Expert Data Mining: k-value

# Introduction

For this next iteration of the Expert Data Mining software, the program now works with k-value instead of just Boolean. K-value refers to how many values that a particular attribute can have. The below k-values follow a convention that they are always 0 through k – 1.. For example, a k-value of 2 is equivalent to Boolean values 0 and 1, but a k-value of 3 is equivalent to values of 0 through 2. Let’s say that we have a dataset of just 2 dimensions, and the k-value for each attribute is 3. The following datapoints will be in the dataset:

(0, 0)

(1, 0)

(0, 1)

(1, 1)

(2, 0)

(0, 2)

(2, 1)

(1, 2)

(2, 2)

In this example, k = (3, 3, kn+1). The dimension of the k-value vector is one more than the dimension of the n-D points. That is because the value of kn+1 refers to the k-value of the function itself. A kn+1 of three would correspond to three possible values that the n-D points could be equal to. Those values would be between 0 and 2 for the current implementation.

Furthermore, the sections below will describe how k-value will be different for the corresponding options. First, k-value attributes will be described. Then, kn+1 will be explored in greater depth.

# K-value Attributes

## Majority Flag

One of the goals of the majority flag is to minimize the number of questions that needs to be answered by the domain expert to restore a k-valued function when n-D points that have multiple values (k-value). The k-valued approach is also the approach which we use for binary functions. We ask a user if they believe that for the function to have a value equal to one or greater, then at least half of the values of the n-D points need to be equal to 1 or greater. We require that half of the Hamming norm of the n-D points will be needed to a function value of 1 or greater.

For example, let’s take a non-binary n-D point, **A**, where the k-value vector is: **k** = (k1,k2…kn, kn+1):

**K** = (3, 5, 2, 2); **A** = (2, 4, 1)

**A** is the largest n-D point that is possible for the given k-values. Obviously, the smallest n-D point will be (0, 0). Its **generalized Hamming norm,** G(x), is the sum of the values; e.g, G(**A**) = 7. However, there is an issue with simply using the generalized Hamming norm for majority flag: there can be “lopsided” n-D points. Since G(**A**) = 5, we look for n-D points where there generalized Hamming norm is 3. For Boolean, this is not an issue because each point in the datapoint can only be one of two values, which means for a majority of those points to be a value of 1, then half or at least half of the points will be a 1. For the given k-values above, then the generalized Hamming norm, G(x), needs to be 4 for the n-D point to be considered a majority datapoint. However, the n-D point (0, 3, 0) would fall into this category. We consider this datapoint to be “lopsided” because although the generalized Hamming norm is the correct value, less than half of the points are equal to 1 or greater. The solution to this is using “middle points.” Middle points are simply k-valued n-D points where at least half of the points are a value of 1 or greater, similar to majority flag for Boolean. However, the generalized Hamming norm still needs to be equal to half that of the maximum generalized Hamming norm possible for the given k-values

For our above k-values, then the middle points are:

(1, 2, 1)

(2, 1, 1)

(1, 3, 0)

(2, 2, 0)

If we only use the generalized Hamming norm, then a possible majority flagged datapoint would be (0, 4, 0). The implementation of middle points is another pilot question we ask the user in the case that they chose to use the majority flag option.

Furthermore, just like the Boolean version, if the user decides to use the majority flag, then all majority vectors are asked first, and once those are done, the normal sequence of questions is asked. If the user does not know roughly how many majority vectors are true, then all majority vectors are asked first. If the user specifies that a number of these majority vectors are true, then we ask at random, and if half of those majority vectors are true, the normal sequence of questions is asked. The idea is that it is likely that for a given dimension, it will take half or slightly over half of the attributes to be true for the vector to be true, but that is not guaranteed. The majority flag can be paired with chain jumping. In this case, after the majority questions are asked, the Hansel Chains which have majority vectors that are true are asked before returning to the normal sequence of Hansel Chains.

