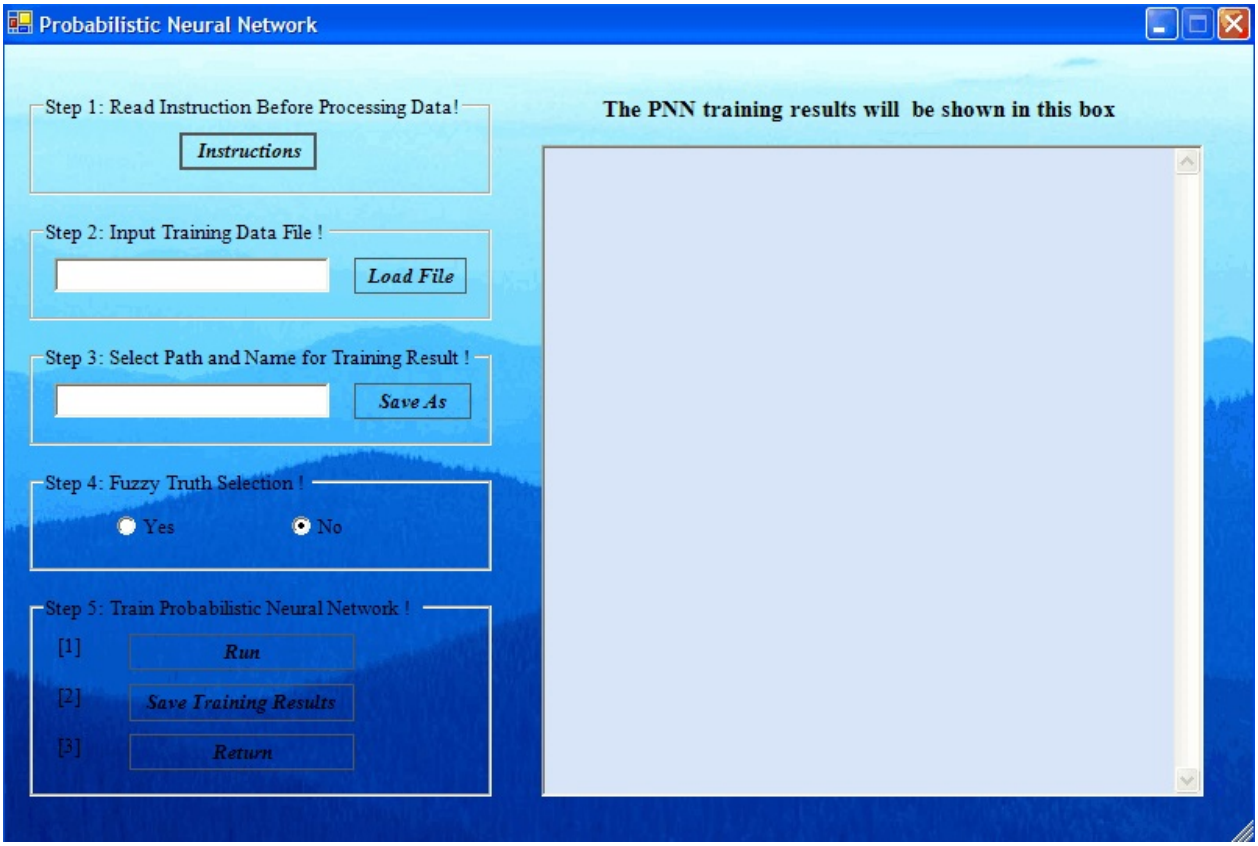
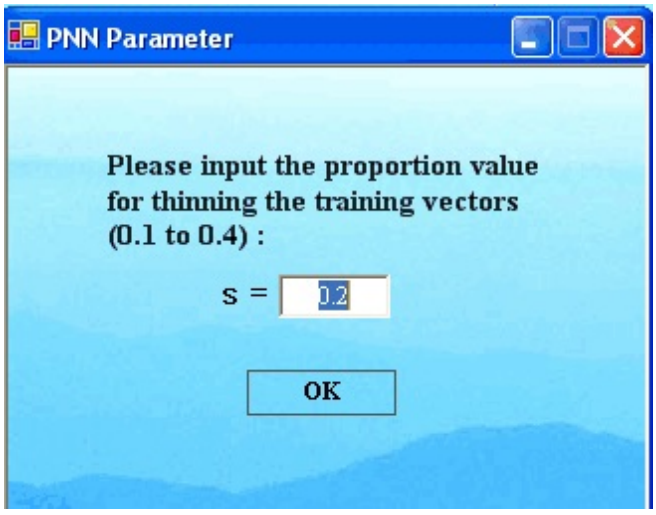
