**Tour of Data Mining Algorithms**

Implementation Project

The Apriori algorithm has been fully implemented. The implementation is composed of an Apriori class, a HashTree class, classes that make up the components of the hash tree, including Item and ItemSet classes. The implementation works primarily by making calls to the HashTree class, in order to manipulate the hash trees storing frequent itemsets and their occurrence counts. The x-means algorithm is finishing up implementation with a visual representation of the clusters. The kD tree was hard to implement but was better than just using a list of points that are in the data set. The algorithm is very computation intensive measuring a lot of distances takes quite a bit of time.

Various problems were encountered during development of the Apriori implementation. Certain aspects of the algorithm’s behavior were unclear from the documentation, with further research failing to produce further clarification. As a result, some decisions and assumptions were made in order to complete the implementation.

One such issue was how an itemset should be added to the hash tree. The algorithm sets a limit on the number of itemsets a leaf node may store (max bucket size). When the bucket limit has been reached and another itemset is to be added to that bucket, the algorithm is to convert the node into an interior hash node, and split the itemsets in the bucket among new child nodes. To determine which child nodes to assign the itemsets to, the itemsets are hashed on their next item. But what happens when there is no next item to hash? Our implementation handles this by simply exceeding the bucket size limit in this case.

Another point of uncertainty was regarding whether or not a sepearate hash tree should be created for itemsets of a particular length. In other words, should there be a 1-itemset tree, a 2-itemset tree, and so on? For our implementation, this was the route we went, resulting in multiple hash trees.

X-Means started with k-means and that worked well then we implemented the kd tree to speed up the process of finding clusters. While testing k-means we used a 2-Gaussian distribution and used a k of 2 with the k means. This worked well but was hard to know for sure without visualizing it, but weka helped figure that out. There was a small issue when figuring out when to reevaluate the centroids a few implementations sent the program into an infinite loop but limiting the number of loops helped.

The command line interface for our project has also been implemented fully for Apriori. The Apache Commons CLI library was used to create the command line interface. When calling the program from the command line, the algorithm and input file must be specified, along with algorithm-specific options such as the minimum support for an itemset to be considered frequent by the Apriori algorithm. Optional options may be specified, such as specifying a specific name for the output file, or specifying the delimiter used to separate attributes in the input file. A help message is also provided when the help option is given, or when invalid or incomplete options are specified.

We just started implementing the id3 algorithm using a unique binary tree.

References:

*An intoductory tutorial on kd-trees* by Andrew W. Moore published with Carnegie Mellon University

Data Mining: Concepts and Techniques, 3rd Ed, by Jiawei Han.