DATA MINING

FALL 2009

FINAL PROJECT : AGNES IMPLEMENTATION

by

VOLKAN PAKSOY

1. Program design

Classes used are as follows :

AGNESClient: Main application. It's a console application and getting the required parameters is done in this class.

AGNES: This is main implementation class. Algorithm is run.

ClusteringMethod: While building distance matrix several methods can be used. This class is used to contain alternative algorithms. It uses a Strategy design pattern.

ClusteringMethodStrategy: This is the absract base class for clustering methods.

SingleLinkClusteringMethod

CompleteLinkClusteringMethod

AverageLinkClusteringMethod: These three classes are the concrete classes for the clustering method implementation.

ClusterObject: An object representing the bits read from the file. Every object has a unique ID and every object has a cluster ID.

Cluster: A collection of ClusterObjects.

ClusterCollection: A container for a collection of clusters.

DistanceMatrix: The class that holds the distance values between clusters.

Database: An abstract class for the data source. Currently only text file database is implemented. But if a need arises to retrieve data from a database, the rest of the code would not be affected. Only connection string should be passed to the concrete database class

TextFileDatabase: A concrete implementation of the Database class. Reads the text file and creates ClusterObjects from the bit strings.

DistanceCalculator: Contains a DistanceMetricStrategy object and calls its corresponding method.

DistanceMetricStrategy: Abstract class for strategy objects to calculate distance.

JacquardTanimotoDistanceMetric: Concrete distance metric calculation class. Currently there is only this one.

Logger: A class to log events. It uses an Observer design pattern. When the program starts LogListener objects attach themselves to the Logger. When an event is needed to be logged, all subscribed listeners are notifed. So that a message can be logged to multiple sources.

LogListener: Abstract base class for listeners.

ConsoleLoglistener: Displays the received message to the console.

TextFileLogListener: Writes the message to the text file. The file of the path is read from the App.config file. This file must in the bin directory. If the configuration file cannot be read then the path will not be set. In this case, TextFileLogListener will not be attached to the logger.

ApplicationException: Custom exception class used in the application.

HelperMethods: Includes some general conversion functions.

ConfigReader: A primitve file reader that finds “App.config” file in the application's folder and reads the value for LogFilePath. Currently no other parameters are needed.

1. Implementation

Implementation starts with creating the ClusterObject objects from the data. ClusterObject contains a BitSet object. After bit vectors are parsed from the database a corresponding ClusterObject is created for each row. When data is parsed all the objects are initially placed into a one big cluster. Before we begin running AGNES algorithm we have to create a cluster for each object. After clusters are created objects are added to their object list. Objects' ID's never change during the run.

After the clusters are prepared we need the initial distance matrix. One of the most important design decisions in this implementation was the data structure to be used as the distance matrix. First, I created a two-dimensional double array. This way I had to use indexes to determine clusters. The deleted clusters would be still be in the matrix. So memory consumption would not decrease after cluster mergers. Then I thought about using an ArrayList. This is a dynamic list so as number of clusters decrease the number of entries will decrease as well. But i'd have to update cluster indexes every time a merger occured. Because like the array I could use only indexes to the find the clusters. My final decision was to store the distance values in a Hashtable. Key of the table is a joint key formed by the cluster IDs concatenated. Such as : Key: 00030004 Value: 0.857143

This entry in the hashtable means the distance between cluster with ID 3 and cluster with ID 4 is 0.857143. By using hashtable, I could assign a new ID to the new clusters without worrying about the indexes. Also, as the matrix is symmetrix and diagonal is zero, more than half of the values would just take space if i used a two dimensional array.

Currently the Ids are left padded up to 4 characters. So it can only hold 10000 clusters. For a greater number of objects prepareKey method of the DistanceMatrix class must be modified.

initializeDistanceMatrix function is called to create the original distance values. It takes the object list and distance metric strategy to be used for the calculations. Jacquard – Tanimoto distance metric is used in this implementation.

After the initial distance values are calculated, a loop starts. In this loop, first we decide which clusters to merge. To find that, we have to find the minimum distance value in the table. After the Ids of the clusters to be merged are found, we call mergeClusters method to actually merge them. Merging two clusters is creating a new clusters and adding the objects from the two clusters to this new one. After we create the new cluster, we send this data to the DistanceMatrix object to calculate new distance values.

