The ECoS Toolbox User Guide
Version 1.24

Michael J. Watts
mike.watts.net.nz

March 8, 2016

Contents

Chapter 1

Introduction

This document gives an overview of the use of the tools in the ECoS toolbox. The ECoS toolbox provides a collection of Windows command line applications that implement and support selected algorithms from the Evolving Connectionist System (ECoS) family. These selected algorithms include the Evolving Fuzzy Neural Network (EFuNN, Chapter 3), the Simple Evolving Connectionist Systems (SECoS, Chapter 4) and the Evolving Clustering Method (ECM, Chapter 5). There are also tools for formatting and manipulating data (Chapter 2) and for working with fuzzy membership functions (MF) and rules (Chapter 7). The tools present in this toolbox were selected to support data modelling and data mining via fuzzy rule discovery. For a comprehensive over-view of the ECoS algorithms implemented in this toolbox, please see the following paper:

Watts, M.J. A Decade of Kasabov's Evolving Connectionist Systems: A Review. IEEE Transactions on Systems, Man and Cybernetics Part C - Applications and Reviews (2009) 39(3) 253-269.

The design philosophy of this toolbox is that each tool should do one thing well. Therefore, each tool will perform one and only one function. As command line applications, it is possible to string these tools together in batch files or using scripting languages such as Python, to perform complex operations.

Each application in this toolbox accept parameters in one of two ways, either from command line arguments (for those applications that have only a few parameters) or from parameter files (for those applications that have more than a few parameters). Parameter files are text files that have tag and value pairs, for example:

```
Inputs  4
InputMF  {3,3,3,3}
Rules  1
OutputMF  {2,2,2}
Outputs  3
FileName  "iris-EFuNN.wgt"
```

This example is for the CreateEFuNN tool (Section 3.2). In this example, `Inputs`, `InputMF`, `Rules`, `OutputMF`, `Outputs`, and `FileName` are all tags. Tags

are case sensitive and must be unique within a parameter file (if there are two or more copies of a tag in a parameter file, then only the first will be used). Tags can be in any position within the file and do not need to be on separate lines. Arguments must be separated from their tags by whitespace.

The arguments to `Inputs`, `Rule` and `Outputs` are numeric and are not delimited in any way. The argument to `FileName` is a string argument. String arguments are delimited with quotes (`"`). The arguments to the `InputMF` and `OutputMF` tags are arrays. Arrays are delimited by braces (started by `{` and ended with `}`) and entries are separated by commas. Comments can be inserted into parameter files, and are begun and ended by the hash (`#`) character. The escape character is the backslash (\) character.

Examples of each of the parameter files, and each of the data, network and fuzzy membership function files, are included in the `egs` directory.

Chapter 2

Data Processing Tools

2.1 Introduction

The applications in the ECoS toolbox load data from files of a certain format (see Section A.1 for an example). Attempting to load data that is not properly formatted will cause the tools to fail. The tools in this chapter allow the user to move data to and from this format, and to manipulate files that are in this format.

2.2 2dat

The 2dat application converts flat data files to the format required by ECoS tools. The application assumes that the number of input and output variables are the same for each row, and that the numbers supplied as parameters are accurate: The resulting data file will be packed with zeros if these numbers are not accurate. An example of the data file format is shown in Section A.1.

```
2dat SourceFile DestinationFile rows input outputs
```

Where:
`SourceFile` is the flat format file to load the data from.
`DestinationFile` is the formatted file to save the data to.
`rows` is the number of rows in the flat file.
`inputs` is the number of input columns in the flat file.
`outputs` is the number of output columns in the flat file.

2.3 2flat

Converts formatted data files to flat files, that is, files with no formatting.

```
2flat SourceFile DestinationFile [s,c,t]
```

Where:
`SourceFile` is the formatted file.
`DestinationFile` is the flat file to save to.
`[s,c,t]` is an optional argument (without brackets) for a delimiter, where `s` is space (default), `c` is comma and `t` is tab.

2.4 shuffle

The shuffle tool will randomly re-arrange the order of the rows in a data file and write the results to another file. The final order of the rows is optionally written to another file. There are two mandatory and one optional argument to the shuffle application:

```
shuffle SourceFile DestinationFile MapFile
```

Where:
`SourceFile` is the data file to shuffle.
`DestinationFile` is the file to write the shuffled data to.
`MapFile` is an optional argument. If this argument is specified then a second data file will be written, which has the same number of rows as the `SourceFile`. Each row consists of a single value. This value specifies the row number that that row was shifted to during the shuffling process. For example, if the contents of the third row of the map file was 72, then that means that the third row in `SourceFile` is now row 72 in `DestinationFile`. This is useful as map files tend to be much smaller than shuffled data files. Thus, if it ever becomes necessary to re-create the shuffled data file, the map file can be used with the mapshuffle application (Section 2.5) to re-generate the shuffled data.

2.5 mapshuffle

The map shuffle tool will re-order the rows in a data file according to the supplied map file. That is, it will shuffle the rows deterministically. This application has three mandatory arguments.

```
mapshuffle SourceFile MapFile DestinationFile
```

Where:
`SourceFile` is the data file to r-order.
`MapFile` specifies the new order of the rows. There must be one row in this file for each row in `SourceFile`. The contents of each row specify the destination row of that row in `SourceFile`. The map files produced by the shuffle application (Section 2.4) are automatically in the right format.
`DestinationFile` is the file to write the re-ordered data to.

2.6 concat

The concat tool concatenates multiple data files together. There is a variable number of arguments for this tool.

```
concat File1 File2 ... FileN DestinationFile
```

Where:
`File1 ... FileN` are the source files. Each file must have the same number of inputs and outputs.
`DestinationFile` is the file to save the concatenated data to.

2.7 hconcat

This application is similar to the concat tool (Section 2.6) but instead of concatenating files to the end of one another, it will concatenate files horizontally. Input columns are concatenated with input columns, and outputs with outputs. Thus, given two data files with two input columns and one output column, the output of this tool will be one file with four input columns and two output columns.

As with the concat application, the number of arguments is variable:

```
hconcat File1 File2 ... FileN DestinationFile
```

Where:
`File1 ... FileN` are the source files. Each file must have the same number of rows.
`DestinationFile` is the file to save the concatenated data to.

2.8 getrows

This tool is used to extract specific rows from a data file. The argument to this application is a parameter file that must be of the following format:

```
SourceFile "iris.trn"
DestinationFile "iris-sub.dat"
Rows {0,10,20}
```

Where:
`SourceFile` is a string argument specifying the file to load the source data from.
`DestinationFile` is a string argument specifying the file to save the extracted data rows to.
`Rows` is an array argument that specifies which rows to extract. These rows are numbered starting from zero. In this example, the first, eleventh and twenty-first rows will be extracted.

2.9 arff2dat

Converts an ARFF format file as used by tools like WEKA to the DAT format required by the ECoS tools. It assumes that the input attributes are NUMERIC or REAL type, it doesn't yet handle encoding of nominal attributes.

```
arff2dat ARFF.file dat.file
```

Where:
`ARFF.file` is the ARFF format file to convert.
`dat.file` is the formatted file to save the data to.

2.10 dat2arff

This tool converts DAT-format files to the ARFF format used by tools like WEKA. The argument to this application is a parameter file that must be of the following format:

```
SourceFile "iris.trn"
DestinationFile "iris.arff"
RelationName "Iris"
InputNames {sepallength,sepalwidth,petallength,petalwidth}
OutputMasks {1:0:0,0:1:0,0:0:1}
OutputTags {Iris-setosa,Iris-versicolor,Iris-virginica}
```

Where:
`SourceFile` is a string argument specifying the DAT file to convert.
`DestinationFile` is a string argument specifying the ARFF file name.
`RelationName` is a string argument that contains the contents of the @RELATION tag in the ARFF file.
`InputNames` is an array of strings that specify the names of the input variables.
`OutputMasks` is an array of strings that specify the output vectors that describe each class. There must be as many masks as there are output variables in the DAT file. Elements in the mask are delimited by colons. There must be as many elements in each mask as there are output variables in the DAT file.
`OutputTags` is an array of strings that specify the name of each class. There must be as many elements in this array as there are output variables in the DAT file. Output tags will be selected by matching output vectors with the output masks. If no match is found an error will occur.

2.11 accuracy

The accuracy tool will calculate several percentage classification accuracies. It will load two data files and compare the output values between them. The arguments for this tool are:

```
accuracy Template.dat Actual.dat Threshold
```

Where:
`Template.dat` is the data file that contains the target output values.
`Actual.dat` is the output values produced by a classification algorithm.
`Threshold` is a floating-point value. Any value above the Threshold value will

be considered to be a positive example. Any value below the Threshold value will be considered to be a negative example.

