Manual for UM

Menaka Lashitha Bandara

Monash University (Clayton),
Victoria, Australia.

lashi@optusnet.com.au

Contents

Chapter 1

Using Console UI: `classify.exe`

1.1 Using `classify.exe`

The utility `classify.exe` is a simple, but powerful console based user interface for the full functionality of the `UserModelling` namespace.

This interface takes command line arguments to alter the behaviour of input, output and the EM algorithm. These arguments resemble the argument format of many Unix™ utilities. Thus, they come in two forms, long and short. It is possible to inter-mix the two forms of arguments when calling `classify.exe`. The tables below explain in detail each of these arguments and their respective parameters, if any. The first column shows the short argument, its respective long argument, and the third column, an explanation of the argument.

Mandatory Arguments

`-i`	`--in`	Takes one parameter - the filename from which to read input data.
`-c`	`--classes`	Takes one parameter. This is the number of classifications to be performed on the data paths.

Optional Arguments

`-o`	`--out`	Takes one parameter - the filename to which output should be written.
`-l`	`--log`	Takes one parameter - the filename to which the internal algorithm should write log data.
`-m`	`--multiple`	Takes one parameter which tells the output module that a path can belong to more than one cluster. The parameter value must be greater than or equal to 0 and less than 1. The value of this parameter determines the range each likelihood should fall within the maximum likelihood observed for that path to be included in another cluster.
`-s`	`--singletons`	Takes no arguments. This option tells the output module to to create an extra cluster to those requested to place all the paths with no transitions.
`-p`	`--probability`	Takes no arguments. This argument tells the output module to print the probability matrix.

Furthermore, these options can be displayed at runtime by executing this utility with no arguments or the argument `--help`.

1.2 Default behaviour of `classify.exe`

The mandatory arguments are necessary for the utility to work on a data set. When only the mandatory arguments are given, a few assumptions are made by the utility. These include:

- The output written by the utility is written to the console or terminal in which the utility was executed from.
- A path may only belong to one cluster. This is the maximum observed likelihood of that path in belonging to a particular cluster in the probability matrix. If there are ties, then the first path with that probability is included.
- Only paths with at least one transition is included in the output. Therefore, paths with zero transitions are not included into a separate cluster.
- The printing of the probability matrix is omitted when output is generated.

1.3 Example of using `classify.exe`

This is the typical use of the `classify.exe` utility:

```
classify.exe -i data_file -c 11 -m 0.01 -l log_file -s -o output_file -p
```

The behaviour expected from this utility is:

- The file to read input data from is `data_file`.
- The algorithm should make 11 classifications of the data.
- A path may be included in more than one cluster if the likelihood of belonging to a cluster is within 0.01 of the observed maximum likelihood.
- A log file should be written to the file `log_file`.
- Cluster 12 should include all paths with zero transitions.
- Output should be written to file `output_file`.
- The generated output should include the probability matrix.

1.4 Inputting Data

The EM algorithm can work with a total of $2^{16} = 65,536$ web pages. The input must be read in from a text file. Each web page must be represented by a single Unicode character, and each data path (ie, each user) must distinguished by either a newline character or a carriage return. Whitespaces are tolerated: that is tabs, and spaces are ignored by this program. Furthermore, lines that contain a newline as the first character are ignored.

These two examples are correct input data. They are identical to each other.

Example 1:

```
HAHHNNNNNNNNHHGGG
HAGGGGGGGGGG
HAHGGGGGGGGGGMMMAAAMMMHMMAAMAAMAAM
HAHNGGGG
HAAHHNNHHBBGGGGGGGGGHASSSSSSSSS
HAAHHGGGGGGHHHHHCCCCHCCCCHCCCHGGGGGGGGGG
HAAGGGGGGGGHNNHH
HAHNHBBBBHHHC
HAH
HAAHCGGH
HAHBG
HAHHHHHHGGGGGGHCHH
HAHHHHHHHHHHHHHBBAMMMMMMMMMMMMMMMMMMMMMMMMMM
HA
HAMMMMMMMMMMMMMMMMMMMMMMMGGGGGGGGGGGGGGGGGGGMMMMMMMGGGGGGGHGGHASSSSHSSSSSGHGH
HAAGGGGGGGGGGGHHCCHNNHCCCCCCC
```

Example 2:

```
H A HHNNNNNNNN    HHGGG

HAGGGGGGGGG
H AHGGGGGGGGGGMMMAAAMMMHMMAAMAAMAAM


H A H    NG   GGG
HAAHHN  NH HB BG GG   GGGGG GHA SS S SSS  SSS
   HAAHHGGGGGGHHHHHCCCCHCCCCHCCCHGGGGGGGGGG
HAAGGGGGGGGHNNHH
  HAHNHBBBBHHHC
HAH
HAAHCGGH
HAHBG
HAHHHHHHGGGGGGHCHH
HA HHHH HHHH H H HH B BA MM M   M M MM MMMMMMMMMMMMMMMM  MMMMM
HA
HAMMMMMMMMMMMMMMMMMMMMMMMGGGGGGGGGGGGGGGGGGGMMMMMMMGGGGGGGHGGHASSSSHSSSSSGHGH
 H A AG   G G  G G G G G GG G HHCCHNNHCCCCCC C
```

1.5 Example of Input Data and Output Data

In section Appendix 1, we display the input data of 100 data paths. The output data is presented in Appendix 2. We do not attach the log file since this is generally very large. We call this utility with the following options.

```
classify.exe -i data -o output -c 3 -p -s -m 0
```

Chapter 2

Programming with `namespace UserModelling`

2.1 Basic Programming using `namespace UserModelling`

We examine a simple example of how to read in a set of training data from a file and create a finite number of classifications on the data paths.

Firstly, we need to build the training data set such that the `ExpectationMaximisation` class understands. We use the `DataBuilder` class for that purpose. We firstly list the minimum number of variables required to use this namespace:

```
Stream                      stream;
DataBuilder                 build;
ushort [][]                 d_train;
ExpectationMaximisation     em;
double [,]                  probability_matrix;
```

We now construct the training data set as required by the class `ExpectationMaximisation` by using `DataBuilder`:

```
stream = File.OpenRead ("testfile");

try {
    build = new DataBuilder (stream);
}
catch (Exception ex) {
    Console.WriteLine (ex); return;
}

d_train = build.get_training_data ();
```

The variable `d_train` contains a complete data set that the class `ExpectationMaximisation` uses:

```
try {
     em = new ExpectationMaximisation (d_train, NUMBER_OF_CLUSTERS);
}
```

```
catch (Exception ex) {
     Console.Error.WriteLine (ex); return;
}
```

We need to tell the class `ExpectationMaximisation` explicitly to run the algorithm by calling the method `run()`.

```
try {
     em.run ();
}
catch (Exception ex) {
     Console.Error.WriteLine (ex); return;
}
```

We can now grab the probability matrix (the probability of each path belonging to a cluster) from the `em` object:

```
probability_matrix = em.get_probability_matrix ();
```

Using this matrix, we can now allocate each path to a cluster. Note that in order to translate from the alphabet that the class `ExpectationMaximisation` understands, back to a human readable format, the method `translate()` in the `DataBuilder` class must be used. An example would be: `build.translate (d_train [i][j])`, which converts the alphabetical state at point `j` of path `i` to the original symbolic state from the input data file. It is important to note that `d_train [i]` does not necessarily correspond to path `i` of the original data set.

Also note that the rows of the probability matrix correspond to indexes of `d_train`, and the columns to the index of each cluster.

2.2 Programming Recommendations

The `UserModelling` namespace provides two classes for the purposes of user-modelling. The first class `DataBuilder` learns an alphabet of a set of data, and maps this to consecutive positive integer states that the second class, `ExpectationMaximisation` understands. The `DataBuilder` class maintains a forward and reverse mapping tables for the purpose of bi-directional translations between the encountered alphabet in the original data stream, and the alphabet that the `ExpectationMaximisation` class understands.

