Data Analytics Design Document

Designed Function: Clustering

Written by Tran Ngoc Minh

Contents

1. Overview

TBD

2. Clustering Input

Currently, we have 2 data input sources for clustering: one from DLS and one from STB. Data format of these sources will be described in turn.

2.1. Data Input Format from DLS

With the DLS source, each user profile is calculated and represented by several Content Based Collaborative Filtering (CBCF) vectors. The data analytics team has designed algorithms to process input data to yield, for each user, a set of CBCF vectors, which will be used as user profile. The process of calculating CBCF vectors is described in details in [1].

A CBCF vector expresses how often a user watches each program attribute which is calculated as percentage frequency. A CBCF vector is represented as [(ai:vi)], i = 1→N, where:

- a_i is the i^{th} attribute such as a main category, a genre or an actor, etc.
- v_i is the percentage frequency of attribute a_i

For example, a movie CBCF vector can be [comedy : 30, drama : 120, thriller : 0, ...]. This shows that the user of this CBCF vector has watched 30% comedy, 120% drama and no thriller. Note that, a value can be larger than 100% in case a user has watched several programs of same genre.

CBCF vectors will be calculated using data collected within a time duration. Currently, we support for time windows of 6 months, 1 month, 2 weeks and 1 week. Within a time window, we calculate several types of CBCF as described in Table 1. A set of combinations of these types will form CBCF vectors as shown in Table 2. The calculated user profiles or CBCFs are stored on HDFS and will be loaded to HBase for permanent storage. The version on HDFS will be used for clustering to increase performance. Furthermore, as there are several time windows calculated for CBCFs, we choose the 6-month time window for clustering as this contains as enough accumulated information as possible.

Type	Example values
Main category	Movies, TV, Sports
Day of week	Monday to Sunday
Genre	Comedy, Drama,…
Actor	Tom Cruise,…

Director	…
Tone	…
Mood	…
Theme	…

Table 1. Data types of CBCF.

Row key	
Account Id	
Columns	
Time window	**Described examples**
6 months	CBCF based on the last 6 months. CBCF vectors are combinations of types in Table1. Some examples are as follows: all.generic.alldays all.sunday, all.monday....all.saturday all.movies, all.sports, all.tv all.sunday.movies, all.sunday,tv, all.sunday.sports all.saturday.movies, all.saturday.tv, all.saturday.sports, …..
1 month	CBCF based on the last month.
2 weeks	CBCF based on the last 2 weeks.
1 week	CBCF based on the last week.
1 day	CBCF by day vectors, for example 20140607.movies, 20140908.actor, etc.

Table 2. Combinations of CBCF types to form CBCF vectors.

2.1.1. Add Series Information

Note that, we should add information on Series into CBCF and use Series as one set of new added features. For example, a user can have a Series vector like [Series1 = XXX, Series2 = YYY,…]. Currently, CBCF data does not have such information for Series, thus one more analytics function should be done to calculate this data.

2.2. Data Input Format from STB

For clustering users with respect to their movie tastes, we will calculate one more type of CBCF, the so-called movie CBCF. This will be done based on classifying movies from genres to sub-genres, movie types and movie groups. This classification is referred to as a movie taxonomy tree, which is described in detailed in Appendix Table 1. According to the taxonomy tree, we will have around 210 movie groups, which are obtained by K-means clustering of TMS and Rovi attributes, together with careful manipulation to ensure separation of family content from adult content, etc. Then, very similar movie groups are combined into movie types, which are collected into sub-genres, and these into genres.

Currently, we do not have data for movie CBCF yet. Therefore, we need to run one more analytics function to generate user data for the taxonomy and use that data for clustering. This is completely feasible as we already have a direct mapping from each movie tmsID to a corresponding movie group. We will need 4 CBCF vectors for each user, including one for movie groups, one for movie types, one for sub-genres and one for genres.

3. Overall Architecture

Figure 1 shows the overall architecture of the clustering process. It includes a number of stages. Firstly, it receives a set of curation rules from the marketing team for classification. The curated data will then be used for feature selection, normalization and weighting. After that, complicated clustering algorithms such as Canopy and K-means are applied to group users into clusters. Results will be evaluated through a number of tests. If passed the tests, results are stored on HDFS, otherwise other clustering jobs must be started by re-doing feature selection. All the stages for clustering will be described in detailed in turns.