Accuracies will be calculated for each output variable in the data files. The number of outputs in the Template must match the number of outputs in the Actual. Accuracies calculated are:

- Number of true positives (TruePositives)
- Number of true negatives (TrueNegatives)
- Number of false positives (FalsePositives)
- Number of false negatives (FalseNegatives)
- True positive percentage (TruePositivePct)
- False positive percentage (FalsePositivePct)
- True negative percentage (TrueNegativePct)
- False negative percentage (FalseNegativePct)
- Overall percentage (OverallPct)
- Sensitivity percentage (SensitivityPct)
- Specificity percentage (SpecificityPct)

Chapter 3

Evolving Fuzzy Neural Networks Tools

3.1 Introduction

EFuNN was the first ECoS network. It is a five neuron layer feed forward network, where each layer performs a specific function. The first neuron layer is the input layer. The second layer is the condition layer. Each neuron in this layer represents a single triangular fuzzy membership function (MF) attached to a particular input, and performs fuzzification of the input values based on that MF. This layer is not fully connected to the input layer, as each condition neuron is connected to a single input neuron, that is, each input neuron is connected to its own subset of condition neurons. The weight of the connection between the condition neuron and its input defines the centre of the condition neuron's MF, where the lower and upper bounds of the MF are defined as the centres of the neighbouring MF. The third layer of neurons is the evolving layer, which is also referred to as the rule layer. The distance measure used in this layer is the distance between the fuzzified input vector and the weight vector. The fourth layer of neurons is the action layer: neurons in this layer represent fuzzy membership functions attached to the output neurons. This layer is similar to the input layer, in that each action neuron is connected only to the output neuron with which its membership function is associated. Also, the value of the connection weight connecting the action neuron to its output defines the centre of the action neuron's membership function. The activation function of the action layer neurons is a simple saturated linear function. The final neuron layer is the output layer. This calculates crisp output values from the fuzzy output values produced by the action layer neurons. The output layer performs centre of gravity defuzzification over the action layer activations to produce a crisp output.

Figure 3.1 shows an idealised EFuNN with three input neurons. Two MF are attached to the first input neuron, three to the second, and two to the third. There are three rule neurons and two outputs, with two MF attached to each output.

The data files loaded by the EFuNN tools must be in the data file format produced by the tools in Chapter 2. The input and output data itself must be

Figure 3.1: EFuNN architecture

within the range of zero to unity.

3.2 CreateEFuNN

The `CreateEFuNN` tool will create a new EFuNN structure and write it to a file. An example of this file is shown in Section A.2. The argument to this tool is a parameter file of the following format:

```
Inputs  4
InputMF  {3,3,3,3}
Rules  1
OutputMF  {2,2,2}
Outputs  3
FileName  "iris-EFuNN.wgt"
```

Where:
`Inputs` is the number of input neurons.
`InputMF` is an array that specifies how many MF to attach to each input neuron. The number of elements in this array must match the number of inputs.
`Rules` is the number of neurons to put in the rule or evolving layer.
`OutputMF` is an array that specifies how many MF to attach to each output neuron. The number of elements in this array must match the number of outputs.
`Outputs` is the number of output neurons.
`FileName` is a string specifying the path to write the EFuNN file to.

3.3 TrainEFuNN

The TrainEFuNN tool will load an EFuNN and data from file, perform ECoS training over the EFuNN, then write the resulting network to a new file. A training log file (see Section A.5) will also be written to file. The parameter file format for this tool is as follows:

```
WeightFile  "iris-EFuNN.wgt"
DestinationFile  "iris-EFuNN-trn.wgt"
TrainingFile  "iris.trn"
Epochs 1
RecallMethod  "OneOfN"
ActivationThreshold 0.5
SensitivityThreshold 0.5
ErrorThreshold 0.01
LearningRateOne 0.5
LearningRateTwo 0.5
LogFile "iris.log"
RuleClustering  "Linear"
MaxRuleNodes 1000
```

Where:
`WeightFile` is the EFuNN to train.
`DestinationFile` is the file to write the trained EFuNN to.
`TrainingFile` is a string argument specifying the data file to use to train the EFuNN.
`Epochs` is the number of times the training data set will be presented to the network. One epoch is usually enough.
`RecallMethod` is a string argument specifying which method to use to propagate activations from the evolving layer. This must be either "`OneOfN`" or "`ManyOfN`".
`ActivationThreshold` is used only with the "`ManyOfN`" recall method. Only evolving layer neurons with activation values greater than this will have their activations propagated. This parameter will be ignored if `OneOfN` recall is used.
`SensitivityThreshold` is the sensitivity threshold training parameter.
`ErrorThreshold` is the error threshold training parameter.
`LearningRateOne` is the learning rate one parameter.
`LearningRateTwo` is the learning rate two parameter.
`LogFile` is a string parameter specifying the file to write the training log to. If this parameter is omitted, the training log will be written to standard output. An example training log file is presented in Section A.5.
`RuleClustering` is a string argument that determines where in the evolving layer new neurons will be inserted. Options are "`Linear`", whereby new neurons are added to the end of the evolving layer, and "`MaxActivation`", where the new neurons will be inserted next to the winning evolving layer neuron.
`MaxRuleNodes` is the maximum number of neurons allowed in the evolving layer.

3.4 RecallEFuNN

The RecallEFuNN tool will load an EFuNN and data file, perform a recall-only operation over the EFuNN using the data, and write the output values only to a new file. It takes a single argument, which is the name of a parameter file and which must be of the following format:

```
WeightFile "iris-EFuNN-trn.wgt"
RecallFile "iris.trn"
OutputFile "iris-EFuNN.out"
RecallMethod "OneOfN"
```

```
ActivationThreshold 1.0
```

Where:
`WeightFile` is a string argument specifying the file name of the EFuNN to recall.
`RecallFile` is a string argument specifying the file name of the data to use to recall the EFuNN.
`OutputFile` is a string argument specifying the file to write the output data to.
`RecallMethod` is a string parameter specifying the recall method of the EFuNN. Must be either "OneOfN" or "ManyOfN".
`ActivationThreshold` is the activation threshold parameter for ManyOfN recall. Ignored if OneOfN is used.

3.5 MakEFuNNLbl

The MakEFuNNLbl tool will extract labels from an EFuNN for each input and output variable, and for each MF attached to those variables. These labels can be edited so that they are more meaningful for the problem at hand, that is, they can be changed from the defaults to reflect the actual variable and MF names. These labels are of the same format as those produced by the MakMFLbl application (Section 7.2). An example label file is presented in Section A.6, where the default labels have been retained. There are two mandatory arguments for this application, as follows:

```
MakEFuNNLbl EFuNNFile LabelFile
```

Where:
`EFuNNFile` is the EFuNN to extract labels from.
`LabelFile` is the file to write the labels to.

3.6 REEFuNN

The REEFuNN tool is used to extract Zadeh-Mamdani fuzzy rules from a trained EFuNN. An example of the kind of rule file generated from an EFuNN is presented in Section A.8. There are three mandatory arguments to this tool, as follows:

```
REEFuNN EFuNNFile LabelFile RuleFile
```

Where:
`EFuNNFile` is the EFuNN file to extract rules from.
`LabelFile` is the label file that assigns linguistic labels to the input and output variables, and the input and output membership functions. This can be the default labels generated by the MakEFuNNLbl tool (Section 3.5).
`RuleFile` is the file to write the fuzzy rules to.

3.7 RIEFuNN

This application will create an EFuNN from Zadeh-Mamdani fuzzy rules. An example of the kind of rule file generated from an EFuNN is presented in Section A.8. There are two mandatory arguments to this tool, as follows:

```
RIEFuNN RuleFile EFuNNFile
```

Where:
`RuleFile` is the fuzzy rule file to construct the EFuNN from.
`EFuNNFile` is the file to write the new EFuNN to.

Chapter 4

Simple Evolving Connectionist Systems Tools

4.1 Introduction

The Simple Evolving Connectionist System (SECoS) was proposed as a minimalist implementation of the ECoS algorithm, that is, it is an architecture that has the minimum number of neuron layers necessary to learn data. Alternatively, SECoS can be viewed as a minimalist EFuNN, with the fuzzification and defuzzification components being removed. Lacking the fuzzification and defuzzification mechanisms of EFuNN, the SECoS model was created for several reasons. Firstly, they are intended as a simpler alternative to EFuNN. Since they lack the fuzzification and defuzzification structures of EFuNN, SECoS are much simpler to implement. Having fewer connection matrices and a smaller number of neurons, there is much less processing involved in simulating a SECoS network. They are also much easier to understand and analyse: while EFuNN expands the dimensionality of the input and output spaces with its fuzzy logic elements, SECoS deals with the input and output space 'as is'. Therefore, rather than dealing with a fuzzy problem space, SECoS deals with the problem space directly. Each neuron that is added to the network during training represents a point in the problem space, rather than a point in the expanded fuzzy problem space. Secondly, for some situations, fuzzified inputs are not only unnecessary but harmful to performance, as they lead to an increase in the dimensionality of the input space and hence an increase in the number of evolving layer neurons. Binary data sets are particularly vulnerable to this, as fuzzification does nothing but increase the dimensionality of the input data. By removing the fuzzification and defuzzification capabilities, the adaptation advantages of EFuNN are retained while eliminating the disadvantages of fuzzification, specifically by eliminating the need to select the number and parameters of the input and output membership functions. For most applications, SECoS are able to model the training data with fewer neurons in the evolving layer than an equivalent EFuNN. A SECoS network consists of three layers of neurons: The input

layer, with linear transfer functions; The evolving layer; And an output layer with a simple saturated linear activation function. The distance measure used in the evolving layer is the normalised Manhattan distance.

As with the EFuNN tools, the data files loaded by the SECoS tools must be in the data file format produced by the tools in Chapter 2. The SECoS tools also expect the input and output data to be within the range of zero to unity.