In addition to this, an embedded namespace `Logging` provides an interface `IVerbose` and concrete classes `NullVerbosity` and `VerboseVerbosity` that allow you to instantiate this class with a stream that points to some logging destination. The class `NullVerbosity` is used so that no logging takes place, and `VerboseVerbosity` is instantiated with an associated stream to which log data is written.

It is quite possible to avoid using the `DataBuilder` class if the training data set `ushort [][] d_train` is built consistent with the specification that `ExpectationMaximisation` expects. However, this is strongly discouraged. It is advised that the `DataBuilder` class is used to construct this training data set.

The `DataBuilder` class *does not* understand the notions of files. It simply reads the input from a data stream that is given at the time the object is constructed. In order to read in data from obscure formats of data, two methods are recommended. The easiest, is to pre-process the data into a pipe or file that is opened as a stream such that the `DataBuilder` class can sanely read the data. The second technique would be to create a derived class of `System.Stream` and overload necessary methods such that it each element read refers directly to a Unicode character corresponding to a symbolic state.

The methods of the class `DataBuilder` are listed in §2.3. When attempting to interpret a forward-mapped symbolic state that class `ExpectationMaximisation` understands, the method `translate()` should always be used.

2.3 Interface: `class DataBuilder`

- `DataBuilder ( Stream raw_stream )`
  This is the constructor. It takes in a raw stream as an argument and returns nothing.
- `DataBuilder ( Stream raw_stream, IVerbose logging )`
  This is another (overloaded) constructor. Along with taking a raw stream as an argument, it also takes an `IVerbose` derived class as an argument. Generally, a reference to a `VerboseVerbosity` class is passed as the second argument.
- `ushort [][] get_training_data ()`
  This returns a multi-dimensional array (the training data set) of type `ushort`. It takes no arguments.
- `ushort [][] get_singleton_data ()`
  This returns a multi-dimensional array (of data that contains no transitions) of type `ushort`. It takes no arguments.
- `char translate ( ushort state )`
  Takes an alphabetic state (that exists in the reverse map) as a parameter and translates that into the Unicode character observed in the original data set. This method takes a `ushort` as an argument to translate.

2.4 Interface: `class ExpectationMaximisation`

Note: `ExpectationMaximisation` uses `UMType` as a type alias for type `double`.

- `ExpectationMaximisation ( ushort [][] d_train, uint k )`
  Constructor returns nothing. It takes a parameter - the input data set as it's first argument (of type `ushort`), and the second argument the number of classifications (or clusters) (of type `uint`).
- `ExpectationMaximisation ( ushort [][] d_train, uint k, IVerbose )`
  Another constructor (overloaded), which takes an `IVerbose` derived class as its last argument. This is typically a reference to an instance of class `VerboseVerbosity`.
- `set_alphas ( UMType _alpha_pi_k, UMType _alpha_i_j_k, UMType _alpha_t_k_j_l )`
  Sets the alpha values (which are added to eliminate zeros in the vectors and matrices). The first two parameters can be equal to zero for the value to be set internally. The last parameter must be between zero and one. If any argument is out of it's respective range, then the corresponding internal variable will not be set.
- `void run ()`
  This method tells the class to run the algorithm. It takes no arguments.
- `UMType [,] get_probability_matrix ()`
  This method returns the probability matrix that the algorithm creates. It takes no arguments.

2.5 Interface: `class Logging.VerboseVerbosity`

Note: This interface is derived from `IVerbose` (abstract class). Only the significant methods of this class are given. The others are only used by `DataBuilder` and `ExpectationMaximisation`.

- `VerboseVerbosity ( TextWriter Stream )`
  This takes a `TextWriter` or derived class of `TextWriter` that is associated with some destination to which to write the log to. This destination is typically `stdout` or a file.

2.6 Exceptions

- `InputStreamException`
  Invalid input data stream. The `DataBuilder` class cannot read from the data stream that has been handed over.
- `NonExistantStateException`
  The `DataBuilder` class does not contain a character state corresponding to the integer parameter.
- `EmptyDataSetException`
  Training data set is empty. The `ExpectationMaximisation` class finds no training data.
- `MalformedDataSetException`
  There is unprocessable data in the training data set. This occurs when `ExpectationMaximisation` finds a data path with zero transitions.
- `ZeroClusterException`
  The `ExpectationMaximisation` class attempts to classify data into zero clusters.
- `ClusterPathsException`
  The `ExpectationMaximisation` class is given more classifications than available data paths.

Chapter 3

Correctness of `namespace UserModelling`

3.1 Validity of `class DataBuilder`

The `DataBuilder` class performs the following tasks:

1. Learns the number of unique alphabet states.
2. Builds a forward mapping table - to translate each symbol appearing in the raw data set into a `ushort` integer.
3. Builds a reverse mapping table - to translate each `ushort` integer back into the symbol that denotes a unique alphabetic state as read from the raw stream.
4. Builds the training data set `d_train`.
5. Builds the singleton data set `s_train`.

These five specifications is easily verified by using a single test case. We design an input file that contains a known number of alphabetic states, and has singleton (data with no transitions) spread throughout the input file. Then, we feed this into the class and write the content read in. All the five points listed above must fully function in order for the input and output data to be identical semantically (they will not maintain the same form because the singletons are kept seperate to the training data set internally).

Thus, we use this following data set with five alphabetic states:

```
AAACEBBCEEDDABAAAABE
C
B
AECCCBEEDAAADBEAAA
ADEEAEE
D
CEEDD
A
E
```

The output generated is:

```
AAACEBBCEEDDABAAAABE
AECCCBEEDAAADBEAAA
ADEEAEE
CEEDD
C
B
D
A
E
```

It is obvious that without those five critical tasks that the `DataBuilder` class performs, the output would not contain the same contents as the input.

3.2 Validity of `class ExpectationMaximisation`

3.2.1 Testing the correct handling of input data

The `ExpectationMaximisation` class throws a number of exceptions to deal with a number of conditions that are impossible for this algorithm to work (as given in §2.6). In order to make sure that this class correctly deals with these conditions, a test case was designed in which these conditions were inflicted upon this class. For each of these conditions, the `ExpectationMaxmisation` class reacted correctly by throwing the appropriate exception.

3.2.2 Testing validity of the initialisation code

We test for the correct construction of $\theta = \{\pi, \theta^I, \theta^T\}$, from the source data by constructing an adequate, but workable data set, from which we could construct θ by hand, and then comparing this to the output of the initial phase of the algorithm.

It is important to note that we change the randomised path allocation code to allocate the paths to cluseters in an equal manner. If any excess paths are left over (because they cannot be distributed evenly), then we simply allocate the excess to the last cluster.

We use the following data set:

```
AAACE
AEC

ADEEAEE
ACEEDD

ABBC
ADAEEAB
```

From this data set, we see there must be two paths per cluster. Furthermore, we can see that the alphabetic states are learnt in the order `A C E D B`. Therefore, the matrices will be a two dimensional arrangement of the alphabetic states in this order. Therefore:

$$\pi = (0.333, 0.333, 0.333)$$

We can also see that for all clusters:

$$\theta^I = (1, 0, 0, 0, 0)$$

The cluster allocations are as follows:

$$\theta_0^T = \begin{bmatrix} 0.5 & 0.25 & 0.25 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\theta_1^T = \begin{bmatrix} 0 & 0.333 & 0.333 & 0.333 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0.2 & 0 & 0.6 & 0.2 & 0 \\ 0 & 0 & 0.5 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\theta_2^T = \begin{bmatrix} 0 & 0 & 0.25 & 0.25 & 0.5 \\ 0 & 0 & 0 & 0 & 0 \\ 0.5 & 0 & 0.5 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 & 0.5 \end{bmatrix}$$

The output we recive from the `ExpectationMaximisation` class is:

```
Using filename: ../testsets/test_em
Data paths: 6
Singletons: 0
Total: 6
Alphabet states: 5
d_train [][] built successfully!
pi = (0.333333333333333, 0.333333333333333, 0.333333333333333)
Cluster Assignments:
2 2 2

Theta^I_k, Cluster 0
1 0 0 0 0

Theta^I_k, Cluster 1
1 0 0 0 0

Theta^I_k, Cluster 2
1 0 0 0 0

Theta^T_k, Cluster 0:
0.5  0.25  0.25  0  0
0  0  1  0  0
0  1  0  0  0
0  0  0  0  0
0  0  0  0  0

Theta^T_k, Cluster 1:
0  0.333333333333333  0.333333333333333  0.333333333333333  0
0  0  1  0  0
0.2  0  0.6  0.2  0
0  0  0.5  0.5  0
0  0  0  0  0

Theta^T_k, Cluster 2:
0  0  0.25  0.25  0.5
0  0  0  0  0
0.5  0  0.5  0  0
```

```
1  0  0  0  0
0  0.5  0  0  0.5
```

3.2.3 Testing validity of the refinement code

We use the technique used in section §3.2.2 to test the refinement code. Firstly, we disable the random allocation of paths and then work each of the refinement equation by hand. Then we compare that to the output of the log created by the algorithm.

