Individual Mandatory Assignment

Data Mining, F2011,
The IT-University of
Copenhagen

Anders Hartzen, andershh@itu.dk

Foreword

This report details the work done in the mandatory assignment of the course in Data Mining taught at The IT-University of Copenhagen in the spring semester of 2011. The main objective was to work with data from a questionnaire answered by the students of the course by employing data mining techniques.

I saw this assignment as a possibility to try different tools from the data mining tool box, and see their effect on a dataset, generated from a real world questionnaire. I saw this as a possibility to learn about the different data mining techniques, by simply implementing them and playing around with them, and seeing what happens.

Therefore my aim wasn't to answer a specific question about the data set, but more to explore it and while doing this learn more about implementing data mining methods practically.

However this aim would turn out to be quite problematic to achieve, as I ran into technical difficulties in getting the preprocessing of the data up and running. Difficulties which ended up absorbing most of my effort used on this assignment, and hurting my answer to the other parts of the assignment.

Preprocessing

In preprocessing I chose two employ two techniques, namely Data Cleaning and Data Reduction.

Data Reduction

On the onset there were four questions, which I decided to eliminate from further deliberations, namely
In the given dataset there are four questions, whose answer really doesn't provide any valuable knowledge, the three questions asking you to write a random numbers (questions 10,11,12) and the rather nonsensical question: *Which do you think therb fortt glag?* (question 20). The conversion between RawDataObject and DataObject

Data Cleaning

Data Cleaning-wise I tried to deal with both missing and erroneous values in two ways: ignoring the tuple and using the attribute mean to fill in the missing/erroneous value.

The implementation

To facilitate the data reduction and cleaning I created four helper classes DataLoader, MeanCalculator, RawDataObject and DataObject. My preprocessing can be divided into 6 phases

Phase 1 – Loading the data in

The DataLoader class begins the "show" by loading the data in as text strings in a two-dimensional array.

Phase 2 – Converting from string to RawDataObject

In phase 2 the string data from are converted to data types we can work with, such as Double or Booleans. This is done through converting each tuple to a RawDataObject. While this happens the tuple attributes are checked to see if their data types match what we expect (for instance Double for age etc.). If not, an error counter is increased by one. This error counter is stored in the RawDataObject for each tuple.

Phase 3 – Removing tuples with too many errors, part 1

The RawDataObjects are now checked to see if their error counter is larger than a defined threshold (which is defined in the Preprocessing class). If they are, they are removed from the data set.

Phase 4 – Converting remaining data to DataObject

The conversion from RawDataObject to DataObject is largely reminiscent of conversion from string values to RawDataObjects, but with one main difference. Instead of checking the data type, we check the actual values to see if they make sense. If not, the error counter is increased. While converting some values are converted to an enum – which is a programming construct, which makes it easier to define one's own categories. For instance the attribute "favourite OS" is converted from a string to an enum. This can be seen a pseudo-generalization as we know from Data Transformation- Pseudo, since I don't translate the attribute to a higher concept, but instead converts it to something which is easier to work with. In other words, here we look at semantics instead of syntax.

Phase 5 - Removing tuples with too many errors, part 2

Same as phase 3, only difference is that is that it is DataObjects that are removed.

Phase 6 – Fill in The Blanks!

In this final phase we now have a collection of tuples with either no errors or missing values; or tuples with a number of missing / erroneous values that is below our threshold. The latter tuples' missing values are now filled in by using attribute mean, which is done in the MeanCalculator class.

The Problem

The big problem I ran into while building this implementation was that the sheer number of attributes collided with my ambition for the preprocessing. It meant that much time was spent writing the code taking account of all the attributes in the different phases, and also resulted in many bugs and that also stole a lot of my time.

However I did manage to get all the preprocessing to work in spite of the mentioned problems.

The algorithms implemented

Apriori

I have implemented the A –priori algorithm, which is an example of a frequent pattern mining method, for my data produced in preprocessing. However due to time spent getting the preprocessing to work, meant it wasn't until very late in the process that I could test this implementation. This has resulted in that there is a crashing bug in the implementation.

K-medoids

The same as said about A-priori can be said about my K-medoids implementation. Only difference is that, the implementation does not have a crashing bug; but it produces an odd result; i.e. it groups all tuples into one cluster, in spite of the k-value.

Perceptron

Wasn't implemented due to the problems mentioned in preprocessing.

Conclusion – what I did learn

What I've learned from this is that being specific is a good thing when dealing with data. I could have worked around the problems in preprocessing, had from the onset decided upon some specific questions I would have wanted to answer. Had I done so, I would have been able to limit the number of attributes to work on and convert in the preprocessing phase. With this done I am confident I would have had the time to get last bugs ridden from my algorithm implementations.

Errors

Preprocessing

Built-in Parse functions can make numbers ten fold larger for instance string “6.5” becomes 65