4.2 CreateSECoS

The `CreateSECoS` tool will create a new SECoS structure and write it to a file. An example of this file format is in Section A.3. The argument to this tool is a parameter file that must be of the following format:

```
Inputs 4
Hidden 1
Outputs 3
MinWeight 0.0
MaxWeight 1.0
FileName "iris-SECoS.wgt"
```

Where:
`Inputs` is the number of input neurons.
`Hidden` is the number of neurons to put in the rule or evolving layer. For implementation reasons this cannot be less than one.
`Outputs` is the number of output neurons.
`MinWeight` is the minimum value of the initial connection weights.
`MaxWeight` is the maximum value of the initial connection weights.
`FileName` is a string specifying the path to write the new SECoS to.

4.3 TrainSECoS

The TrainSECoS tool will load a SECoS and data from file, perform ECoS training over the SECoS, then write the resulting network to a new file. A training log file (see Section A.5) will also be written to file or standard output.

```
WeightFile  "iris-SECoS.wgt"
DestinationFile  "iris-SECoS-trn.wgt"
TrainingFile  "iris.trn"
Epochs 1
RecallMethod  "OneOfN"
ActivationThreshold 0.5
SensitivityThreshold 0.5
ErrorThreshold 0.1
LearningRateOne 0.5
LearningRateTwo 0.5
RuleClustering  "Linear"
LogFile "iris.log"
MaxRuleNodes 1000
```

Where:
`WeightFile` is the SECoS to train.
`DestinationFile` is the file to write the trained SECoS to.
`TrainingFile` is a string argument specifying the data file to use to train the SECoS.
`Epochs` is the number of times the training data set will be presented to the network. One epoch is usually enough.
`RecallMethod` is a string argument specifying which method to use to propagate activations from the evolving layer. This must be either "`OneOfN`" or "`ManyOfN`".
`ActivationThreshold` is used only with the "`ManyOfN`" recall method. Only evolving layer neurons with activation values greater than this will have their activations propagated. This parameter will be ignored if `OneOfN` recall is used.
`SensitivityThreshold` is the sensitivity threshold training parameter.
`ErrorThreshold` is the error threshold training parameter.
`LearningRateOne` is the learning rate one parameter.
`LearningRateTwo` is the learning rate two parameter.
`LogFile` is a string parameter specifying the file to write the training log to. If this parameter is omitted, the training log will be written to standard output. An example training log file is presented in Section A.5.
`RuleClustering` is a string argument that determines where in the evolving layer new neurons will be inserted. Options are "`Linear`", whereby new neurons are added to the end of the evolving layer, and "`MaxActivation`", where the new neurons will be inserted next to the winning evolving layer neuron.
`MaxRuleNodes` is the maximum number of neurons allowed in the evolving layer.

4.4 RecallSECoS

The RecallSECoS tool will load a SECoS and data file, perform a recall-only operation over the SECoS using the data, and write the output values only to a new file. It take a single argument, which is the name of a parameter file and must be of the following format:

```
WeightFile "iris-SECoS-trn.wgt"
RecallFile "iris.trn"
OutputFile "iris-SECoS.out"
RecallMethod "OneOfN"
ActivationThreshold 1.0
```

Where:
`WeightFile` is a string argument specifying the file name of the SECoS to recall.
`RecallFile` is a string argument specifying the file name of the data to use to recall the SECoS.
`OutputFile` is a string argument specifying the file to write the output data to.
`RecallMethod` is a string parameter specifying the recall method of the SECoS. Must be either "`OneOfN`" or "`ManyOfN`".
`ActivationThreshold` is the activation threshold parameter for `ManyOfN` recall. Ignored if `OneOfN` is used.

4.5 SECoSFRE

This application will extract Zadeh-Mamdani fuzzy rules from a trained SECoS using externally supplied fuzzy membershp functions (MF). The MF can be generated using the MakMF tool (Section 7.3). An example of the kind of fuzzy rule file that is extracted from SECoS is presented in Section A.8. The argument to the SECoSFRE tool is a parameter file, which must be of the following format:

```
NetworkFile "iris-SECoS-trn.wgt"
InputMFFile "iris-in.mf"
OutputMFFile "iris-out.mf"
RuleFile "iris-SECoS.rul"
```

Where:
`NetworkFile` is a string specifying the SECoS file to extract the fuzzy rules from.
`InputMFFile` is a string argument specifying the file containing the input MF. The number of variables in this file must match the number of inputs in the SECoS.
`OutputMFFile` is a string argument specifying the file containing the output MF. The number of variables in this file must match the number of outputs in the SECoS.
`RuleFile` is a string argument specifying the file to write the extracted fuzzy rules to.

4.6 SECoSFRI

The SECoSFRI tool will defuzzify a set of Zadeh-Mamdani fuzzy rules and use them to construct a SECoS. This application has three mandatory arguments, as follows:

```
SECoSFRI RuleFile SECoSFile Steps
```

Where:
`RuleFile` is the fuzzy rule file to construct the SECoS from.
`SECoSFile` is the file to write the constructed SECoS to.
`Steps` is the number of steps to use in the defuzzification of the antecendents and consequents.

4.7 GetSECoSEx

The GetSECoSEx tool will extract the exemplars that are contained in the input to evolving layer connection weights of a SECoS. These exemplars are the prototypes that the SECoS uses to model new data. This application has two mandatory arguments, as follows:

```
GetSECoSEx SECoSFile DataFile
```

Where:
`SECoSFile` is the file to extract the exemplars from.
`DataFile` is the file to write the exemplars to. There will be one row for each neuron in the evolving layer of the SECoS. The number of inputs will be equal to the number of inputs in the SECoS. There will be no outputs.

4.8 CompileSECoS

Compiles a trained SECoS file into a code file that implements that particular SECoS. The SECoS in the code file produced is fixed and cannot be further trained but can be recalled. Five languages are supported by this tool: C++, C#, Python, Java and PHP.

```
CompileSECoS SECoSFile Classname [CPP, CS, PY, JV, PHP]
```

Where:
`SECoSFile` is the SECoS weight file to compile.
`Classname` is the name to give to the class that is output in code.
`[CPP, CS, PY, JV]` choose one (without the brackets), specifies which language to output. CPP is C++, CS is C#, PY is Python, JV is Java and PHP is PHP.

Examples of the code produced by this tool, and how to use it, are included in the `eg/CompileSECoS` subdirectory of the ECoS Toolbox. These examples were all compiled from the same SECoS network, which was trained on the iris data set.

The Java output portion of the compiler output was created with the assistance of Binglan Han.

Chapter 5

Evolving Clustering Method Tools

5.1 Introduction

The evolving clustering method (ECM) is an application of the ECoS principles to data clustering. This is based on the concept of dynamically adding and modifying the clusters as new data is presented, where the modification to the clusters affects both the position of the clusters and the size of the cluster, in terms of a radius parameter associated with each cluster that determines the boundaries of that cluster. ECM has only one parameter, which drives the addition of clusters, known as the distance threshold D_{thr}. When new clusters are added, their centres are set to equal the example that triggered their creation, and the radius R of a new cluster is initially set to zero. R grows as more vectors are allocated to the cluster. Due to the mechanism by which R is updated, it cannot exceed D_{thr}. There are two tools available for ECM. The first, TrainECM, will create the cluster centers and R values of the clusters. The second, ClusterECM, uses these cluster centres and R values to cluster a complete data set.

As before, the data files loaded by the ECM tools must be in the data file format produced by the tools in Chapter 2. Although the tools described in this chapter do not expect the data to be normalised, better results will be obtained if the data is normalised.

5.2 TrainECM

The TrainECM tool will create a set of cluster centres and cluster R values for a data set. This can be done incrementally, that is, existing centres and R values can be loaded and modified, or they can be created from scratch.

```
SourceFile "iris.trn"
DistanceThreshold 0.1
SourceCentresFile "iris_centres1.dat"
SourceRValuesFile "iris_R_values1.dat"
CentresFile "iris_centres2.dat"
```

```
RValuesFile "iris_R_Values2.dat"
DistanceMeasure "Euclidean"
```

Where:
`SourceFile` is a string parameter giving the name of the data file to cluster.
`DistanceThreshold` is the D_{thr} parameter.
`SourceCentresFile` is an optional string parameter that specifies an existing cluster centres file to load.
`SourceRValuesFile` is an optional string parameter that specifies an existing R values file to load. If `SourceCentresFile` is used, then `SourceRValuesFile` must also be present.
`CentresFile` is a string parameter that specifies the filename to save the cluster centres to. This is in the data file format.
`RValuesFile` is a string parameter that specifies the filename to save the R values to. This is also in the data file format.
`DistanceMeasure` is a string parameter that specifies which distance measure to use. This parameter was included for future growth: the only permissible option at this time is `Euclidean`.

5.3 ClusterECM

This application will perform clustering using the centres and R values produced by the TrainECM tool (Section 5.2).

```
SourceFile "iris.trn"
CentresFile "iris_centres1.dat"
RValuesFile "iris_R_Values1.dat"
MapFile "iris_map.dat"
ClusterPrefix "iris-"
ClusterSuffix ".dat"
DistanceMeasure "Euclidean"
SizesFile "ECM_Cluster_Sizes.dat"
```

Where:
`SourceFile` is a string parameter giving the name of the data file to cluster.
`CentresFile` is a string parameter giving the name of the cluster centres file to load.
`RValuesFile` is a string parameter giving the name of the R values file to load.
`MapFile` is a string parameter specifying the file to save the cluster map to. This map specifies which cluster each data row is assigned to. There are therefore as many rows in this file as there are in `SourceFile`.
`ClusterPrefix` is the prefix for each cluster data file. There will be one file for each cluster, and each will have this prefix, followed by a unique, sequential, number, then the suffix specified by the `ClusterSuffix` parameter.
`DistanceMeasure` is a string parameter that specifies which distance measure to use. This parameter was included for future growth: the only permissible option at this time is `Euclidean`.
`SizesFile` is a string parameter giving the name of the data file to save the size of each cluster to. That is, the number of vectors in each cluster.