It is important to note that we do not follow the algorithm until convergence is reached. The convergence rule is a comparitively small algorithm, and we safely neglect it. Furthermore, it is rather tedious to perform all these calculations by hand, even for this tiny data set.

We test this code for correctness in a way similar to proof by induction. Since we know that the refinement code is the same at each refinement step, we only perform the calculations for one iteration. We induce that it is correct for all other steps because the first step matches the output generated by the first iteration.

Below, we show the different variables for which we perform these calculations by hand:

$$P(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\pi = (0.333, 0.333, 0.333)$$

$$\theta_1^I = (0.99602, 0.00099502, 0.0009502, 0.0009502, 0.0009502)$$

$$\theta_2^I = (0.99602, 0.00099502, 0.0009502, 0.0009502, 0.0009502)$$

$$\theta_3^I = (0.99602, 0.00099502, 0.0009502, 0.0009502, 0.0009502)$$

$$\theta_1^T = \begin{bmatrix} 0.49925 & 0.24988 & 0.24988 & 0.00049875 & 0.00049875 \\ 0.0019802 & 0.0019802 & 0.99208 & 0.0019802 & 0.0019802 \\ 0.0019802 & 0.99208 & 0.0019802 & 0.0019802 & 0.0019802 \\ 0.20000 & 0.20000 & 0.20000 & 0.20000 & 0.20000 \\ 0.20000 & 0.20000 & 0.20000 & 0.20000 & 0.20000 \end{bmatrix}$$

$$\theta_2^T = \begin{bmatrix} 0.00066445 & 0.33289 & 0.33289 & 0.33289 & 0.00066445 \\ 0.0019802 & 0.0019802 & 0.99208 & 0.0019802 & 0.0019802 \\ 0.20000 & 0.00039920 & 0.59920 & 0.20000 & 0.00039920 \\ 0.00099502 & 0.00099502 & 0.49851 & 0.49851 & 0.00099502 \\ 0.20000 & 0.20000 & 0.20000 & 0.20000 & 0.20000 \end{bmatrix}$$

$$\theta_3^T = \begin{bmatrix} 0.00049875 & 0.00049875 & 0.24988 & 0.24988 & 0.49925 \\ 0.20000 & 0.20000 & 0.20000 & 0.20000 & 0.20000 \\ 0.49851 & 0.00099502 & 0.49851 & 0.00099502 & 0.00099502 \\ 0.99208 & 0.0019802 & 0.0019802 & 0.0019802 & 0.0019802 \\ 0.00099502 & 0.49851 & 0.00099502 & 0.00099502 & 0.49851 \end{bmatrix}$$

The output generated by the algorithm is as follows (for the iterative steps that have been calculated above):

```
Probability Matrix:
1 0 0
1 0 0
0 1 0
```

```
0 1 0
0 0 1
0 0 1

pi = (0.333333333333333 0.333333333333333 0.333333333333333)

Theta^I, cluster 1:
0.996019900497513 0.000995024875621891 0.000995024875621891
        0.000995024875621891 0.000995024875621891
Theta^I, cluster 2:
0.996019900497513 0.000995024875621891 0.000995024875621891
        0.000995024875621891 0.000995024875621891
Theta^I, cluster 3:
0.996019900497513 0.000995024875621891 0.000995024875621891
        0.000995024875621891 0.000995024875621891

Theta^T, cluster 1:
0.49925187032419 0.249875311720698 0.249875311720698
        0.000498753117206983 0.000498753117206983
0.0019801980198 0.0019801980198 0.992079207920792
        0.0019801980198 0.0019801980198
0.0019801980198 0.992079207920792 0.0019801980198
        0.0019801980198 0.0019801980198
0.2 0.2 0.2
        0.2 0.2
0.2 0.2 0.2
        0.2 0.2

Theta^T, cluster 2:
0.000664451827242525 0.332890365448505 0.332890365448505
        0.332890365448505 0.000664451827242525
0.0019801980198 0.0019801980198 0.992079207920792
        0.0019801980198 0.0019801980198
0.2 0.000399201596806387 0.599201596806387
        0.2 0.000399201596806387
0.000995024875621891 0.000995024875621891 0.498507462686567
        0.498507462686567 0.000995024875621891
0.2 0.2 0.2
        0.2 0.2

Theta^T, cluster 3:
0.000498753117206983 0.000498753117206983 0.249875311720698
        0.249875311720698 0.499251870324189
0.2 0.2 0.2
        0.2 0.2
0.498507462686567 0.000995024875621891 0.498507462686567
        0.000995024875621891 0.000995024875621891
0.992079207920792 0.0019801980198 0.0019801980198
        0.0019801980198 0.0019801980198
0.000995024875621891 0.498507462686567 0.000995024875621891
        0.000995024875621891 0.498507462686567
```

References and Development Information

The work presented in this document is based on the paper:

Cadez, I, et al. *Model-Based Clustering and Visualization of Navigation patterns on a Web Site.* ftp://ftp.research.microsoft.com/pub/tr/tr-2000-18.pdf

The development of this algorithm was done using the **C#** language. The primary development was on a Linux^{TM} `powerpc` machine using the **mcs** compiler (www.go-mono.org), a *.NET* implementation for Unix^{TM} . This application was tested and run successfully on a Windows^{TM} NT machine and compiled without errors under **Microsoft Visual Studio .NET**.

All source code was written by **Menaka Lashitha Bandara** (`lashi@optusnet.com.au`) as a project for **SummerVac 2002/2003, Monash University, Clayton, Victoria, Australia**.

Appendix 1

```
HAHHNNNNNNNNHHGGG
HAGGGGGGGGGG
HAHGGGGGGGGGGMMMAAAMMMHMMAAMAAMAAM
HAHNGGGG
HAAHHNNHHBBGGGGGGGGGHASSSSSSSSS
HAAHHGGGGGGHHHHHCCCCHCCCCHCCCHGGGGGGGGGG
HAAGGGGGGGGHNNHH
HAHNHBBBBHHHC
HAH
HAAHCGGH
HAHBG
HAHHHHHHGGGGGGHCHH
HAHHHHHHHHHHHHHBBAMMMMMMMMMMMMMMMMMMMMMMMMMM
HA
HAMMMMMMMMMMMMMMMMMMMMMMGGGGGGGGGGGGGGGGGGGMMMMMMMGGGGGGGHGGHASSSSHSSSSSGHGH
HAAGGGGGGGGGGGHHCCHNNHCCCCCCC
HACCH
HAHHHHHH
HAHGG
HASSSS
HAH
HAAHCGGHCCHBBH
HAAHBBHCCCCHCCHGGGGH
HAAGGGGGGGGGGGHBBBBBBBHHBHHTTGBBBBHBGGHTTTHMMMH
HAHTHBBBBBHG
HAHBGGDDDMDFFFFFDD
HAHCCHGGGGHNNNHH
HASSSAN
HAHTTNNAHBMMHTT
HAHTTHTTHDDFDDFDFFFHTTHBHH
HAAAAAA
HAHBBBGGGG
HAAHAAAAAAAAAHAAAA
HAHAA
HAAHMMHB
HAAGGHFFFFFFF
HAHTTTHFFFFHH
HAAHHGGGGGGGGGG
HAAHGGHCC
HAAHBA
H
H
H
H
HAAHHAACCH
```