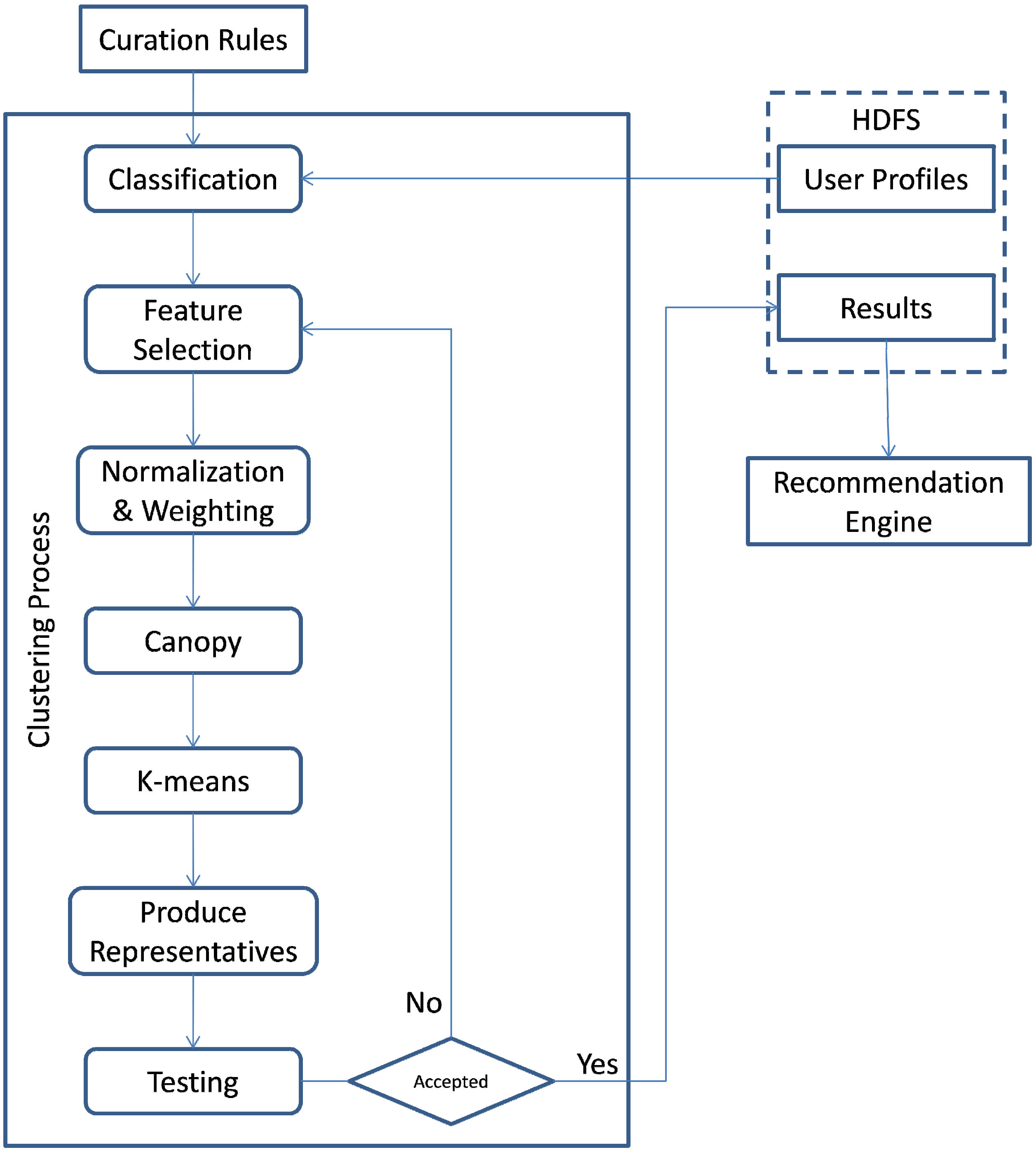

Figure 1. The overall architecture of the clustering process.

3.1. Classification

Classification is an initial step that sorts customers into common groups based on the viewing history. Like a group that has a high percentage of kids shows, a group for high percentage sports, high percentage movies, high percentage reality, high online viewing, etc. Then these classified groups would later be broken down with K-means into smaller groups. These classifications might match customer groups that marketing uses.