Chapter 6

Nominal-scale Evolving Connectionist Systems Tools

6.1 Introduction

A NECoS network consists of three layers of neurons: The first layer is the input layer, where each neuron corresponds to a single input variable. Each input neuron has a set of "signal counters" associated with it, with there being one counter for each of the classes associated with that particular variable. Whenever an input vector is propagated through the input layer, the signal counters for each neuron are updated, with the signal counter for the class observed at each neuron being incremented.; The evolving layer; And an output layer with a simple saturated linear activation function. The similarity measure used in the evolving layer is the similarity coefficient.

As with the EFuNN and SECoS tools, the data files loaded by theNSECoS tools must be in the data file format produced by the tools in Chapter 2. The NECoS tools also expect the output data to be within the range of zero to unity. The input data, however, is assumed to be numerical labels. That is, a number is a label for a particular category. The numbers are therefore not constrained to any particular range.

The learning algorithm for NECoS networks is based heavily upon the standard ECoS algorithm. The only difference between the two is the manner in which learning is affected in the input to evolving layer of connections. In the standard ECoS algorithm, the amount the incoming weight vector moves closer to the input vector is determined by the learning rate one parameter (η_1), where a higher η_1 corresponds to a greater change. It is desirable to maintain this semantic in the NECoS training algorithm. Whereas the standard ECoS learning algorithm alters the weights in the input to evolving layer so that the neuron is spatially closer to $\mathbf{I}$, in NECoS this would not make sense, because nominal scale weights do not describe a position is space. Instead, the learning algorithm is based on the concept of making the incoming weight vector more similar to $\mathbf{I}$ by changing some of the connection labels to match the labels in $\mathbf{I}$, where

the degree to which the incoming weight vector is changed is determined by η_1. This is achieved by the following algorithm:

- Find each connection $\mathbf{W}_{i,j}$ that does not match the corresponding element c of $\mathbf{I}_i$, that is, find the mismatching incoming connections of j. Let n_m be the number of mismatched connections, m be the set of mismatched connections, and $\mathbf{I}_c$ be the set of mismatched input vector class labels.
- For each mismatched connection m_c find, from the input neuron signal counters, the corresponding frequency f_c of the target category for that input.
- Rank the mismatching connections m_r in descending order of f_c.
- Calculate the number of connections to change $n_c = \lceil \eta_1 n_m$.
- For each of the top n_c connections in m_r, set $\mathbf{W}_{i,j}$ to $\mathbf{I}_i$

Thus, the mismatching connections are changed to equal the most frequently occurring class labels, thereby making the incoming weight vector more similar to the current input vector.

6.2 CreateNECoS

The `CreateNECoS` tool will create a new NECoS structure and write it to a file. An example of this file format is in Section A.4. The argument to this tool is a parameter file that must be of the following format:

```
Inputs 4
Hidden 1
Outputs 3
FileName "lenses-NECoS.wgt"
DistanceMeasure "SimilarityMeasure"
```

Where:
`Inputs` is the number of input neurons.
`Hidden` is the number of neurons to put in the rule or evolving layer. For implementation reasons this cannot be less than one.
`Outputs` is the number of output neurons.
`FileName` is a string specifying the path to write the new SECoS to.
`DistanceMeasure` specifies the similarity measure to use. At the moment only `SimilarityCoefficient` can be used.

6.3 TrainNECoS

The TrainNECoS tool will load a NECoS and data from file, perform ECoS training over the NECoS, then write the resulting network to a new file. A training log file (see Section A.5) will also be written to file or standard output.

```
WeightFile  "lenses-NECoS.wgt"
DestinationFile  "lenses-NECoS-trn.wgt"
TrainingFile  "lenses.trn"
Epochs 1
SensitivityThreshold 0.5
ErrorThreshold 0.1
LearningRateOne 0.5
LearningRateTwo 0.5
RuleClustering  "Linear"
LogFile "iris.log"
MaxRuleNodes 1000
```

Where:
`WeightFile` is the NECoS to train.
`DestinationFile` is the file to write the trained NECoS to.
`TrainingFile` is a string argument specifying the data file to use to train the NECoS.
`Epochs` is the number of times the training data set will be presented to the network. One epoch is usually enough.
`SensitivityThreshold` is the sensitivity threshold training parameter.
`ErrorThreshold` is the error threshold training parameter.
`LearningRateOne` is the learning rate one parameter.
`LearningRateTwo` is the learning rate two parameter.
`LogFile` is a string parameter specifying the file to write the training log to. If this parameter is omitted, the training log will be written to standard output. An example training log file is presented in Section A.5.
`RuleClustering` is a string argument that determines where in the evolving layer new neurons will be inserted. Options are "`Linear`", whereby new neurons are added to the end of the evolving layer, and "`MaxActivation`", where the new neurons will be inserted next to the winning evolving layer neuron.
`MaxRuleNodes` is the maximum number of neurons allowed in the evolving layer.

6.4 RecallNECoS

The RecallNECoS tool will load a trained NECoS and data file, perform a recall-only operation over the NECoS using the data, and write the output values only to a new file. It take a single argument, which is the name of a parameter file and must be of the following format:

```
WeightFile "lenses-NECoS-trn.wgt"
RecallFile "lenses.trn"
OutputFile "lenses-NECoS.out"
```

Where:
`WeightFile` is a string argument specifying the file name of the NECoS to recall.
`RecallFile` is a string argument specifying the file name of the data to use to recall the NECoS.
`OutputFile` is a string argument specifying the file to write the output data to.

6.5 GetNECoSEx

The GetNECoSEx tool will extract the exemplars that are contained in the input to evolving layer connection weights of a NECoS. These exemplars are the prototypes that the NECoS uses to model new data. This application has two mandatory arguments, as follows:

```
GetNECoSEx NECoSFile DataFile
```

Where:
`NECoSFile` is the file to extract the exemplars from.
`DataFile` is the file to write the exemplars to. There will be one row for each neuron in the evolving layer of the NECoS. The number of inputs will be equal to the number of inputs in the NECoS. The number of outputs will be equal to the number of outputs in the NECoS.

6.6 GetNECoSSignals

The GetNECoSSignals tools will extract the signals counter data from a trained NECoS network. These are the category labels and the number of times each of those categories has been seen by each input neuron.

```
GetNECoSSignals NECoSFile
```

Where:
`NECoSFile` is the file to extract the signals from.

The output places the data for each input on one line. The start of the line is the tag `InputN`, where `N` is the number of the input, followed by an array of pairs of numbers in the format `X:Y`, where `X` is the category label and `Y` is the count. For example:

```
Input0 {0:8,1:8,2:8}
Input1 {0:12,1:12}
Input2 {0:12,1:12}
Input3 {0:12,1:12}
```

Chapter 7

Fuzzy Membership Function and Fuzzy Rule Tools

7.1 Introduction

The fuzzy tools included in the ECoS Toolbox are intended to be used for two activities. Firstly, the evaluation of fuzzy rules extracted from EFuNN and SECoS. Secondly, to support the extraction of fuzzy rules from SECoS. Only Zadeh-Mamdani rules are supported at this time.

7.2 MakMFLbl

The MakMFLbl tool is used to generate label files of the format needed to generate fuzzy MF files using the MakMF application (Section 7.3). The label files only describe the names (labels) attached to each variable and MF and not the type or parameters of the MF themselves. The labels generated must be manually edited if the user wants to change the defaults. Labels cannot contain spaces or other special characters. An example of a label file is shown in Section A.6. The argument to the MakMFLbl tool is the name of a parameter file, which is of the following format.

```
InputMF {3,3,3,3}
OutputMF {2,2,2}
LabelFile "iris.lbl"
```

Where:
`InputMF` is an array with one entry for each input variable. The number of input variables is inferred from the length of this array.
`OutputMF` is an array with one entry for each output variable. Again, the number of output variables is inferred from the length of this array.
`LabelFile` is a string that specifies the file to write the labels to.

The label file produced by these parameters is shown in Section A.6, where the default labels have been retained.

7.3 MakMF

This tool is used to create fuzzy MF files from labels. The label file is produced using the MakMFLbl tool (Section 7.2). An example of the file format produced by this tool is presented in Section A.7. The MakMF application takes a single argument, which is the name of a parameter file of the following format:

```
LabelFile "iris.lbl"
MFType "Triangular"
InputMFSetFile "iris-in.mf"
OutputMFSetFile "iris-out.mf"
```

Where:
`LabelFile` is a string argument specifying the name of the label file. The number of variables and the number of MF attached to each variable, as well as the labels for each variable and MF, are all inferred from this file.
`MFType` is a string argument specifying the type of the MF to construct. Currently, the only option is "`Triangular`".
`InputMFSetFile` is a string argument specifying the file to save the input MF set to.
`OutputMFSetFile` is a string argument specifying the file to save the output MF set to.

The input MF set file produced by these parameters is presented in Section A.7, where the labels have been changed from the default to better reflect the iris data set.