HASSSGGG
HAHB
HAAGGGGGGHGGGGGGGGGGGGGGGGGGGGGGGGMMMMGGGGGGHHBBHHAAHHHHHMMMGGGGGGGGGHBBHGGGGGG
HAAHCCHBGGGGGGHHNNH
HAAAAAA
HAAHHBGG
HAAGGHHHBBMM
HASSSAAAAAAAHAAAAAAAAAAAH
HAAAAAAAHHAAAAAAAAAAA
HA
HAAAAAAAAAAAAAAAAAAAA
HAAHHBBG
HAHBBBH
HAAHBBHBBHCFFFFFFFFHGGFFFFFF
HAAMM
HHHHBBBGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGHH
HHHCCGGGGGG
HHHCCHHHTTTHHTTTTHTTTTHHMMGGGG
HHHCCCCC
HHH
H
H
HNNHHCCCCHH
HHAASSSSSSSSS
HHHTTTAMMHMMAASSSSSSSSSSSSSSSHHSSAASSSS
HHBHTTT
HH
HAAHGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGHHGGGGGGGGGG
HAAHHCCCCHHBBBBBBHHHNHBBBB
HAAAAA
HAAHHBBBHBBBHH
HAAAAAAAAAAA
HAAAAAAAAAAAA
HAAAAAAAAAAAAAA
HAAHBB
HAAHHAASSSSS
HAAHGGGGGGGGHHHFFFFFFFFGGF
HAHFFFFFFFFGGFFFFGHGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGH
HAAHBBGGGGGGGGGGGGGGGGGGGHTTTTTHBBBBBBAANNNNNH
HAAGGGGGGGGHBBBHGGGGGG
HAAGGGGGGGGGGGGGGGGHTT
HAAHGGGGGGGGGGGGGG
HAAHBHBHCCCCHGGGG
HAAHGGGGGGGGGGGGGGGGGAAAAAAAAAAAGHHAASSSSSSSSSSSSSSSS
HAAHGGHMHHM
HAAAHAAHPPPPPPDDFFFPHH
HAAASSSSSSSSSSHAASSHAASSSS
HASSSSSSSSS
HAAGGGGGGGGGHBBBBHHHHTTTHCCCCCCH
HHTTTTAAAAAAAAAAAHTTTGGHAAAAHH
HAAHGGGGGGGGGGGGGGHTTHHGGGGGGGHAASSSSSSSAA
HAAHTT
HAAHGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGG
HAAHAA
HAAAAPPPPHPPPAAAAPPPPPAPPAAAPPPPPPHAAAAPPPPPPPPAAAPPPP

