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: overall algorithm

For Apriori, it starts by creating a 1-itemset hash tree, adding all 1-itemsets from the given transactions, counting frequencies as it goes. Any itemsets that do not meet minimum support are removed.

After that it enters a while loop, that terminates when the previously generated tree has no items. Each hash tree is added to an array, which will contain a 1-itemset tree, a 2-itemset tree, and so on. The k-itemset tree then becomes the k-1-itemset tree. A new k-itemset tree is generated by self-joining the itemsets in the previous tree. It is then pruned by looking in the previous tree to see if any itemset in the new tree has subsets that are not present in the old tree. The frequencies for the remaining itemsets are counted, and those that don’t meet minimum support are removed.

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 items that have already been chosen and hashed on, the other containing unchosen items. If it’s a bucket node, it calls the addToBucket method, otherwise the addToChildren method is called, which hashes on the next item, and passes it to the appropriate children based on the outome of the hash. For addToBucket, it checks to see if k items have been chosen. If so, it calls the putInBucket method, which puts the set in the bucket, or updates its count if it’s already there. If there are more items to chose, but everything will fit in the bucket, it puts it in the bucket. If the they won’t fit, it converts itself to a hash node, hashes all itemsets on their next item, and passes them to the appropriate children.

ID3

The ID3 algorithm has a run method to build the decision tree, and the other important method is classify, which uses the tree it built previously to classify unlabled tuples. Each node is passed a partition of data, and keeps track of which attribute value the data was split on to create the partition it just received. It first checks to see if all the tuples it received are of the same class, if so it labels itself with that class and is done. After that, if there are no attributes left to split the data on, it labels itself with the class determined by majority voting. Otherwise, there are more attributes it can use to split the data on, so it uses information gain to determine the splitting criterion to use. Once it has that, the data is split, children are created, and the children receive their partition.

ID3: Information Gain

For information gain, the information gained by splitting on a particular attribute is the total entropy minus the entropy of the partitions created by the split. First it calls countAttributeFrequencies, which counts how many times each value appears for each attribute in the data. Then it can calculate the total entropy. After that this for loop calculates the entropy after splitting on each attribute. In the loop, the gain is calculated for each attribute and stored in the gains array. Once that is done for each attribute, the last for loop simply goes through the gains array and finds the highest gain, and stores the index and title of the chosen attribute, which can be retrieved with getter methods.

The calculateEntropy method implements this formula here. The for loop performs the summation of the probabilty of each class times the log of that probability, with the negative here. Then it just returns the result.

The calculateEntropyAfterSplit method does just that for a particular attribute, which is identfied by the attrIndex argument. To do that, it has to group together tuples having a particular value for that attribute, and pass those to the previous method here. That value is then multiplied by a weight here. Then it returns the result.

The classify method is passed an unlabled tuple. First it checks to see if it is a leaf node, in which case it labels the tuple with its chosen class. If it’s not a leaf node, it goes through its children, looking at which value they were split on, this value here, and passes it to the matching child.