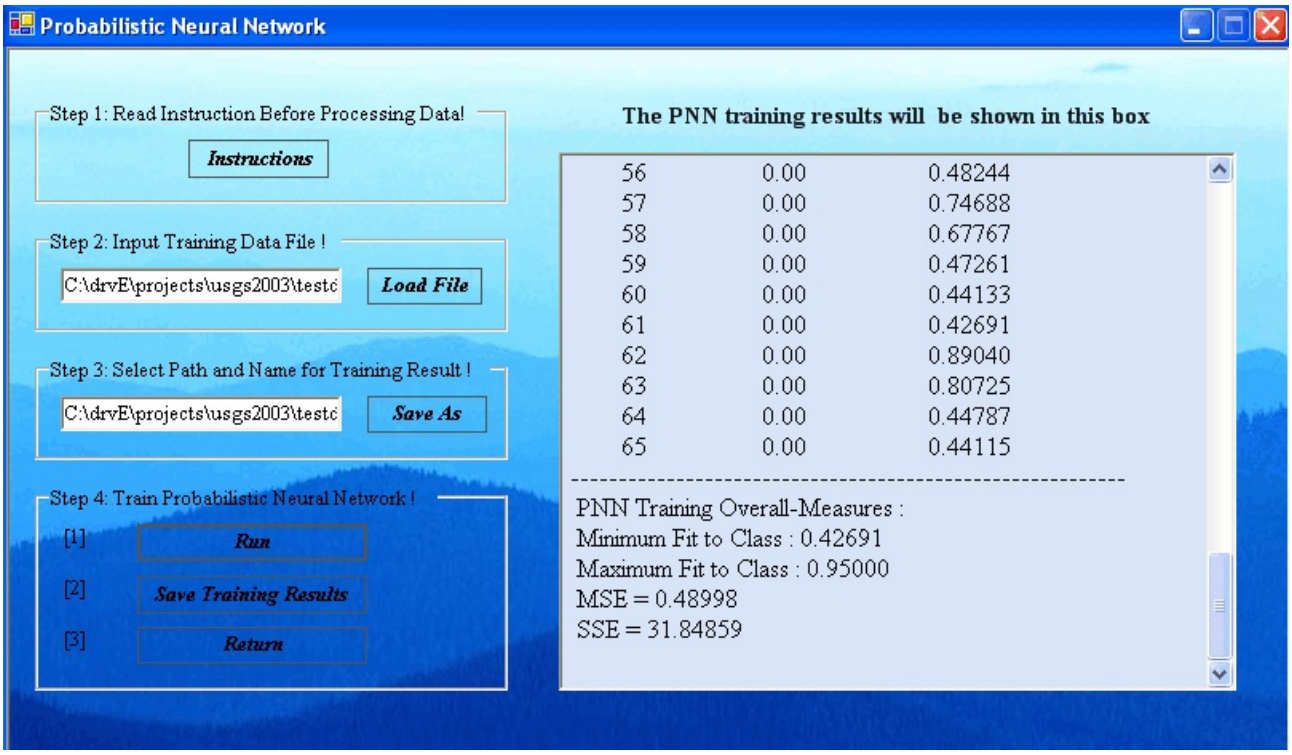