Depending on the number of curation rules that the marketing team defines, we will do filtering on the data and have a corresponding number of clustering jobs.

As for clustering on the STB data input, we select all the user viewing histories where their movie viewing percentage of their total viewing is larger than a pre-configured cut-off, e.g. 20%.

3.2. Feature Selection

We have several different strategies for feature selection, depending on the kind of input data we use for clustering.

3.2.1. Clustering on Movie CBCFs

This is corresponding to clustering on the movie taxonomy tree mentioned above. According to this, we have 4 CBCF vectors for each user, including one for genres, one for sub-genres, one for movie types and one for movie groups. We denote the 4 vectors as L1, L2, L3, L4, respectively.

The strategy for selecting features is described as follows. We start by clustering users with [L4] vectors. Then, we keep all clusters that satisfies the tests presented in Section 3.7. Those clusters that do not satisfy the tests will be grouped together and then be clustered again with [L3,L4] vectors. Repeat this process, i.e., keep satisfied clusters and do clustering with the rest by [L2,L3,L4], [L1,L2,L3,L4], etc. until we get all clusters passing the tests or until we cannot move anymore, where we will have the final clustering result.

In addition to the strategy for feature selection described above, we have another alternative. That is we start by clustering users with [L1,L2,L3,L4]. Assume that we get a result of 6 clusters, where 2 passes the tests, then users belonging to the 4 failed clusters will be grouped and clustered again, but this time we represent each user as [L2,L3,L4]. This process will repeat as for [L3,L4], and then [L4] until we get all clustered passing the tests or until we cannot move anymore.

3.2.2. Clustering on General CBCFs

This is to apply on CBCF vectors presented in Section 2.1, the so-called general CBCFs. For each user, we have a set of several general CBCF vectors for representation. For example for each main category, we have a vector of genres, a vector of directors, a vector of actors, a vector of moods, a vector of tones, etc. As such, we have several choices of combining these vectors together for clustering. For initial simplicity, we will aggregate all vectors for clustering. New strategies such as multi-level based clustering can be easily added as in Section 3.2.1.

3.3. Normalization and Weighting

3.3.1. Normalization

Feature values of CBCF vectors need to be normalized into the range [0,1] such as [X-min(X)] / [max(X)-min(X)].

3.3.2. Weighting

One way to reduce the influence of noise in K-Means is to use a distance function with weights. We can heavily weight factors that we have a high degree of confidence in and underweight less reliable ones.

To find factors that we overweight and underweight we can look at statistics on the distribution of the factors in a sample of customers. This requires one more task for data analytics. However, at first implementation, we will tune and assign weights manually.

3.4. Clustering Initialization

For initializing K, we can use the Canopy clustering algorithm [2] which is often used as a pre-processing step for K-means to determine a suitable value for K, also Canopy will help solve the issue of initializing centroids. A detailed description of the Canopy algorithm is given in Section 4. As there are no parallel version of Canopy on R Revolution, we need a research effort to parallelize this algorithm on R Revolution.

3.5. Core Clustering Algorithm

There are several clustering algorithms available in the literature such as K-means, Fuzzy, Dirichlet, etc. Here, we will use K-means because of the following reasons: 1. We use R Revolution for implementation, 2. R Revolution only supports K-means and 3. We try to use something tested for stability and minimize our effort on implementing a new one. A detailed description of K-means is given in Section 5.

3.6. Produce Representatives

After clustering is done, similar user profiles, i.e. CBCFs, are now grouped into same cluster. Then, a new problem occurs: how to select a good representative for individuals of a cluster. There are a number of methods to build a representative for a cluster. Normally, an average of all individuals within the cluster will be calculated and selected as the representative. A more complicated approach is to find an individual in the cluster that has a smallest difference with the others. The user profile of this individual will be selected as the representative of the cluster. We refer the user profile of the selected representative to a superprofile. As such, each generated cluster will be represented by a superprofile which is then stored in HDFS or HBase. The set of superprofiles will then be transferred to the DigitalSmith engine for recommendation.

It should be noted that not only the clustering algorithm is important, but the approach of selecting good representatives is also crucial in the fact that a bad representative will lead to a bad recommendation. This consequence is quite reasonable according to the GIGO principle: garbage in then garbage out. Therefore, different approaches of selecting representatives will be considered when recommendation quality is evaluated.

