# Assessing Distributed Content Delivery via Content and Consumer Clustering

Josh Kimmel
*UCLA, Computer Science*

Peter Jourgensen
*UCLA, Computer Science*

## Abstract

**With the explosion of media streaming services such as Netflix, a major issue in the quality of content delivery is bottlenecking at the content delivery nodes deployed across Content Delivery Networks ("CDN's"). Our aim is to assess the feasibility of constraining the storage requirements of individual content delivery nodes in the CDN by fixing the genres of the content each one supplies. If the genre groups are sufficiently unique and content is well-distributed across these groups, the storage required of individual nodes can be constrained in terms of its rate of growth. This can help financially justify the deployment of more nodes in the network, thereby reducing bottlenecking. Furthermore, if the genre distribution is such that consumer genre preferences can be mapped close to injectively on to it, applying such a mapping when serving consumer requests would help distribute the load across more servers, further reducing bottlenecking. The reduction of bottlenecking as described above would result in a better and more consistent end user experience for streaming services. Our results indicate that such groupings can lead to more favorable constraints for content delivery node resources.**

## 1 Introduction

With the rise of media streaming services such as Netflix, the landscape for how users consume digital content (movies, TV shows, etc.) has changed dramatically in recent years. Many major media streaming service providers utilize a Content Delivery Network (CDN) to serve requests from users for content. CDN's are designed to help ease the load of requests for content by distributing the location of the provider of this content geographically [8]. A CDN typically consists of many content servers to which a given user can be directed to obtain his/her requested content. The direction is typically performed by some master software controller in a Cloud environment.

However, as the demand for content continues to grow at an exponential rate, a major limitation to CDN's today is their ability to deal effectively with bottlenecks at the content delivery servers [1]. Bottlenecks ultimately result in poor user experiences such as buffering, lags, and crashes.

The obvious solution to this problem is to expand the network by adding more content delivery servers - each servicing a more confined geographic region. However, the feasibility of such a solution is severely limited by the cost of deployment and maintenance of these additional servers. Especially given the extremely large base of content that must be delivered, each server in the network must have a storage system with a high and very scalable capacity.

However, this problem can in theory be mitigated by taking a slightly different approach. Instead of focusing on geographic location as the major decision factor in request routing, our proposal is to focus on content genre relative to requestor genre preferences.

There has been some initial thought and exploration into quantifying consumer preferences [7] as well as clustering media content [10], but there is no well-known technique (or even attempt[1]) at combining the two in the context of content delivery.

Our hypothesis is that content genres can be clustered in such a way that maps bijectively[2] to clusters of consumers in terms of the content genres that they prefer. A large enough variation in content genre clusters could help impose storage constraints[3] on the servers that deliver content of the given genre. Furthermore, such a mapping could help better distribute requests for content across the network. Both consequences would ultimately result in a better user experience for such prevalent media streaming applications which, in such a competitive market, could make a huge difference for these businesses.

[1] At least per the authors' research.

[2] Or at least close thereto.

[3] Namely, on the expected growth rate of storage capacity. This assumes that content is well-distributed overall across these clusters.

## 2 Clustering Techniques

As the primary means for our analysis is clustering of both content by genre and consumer by genre preferences, the clustering algorithm we use has major implications towards our ultimate conclusions. For this reason, we spent quite some time assessing which clustering algorithms suited our research best.

Our desired qualities out of a clustering algorithm are:

- Either yields a configurable number of clusters or tends to yield a large number of clusters. This is because more clusters ideally implies less content per cluster and fewer consumer requests mapped to a given cluster.
- Tends to yield clusters that are close in size. This is because, assuming our data set is representative, small clusters would under-utilize storage and large cluster would over-utilize storage, rendering our solution less cost effective.
- Lower asymptotic complexity - especially in the size of the data set. We are working with very large data sets on modest machines and need to run many simulations.

In examining various popular clustering algorithms, we concluded that Mean-Shift [2], Affinity Propagation [11], and Agglomerative Clustering [4] are not scalable enough. Spectral Clustering tends to yield too few clusters for our purposes [5]. DBSCAN tends to yield uneven cluster sizes [3]. This left us with K-Means and, more generally, Gaussian Mixtures.

Given the relative simplicity, configurability, and suitability to our needs of K-Means clustering, we decided to implement this algorithm for our research. The implementation of our K-Means library can be found at `https://github.com/joshkimmel16/ContentCluster`.

We chose to implement our own K-Means clustering library[4] because we wanted to play with various formats for the input data and thus various interpretations of the distance between two points and the mean of a set of points. Our implementation allows for generic functions to be plugged in that best suit the structure of the input data set.