Appendix 2

```
*** Probability Matrix ***
0.999988368705329 7.19050168546129e-26 1.16312946710805e-05
0.849291810986282 0.0521882473269126 0.098519941686805
4.41257989246422e-24 0.999999998178963 1.82103673856901e-09
0.000652961217853775 0.99934696923723 6.95449160420525e-08
0.999978946725874 1.98575441510352e-11 2.10532542684762e-05
0.999999010216845 9.8978315499997e-07 2.04794394788901e-23
0.999971322344497 4.52227318493737e-08 2.86324327711179e-05
0.999989512910441 1.04863673658856e-05 7.22193256210255e-10
0.676353579864686 0.150881663700525 0.172764756434789
0.105580380629862 0.894359387643808 6.02317263294228e-05
0.820426300357849 0.000291409394372589 0.179282290247778
0.999994646096148 5.20911875719149e-06 1.44785095336894e-07
0.999999999996796 9.82339672352275e-19 3.20419164962367e-12
0.386759942318281 0.208917244815089 0.404322812866629
1 4.10700533858707e-17 2.22571090327964e-20
0.999999998269627 1.73037348364329e-09 4.6072032969511e-20
0.999773963262767 0.000218915392883526 7.12134434914941e-06
0.99274365248787 1.64538336357676e-05 0.0072398936784946
0.659337985912983 0.225494482621021 0.115167531465996
0.45496183826147 0.216424172482379 0.328613989256151
0.676353579864686 0.150881663700525 0.172764756434789
0.080646344536406 0.919353655283403 1.80190616023167e-10
0.799080223472322 0.200919776527541 1.37630543373134e-13
4.27385677821766e-12 6.08091019457928e-19 0.999999999995726
0.852247027253984 3.65138114373072e-05 0.147716458934579
0.999999999723084 3.12390762507994e-15 2.76912688490863e-10
0.99999999979677 1.13206002249888e-10 9.00244046641532e-11
1.3359759608903e-06 6.83108041856569e-07 0.999997980915997
5.13268060320821e-12 4.61334666280844e-18 0.999999999994867
0.999999999330416 1.83712405584685e-30 6.69584301504985e-10
0.0154213522562243 0.0675880802521871 0.916990567491589
0.822655799387656 0.000143056624726185 0.177201143987618
0.000284035537435203 0.0137346642183775 0.985981300244187
0.371885351325071 0.234251998867336 0.393862649807593
1.19564327302866e-05 0.0553595862276676 0.944628457339602
0.0307598150524951 0.968710596411389 0.000529588536115424
0.994607143931366 5.76165080072885e-07 0.00539227990355359
0.871798612207019 0.0303135155708587 0.0978878722221225
0.573021492998884 0.426972215495561 6.29150555507728e-06
0.406065426650841 0.00396031385366808 0.58997425949549
0.999934090123264 5.7959377893859e-05 7.95049884208576e-06
0.999308733364841 0.000520313658150226 0.000170952977009175
0.659480140590464 0.15070906207796 0.189810797331576
0.999999999998612 9.98674156875778e-13 3.89732370877468e-13
```

```
0.999999999719116 1.00767889151768e-10 1.80115968848695e-10
0.0154213522562243 0.0675880802521871 0.916990567491589
0.815091564912075 6.01099473340578e-05 0.184848325140591
0.713691680092216 1.70692719758075e-05 0.286291250635809
1.91199051587911e-08 2.81742641176722e-06 0.999997163453683
4.77331254241853e-05 0.000235658230190287 0.999716608644385
0.386759942318281 0.208917244815089 0.404322812866629
2.04995600585559e-07 0.000315713953067967 0.999684081051332
0.792659332883392 5.11258556378892e-05 0.20728954126097
0.646187256138258 0.215534014731714 0.138278729130027
2.37116994050261e-10 0.999999999541218 2.21664883354353e-10
0.402888233512931 0.493372953920641 0.103738812566428
0.997081149499776 2.78362408381148e-09 0.00291884771660044
0.960041781413245 0.0399512841800798 6.93440667560615e-06
0.999999999841695 3.94206861259987e-28 1.58304698985969e-10
0.998761747109329 0.00123824923964956 3.65102128616539e-09
0.874965064088349 0.0056746861627322 0.119360249748919
0.999999997197024 2.80186877852074e-09 1.10694039975924e-12
0.517417413516329 0.0530171562898762 0.429565430193795
3.12684230847745e-15 9.57021993588013e-13 0.99999999999904
0.621237607898335 4.74063662948974e-07 0.378761918038002
0.756867195540276 0.032868308544386 0.210264495915338
0.996792614868102 0.000288520840801391 0.00291886429109695
0.999999978445911 2.15537312430272e-08 3.57420587706371e-13
0.032912353063143 0.0948994327917779 0.872188214145079
0.897744576645575 0.0151787103468554 0.0870767130075691
0.000292504524155287 0.0104014788875234 0.989306016588321
0.000130819295207593 0.00707094207237182 0.992798238632421
2.60855581417559e-05 0.00325755343353626 0.996716361008322
0.505228230468054 0.140385363238317 0.354386406293629
0.516071987703484 0.0643073446510042 0.419620667645511
3.60397053102849e-08 0.999999841065531 1.2289476378946e-07
9.08102604266184e-13 0.999999999120579 8.78512763770833e-10
0.000589936528458167 2.21107238316766e-31 0.999410063471542
0.654893840951729 0.192630501805232 0.152475657243039
0.517331843840896 2.72162385913651e-06 0.482665434535244
0.730354093184162 0.130100439489288 0.13954546732655
0.616853991990258 0.38314600023661 7.77313166621272e-09
7.67612645793144e-07 4.78896302819158e-06 0.999994443424326
2.0511146403271e-06 0.0170337596578886 0.982964189227471
0.999999998183527 1.71788499335262e-09 9.85882152560748e-11
0.0121863374072375 0.832170338425668 0.155643324167094
0.453543296101614 0.204718791111479 0.341737912786907
0.9999999998181 1.25327449335948e-10 5.65723298765018e-11
2.05619717059782e-11 7.02918295074975e-16 0.999999999979437
0.000640531782471319 3.02568022035444e-09 0.999359465191848
0.374275620604091 7.43538849345134e-06 0.625716944007416
0.948816307216424 0.0151843102269814 0.0359993825565951
0.230739459502532 0.220922055230531 0.548338485266938
1.38967582077493e-40 1 4.98428439168828e-53

*** Cluster 1 ***
1. HAHHNNNNNNNNHHGGG
2. HAGGGGGGGGG
5. HAAHHNNHHBBGGGGGGGGHASSSSSSSSS
6. HAAHHGGGGGGHHHHHCCCCHCCCCHCCCHGGGGGGGGGG
7. HAAGGGGGGGGHNNHH
8. HAHNHBBBBHHHC
```

```
9. HAH
11. HAHBG
12. HAHHHHHHGGGGGGHCHH
13. HAHHHHHHHHHHHHBBAMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM
15. HAMMMMMMMMMMMMMMMMMMMMMMGGGGGGGGGGGGGGGGGGGMMMMMMMMGGGGGGGHGGHASSSSHSSSSSGHGH
16. HAAGGGGGGGGGGGHHCCHNNHCCCCCCC
17. HACCH
18. HAHHHHHH
19. HAHGG
20. HASSSS
21. HAH
23. HAAHBBHCCCCHCCHGGGGH
25. HAHTHBBBBBHG
26. HAHBGGDDDMDFFFFFDD
27. HAHCCHGGGGHNNNHH
30. HAHTTHTTHDDFDDFDFFFHTTHBHH
32. HAHBBBGGGG
37. HAHTTTHFFFFHH
38. HAAHHGGGGGGGGGGG
39. HAAHGGHCC
41. HAAHHAACCH
42. HASSSGGG
43. HAHB
44. HAAGGGGGHGGGGGGGGGGGGGGGGGGGGGGGGGMMMMGGGGGGHHBBHHAAHHHHHMMMGGGGGGGGGHBBHGGGGGG
45. HAAHCCHBGGGGGGHHNNH
47. HAAHHBGG
48. HAAGGHHHBBMM
53. HAAHHBBG
54. HAHBBBH
57. HHHHBBBGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGHH
58. HHHCCGGGGGG
59. HHHCCHHHTTTHHTTTTHTTTTHHMMGGGG
60. HHHCCCCC
61. HHH
62. HNNHHCCCCHH
63. HHAASSSSSSSSS
65. HHBHTTT
66. HH
67. HAAHGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGHHGGGGGGGGGG
68. HAAHHCCCCHHBBBBBBHHHNHBBBB
70. HAAHHBBBHBBBHH
74. HAAHBB
75. HAAHHAASSSSS
79. HAAGGGGGGGGHBBBHGGGGGG
80. HAAGGGGGGGGGGGGGGGGHTT
81. HAAHGGGGGGGGGGGGGG
82. HAAHBHBHCCCCHGGGG
85. HAAAHAAHPPPPPPDDFFFPHH
87. HASSSSSSSSS
88. HAAGGGGGGGGGHBBBBHHHHTTTHCCCCCCH
92. HAAHGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGG

*** Cluster 2 ***
3. HAHGGGGGGGGGGMMMAAAMMMHMMAAMAAMAAM
4. HAHNGGGG
10. HAAHCGGH
22. HAAHCGGHCCHBBH
36. HAAGGHFFFFFFF
```

```
55. HAAHBBHBBHCFFFFFFFFHGGFFFFFF
56. HAAMM
76. HAAHGGGGGGGGGHHHFFFFFFFGGF
77. HAHFFFFFFFFGGFFFFGHGGGGGGGGGGGGGGGGGGGGGGGGGGGH
86. HAAASSSSSSSSSSHAASSHAASSSS
94. HAAAAPPPPHPPPAAAAPPPPPAPPAAAPPPPPPHAAAAPPPPPPPPAAAPPPP

*** Cluster 3 ***
14. HA
24. HAAGGGGGGGGGGGHBBBBBBBBHHBHHTTGBBBBHBGGHTTTHMMMH
28. HASSSAN
29. HAHTTNNAHBMMHTT
31. HAAAAAA
33. HAAHAAAAAAAAAHAAAA
34. HAHAA
35. HAAHMMHB
40. HAAHBA
46. HAAAAAA
49. HASSSAAAAAAAHAAAAAAAAAAAH
50. HAAAAAAAHHHAAAAAAAAAA
51. HA
52. HAAAAAAAAAAAAAAAAAAA
64. HHHTTTAMMHMMAASSSSSSSSSSSSSSSHHSSAASSSS
69. HAAAAA
71. HAAAAAAAAAAA
72. HAAAAAAAAAAAA
73. HAAAAAAAAAAAAA
78. HAAHBBGGGGGGGGGGGGGGGGGHTTTTTHBBBBBBBAANNNNNH
83. HAAHGGGGGGGGGGGGGGGGAAAAAAAAAAAGHHAASSSSSSSSSSSSSSSS
84. HAAHGGHMHHM
89. HHTTTTAAAAAAAAAAAHTTTGGHAAAAHH
90. HAAHGGGGGGGGGGGGGHTTHHGGGGGGGGHAASSSSSSSAA
91. HAAHTT
93. HAAHAA

*** Cluster 4 ***
94. H
95. H
96. H
97. H
98. H
99. H
```

Appendix 3

Build options:

```
$(C_HASH_COMPILER) /target:library /out:UserModelling.dll \
        em.cs databuilder.cs verbose.cs exception.cs ll.cs
```

File: `exception.cs`

```
namespace UserModelling {
  using System;

  public class InputStreamException : Exception {
    public InputStreamException (string s) : base (s) { }
  };

  public class NonExistantStateException : Exception {
    public NonExistantStateException (string s) : base (s) { }
  };

  public class EmptyDataSetException : Exception {
    public EmptyDataSetException (string s) : base (s) { }
  };

  public class MalformedDataSetException : Exception {
    public MalformedDataSetException (string s) : base (s) { }
  };

  public class ZeroClusterException : Exception {
    public ZeroClusterException (string s) : base (s) { }
  };

  public class ClusterPathsException : Exception {
    public ClusterPathsException (string s) : base (s) { }
  };

  public class NullLoggingObjectException : Exception {
    public NullLoggingObjectException (string s) : base (s) { }
  };
};
```