3.7. Testing

For a clustering to be good we want for each cluster in the clustering (The s% numbers could be parameterized):

- The top 10% most frequent Series in a cluster will appear in more than 75% of the customers viewing history in that cluster.
- The bottom 10% least frequent Series will appear in less than 25% of the customer viewing history in that cluster.
- The top 10% most frequent Movies will appear in more than 60% of the customers viewing history in that cluster.
- The bottom 10% least frequent Movie will appear in less than 10 % of the customers viewing history in that cluster.
- The top 10% most frequent Sport will appear in more than 50% of the customers viewing history in that cluster.
- The bottom 10% least frequent Sports will appear in less than 5% of the customers viewing history in that cluster.
- The top 10% most frequent events will appear in more than 30% of the customers viewing history in that cluster.
- The bottom 10% least frequent events will appear in less than 5% of the customers viewing history in that cluster.

4. Canopy Clustering Algorithm

4.1. Introduction

Canopy Clustering Technique is first introduced in [2]. This technique is efficient when the problem is large in all of three ways at once:

- there are a large number of elements in the dataset,
- each element has many features,
- there are many clusters to discover.

The key idea is to use a cheap distance measure to efficiently divide the data into overlapping subsets we call canopies. The number of canopies can be used to guess the number of clusters K and the initial centroids for K-means that is not usually specified.

The Canopy Clustering algorithm has gained a certain level of popularity over the years as demonstrated by the Google Lecture on Cluster Computing and MapReduce, the open source implementation of the algorithm in the Apache project Mahout [3].

4.2. Algorithm

The algorithm creates a list of canopies where each canopy is assigned elements from the data set. The canopies may or may not have elements in common. After the canopies are created they can be used to reduce the number of distance operations performed in any clustering algorithm such as K-means or divisive agglomerative hierarchical clustering.

Inputs: a set of points D, a set of canopies C, a distance metric, d(x,y), thresholds T1, T2 where T1 > T2.

Outputs: a set of canopies C.

Algorithm:

```
while D is not empty:
begin

   Select element d from D to initialize canopy c.
   Remove d from D.


   Loop through remaining elements x in D:
   begin

      if d(x,c) < T1: add element to the canopy c.

      if d(x,c) < T2: remove element from the set D.

   end

   add canopy c to the list of canopies C.
end
```

4.3. Example

We consider an example as in Figure 2. The set of points are divided into 5 canopies with centers A, B, C, D and E. Points belonging to the same cluster are colored in the same shade of gray. First, point A was selected at random and forms a canopy consisting of all points within the threshold. Points inside the inner threshold are excluded from being the center of, and forming new canopies. Canopies for B, C, D, and E were formed similarly to A. Notice that while there is some overlap, there are many points excluded by each canopy. Expensive distance measurements will only be made between pairs of points in the same canopies, far fewer than all possible pairs in the dataset.

