Presentation Script

Personal notes so far: this is super long, and I will try to trim it down. Also, I will try to speak very, very quickly, and point at the code as I speak.

Apriori

Apriori: overall algorithm

The first algorithm we implemented was the Apriori, a frequent itemset mining algorithm. For our implentation, the user is required to specify the input file, the absolute minimum support for an itemset to be considered frequent, and they may optionally specify the number of children per node in the constructed hash trees, as well as the maximum bucket size of the bucket (leaf) nodes in the hash tree.

After the input is read, the implementation begins creating a 1-itemset hash tree by counting occurrences of items in the given transactions. Itemsets not meeting minimum support are then removed. So now the 1-itemset hash tree contains only frequent 1-itemsets.

After that, the implementation enters a while loop, that terminates when the previously generated hash tree is empty. All non-empty hash trees are added to an array of hash trees, which will contain a 1-itemset hash tree, a 2-itemset hash tree, and so on. At this point, the previously generated tree is renamed to the k-1-itemset tree. A new tree is generated from the k-1-itemset tree by performing a self-join on the itemsets in the k-1-itemset tree. The new tree is named as the current k-itemset tree. All frequencies in this new tree are still set to 0 at this point. The new tree is pruned, by looking at the k-1-itemset tree and to see if any itemset in the new tree has subsets that are not present in the old tree. After pruning, the transactions are scanned again, counting occurrences of the itemsets in the newly pruned tree. After counting, any itemsets in the new tree that do not meet minimum support are removed. The loop continues until the termination condition above is met.

Apriori: Hash Tree

Each node in the hash tree is responsible for determining what to do with an itemset it has received. The itemset it receives is passed as a combination of two itemsets: one containing those items which have already been chosen and hashed on, and the other containing the remaining unchosen items. If the node is a bucket node, it calls the addToBucket method, which checks the set of chosen items to see if all k items have been chosen. If so, it calls the putInBucket method, which puts the itemset in its bucket, or updates the itemset’s frequency if it is already present. If it is a bucket node and there are more items to choose, and the bucket size limit will not be breached, it generates all k-itemsets it can from the received sets, and adds them to the bucket. If the itemset limit will be breached by adding those itemsets, it instead converts itself to a hash node, chooses the next item, hashes it to determine the child node to pass it down to, creates the appropriate child nodes, and passes the chosen and unchosen items to the appropriate children. Any children that were already in the bucket are also hashed and passed to the approriate children. Each node keeps track of its level in the tree so it always knows which item to hash on.

ID3