File: `em.cs`

```
namespace UserModelling {
  using UserModelling.LinkedList;
```

```
using UserModelling.Logging;
using System;

using UMType = System.Double;

/* The EM Class */
public class ExpectationMaximisation {
  // __x should be read as X.
  private uint     __m;     // Elements of alphabet.
  private uint    __n;     // Number of data paths.
  private uint    __k;     // Number of clusters.

  private ushort [][]  d_train;   // Data Paths.
  private UMType [,]  prob_theta;   // Probability Matrix (N x K).

  // Set Theta:
  private UMType [][]  theta_i;   // Initial Parameters (M vectors).
  private UMType [][,]  theta_t;   // Transitional Matrices (M x M).
  private UMType []  pi;     // Mixture Weights (K set).

  // Log posterior values:
  private UMType    log_post_old;  // Old Log...
  private UMType    log_post_cur;  // Cur Log...

  // Fudge factors
  private UMType    alpha_pi_k;  // Mixture weight fudge.
  private UMType    alpha_i_k_j;   // Initial State Probability fudge.
  private UMType    alpha_t_k_j_l;  // Transition Probability fudge.

  // Logging!
  private IVerbose  logging;  // Logging.


  /* Interface. */

  public
  ExpectationMaximisation ( ushort [][] d_train, uint k )
  {
    if ( d_train == null )
      throw new EmptyDataSetException ("Cannot work with empty data set!");


    this.d_train = d_train;
    this.__n = (uint) d_train.Length;

    this.__k = k;

    this.logging = new NullVerbosity ();

    // Defaults...
    this.alpha_pi_k = 0;
    this.alpha_i_k_j = 0;
    this.alpha_t_k_j_l = 0; // bit of a misnomer, this default changes...

    // Exceptions:
    if ( this.__k == 0 )
      throw new ZeroClusterException ("Cluster size must be greater than zero!");
```

```
  if ( this.__n < this.__k )
    throw new ClusterPathsException ("There cannot be more clusters than paths!");
}

public
ExpectationMaximisation ( ushort [][] d_train, uint k, IVerbose log ) : this (d_train, k)
{
  if ( log == null )
    throw new NullLoggingObjectException ("A non-existant logging object was handed over!");

  this.logging = log;
}

public void
set_alphas (UMType _alpha_pi_k, UMType _alpha_i_k_j, UMType _alpha_t_k_j_l)
{
  if ( (_alpha_pi_k >= 0) && (_alpha_pi_k < 1) )
    this.alpha_pi_k = alpha_pi_k;

  if ( (_alpha_i_k_j >= 0)  && (_alpha_i_k_j < 1) )
    this.alpha_i_k_j = _alpha_i_k_j;

  // This must be greater than 0 and less than 1. Otherwise
  // we're in deep shit.
  if ( (_alpha_t_k_j_l > 0 ) && (_alpha_t_k_j_l < 1) )
    this.alpha_t_k_j_l = _alpha_t_k_j_l;

  return;
}

public void
run ()
{
  /* Neccessary (and boring) stuff */
  this.determine_alphabet_lengths ();
  this.create_default_alphas ();
  this.allocate_memmory_theta_i ();
  this.allocate_memmory_theta_t ();
  this.allocate_memmory_pi ();

  /* Initialisation Code */
  this.create_theta ();

  /* Refinement Code */
  this.allocate_memmory_prob_theta ();
  this.init_log_post ();

  this.compute_probability_matrix ();

  this.refine ();

  return;
}

public UMType [,]
get_probability_matrix ()
{
  return prob_theta;
```

```
}


/* Private methods... only used by class. */

private void
determine_alphabet_lengths ()
{
  uint  i, j;

  /* We make the alphabet initially 0. This is
   * still impossible though, because for there
   * to be a transition, there must be at least
   * two states */
  this.__m = 1;

  for ( i = 0; i < this.__n ; i++ ) {

    /* Whoever calls this with no transitions
     * need to get his/her head checked. If
     * you get this exception, make sure you read
     * the docs! */
    if ( this.d_train [i].Length < 2 )
      throw new MalformedDataSetException ("Data Set Contains a no transition element!");

    for ( j = 0; j < this.d_train [i].Length ; j++ ) {

      /* Larger alphabetical state */
      if ( this.d_train [i][j] > this.__m )
        this.__m = this.d_train [i][j];
    }
  }

  // We begin with alphabet at 0. So,
  // The number of states in alphabet is
  // one more than appears in the training data path.
  this.__m++;

  return;
}

private void
create_default_alphas ()
{
  // If not already set by set_alphas ()
  if ( alpha_t_k_j_l == 0 )
    alpha_t_k_j_l = 0.01/this.__m;

  if ( alpha_i_k_j == 0 )
    alpha_i_k_j = 0.01/this.__m;

  return;
}

private void
allocate_memmory_theta_i ()
{
  uint  i;
```

```
  // K clusters...
  this.theta_i = new UMType [this.__k][];

  // theta_i is an M-vector...
  for ( i = 0; i < this.__k; i++ )
    this.theta_i [i] = new UMType [this.__m];

  return;
}

private void
allocate_memmory_theta_t ()
{
  uint  i;

  this.theta_t = new UMType [this.__k][,];


  // theta_t is an M x M matrix...
  for ( i = 0; i < this.__k; i++ ) // M x M matrix.
    this.theta_t [i] = new UMType [this.__m, this.__m];

  return;

}

private void
allocate_memmory_pi ()
{
  this.pi = new UMType [this.__k];

  return;
}

private void
allocate_memmory_prob_theta ()
{
  this.prob_theta = new UMType [this.__n, this.__k];

  return;
}

private void
create_theta ()
{
  uint    k;    // Indexer
  List []    cluster_list;   // Linked List of path assignments.
  System.Random  rng;    // Random Number generator object.

  // Temporary test code...
  uint    paths_per_cluster;
  uint    i,j;

  // Init Random Number Generator
  rng = new System.Random ();
```

```
  // Init Cluster Linked Lists...
  cluster_list = new List [this.__k];
  for ( k = 0; k < this.__k ; k++ )
    cluster_list [k] = new List ();

  // Randomly assign paths to clusters...
  // Note: i denotes the current path we're working on.
  for ( k = 0; k < this.__n ; k++ )
    cluster_list [rng.Next ((int) this.__k)].insert (k);

  // Un-random this bit:
  /*
  paths_per_cluster = this.__n / this.__k;

  for ( k = 0, j = 0; k < this.__k ; k++ )
    for ( i = 0; i < paths_per_cluster; i++ , j++ )
      cluster_list [k].insert (j);

  // Last Cluster assignment...
  for ( ; j < this.__n; j++ )
    cluster_list [k - 1].insert (j);
    */

  // Create pi:
  this.create_pi (cluster_list);
  this.logging.print (this.pi);


  // Create theta_i:
  for ( k = 0; k < this.__k ; k++ )
    this.create_theta_i_k (k, cluster_list [k]);

  this.logging.print (this.theta_i);


  // Create theta_t:
  for ( k = 0; k < this.__k ; k++ )
    this.create_theta_t_k (k, cluster_list [k]);

  this.logging.print (this.theta_t);

  return;

}

private void
create_pi (List [] cluster_list)
{
  uint  k;

  for ( k = 0; k < this.__k ; k++ )
    this.pi [k] =
      ((UMType) cluster_list [k].length) /
      ((double) this.__n );
  return;
}

private void
```

```
create_theta_i_k (uint k, List cluster)
{
  uint  i;
  uint  path;

  cluster.reset ();

  // We havn't got anything in this cluster.
  if ( cluster.length == 0 )
    return;

  for ( i = 0 ; i < cluster.length ; i++ ) {
    path = cluster.get_next ();

    // We look at the first element of each path
    // in the cluster, and increment that by one
    // in the vector.
    this.theta_i [k] [ this.d_train [path] [0] ]++;
  }

  // We normalise what we got before... nice :)
  for ( i = 0 ; i < this.__m ; i++ )
    this.theta_i [k] [i] /= (UMType) cluster.length;

  return;
}

private void
create_theta_t_k (uint k, List cluster)
{
  uint    i, j;    // Indexers...
  uint    path;    // Path index...
  uint []    occurance_list; // Occurance list for each alphabetical m.

  // We normalise the matrix using this after
  // it's built.
  occurance_list = new uint [this.__m];

  for ( i = 0; i < this.__m ; i++ )
    occurance_list [i] = 0;

  // We reset the cluter path data
  cluster.reset ();

  // We first count the number of transitions, and
  // number of occrances of each alphabetical state for
  // all paths of this cluster.

  for ( i = 0; i < cluster.length; i++ ) {
    path = cluster.get_next ();

    // Note we start on the second element,
    // and we look back to compare!
    for ( j = 1; j < this.d_train [path].Length; j++ ) {

      // (theta^t_k)_(d_train [path] [j-1], d_train [path] [j])++
      theta_t [k] [ this.d_train [path][j - 1], this.d_train [path][j] ]++;
```

```
        occurance_list [ this.d_train [path] [j-1] ]++;
      }

      //occurance_list [ this.d_train [path] [j-1] ]++;
    }

    // Now we normalise.
    for ( i = 0; i < this.__m ; i++ )
      if ( occurance_list [i] != 0)
        for ( j = 0; j < this.__m; j++ )
          this.theta_t [k] [i,j] /= ((UMType) occurance_list [i]);

    // Ahhh...
    return;
  }


  /* Refinement Code */

  private void
  refine ()
  {
    uint  i;

    // Initially, we run this, so that alpha_t_k_j_l
    // gets included into set theta.
    this.compute_mixture_weights ();
    this.compute_initial_state_probabilities ();
    this.compute_transitional_probabilites ();
    this.init_log_post (); // We reset log_p_cur & log_p_old.
    this.compute_probability_matrix ();

    // Upto here, we're ensured that inf can never
    // occur in any log, because there are no __zero__
    // probabilities... ie, log (0) is bad!
    this.convergence ();

    // At each iteration, we are guarenteed
    // that log_p_old will contain the previous
    // log_p_cur value, and log_p_cur = 0, because
    // convergence () resets it to this condition.
    for ( i = 0; i < 1000 ; i++ ) {
      this.compute_mixture_weights ();
      this.compute_initial_state_probabilities ();
      this.compute_transitional_probabilites ();
      this.compute_probability_matrix ();

      if ( this.convergence () )
        break;
    }

    if ( i == 1000 )
      this.logging.print ("No Convergence!");
    else
      this.logging.print (i + 1);

    return;
```

```
}

private bool
convergence ()
{
  uint    i;
  UMType    convergence_val;
  bool    val;

  val = false;

  convergence_val = Math.Abs ( (Math.Abs (this.log_post_cur) -
        Math.Abs (this.log_post_old) ) /
      this.log_post_cur );

  this.logging.print (this.log_post_old,
      this.log_post_cur, convergence_val * 100);

  if ( convergence_val < (0.01/100) )
    val = true;

  // Reset...
  this.log_post_old = this.log_post_cur;
  this.log_post_cur = 0;

  return val;
}

private void
init_log_post ()
{
  this.log_post_old = 0;
  this.log_post_cur = 0;
}

private void
compute_probability_matrix ()
{
  uint  i, k, t;

  for ( i = 0; i < this.__n ; i++ )
    compute_prob_theta_i (i);

  this.logging.print (prob_theta);


  return;
}

private void
compute_prob_theta_i (uint i)
{
  uint     k;
  UMType    ret_val;
  UMType    normal;

  // We simply do this computation.
  for ( k = 0, normal = 0; k < this.__k ; k++ ) {
```

```
      prob_theta [i,k] =
        this.pi [k] *
        this.theta_i [k] [this.d_train [i][0]] *
        theta_t_k_multi (i,k);

      // We also compute normal here.
      normal += this.prob_theta [i,k];
    }

    // Hmm... are you sure?
    this.log_post_cur += Math.Log (normal);

    // Now we normalise the prob matrix.
    for ( k = 0; k < this.__k; k++ )
      this.prob_theta [i,k] /= normal;


    return;
  }

  private UMType
  theta_t_k_multi (uint i, uint k)
  {
    uint    t;
    UMType    ret_val;

    for ( t = 1, ret_val = 1 ; t < d_train [i].Length ; t++ )
      ret_val *= theta_t [k] [ this.d_train [i] [t-1], this.d_train [i] [t] ];

    return ret_val;
  }

  private void
  compute_mixture_weights ()
  {
    uint    k, i;
    UMType    pi_k;
    UMType    normal;

    // Compute the vals and normal
    for ( k = 0, normal = 0; k < this.__k ; k++ ) {
      for ( i = 0, pi_k = 0 ; i < this.__n ; i++ )
        pi_k += this.prob_theta [i,k];

      // We also use an additive - the mixture weight fudge
      this.pi [k] = pi_k + alpha_pi_k;

      normal += this.pi [k];
    }

    // Normalise
    for ( k = 0; k < this.__k ; k++ )
      this.pi [k] /= normal;

    this.logging.print (this.pi);

    return;
```

```
}

private void
compute_initial_state_probabilities ()
{
  uint    k;

  for ( k = 0; k < this.__k ; k++ )
    compute_theta_i_k (k);

  this.logging.print (this.theta_i);

  return;
}

private void
compute_theta_i_k (uint k)
{
  uint    j, i;
  UMType    sum_over_n;
  UMType    normal;

  // Compute values initally for \Theta^I_k
  for ( j = 0, normal = 0; j < this.__m ; j++ ) {
    for ( i = 0, sum_over_n = 0 ; i < this.__n ; i++ )
      sum_over_n +=
        this.prob_theta [i,k] *
        this.delta (this.d_train [i][0], j);

    this.theta_i [k] [j] = sum_over_n + alpha_i_k_j;

    normal += this.theta_i [k] [j];
  }

  // Normalise.
  for ( j = 0; j < this.__m ; j++ )
    this.theta_i [k] [j] /= normal;

  return;
}

private uint
delta (ushort x_i_init, uint j)
{
  if ( ((uint) x_i_init ) == j )
    return 1;

  return 0;
}

private void
compute_transitional_probabilites ()
{
  uint    k;

  for ( k = 0; k < this.__k ; k++ )
    compute_theta_t_k (k);
```

```
  this.logging.print (this.theta_t);

  return;

}

private void
compute_theta_t_k (uint k)
{
  uint    j, l;
  UMType    normal;


  for ( j = 0; j < this.__m ; j++ ) {
    for ( l = 0, normal = 0; l < this.__m ; l++ ) {
      this.theta_t [k] [j,l] =
        iteration_prob_theta_t_n (j, l, k) +
        alpha_t_k_j_l;

      normal += this.theta_t [k] [j,l];
    }

    for ( l = 0 ; l < this.__m ; l++ )
      // Sometimes... this can be the way it is!
      // Because if alpha_t_k_j_l == 0,
      // we're in deep shit!
      if ( normal != 0 )
        this.theta_t [k] [j,l] /= normal;
  }

  return;
}

private UMType
iteration_prob_theta_t_n (uint j, uint l, uint k)
{
  UMType    ret_val;
  uint    i;

  for ( i = 0, ret_val = 0 ; i < this.__n ; i++ )
    ret_val += (prob_theta [i, k] *
        (UMType) count_n (j, l, i));

  return ret_val;
}

private uint
count_n (uint j, uint l, uint i)
{
  uint  count = 0;
  uint  q;

  for ( q = 1; q < this.d_train [i].Length; q++ )
    // We look for a transition from
    // each d_train element to the next, and
    // if they are equal to the arguments, then we
    // increment our counter...
    if (  (d_train [i] [q - 1] == j)
```

```
            &&  ( d_train [i] [q] == 1 ) )

          count++;

      return count;
    }

  };
};
```