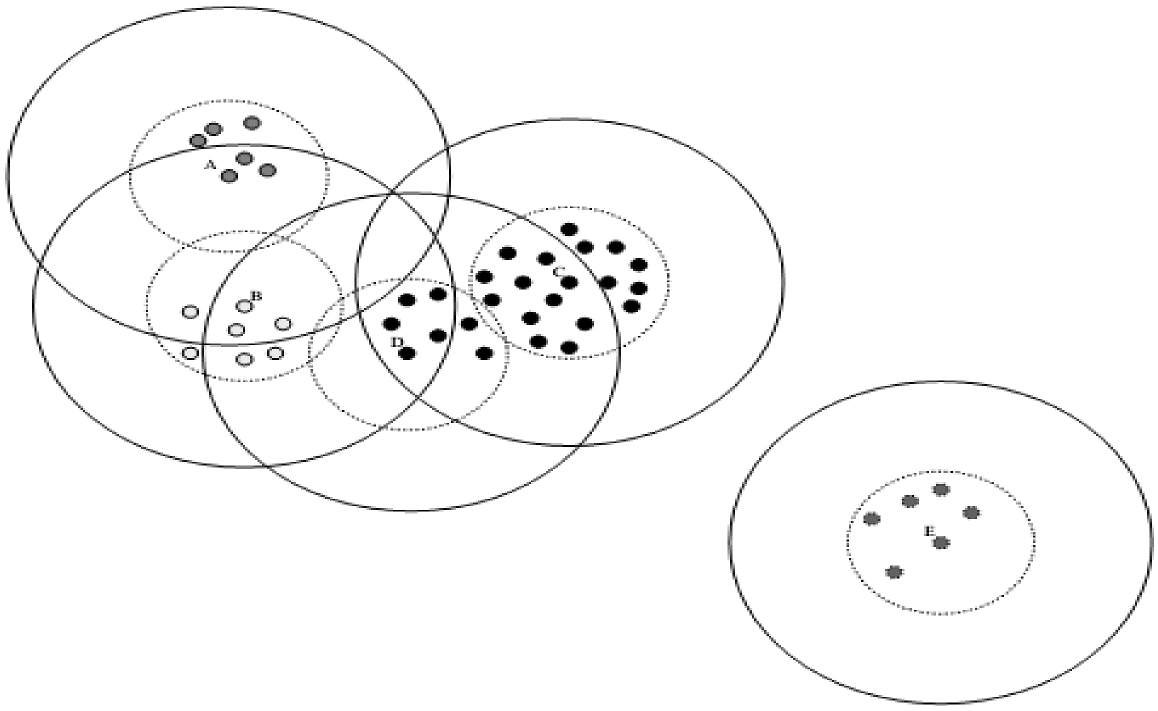

Figure 2. An example of four data clusters and the canopies that cover them.

4.4. Combining with K-means

After executing Canopy Clustering, we can guess the number of clusters K. Then we execute K-means by using the accurate distance measure. We do not need to calculate the distance between two points that never appear in the same canopy. If the canopies are not too large and do not overlap too much, then a large number of expensive distance measurements will be avoided, and the amount of computation required for clustering will be greatly reduced.

A relation between Canopy clustering and K-means is that: Canopy clustering may not lose any accuracy of K-means while still increasing computational efficiency if for every cluster of K-means, there exists a canopy such that all elements of the cluster are in the canopy.

4.5. Efficiency of Canopy Clustering

The efficiency of Canopy Clustering depends on how to choose thresholds and distances from the canopy center. Usually, we choose a cheap distance measure for Canopy Clustering. The thresholds T1, T2 are chosen by experiments.

For a 2D geographic dataset, a domain expert can define the distance thresholds easily. We can compute the distances between all pairs. Plot them as a histogram in various distance ranges. From this information you can choose T1 and T2 (e.g. choose them so that we can cover the distance range that is the most populated). See Figure 3.

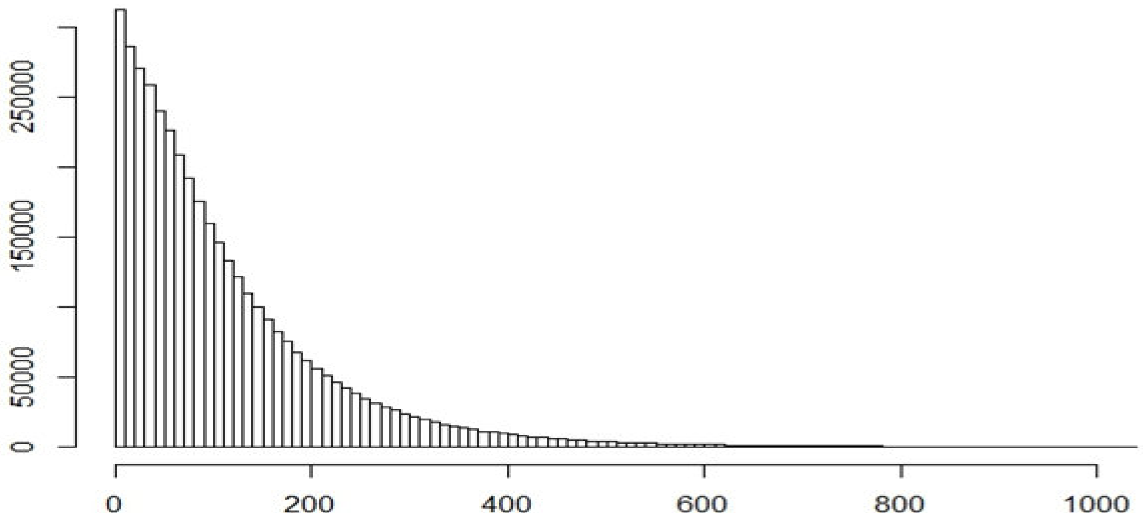

Figure 3: A histogram in various distance ranges.