7.4 GetNumMF

This tool is used to analyse fuzzy MF files and will write to standard output the number of MF attached to each variable. There is one mandatory argument, as follows:

```
GetNumMF MFFile
```

Where:
`MFFile` is the file containing the MF to analyse.

The output of this tool is a line displaying the number of MF attached to each variable. There will be one number for each variable.

7.5 FR2MF

This tool will extract the fuzzy MF from a Zadeh-Mamdani fuzzy rule file and write them to separate files. There are three mandatory arguments, as follows:

```
FR2MF RuleFile InputMFFile OutputMFFile
```

Where:
`RuleFile` is the fuzzy rule file to extract the MF from.
`InputMFFile` is the file to write the input MF to.
`OutputMFFile` is the file to write the output MF to.

7.6 mPlotMF

This application will produce MATLAB code to plot the MF in the specified file.

```
mPlotMF MFFile mFile xSteps PlotTitle
```

Where:
`MFFile` is the file to read the MF from.
`mFile` is the file to write the MATLAB code to.
`xSteps` is the number of points on the x-axis of the plot, that is, the number of points to use to plot the MF.
`PlotTitle` is the title of the plot. This should be in quotes.

The code this tool produces has been confirmed to work with the free MATLAB clone Octave.

7.7 SFIE

This tool will perform a simplified fuzzy inference over a fuzzy rule set. Only the best-matching fuzzy rule is allowed to activate. This better matches the semantics of ECoS networks and therefore provides a better method of fuzzy inference for fuzzy rules that have been extracted from ECoS networks. This application has four mandatory and one optional argument, as follows:

```
SFIE RuleFile InputDataFile OutputDataFile MFInferenceMethod Steps
```

Where:
`RuleFile` is the fuzzy rule file.
`InputDataFile` is the file to read the input data from.
`OutputDataFile` is the file to write the output data to.
`MFInferenceMethod` is the inference method to use. Options are "`Max`" or "`Prod`".
`Steps` is the number of steps to use in the defuzzification step. This is an optional argument.

Appendix A

File Formats

A.1 Data File Format

This file is a subset of the famous iris data set. There are thirty rows in this file, with four input variables and three output variables. Tags in this and the following file formats are contained in square brackets. The `Rows`, `Inputs` and `ExpectedOutputs` tags specify the number of data row, input and output variables respectively. The `GeneratedOutputs` and `Error` tags were included for future developments and are not used by the ECoS toolbox. All of the data is between the `Data` and `~Data` tags. This data file is included in the toolbox as `iris.trn`.

```
[FormatVersion = 1.0]

[DataSet]
[Rows = 30]
[Inputs = 4]
[ExpectedOutputs = 3]
[GeneratedOutputs = 0]
[Error = 0]

[Data]
0.30555555555487 0.708333333332903 0.0847457627118434 0.0416666666666575   1 0 0
0.194444444443714 0 0.423728813559217 0.374999999999994   0 1 0
0.611111111110549 0.499999999999696 0.694915254237116 0.791666666666665   0 0 1
0.138888888888136 0.583333333332978 0.101694915254212 0.0416666666666575   1 0 0
0.444444444443815 0.416666666666413 0.542372881355798 0.583333333333329   0 1 0
0.58333333333276 0.291666666666489 0.728813559321854 0.749999999999998   0 0 1
0.138888888888136 0.416666666666413 0.0677966101694747 0   1 0 0
0.472222222221604 0.0833333333332827 0.508474576271061 0.374999999999994   0 1 0
0.694444444443916 0.416666666666413 0.762711864406591 0.833333333333332   0 0 1
0 0.416666666666413 0.0169491525423687 0   1 0 0
0.499999999999393 0.374999999999772 0.627118644067642 0.541666666666662   0 1 0
0.388888888888237 0.208333333333207 0.677966101694748 0.791666666666665   0 0 1
0.416666666666026 0.833333333332826 0.0338983050847374 0.0416666666666575   1 0 0
0.361111111110448 0.374999999999772 0.440677966101586 0.499999999999995   0 1 0
```

```
0.416666666666026 0.333333333333131 0.694915254237116 0.958333333333333   0 0 1
0.388888888888237 0.999999999999392 0.0847457627118434 0.124999999999992   1 0 0
0.666666666666127 0.458333333333055 0.576271186440535 0.541666666666662   0 1 0
0.58333333333276 0.499999999999696 0.728813559321854 0.916666666666666   0 0 1
0.30555555555487 0.791666666666185 0.0508474576271061 0.124999999999992   1 0 0
0.361111111110448 0.416666666666413 0.593220338982904 0.583333333333329   0 1 0
0.611111111110549 0.416666666666413 0.762711864406591 0.708333333333331   0 0 1
0.222222222221503 0.62499999999962 0.0677966101694747 0.0833333333333246   1 0 0
0.416666666666026 0.291666666666489 0.525423728813429 0.374999999999994   0 1 0
0.944444444444017 0.749999999999544 0.966101694915015 0.874999999999999   0 0 1
0.388888888888237 0.749999999999544 0.118644067796581 0.0833333333333246   1 0 0
0.527777777777182 0.0833333333332827 0.593220338982904 0.583333333333329   0 1 0
0.944444444444017 0.249999999999848 0.999999999999753 0.916666666666666   0 0 1
0.222222222221503 0.749999999999544 0.0847457627118434 0.0833333333333246   1 0 0
0.361111111110448 0.208333333333207 0.491525423728692 0.416666666666661   0 1 0
0.472222222221604 0.0833333333332827 0.677966101694748 0.583333333333329   0 0 1
[~Data]

[~DataSet]
```

A.2 EFuNN File Format

This file is for an EFuNN with four input neurons with three MF attached to each. There is only one neuron in the rule layer. There are three output neurons with two MF attached to each. Basic information on the sturcture of the network is defined in the block delimited by the `NetworkInformation` and`~NetworkInformation` tags. Information about the individual neuron layers is between `Layer` and `~Layer` tags. The `LayerType` tag specifies which type of neurons are in the layer.The `FuzzyCondition` layer represents the input MF, while the output MF are specified in the `Defuzzify`. layer. Layer 0 is included for future growth and is not used by EFuNN. Information on the connection weight layers is delimited by the `ConnectionLayer` and `~ConnectionLayer` tags. The `LayerFrom` and `LayerTo` tags specify which neuron layers the connections connect. For the connections from layer 1 to 2 in EFuNN the connection weights specify the centres of each MF. There is a single entry for each connection, where the `From` and `To` tags specify which neurons in layers 1 and two the connections are between. There is a matrix of connections between layers 2 and 3, and 3 and 4 (since there is only one rule neuron in this example, these matrices take the forms of single columns and rows, respectively). Comments in this file format follow the C-style, that is, they are started with `/*` and ended with `*/`. The weights from neuron layer 4 to neuron layer 5 are of the same format as those from layers 1 to 2. This file is included in the toolbox as `iris-EFuNN.wgt`.

```
[FormatVersion = 1.0]

[NeuralNetwork]
[NetworkType = EFuNN]

[NetworkInformation]
```

```
[NumberOfLayers = 6]
[InputLayer = 1]
[OutputLayer = 5]
[NumberOfInputs = 4]
[NumberOfOutputs = 3]
[~NetworkInformation]

[Layer = 0]
[LayerType = Simple]
[NumberOfNeurons = 0]
[All ActivationFunction=null ]
[~Layer]

[Layer = 1]
[LayerType = Input]
[NumberOfNeurons = 4]
[All ActivationFunction=linear ]
[~Layer]

[Layer = 2]
[LayerType = FuzzyCondition]
[NumberOfNeurons = 12]
[All ActivationFunction=FuNN-condition ]
[InputMF]
[Input=0 MF=3 ]
[Input=1 MF=3 ]
[Input=2 MF=3 ]
[Input=3 MF=3 ]
[~InputMF]
[~Layer]

[Layer = 3]
[LayerType = EFuNNRule]
[NumberOfNeurons = 1]
[All ActivationFunction=saturated-linear ]
[~Layer]

[Layer = 4]
[LayerType = Simple]
[NumberOfNeurons = 6]
[All ActivationFunction=saturated-linear ]
[~Layer]

[Layer = 5]
[LayerType = Defuzzify]
[NumberOfNeurons = 3]
[All ActivationFunction=cog ]
[OutputMF]
[Output=0 MF=2 ]
[Output=1 MF=2 ]
```

```
[Output=2 MF=2 ]
[~OutputMF]
[~Layer]

[ConnectionLayer]
[LayerFrom = 1]
[LayerTo = 2]
[ConnectionType = ToConditionConnections]
[Attributes Pruneable Freezable ]
[WeightList]
[From=0 To=0 Weight=0.000000000000000 Trainable ]
[From=0 To=1 Weight=0.500000000000000 Trainable ]
[From=0 To=2 Weight=1.000000000000000 Trainable ]
[From=1 To=3 Weight=0.000000000000000 Trainable ]
[From=1 To=4 Weight=0.500000000000000 Trainable ]
[From=1 To=5 Weight=1.000000000000000 Trainable ]
[From=2 To=6 Weight=0.000000000000000 Trainable ]
[From=2 To=7 Weight=0.500000000000000 Trainable ]
[From=2 To=8 Weight=1.000000000000000 Trainable ]
[From=3 To=9 Weight=0.000000000000000 Trainable ]
[From=3 To=10 Weight=0.500000000000000 Trainable ]
[From=3 To=11 Weight=1.000000000000000 Trainable ]
[~WeightList]
[~ConnectionLayer]

[ConnectionLayer]
[LayerFrom = 2]
[LayerTo = 3]
[ConnectionType = FullyConnected]
[Attributes UnPrunable UnFreezable ]
[Matrix Weights ]
/* To->            0  From */
   0.105807672353282  /* 0 */
   0.386974700155644  /* 1 */
   0.114749595629749  /* 2 */
   0.461378826258126  /* 3 */
   0.600665303506577  /* 4 */
   0.838251899777215  /* 5 */
  0.0805993835261086  /* 6 */
   0.223914304025391  /* 7 */
   0.475173192541276  /* 8 */
   0.300515762810144  /* 9 */
   0.719534897915586  /* 10 */
   0.380413220618305  /* 11 */
[~Matrix]
[~ConnectionLayer]

[ConnectionLayer]
[LayerFrom = 3]
[LayerTo = 4]
```

```
[ConnectionType = FullyConnected]
[Attributes UnPrunable UnFreezable ]
[Matrix Weights ]
/* To->            0                    1                    2
   0.335154271065401   0.0858180486465041    0.207251197851497
                3                    4                    5  From */
0.567827387310404    0.275917844172491    0.389263588366344  /* 0 */
[~Matrix]
[~ConnectionLayer]

[ConnectionLayer]
[LayerFrom = 4]
[LayerTo = 5]
[ConnectionType = ToOutputConnections]
[Attributes Pruneable Freezable ]
[WeightList]
[From=0 To=0 Weight=0.000000000000000 Trainable ]
[From=1 To=0 Weight=1.000000000000000 Trainable ]
[From=2 To=1 Weight=0.000000000000000 Trainable ]
[From=3 To=1 Weight=1.000000000000000 Trainable ]
[From=4 To=2 Weight=0.000000000000000 Trainable ]
[From=5 To=2 Weight=1.000000000000000 Trainable ]
[~WeightList]
[~ConnectionLayer]

[~NeuralNetwork]
```