File: databuilder.cs

```
namespace UserModelling {
  using UserModelling.Logging;

  using System;
  using System.IO;
  using System.Collections;

  public class DataBuilder {
    private ushort [][]  d_train;    // Training data set...
    private ushort [][]  s_train;    // Singleton data set...
    private Hashtable  forward_mapping;   // Forward mapping table...
    private ArrayList  reverse_mapping;  // Reverse mapping table...
    private uint     singletons;     // Number of single transitions...
    private Stream    raw_stream;    // Raw Stream, for doing raw ops...
    private StreamReader  smart_stream;    // Apparently performs Endian/Unicode...

    // Note reverse_mapping.Count gives us the number
    // of states in the alphabet.


    /* Interface */

    public
    DataBuilder (Stream raw_stream)
    {
      if ( raw_stream == null )
        throw new InputStreamException ("Input (Raw) Stream is invalid!");


      this.raw_stream = raw_stream;
      this.smart_stream = new StreamReader (this.raw_stream);

      this.singletons = 0;

      this.build_mapping_tables_training_sets ();

      this.build_training_sets ();
    }

    public
    DataBuilder (Stream raw_stream, IVerbose logging) : this (raw_stream)
    {
      logging.print ((uint) this.d_train.Length,
          (uint) this.singletons,
          (uint) this.reverse_mapping.Count);
    }
```

```
public ushort [][]
get_training_data ()
{
  return this.d_train;
}

public ushort [][]
get_singleton_data ()
{
  return this.s_train;
}

public char
translate (ushort state)
{
  if ( state >= this.reverse_mapping.Count )
    throw new NonExistantStateException ("Unrecognised Alphabetic State!");

  return (char) this.reverse_mapping [ (int) state ];
}



/* Private methods... class use only! */

private void
build_mapping_tables_training_sets ()
{
  string    cur_path;
  int    i;
  ArrayList  d_train_meta;

  // We use this data structure to keep the length
  // of each path in the data set.
  d_train_meta = new ArrayList ();

  // Allocate memmory to Hashtable...
  this.forward_mapping = new Hashtable ();

  // Allocate reverse mapping table
  this.reverse_mapping = new ArrayList ();

  // Reset stream before anything...
  this.raw_stream.Position = 0;

  // Ok. we attempt this again, for the millionth time.
  // /djalkdfjasd;lfkjasdl;!LJLVKEJFOEFJL:DfijsLV lDKSJ FLD:SAfj
  // It's 5:49am!!! FULKJCLKXJCL:JDL:SA:JKLFDAJF:DSJF:

  cur_path = this.anti_wspace_readline ();

  while ( cur_path != null ) {
    if ( cur_path.Length > 1 ) {
      for ( i = 0 ; i < cur_path.Length; i++ )
        // Element doesn't exist.
        if ( this.forward_mapping [ cur_path [i] ] == null ) {
          // Assign to forward map.
```

```
          this.forward_mapping [ cur_path [i] ] = (ushort)
            this.reverse_mapping.Count;

          // Assign to reverse map.
          this.reverse_mapping.Add ( cur_path [i] );
        }

      // Add length of path to out d_train_meta.
      d_train_meta.Add (cur_path.Length);
    }
    else this.singletons++;

    cur_path = this.anti_wspace_readline ();
  }

  // Allocate memmory to the d_train... somewhat like
  // the money train, but not quite.
  this.d_train = new ushort [d_train_meta.Count] [];
  this.s_train = new ushort [this.singletons] [];

  for ( i = 0; i < d_train_meta.Count ; i++ )
    this.d_train [i] = new ushort [(int) d_train_meta [i]];

  for ( i = 0; i < this.singletons ; i++)
    this.s_train [i] = new ushort [1];

  return;
}

private string
anti_wspace_readline ()
{
  string  s = null;
  int   c;

  c = this.smart_stream.Read ();

  // We have to handle the case where we encounter
  // a newline initially. We need to ignore this.
  while ( c != -1 && (char) c == '\n' )
    c = this.smart_stream.Read ();

  // Consequitive reads...
  while ( c != -1 && (char) c != '\n' && (char) c != '\r' ) {
    // Skip over whitespace
    if ( (char) c != '\t'
        && (char) c != ' '
        && (char) c != '\r'
        && (char) c != '\n' )
      s += (char) c;

    c = this.smart_stream.Read ();
  }

  // We don't need to explicitly handle
  // the case where we encounter EOF.
  // If it occurs initially, then both
  // while statements would be false, and
```

```
      // returning s would mean returning null.

      return s;
    }

    private void
    build_training_sets ()
    {
      string  cur_path;
      int  i, j, k;

      // Reset Stream...
      this.raw_stream.Position = 0;

      cur_path = this.anti_wspace_readline ();

      i = 0;
      j = 0;
      while ( cur_path != null ) {
        if ( cur_path.Length > 1 ) {
          for ( k = 0; k < cur_path.Length ; k++ )
            d_train [i][k] = (ushort) this.forward_mapping [ cur_path [k] ];

          i++;
        }
        else if ( cur_path.Length == 1 ) {
          s_train [j][0] = (ushort) this.forward_mapping [ cur_path [0] ];

          j++;
        }

        cur_path = this.anti_wspace_readline ();
      }


      return;
    }

  };
};
```