5. K-means Clustering Algorithm

5.1. Introduction

K-means is one of the simplest unsupervised learning algorithms that solve the well-known clustering problem. The procedure follows a simple and easy way to classify or partition a set of n data points (a set that contains no information as to class identity) into k clusters in such a way that data points that belong to a cluster are similar in some way. Formally, given an integer K and a set of n data points, the goal is to choose K centers so as to minimize the function θ, the total squared distance between each point and its closest center.

5.2. Algorithm

It is a standard practice to choose the initial centers uniformly at random from the data set D. The idea here is that the loop While is guaranteed to decrease θ, so the algorithm makes local improvements to an arbitrary clustering until it is no longer possible to do that.

Inputs: a set of points D, number of clusters K, a set of K clusters $C = \{C_1, C_2, \dots, C_k\}$ where $C_i = \{\emptyset\}$.

Outputs: a set of K clusters C.

Algorithm:

```
Arbitrarily choose K initial centers $\{c_1, c_2, \dots, c_k\}$.
While set C changes:
begin
    For each $i \in \{1,2,\dots,K\}$, set the cluster $C_i$ to be the set of points
    in D that are closer to $c_i$ than they are to $c_j$ for all $j \neq i$.

    For each $i \in \{1,2,\dots,K\}$, set $c_i$ to be the center of mass of all
    points in $C_i$: $c_i = \frac{1}{|C_i|}\sum_{x \in C_i} x$.

end
```

5.3. Distance Functions

We calculate the distance between user u1 and user u2 based on the distance between two multisets representing the 2 users. For example, if users are represented by a combination of CBCF vectors like genres, directors, actors, moods, tones, and themes, we then calculate $\mathrm{d_{theme}(u1,u2)}$, $\mathrm{d_{genre}(u1,u2)}$, $\mathrm{d_{director}(u1,u2)}$, $\mathrm{d_{actor}(u1,u2)}$, $\mathrm{d_{mood}(u1,u2)}$ and $\mathrm{d_{tone}(u1,u2)}$. Then we compute the distance between u1 and u2 using weights for each CBCF vector as follows:

$$\begin{aligned}\mathrm{d(u1,u2)} = {} & \mathrm{w_{theme}d_{theme}(u1,u2)} + \mathrm{w_{genre}d_{genre}(u1,u2)} + \mathrm{w_{director}d_{director}(u1,u2)} \\ & + \mathrm{w_{actor}d_{actor}(u1,u2)} + \mathrm{w_{mood}d_{mood}(u1,u2)} + \mathrm{w_{tone}d_{tone}(u1,u2)}\end{aligned}$$

As mentioned in Section 3.3.2, the weight of a factor is determined by looking at statistics on the distribution of the impact factor in a sample of customers. This requires one more task for data analytics. However, at first implementation, we will tune and assign weights manually. For each type of CBCF vector described in Sections 2.1 and 2.2, there will be a corresponding weight.

There are several methods for calculating the distance between 2 vectors such as Hamming distance, Jaccard distance, Cosine distance, Euclidean distance, etc. It is not clear right now which is the best function selected for computing vector distance or vector similarity. Currently there are a number of functions supported in R Revolution, thus, we will start with the traditional Euclidean function and then tuning gradually.

6. Execution Plan

TBD

7. Conclusion

We have presented in this document a detailed design of clustering. The document described different input data requested and suggested some more required analytics functions. In this report, we also presented in detailed 2 clustering algorithms namely Canopy and K-means. An execution plan was also suggested for implementation. However, the whole design will not be implemented all in one, instead done in step-by-steps and hence, clustering can be evaluated, tuned and updated according to business demands.

8. References

[1] Content Based Collaborative Filtering Calculation, DirecTV Data Analytics Design Document, Designed Function: Content Based Collaborative Filtering, Oct 2014.

[2] A. McCallum, K. Nigam, L.H. Ungar, “Efficient Clustering of High-Dimensional Data Sets with Application to Reference Matching”, in Proceedings of 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 169-178, 2000.

[3] Apache Mahout, https://mahout.apache.org/, Jul 2014.