A.3 SECoS File Format

The tags in the SECoS file format have the same meaning as those in the EFuNN file format. The SECoS in this example has four input neurons, one neuron in the evolving layer, and three neurons in the output layer. This file is included in the toolbox as `iris-SECoS.wgt`.

```
[FormatVersion = 1.0]

[NeuralNetwork]
[NetworkType = SECoS]

[NetworkInformation]
[NumberOfLayers = 4]
[InputLayer = 1]
[OutputLayer = 3]
[NumberOfInputs = 4]
[NumberOfOutputs = 3]
[~NetworkInformation]

[Layer = 0]
[LayerType = Simple]
```

```
[NumberOfNeurons = 0]
[All ActivationFunction=null ]
[~Layer]

[Layer = 1]
[LayerType = Input]
[NumberOfNeurons = 4]
[All ActivationFunction=linear ]
[~Layer]

[Layer = 2]
[LayerType = SECoSRule]
[NumberOfNeurons = 1]
[All ActivationFunction=saturated-linear ]
[~Layer]

[Layer = 3]
[LayerType = Gain]
[NumberOfNeurons = 3]
[All ActivationFunction=saturated-linear ]
[All Gain=1.000000000000000 ]
[~Layer]

[ConnectionLayer]
[LayerFrom = 1]
[LayerTo = 2]
[ConnectionType = FullyConnected]
[Attributes UnPrunable UnFreezable ]
[Matrix Weights ]
/* To->            0  From */
   0.254341257972961  /* 0 */
   0.405438398388623  /* 1 */
   0.337137974181341  /* 2 */
   0.590990936002686  /* 3 */
[~Matrix]
[~ConnectionLayer]

[ConnectionLayer]
[LayerFrom = 2]
[LayerTo = 3]
[ConnectionType = FullyConnected]
[Attributes UnPrunable UnFreezable ]
[Matrix Weights ]
/* To->            0                    1                    2  From */
   0.770012512588885    0.503372295297098    0.197485274819178  /* 0 */
[~Matrix]
[~ConnectionLayer]

[~NeuralNetwork]
```

A.4 NECoS File Format

The tags in the NECoS file format have the same meaning as those in the SECoS file format. The only differences are that the NECoS file format include a NumberOfCategories tag for each input neuron, which counts how many unique categories the neuron has encountered during training, and a SignalCounts tag for each category that counts how many times each unique category has been encountered. The NECoS in this example has four input neurons, one neuron in the evolving layer, and three neurons in the output layer. The first input has three categories and the other three have two each. This file is included in the toolbox as `lenses-NECoS.wgt`.

```
[FormatVersion = 1.0]

[NeuralNetwork]
[NetworkType = NECoS]

[NetworkInformation]
[NumberOfLayers = 4]
[InputLayer = 1]
[OutputLayer = 3]
[NumberOfInputs = 4]
[NumberOfOutputs = 3]
[~NetworkInformation]

[Layer = 0]
[LayerType = Simple]
[NumberOfNeurons = 0]
[All ActivationFunction=null ]
[~Layer]

[Layer = 1]
[LayerType = NECoSInputLayer]
[NumberOfNeurons = 4]
[All ActivationFunction=linear ]
[NECoSInputLayer]
[NECoSInputNeuron]
[Index = 0]
[NumberOfCategories = 3]
[SignalCounts]
[Category=0.0000000 Count=8.0000000 ]
[Category=1.0000000 Count=8.0000000 ]
[Category=2.0000000 Count=8.0000000 ]
[~SignalCounts]
[~NECoSInputNeuron]
[NECoSInputNeuron]
[Index = 1]
[NumberOfCategories = 2]
[SignalCounts]
[Category=0.0000000 Count=12.0000000 ]
```

```
[Category=1.0000000 Count=12.0000000 ]
[~SignalCounts]
[~NECoSInputNeuron]
[NECoSInputNeuron]
[Index = 2]
[NumberOfCategories = 2]
[SignalCounts]
[Category=0.0000000 Count=12.0000000 ]
[Category=1.0000000 Count=12.0000000 ]
[~SignalCounts]
[~NECoSInputNeuron]
[NECoSInputNeuron]
[Index = 3]
[NumberOfCategories = 2]
[SignalCounts]
[Category=0.0000000 Count=12.0000000 ]
[Category=1.0000000 Count=12.0000000 ]
[~SignalCounts]
[~NECoSInputNeuron]
[~NECoSInputLayer]
[~Layer]

[Layer = 2]
[LayerType = NECoSEvolvingLayer]
[NumberOfNeurons = 13]
[All ActivationFunction=saturated-linear ]
[NECoSEvolvingLayer]
[DistanceMeasure = SimilarityCoefficient]
[NumVariables = 4]
[~NECoSEvolvingLayer]
[~Layer]

[Layer = 3]
[LayerType = Gain]
[NumberOfNeurons = 3]
[All ActivationFunction=saturated-linear ]
[All Gain=1.000000000000000 ]
[~Layer]

[ConnectionLayer]
[LayerFrom = 1]
[LayerTo = 2]
[ConnectionType = FullyConnected]
[Attributes UnPrunable UnFreezable ]
[Matrix Weights ]
/* To->           0                   1                   2                   3
                  0                   0                   2                   2
                  0                   0                   1                   1
                  0                   0                   0                   1
                  0                   0                   1                   0
```

```
[~Matrix]
[~ConnectionLayer]

[ConnectionLayer]
[LayerFrom = 2]
[LayerTo = 3]
[ConnectionType = FullyConnected]
[Attributes UnPrunable UnFreezable ]
[Matrix Weights ]
/* To->            0                    1                    2  From */
                   0                    0                    0  /* 0 */
                   0                    0                    1  /* 1 */
                   0     1.20956420898438                    0  /* 2 */
                   0                    0     1.20956420898438  /* 3 */
        1.1611328125                    0                    0  /* 4 */
                   0                    0         1.1611328125  /* 5 */
                   1                    0                    0  /* 6 */
                   0                    0         1.1611328125  /* 7 */
                   0                    1                    0  /* 8 */
                   0                    0                    1  /* 9 */
                   0                    0                    1  /* 10 */
                   0                    0                    1  /* 11 */
                   0                    0                    1  /* 12 */
[~Matrix]
[~ConnectionLayer]

[~NeuralNetwork]
```

A.5 ECoS Train Log Format

This is an example of the output produced by the training tools for EFuNN and SECoS networks. The training in this example was performed over the iris data set shown in Section A.1. There is one line for each training example ("`Pattern`") and they are numbered from zero. There is one "`Error`" tag for each output neuron, and they are numbered according to the output neuron they correspond to. The error is the absolute difference between the target output value for that training example and the output value produced by the output neuron. The "`MaxAct`" tag is the maximum activation value of the evolving layer neurons, while the "`MinAct`" tag is the minimum activation value. The tag "`AddNode`" is inserted if a new node or neuron is added to the evolving layer. This is followed by the position in which the neuron is inserted into the evolving layer. Finally, the "`Nodes`" tag displays the total number of neurons in the evolving layer after any further neurons were inserted.