File: `verbose.cs`

```
namespace UserModelling.Logging {
  using System;
  using System.IO;

  public interface IVerbose {
    void print (double [] pi);
    void print (double [][] theta_i);
    void print (double [][,] theta_t);
    void print (double [,] prob_theta);
    void print (uint data_path_length, uint singletons, uint alphabetic_states);
    void print (string s);
    void print (uint i);
    void print (double log_old, double log_cur, double convergence_val);
  };
```

```
public class NullVerbosity : IVerbose {
  public void print (double [] var) { return; }
  public void print (double [][] var) { return; }
  public void print (double [][,] var) { return; }
  public void print (double [,] var) { return; }
  public void print (uint var1, uint var2, uint var3) { return; }
  public void print (string s) { return; }
  public void print (uint i) { return; }
  public void print (double log_old, double log_cur, double convergence_val) { return; }
};

public class VerboseVerbosity : IVerbose {
  private TextWriter  logging_stream;

  public
  VerboseVerbosity (TextWriter stream)
  {
    this.logging_stream = stream;
  }

  public void
  print (double [] pi)
  {
    uint  i;

    this.logging_stream.Write ("pi = (");

    for ( i = 0; i < pi.Length - 1 ; i++ )
      this.logging_stream.Write ("{0} ", pi [i]);

    this.logging_stream.Write ("{0})\n\n", pi [i]);

    return;

  }

  public void
  print (double [][] theta_i)
  {
    uint  i,j;

    for ( i = 0; i < theta_i.Length ; i++ ) {

      this.logging_stream.Write ("Theta^I, cluster {0}:\n", i + 1);

      for ( j = 0 ; j < theta_i [i].Length ; j++ )
        this.logging_stream.Write ("{0} ", theta_i [i][j]);

      this.logging_stream.Write ("\n");
    }

    this.logging_stream.Write ("\n");

    return;

  }

  public void
```

```
print (double [][,] theta_t)
{
  uint  i, j, k;

  for ( i = 0 ; i < theta_t.Length ; i++ ) {

    this.logging_stream.Write ("Theta^T, cluster {0}:\n", i + 1);

    for ( j = 0; j < theta_t [i].GetLength (0) ; j++ ) {
      for ( k = 0; k < theta_t [i].GetLength (1); k++ )
        this.logging_stream.Write ("{0} ", theta_t [i] [j,k]);

      this.logging_stream.Write ("\n");
    }

    this.logging_stream.Write ("\n");
  }

  this.logging_stream.Write ("\n");

  return;
}

public void
print (double [,] prob_matrix)
{
  uint   i,j;


  this.logging_stream.Write ("Probability Matrix:\n");

  for ( i = 0; i < prob_matrix.GetLength (0) ; i++ ) {
    for ( j = 0; j < prob_matrix.GetLength (1) ; j++ ) {
      this.logging_stream.Write ("{0} ", prob_matrix [i,j]);
    }

    this.logging_stream.Write ("\n");
  }

  this.logging_stream.Write ("\n");

  return;
}

public void
print (uint data_path_length, uint singletons, uint alphabetic_states)
{
  this.logging_stream.WriteLine ("Data paths: {0}", data_path_length);
  this.logging_stream.WriteLine ("Singletons: {0}", singletons);
  this.logging_stream.WriteLine ("Total: {0}", data_path_length + singletons);
  this.logging_stream.WriteLine ("Alphabet states: {0}", alphabetic_states);

  return;
}

public void
print (string s)
{
```

```
      this.logging_stream.WriteLine (s);

      return;
    }

    public void
    print (uint i)
    {
      this.logging_stream.WriteLine ("Convergence after {0} iterations!", i);

      return;
    }

    public void
    print (double log_old, double log_cur, double convergence_val)
    {
      this.logging_stream.WriteLine ("log_e (P (d_train | c_k, \\theta))_old = {0}", log_old);
      this.logging_stream.WriteLine ("log_e (P (d_train | c_k, \\theta))_cur = {0}", log_cur);
      this.logging_stream.WriteLine ("Percentage difference: {0}%\n", convergence_val);

      return;
    }
  };

};
```