Figure 19. The run parameter window for PNN.



Click the ***OK*** button of Figure 19 for the training with the proportion selected. This will run the training and build the *probility density functions* as mixed sums of Gaussians. The results are shown in Figure 20. The overall goodness of training measures are at the bottom of the textbox and are: 1) mininum fit to the default (desired) class; 2) maximum fit to the default class; and 3) the mean-square and sum-squared errors. Each fit value is a maximum aposteriori pdf value (that also serves as a fuzzy truth of membership of a vector in the default class).

Figure 20. The results of training the *PNN*.

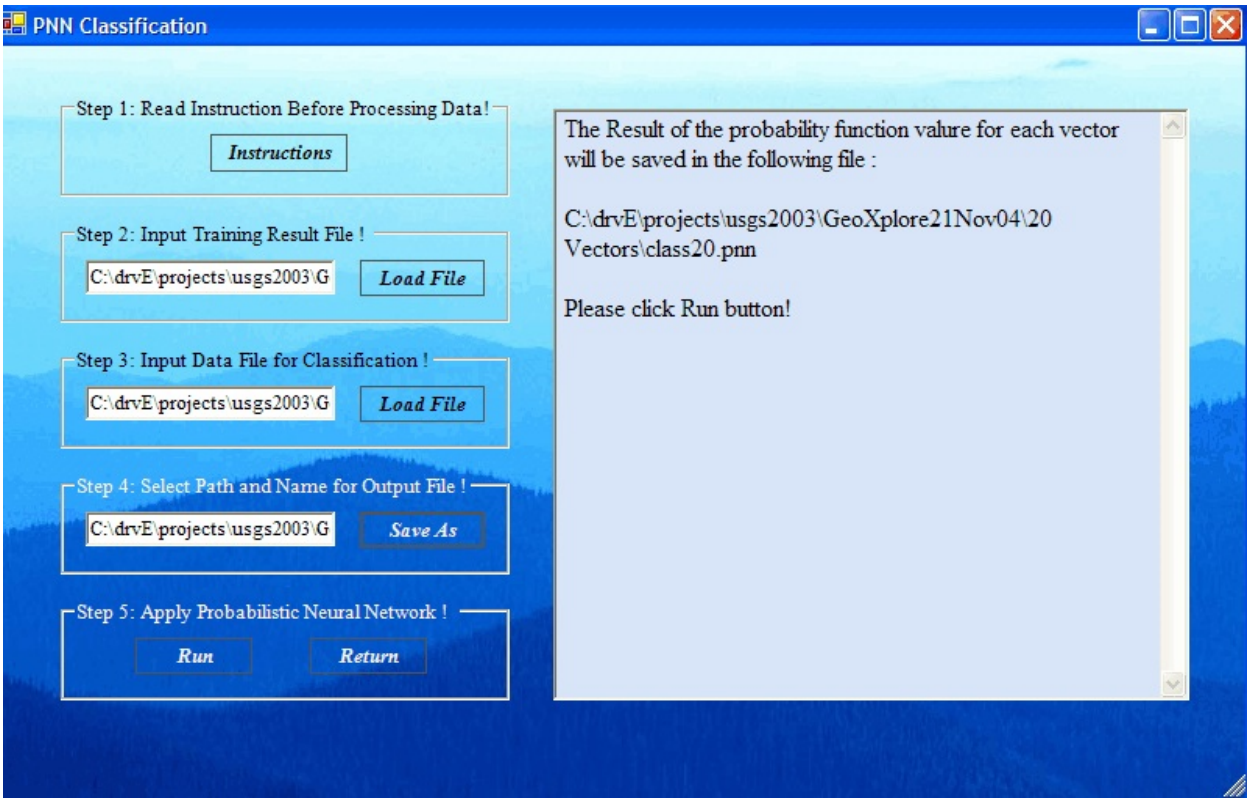


The training may be repeated multiple times with different proportions until the overall measures of goodness of training are near optimal. These measures are shown at the bottom of the textbox displayed in Figure 20. When the user is ready to accept this training, then the training parameters can be saved by clicking on the ***Save Training Results*** button. Click the ***Return*** buttons of Figure 20 and 18 to go back to the window shown in Figure 17. Unknown feature vector data from similar sources can now be classified by use of the the trained PNN.

4.2 Classifying Unknown Data with the Trained PNN System. To start the classifying process, click the return buttons to get back to Figure 17 and then click the ***Do PNN Classification*** button of Figure 17. This will bring up the PNN classification window as shown in Figure 21. Now we click the top ***Load File*** button and select the file of parameters that was saved during the training, which has a name of the form **.prb*. Next we click the lower ***Load File*** button and select the file of data to be classified whose name has the form **.dta*. Now we must provide a file name for the file where

the results of the classification will be stored in a file with name format **.pnn*, so click the ***Save As*** button and provide a path and file name. Then click the ***Run*** button.

Figure 21. Loading the training parameters and data for PNN classification.



The PNN network executes on the loaded training parameters and processes the data input for classification into a class for each input unknown feature vector. The results are written to the file with name format **.pnn* selected above and can be viewed or printed out with an editor program. The window now appears like the window shown in Figure 22.

The goodness of classification numbers are shown at the bottom of the textbox in Fiugre 22 in the form of a minimum goodness of fit and an average goodness of fit to the default class. These should be close to zero (failure to belong to the default class) and close to unity (strong membership in the default class). The classification results file have the same format as shown in Section 1.5.

Figure 22. The results of running the PNN classification function.