This training log is included in the toolbox as `iris.log`. Some line wrapping has been done here to improve readability.

```
Pattern 0 Error0 1 Error1 0 Error2 0 MaxAct 0.719596 MinAct 0.719596
          AddNode {1} Nodes 2
```

```
Pattern 1 Error0 0.0404229 Error1 0 Error2 0 MaxAct 0.959577 MinAct 0.738633
          Nodes 2
Pattern 2 Error0 0.0104003 Error1 0 Error2 0 MaxAct 0.9896 MinAct 0.7171
          Nodes 2
Pattern 3 Error0 0.0361882 Error1 0 Error2 0 MaxAct 0.963812 MinAct 0.753686
          Nodes 2
Pattern 4 Error0 0.0708647 Error1 0 Error2 0 MaxAct 0.929135 MinAct 0.714791
          Nodes 2
Pattern 5 Error0 0.0276322 Error1 0 Error2 0 MaxAct 0.972368 MinAct 0.698651
          Nodes 2
Pattern 6 Error0 0.0593512 Error1 0 Error2 0 MaxAct 0.940649 MinAct 0.697417
          Nodes 2
Pattern 7 Error0 0.0639847 Error1 0 Error2 0 MaxAct 0.936015 MinAct 0.68262
          Nodes 2
Pattern 8 Error0 0.0607172 Error1 0 Error2 0 MaxAct 0.939283 MinAct 0.723116
          Nodes 2
Pattern 9 Error0 0.060977 Error1 0 Error2 0 MaxAct 0.939023 MinAct 0.695157
          Nodes 2
Pattern 10 Error0 0.731681 Error1 1 Error2 0 MaxAct 0.731681 MinAct 0.703446
           AddNode {2} Nodes 3
Pattern 11 Error0 0 Error1 0.0180347 Error2 0 MaxAct 0.981965 MinAct 0.687876
           Nodes 3
Pattern 12 Error0 0 Error1 0.0380879 Error2 0 MaxAct 0.961912 MinAct 0.680499
           Nodes 3
Pattern 13 Error0 0 Error1 0.033123 Error2 0 MaxAct 0.966877 MinAct 0.70973
           Nodes 3
Pattern 14 Error0 0 Error1 0.0317255 Error2 0 MaxAct 0.968275 MinAct 0.698362
           Nodes 3
Pattern 15 Error0 0 Error1 0.178486 Error2 0 MaxAct 0.821514 MinAct 0.673798
           AddNode {3} Nodes 4
Pattern 16 Error0 0 Error1 0.0578015 Error2 0 MaxAct 0.942199 MinAct 0.656024
           Nodes 4
Pattern 17 Error0 0 Error1 0.0838016 Error2 0 MaxAct 0.916198 MinAct 0.74489
           Nodes 4
Pattern 18 Error0 0 Error1 0.0529611 Error2 0 MaxAct 0.947039 MinAct 0.716135
           Nodes 4
Pattern 19 Error0 0 Error1 0.0760903 Error2 0 MaxAct 0.92391 MinAct 0.642379
           Nodes 4
Pattern 20 Error0 0 Error1 0.884834 Error2 1 MaxAct 0.884834 MinAct 0.634055
           AddNode {4} Nodes 5
Pattern 21 Error0 0 Error1 0 Error2 0.0513654 MaxAct 0.948635 MinAct 0.624757
           Nodes 5
Pattern 22 Error0 0 Error1 0 Error2 0.0366107 MaxAct 0.963389 MinAct 0.631379
           Nodes 5
Pattern 23 Error0 0 Error1 0.892364 Error2 1 MaxAct 0.892364 MinAct 0.679992
           AddNode {5} Nodes 6
Pattern 24 Error0 0 Error1 0 Error2 0.0818112 MaxAct 0.918189 MinAct 0.61293
           Nodes 6
Pattern 25 Error0 0 Error1 0 Error2 0.0569604 MaxAct 0.94304 MinAct 0.669393
           Nodes 6
```

```
Pattern 26 Error0 0 Error1 0 Error2 0.0549021 MaxAct 0.945098 MinAct 0.637114
           Nodes 6
Pattern 27 Error0 0 Error1 0 Error2 0.054068 MaxAct 0.945932 MinAct 0.631904
           Nodes 6
Pattern 28 Error0 0 Error1 0.935515 Error2 1 MaxAct 0.935515 MinAct 0.658472
           AddNode {6} Nodes 7
Pattern 29 Error0 0 Error1 0 Error2 0.0800894 MaxAct 0.919911 MinAct 0.561931
           Nodes 7
```

A.6 Fuzzy MF Label File Format

This file describes the names or labels attached to each variable and membership function (MF) in a fuzzy MF set. It does not describe the structure of the MF or rules themselves. These labels are used to extract fuzzy rules from EFuNN (using the `REEFuNN` tool, Section 3.6) and for creating sets of MF for fuzzy rule extraction from SECoS (using the `MakMF` and `SECOSFRE` tools respectively, Sections 7.3 and 4.5). Information on the number of variables and MF attached to each is contained in the `StructureInfo` block. The labels attached to the input variables are in the `InputLabels` block, while the labels attached to the input MF are in the `InputMFLabels`. These labels can all be edited, but they must not contain spaces or other special characters. The output variable labels are in the `OutputLables` block and the output MF labels are in the `OutputMFLabels` block. This file is included in the toolbox as `iris.lbl`.

```
[FuNNLabels]

[StructureInfo]

[Inputs = 4]
[InputMF]
[Input=0 MF=3 ]
[Input=1 MF=3 ]
[Input=2 MF=3 ]
[Input=3 MF=3 ]
[~InputMF]

[Outputs = 3]
[OutputMF]
[Output=0 MF=2 ]
[Output=1 MF=2 ]
[Output=2 MF=2 ]
[~OutputMF]

[~StructureInfo]

[InputLabels]
[Input=0 Label=InputA ]
[Input=1 Label=InputB ]
[Input=2 Label=InputC ]
```

```
[Input=3 Label=InputD ]
[~InputLabels]

[InputMFLabels]
[Condition=0 Label=A ]
[Condition=1 Label=B ]
[Condition=2 Label=C ]
[Condition=3 Label=A ]
[Condition=4 Label=B ]
[Condition=5 Label=C ]
[Condition=6 Label=A ]
[Condition=7 Label=B ]
[Condition=8 Label=C ]
[Condition=9 Label=A ]
[Condition=10 Label=B ]
[Condition=11 Label=C ]
[~InputMFLabels]

[OutputLabels]
[Output=0 Label=OutputA ]
[Output=1 Label=OutputB ]
[Output=2 Label=OutputC ]
[~OutputLabels]

[OutputMFLabels]
[Action=0 Label=A ]
[Action=1 Label=B ]
[Action=2 Label=A ]
[Action=3 Label=B ]
[Action=4 Label=A ]
[Action=5 Label=B ]
[~OutputMFLabels]

[~FuNNLabels]
```

A.7 Fuzzy MF File Format

This file describes a set of fuzzy membership functions for four variables. There are three MF attached to each variable. The information for individual MF are in the blocks delimited with `MF` and `~MF`. All of the MF attached to a variable are in blocks delimited with `MFSetInfo` and `~MFSetInfo`. An `MFList` contains the MF information for several variables. The MF specified in this file are triangular MF. This file in included in the toolbox as `iris-in.mf`.

```
[MFList]

[MFListInfo]
[VariablesRepresented = 4]
[ListName = Inputs]
```

```
[~MFListInfo]

[MFSet]

[MFSetInfo]
[Name=SepalLength Index=0 Type=Triangular ]
[LowerBound=0.0000000 UpperBound=1.0000000 ]
[NumberOfMF = 3]
[~MFSetInfo]

[MF]
[Name=Short Index=0 Type=LeftShoulderedTriangular ]
[Centre=0.0000000 RightBound=0.5000000 ]
[~MF]

[MF]
[Name=Medium Index=1 Type=Triangular ]
[LeftBound=0.0000000 Centre=0.5000000 RightBound=1.0000000 ]
[~MF]

[MF]
[Name=Long Index=2 Type=RightShoulderedTriangular ]
[LeftBound=0.5000000 Centre=1.0000000 ]
[~MF]

[~MFSet]

[MFSet]

[MFSetInfo]
[Name=SepalWidth Index=1 Type=Triangular ]
[LowerBound=0.0000000 UpperBound=1.0000000 ]
[NumberOfMF = 3]
[~MFSetInfo]

[MF]
[Name=Short Index=0 Type=LeftShoulderedTriangular ]
[Centre=0.0000000 RightBound=0.5000000 ]
[~MF]

[MF]
[Name=Medium Index=1 Type=Triangular ]
[LeftBound=0.0000000 Centre=0.5000000 RightBound=1.0000000 ]
[~MF]

[MF]
[Name=Long Index=2 Type=RightShoulderedTriangular ]
[LeftBound=0.5000000 Centre=1.0000000 ]
[~MF]
```

```
[~MFSet]

[MFSet]

[MFSetInfo]
[Name=PetalLength Index=2 Type=Triangular ]
[LowerBound=0.0000000 UpperBound=1.0000000 ]
[NumberOfMF = 3]
[~MFSetInfo]

[MF]
[Name=Short Index=0 Type=LeftShoulderedTriangular ]
[Centre=0.0000000 RightBound=0.5000000 ]
[~MF]

[MF]
[Name=Medium Index=1 Type=Triangular ]
[LeftBound=0.0000000 Centre=0.5000000 RightBound=1.0000000 ]
[~MF]

[MF]
[Name=Long Index=2 Type=RightShoulderedTriangular ]
[LeftBound=0.5000000 Centre=1.0000000 ]
[~MF]

[~MFSet]

[MFSet]

[MFSetInfo]
[Name=PetalWidth Index=3 Type=Triangular ]
[LowerBound=0.0000000 UpperBound=1.0000000 ]
[NumberOfMF = 3]
[~MFSetInfo]

[MF]
[Name=Short Index=0 Type=LeftShoulderedTriangular ]
[Centre=0.0000000 RightBound=0.5000000 ]
[~MF]

[MF]
[Name=Medium Index=1 Type=Triangular ]
[LeftBound=0.0000000 Centre=0.5000000 RightBound=1.0000000 ]
[~MF]

[MF]
[Name=Long Index=2 Type=RightShoulderedTriangular ]
[LeftBound=0.5000000 Centre=1.0000000 ]
[~MF]
```

```
[~MFSet]

[~MFList]
```

A.8 Fuzzy Rule File Format

This file describes the fuzzy rules extracted from an ECoS network for the iris problem. The rules are in the same format whether they are extracted from EFuNN or SECoS. The fuzzy MF used to extract the rules are embedded in the fuzzy rule file and are of the same format as in Section A.7. Following the fuzzy MF, in the block delimited by the `RuleList` and `~RuleList` tags, are the fuzzy rules. Each fuzzy rule starts with `if` and each atom of the antecedents and consequents are delimited by `<` and `>`. The antecedents and consequents are delimited by `then`. The numbers at the end of each atom are certainty factors.