## True Attributes

Originally, a true attribute was an attribute that needed to be true for the datapoint to have a class of 1. That is still the case with k-value, but now the value of that attribute needs to be specified as well if the k-value is greater than 2. For example, if the k-value of an attribute is 4, then the user could specify that the value of that attribute needs to be 2 for that attribute to be true.

## f-changes

f-changes are mostly the same as they were previously. However, if the f-change causes a violation of monotonicity and the user decides to add an attribute to fix the violation, then we need to ask the k-value for the new attribute. Otherwise, this feature will function the same as before.

## Nested Functions

k-valued attributes do not change anything for nested functions. However, k-value function values will change how it works (more on this later).

# Kn+1 (k-value function)

As stated before, kn+1 refers to the number of functions values. Similar to how the Hansel Chains are ordered, the values are generally ordered in a monotonically increasing fashion. This is because we need to be able to expand from one n-D point to another. If the function values are nominal, then we can’t expand across function values. For example, if we have some generic datapoint, **A** and **B,** and **A** expands **B** in the positive direction,then the f(**B**) >=f(**A**). If f(**A**) = 1 and assuming that kn+1 = 3, then f(**B**) >= 1. If f(**A**) = 2 and assuming that kn+1 = 3, then f(**B**) = 2. Without being able to assume that a function value of 2 is monotonically greater than a value of 1. These types of expansions would not be possible.

Furthermore, expansions are slightly different for values of kn+1 > 2. For Boolean (kn+1  = 2), we always expand in one direction and one direction only. If a datapoint has a function value of 1, then can expand another datapoint to have a value of 1. The same is true for a function value of 0. Previously, we called these one-to-one expansions and zero-to-zero expansions. However, for of kn+1 > 2, when it comes to “weak” values, we can expand in both directions. For example, we have some Boolean datapoints datapoints:

**A =** (1, 0, 1),

**B =** (1, 1, 1),

**C =** (1, 0, 0)

If kn+1 = 3 and f(**A**) = 1, then we will expand both **B and C.** Therefore, f(**C**) <= f(**A**) <=f(**B**). The term “one-to-one” and “zero-to-zero” do not apply anymore. Instead, we generalize the terminology: an “up expansion” is an expansion that occurs to a datapoint that is monotonically greater than the source of the expansion. A “down expansion” is an expansion that occurs to a datapoint that is monotonically lesser than the source of expansion. In this example, f(**B**) is the up expansion and f(**C**) is the down expansion. One caveat with up and down expansions is that we don’t necessarily know the exact function value for f(**C**) and f(**B**), just that the function value for them are f(**C**) <= 1 and f(**B**) >= 1. We still need to confirm with the user if f(**C**) = 0 or 1 and if f(**B**) = 1 or 2. Therefore, we refer to the middle values for any value of kn+1 to be weak values. If a user assigns a function value to a datapoint, and that function value is the smallest or greatest possible value with respect to kn+1, then one of expansions from that datapoint will be a “strong” value. Using the previous example, if f(**A**) = 2, then f(**B**) = 2 and f(**C**) <= 2. Strong answers using the lowest and highest possible function value will reduce the number of questions that need to be asked to the user.

## True Attributes

For true attributes, it may seem like we would need to specify what is “true” when kn+1  > 2, but its actually quite simple: it’s the same as Boolean it just means that if an attribute is chosen to be needed for a “true” function value, then that value will need to be at least 0. The user will still need to specify the exact class, which is the same as Boolean. If the true attribute is not present, then we know that the function value for that datapoint is 0.

## Nested Functions

If there is only one level of function, the value of kn+1 is unrestricted. However, let’s say that we have a two-level function, a parent and a child. The child function is representative of an attribute of the parent function. For Boolean, it’s assumed that is some attribute, say **x1** is a Boolean attribute, then the child must be a Boolean function because it determines of **x1** is true or false. Therefore, if **x1** is k-value, let’s say that k = 3, then the child function must have kn+1 = 3. The k-value of the parent must be equal to the kn+1 value of the child. This is automatically determined by the program. If a parent attribute is given a k-value of 3, then when the child function is created, the value of kn+1 is automatically assumed to be 3 as well.