CalculateDistanceMatrix method loops through the hashtable and checks the keys. It creates a new hashtable that will replace the old one at the end of the method.

Every key contains 2 cluster Ids and the method takes the merged cluster Ids. So there are 3 cases to be considered:

1. The Ids in the key both match with the Ids of the clusters being merged.
2. One of the IDs matches with one of the Ids of the clusters being merged.
3. None of the Ids match.

First case means that the value we found is no longer necessary. Since the clusters merged, the old distance between them is no longer a valid value.

In the second case, it creates a key to search in the new distance matrix if it is already inserted. If it's not it inserts the value. If it's already inserted, it takes this value and the one from the old hashtable and calls clustering method objects calculate method. Whatever this method returns is saved as the new distance. (This only works for Single-Link and Complete-Link clustering methods. Average-Link clustering method requires the original distance values so they are calculated in a different method.)

Thirds case means, the objects are irrelvant with the current merger so the old value is added to the new table.

For example if the values in th hashtable are as follows

00010002 1

00010003 2

00010004 3

00020003 4

00020004 5

00030004 6

and let clusters 1 and 2 to be merged into cluster 5.

While looping, first we get the key 00010002. This means we no longer this value so we skip it.

Then we get 00010003. This means one of the clusters is in this distance is being merged. So we add a new value to the new table as 00050003 with value 2 since it doesn't exist already. At 00010004, we check for 00050004 and add value 3. At 00020003 we check for 00050003 and since it is already in the new table we send this value (2) and curernt value (4) to the clustering method. If it is a Single-Link clustering method it will return 2, Complete-Link method will return 4. So we update the current value. At 00020004, we do the same for values 3 and 5. At 00030004, since it's unrelated to the merger, we add 6 with the same key.

When Average-Link clustering method is used calculateDistanceMatrixForAverage is called. Actually this is not a very good object-oriented programming practice. But since it works differently on different data (it uses the initial distance values), i accepted as an exceptional case. Average method starts like the other method. It loops through all the keys in the current table. And it behaves the same in case 1 and case 3. In case 2, it checks the initial table and finds all the distance values from the clusters being merged to the other cluster and calculates the average value. It only calculates once if the new key is not in the table.

After new distance are calculated, the old clusters are removed from the collection. At the end of the loop we end up with a one cluster that contains all the objects.

1. Test results:

3 data files and detailed logs can be found under tests folder. Only the inputs and outputs are included in this document.

* 1. Test 1

Input file : data1

Clustering method: Single-Link

Data :

0,1,1,1,1,0,0,0,1,1

0,1,1,1,1,0,0,0,0,0

0,1,1,1,1,0,0,1,1,1

1,1,1,1,1,0,0,0,1,1

0,0,0,0,0,0,0,0,0,1

Results:

0 merged with 3

0,3 merged with 2

0,3,2 merged with 1

0,3,2,1 merged with 4

* 1. Test 2

Input file : data2

Clustering method: Complete-Link

Data :

1,0,0,0,1,0,1,0

1,1,1,1,1,1,1,1

0,0,0,0,0,0,0,0

1,1,0,0,1,1,0,0

1,0,1,0,1,0,1,0

1,1,1,0,0,0,0,0

Results:

0 merged with 4

1 merged with 3

1,3 merged with 5

1,3,5 merged with 0,4

1,3,5,0,4 merged with 2

* 1. Test 3

Input file : data3

Clustering method: Average-Link

Data :

1,0,0,1,1,0,1,0,1

1,0,0,1,1,1,1,1,1

0,1,0,0,0,0,1,0,0

1,1,0,0,1,1,0,1,0

1,0,1,0,1,0,1,0,1

0,0,1,0,0,1,0,0,1

1,1,0,0,0,1,1,0,1

0,0,1,1,1,0,0,1,0

1,1,0,0,1,0,1,0,1

Results:

0 merged with 1

4 merged with 8

4,8 merged with 0,1

4,8,0,1 merged with 6

4,8,0,1,6 merged with 3

4,8,0,1,6,3 merged with 2

5 merged with 7

5,7 merged with 4,8,0,1,6,3,2