This file is included in the toolbox as `iris.rul`. Some line-wrapping has been done for clarity.

```
[FormatVersion = 1.0]

[FuzzyRules]
[RuleType = SECoSFRE]

[MFList]

[MFListInfo]
[VariablesRepresented = 4]
[ListName = Inputs]
[~MFListInfo]

[MFSet]

[MFSetInfo]
[Name=SepalLength Index=0 Type=Triangular ]
[LowerBound=0.0000000 UpperBound=1.0000000 ]
[NumberOfMF = 3]
[~MFSetInfo]

[MF]
[Name=Short Index=0 Type=LeftShoulderedTriangular ]
[Centre=0.0000000 RightBound=0.5000000 ]
[~MF]

[MF]
[Name=Medium Index=1 Type=Triangular ]
[LeftBound=0.0000000 Centre=0.5000000 RightBound=1.0000000 ]
[~MF]

[MF]
[Name=Long Index=2 Type=RightShoulderedTriangular ]
```

```
[LeftBound=0.5000000 Centre=1.0000000 ]
[~MF]

[~MFSet]

[MFSet]

[MFSetInfo]
[Name=SepalWidth Index=1 Type=Triangular ]
[LowerBound=0.0000000 UpperBound=1.0000000 ]
[NumberOfMF = 3]
[~MFSetInfo]

[MF]
[Name=Short Index=0 Type=LeftShoulderedTriangular ]
[Centre=0.0000000 RightBound=0.5000000 ]
[~MF]

[MF]
[Name=Medium Index=1 Type=Triangular ]
[LeftBound=0.0000000 Centre=0.5000000 RightBound=1.0000000 ]
[~MF]

[MF]
[Name=Long Index=2 Type=RightShoulderedTriangular ]
[LeftBound=0.5000000 Centre=1.0000000 ]
[~MF]

[~MFSet]

[MFSet]

[MFSetInfo]
[Name=PetalLength Index=2 Type=Triangular ]
[LowerBound=0.0000000 UpperBound=1.0000000 ]
[NumberOfMF = 3]
[~MFSetInfo]

[MF]
[Name=Short Index=0 Type=LeftShoulderedTriangular ]
[Centre=0.0000000 RightBound=0.5000000 ]
[~MF]

[MF]
[Name=Medium Index=1 Type=Triangular ]
[LeftBound=0.0000000 Centre=0.5000000 RightBound=1.0000000 ]
[~MF]

[MF]
[Name=Long Index=2 Type=RightShoulderedTriangular ]
```

```
[LeftBound=0.5000000 Centre=1.0000000 ]
[~MF]

[~MFSet]

[MFSet]

[MFSetInfo]
[Name=PetalWidth Index=3 Type=Triangular ]
[LowerBound=0.0000000 UpperBound=1.0000000 ]
[NumberOfMF = 3]
[~MFSetInfo]

[MF]
[Name=Short Index=0 Type=LeftShoulderedTriangular ]
[Centre=0.0000000 RightBound=0.5000000 ]
[~MF]

[MF]
[Name=Medium Index=1 Type=Triangular ]
[LeftBound=0.0000000 Centre=0.5000000 RightBound=1.0000000 ]
[~MF]

[MF]
[Name=Long Index=2 Type=RightShoulderedTriangular ]
[LeftBound=0.5000000 Centre=1.0000000 ]
[~MF]

[~MFSet]

[~MFList]

[MFList]

[MFListInfo]
[VariablesRepresented = 3]
[ListName = Outputs]
[~MFListInfo]

[MFSet]

[MFSetInfo]
[Name=IrisSetosa Index=0 Type=Triangular ]
[LowerBound=0.0000000 UpperBound=1.0000000 ]
[NumberOfMF = 2]
[~MFSetInfo]

[MF]
[Name=IsNot Index=0 Type=LeftShoulderedTriangular ]
[Centre=0.0000000 RightBound=1.0000000 ]
```

```
[~MF]

[MF]
[Name=Certain Index=1 Type=RightShoulderedTriangular ]
[LeftBound=0.0000000 Centre=1.0000000 ]
[~MF]

[~MFSet]

[MFSet]

[MFSetInfo]
[Name=IrisVersicolor Index=1 Type=Triangular ]
[LowerBound=0.0000000 UpperBound=1.0000000 ]
[NumberOfMF = 3]
[~MFSetInfo]

[MF]
[Name=IsNot Index=0 Type=LeftShoulderedTriangular ]
[Centre=0.0000000 RightBound=0.5000000 ]
[~MF]

[MF]
[Name=UnCertain Index=1 Type=Triangular ]
[LeftBound=0.0000000 Centre=0.5000000 RightBound=1.0000000 ]
[~MF]

[MF]
[Name=Certain Index=2 Type=RightShoulderedTriangular ]
[LeftBound=0.5000000 Centre=1.0000000 ]
[~MF]

[~MFSet]

[MFSet]

[MFSetInfo]
[Name=IrisVirginica Index=2 Type=Triangular ]
[LowerBound=0.0000000 UpperBound=1.0000000 ]
[NumberOfMF = 3]
[~MFSetInfo]

[MF]
[Name=IsNot Index=0 Type=LeftShoulderedTriangular ]
[Centre=0.0000000 RightBound=0.5000000 ]
[~MF]

[MF]
[Name=UnCertain Index=1 Type=Triangular ]
[LeftBound=0.0000000 Centre=0.5000000 RightBound=1.0000000 ]
```

```
[~MF]

[MF]
[Name=Certain Index=2 Type=RightShoulderedTriangular ]
[LeftBound=0.5000000 Centre=1.0000000 ]
[~MF]

[~MFSet]

[~MFList]

[RuleList]

if <SepalLength is Long 0.655448> and <SepalWidth is Long 0.589892> and
   <PetalLength is Long 0.70782> and <PetalWidth is Long 0.886445>
  then <IrisSetosa is IsNot 1> and <IrisVersicolor is IsNot 1> and
       <IrisVirginica is Certain 1>

if <SepalLength is Medium 0.601663> and <SepalWidth is Medium 0.69268> and
   <PetalLength is Medium 0.597808> and <PetalWidth is Medium 0.563703>
  then <IrisSetosa is IsNot 1> and <IrisVersicolor is IsNot 1> and
       <IrisVirginica is Certain 1>

if <SepalLength is Long 0.727569> and <SepalWidth is Medium 0.687862> and
   <PetalLength is Long 0.667716> and <PetalWidth is Medium 0.502216>
  then <IrisSetosa is IsNot 1> and <IrisVersicolor is IsNot 1> and
       <IrisVirginica is Certain 1>

if <SepalLength is Long 0.691592> and <SepalWidth is Long 0.553499> and
   <PetalLength is Long 0.528793> and <PetalWidth is Long 0.609417>
  then <IrisSetosa is IsNot 1> and <IrisVersicolor is IsNot 1> and
       <IrisVirginica is Certain 1>

if <SepalLength is Long 0.741096> and <SepalWidth is Long 0.534557> and
   <PetalLength is Medium 0.527419> and <PetalWidth is Medium 0.644126>
  then <IrisSetosa is IsNot 1> and <IrisVersicolor is Certain 1> and
       <IrisVirginica is IsNot 1>

if <SepalLength is Long 0.561781> and <SepalWidth is Long 0.597345> and
   <PetalLength is Medium 0.906948> and <PetalWidth is Medium 0.95859>
  then <IrisSetosa is Certain 1> and <IrisVersicolor is IsNot 1> and
       <IrisVirginica is IsNot 1>

if <SepalLength is Long 0.502386> and <SepalWidth is Medium 0.746384> and
   <PetalLength is Medium 0.706567> and <PetalWidth is Medium 0.827206>
  then <IrisSetosa is IsNot 1> and <IrisVersicolor is Certain 1> and
       <IrisVirginica is IsNot 1>

if <SepalLength is Medium 0.639418> and <SepalWidth is Medium 0.640534> and
   <PetalLength is Medium 0.840115> and <PetalWidth is Medium 0.793157>
```

```
  then <IrisSetosa is Certain 0.770013> and
       <IrisVersicolor is UnCertain 0.993255> and <IrisVirginica is IsNot 0.605029>

[~RuleList]

[~FuzzyRules]
```