**Background**

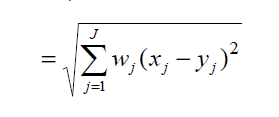
I am looking to compute a measure of how close a set of items are to each other, based on individual features of the items. Consider the following input:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| I1  (f1,v1)  (f2,v2)  (f3,v3)  .  .  .  (fn,vn) | I2  (f1,v1)  (f2,v2)  (f3,v3)  .  .  .  (fm,vm) | I3  (f1,v1)  (f2,v2)  (f3,v3)  .  .  .  (fk,vk) | . . . . . . . | In  (f1,v1)  (f2,v2)  (f3,v3)  .  .  .  (fj,vj) |

where In is a uniquely identifiable item, and (fn,vn) are feature-value pairs that describe attributes of the item. The goal is to come up with a real number between 0 and 100 that quantifies the fit between Ix and Iy.

The values for the features will consist of either real numbers, or integers. In some cases, features will be binary, and consist of the integers 1 or 0 to denote existence.

The current process we use is to derive the fit from a weighted Euclidean distance:



The number of items will be on the order of 20,000, but will get bigger as time goes on. The number of features per item shouldn’t exceed 100.

This function will be executed on an R instance running on a 64 bit Linux instance on Amazon’s Elastic Compute Cloud (EC2), so there will be plenty of processor power and memory (up to 8 cores, 64GB of memory). The EC2 instance will only exist to execute this code, and will be torn down upon completion, so billing will be minimal ($2/hr for the instance with the most resources).

**Current Process Detailed**

1. Normalize the values of all the features across the items:

vnornm1 = (v1-avg(v1, v1, v1, v1, v1,...)) / standard dev(v1, v1, v1, v1, v1,...)

Resulting in:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| I1  (f1,v1)  (f2,v2)  (f3,v3)  .  (fn,vn)  (f1norm,v1norm)  (f2norm,v2norm)  (f3norm,v3norm) .  (fnnorm,vnnorm) | I2  (f1,v1)  (f2,v2)  (f3,v3)  .  (fn,vn)  (f1norm,v1norm)  (f2norm,v2norm)  (f3norm,v3norm) .  (fnnorm,vnnorm) | I3  (f1,v1)  (f2,v2)  (f3,v3)  .  (fn,vn)  (f1norm,v1norm)  (f2norm,v2norm)  (f3norm,v3norm) .  (fnnorm,vnnorm) | . . . | In  (f1,v1)  (f2,v2)  (f3,v3)  .  (fn,vn)  (f1norm,v1norm)  (f2norm,v2norm)  (f3norm,v3norm) .  (fnnorm,vnnorm) |

It should be noted that the average and standard deviation on this calculation will be grouped by certain features. The grouping features will be annotated in a support input table.

1. Compute the Euclidean distance by summing the weighted differences between each of the corresponding features between Ix and Iy:

It should be noted that the summation on this calculation will be grouped by certain features. The grouping features will be annotated in a support input table.

1. Compute the fit between Ix and Iy :

It should be noted that the summation on this calculation will be grouped by certain features. The grouping features will be annotated in a support input table.

The support table will contain the following information:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Feature | Type | Include in dist calc? | Group by in normalization? | Group by in distance? | Group by in fit? | Weight  factor |
| f1 | String | No | Yes | No | No | Null |
| f2 | String | No | Yes | Yes | Yes | Null |
| f3 | String | No | Yes | No | No | Null |
| f4 | Integer | Yes | Null | Null | Null | 1 |
| f5 | Integer | Yes | Null | Null | Null | 1 |
| f6 | Real | Yes | Null | Null | Null | 1 |
| f7 | Real | Yes | Null | Null | Null | 2 |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

**Inputs and Outputs**

I’d like to have the function declaration be

*compute\_comparables(input\_file, support\_file, output\_file)*

where *input\_file* is a ‘|’ (pipe) delimited text file in the following format:

I1f1|v1  
f2 |v2  
f3 |v3  
fn |vnI2f1|v1  
f2 |v2  
f3 |v3  
fn |vn.  
.  
Inf1|v1  
f2 |v2  
f3 |v3  
fn |vn

The *output\_file* will be a ‘|’ (pipe) delimited in the following format:

I1 |I2 |FI1I2I1 |I3 |FI1I3I1 |I4 |FI1I4

The *support\_file* will be a ‘|’ (pipe) delimited in the following format:  
  
Feature | Type | Dist calc |Group by Norm | Group by Dist |Group by fit | Weight Factor

f1|String|No|Yes|No|No|Null  
f2|String|No|Yes|Yes|Yes|Null  
f3|String|No|Yes|No|Yes|Null  
f4|Integer|Yes|Yes|No|Yes|Null

**Other Notes**

1. Since FIxIy = FIyIx , there is no reason to calculate both.
2. The function should return FIxIx = 100.
3. If an item does not have a feature for comparison, use a value of 0 for the missing feature.
4. Some features will be binary (either they will exist or not) in this case, the values for these features will be the integer values 1 or 0.
5. I have a set of test data with expected output I can post.
6. Everything in this document is negotiable, including the statistical methods.