### 2.1 K-Means Clustering Considerations

One major factor in both the performance and ultimate results for K-Means clustering is the formation of an initial partition on the data set. Though there has been considerable research [9] into the optimization of initial partitions, we opted to use a simple random partitioning technique as none seemed to particularly suit our experiment.

As randomness can inherently result in different outcomes across executions, our clustering experiments were run multiple times to ensure that the results were not drastically different.

## 3 Content Clustering by Genre

The source data set of the content clustering portion of our research is from themoviedb.org (`https://www.kaggle.com/stephanerappeneau/350-000-movies-from-themoviedborg`). This data set provides one file in particular - `AllMoviesDetailsCleaned.csv` - that tags each of the 350,000[5] movies with a subset of a predefined set of genres. These genres are: **Drama, Crime, Comedy, Action, Thriller, Documentary, Adventure, Science Fiction, Animation, Family, Romance, Mystery, Music, Horror, Fantasy, War, History, Western, Foreign, TV Movie**.

### 3.1 Pre-Processing

In our pre-processing step, we use a Python script to generate the ultimate data set that is fed into the clustering algorithm. This script examines each movie and outputs an array of objects containing: **MovieID, MovieName, GenreBitVector**. The `GenreBitVector` is an array of bits where $\text{bit}_i$ is asserted if and only if the given movie has been tagged with the $i^{th}$ genre per the ordering of genres above.

### 3.2 Distance Functions

We have explored several possible definitions of the "distance" between two points in the data set. Each of these definitions is elaborated upon below:

**DirectCompare** Takes a scalar parameter $\lambda$. Compares the input `GenreBitVectors` of each data point bit-wise and returns $\sum_{MismatchedBits} \lambda$.

**DirectDiff** Takes a scalar parameter $\lambda$. Compares the input `GenreBitVectors` of each data point bit-wise and returns $\lambda \times \sum_i |\texttt{GenreBitVector}_0[i] - \texttt{GenreBitVector}_1[i]|$.

Our analysis concluded that `DirectDiff` lends itself better[6] to computations pertaining to the mean of a subset of data points with the given structure and thus we used this definition of distance for clustering.

[4] As opposed to using an existing implementation.

[5] This is an approximation.

[6] Means are better suited to have ranges of values in $[0,1]$ making `DirectCompare` nonsensical when evaluating the distance between a point and a mean.

### 3.3 Mean Functions

There was only one function that we deemed appropriate for the computation of the mean of a subset of data points in the content data set. This function is described below:

**ArithMean** Computes the arithmetic means of each data point bit-wise and returns a vector of these means. The arithmetic mean for the $i^{th}$ bit is defined as: $\frac{\sum_{j=0}^{j=\#ofDataPoints} \texttt{GenreBitVector}_j[i]}{\#ofDataPoints}$.

### 3.4 Error Functions

In computing the overall error associated with assigning a given data point to a given cluster, there were 3 major factors that we tried to incorporate:

- Minimal aggregate distance of points within a cluster to the cluster's mean. In the limit, this implies that movies within a cluster are noticeably similar.
- Maximal aggregate distance of points within a cluster to the means of other clusters. In the limit, this implies that movies in different clusters are noticeably different.
- Minimal variance in the size of each cluster from the ideal cluster size. The ideal cluster size is defined as: $\frac{\#ofDataPoints}{k}$. In the ideal cluster size scenario, assuming our data set is representative of all content, this implies that each cluster requires the same amount of storage for its content and total required storage is evenly distributed across the clusters.

We explored several possible functions:

**IntraErrorOnly** Selects the cluster that minimizes the distance[7] between the given point to that cluster's mean[8].

**IntraClusterSize** Selects the cluster that minimizes `IntraErrorOnly` + $\lambda \times$ *ClusterSizePenalty*. `ClusterSizePenalty` penalizes clusters that deviate in size from the ideal cluster size.

The `IntraErrorOnly` component of the error functions, by construction of the K-Means algorithm, accomplishes our goals of minimizing aggregate error within clusters and maximizing error between clusters [6]. The `ClusterSizePenalty` component attempts to bias clusters towards their ideal cluster sizes.

#### 3.4.1 Cluster Size Penalty Functions

We explored two functions for computing the `ClusterSizePenalty` component of the error functions:

[7] Defined in section 3.2

[8] Defined in section 3.3

**SimpleDiff** Returns |ActualClusterSize - IdealClusterSize|.

**GaussianDiff** Returns $\frac{1}{P(ActualClusterSize)}$ where the distribution is a Gaussian of mean *IdealClusterSize* and variance $\frac{(k-1)n^2}{k^2}$ [9].

Although `GaussianDiff` has a better biasing effect given its aversion to cluster sizes that deviate greatly from the ideal cluster size, with a small enough dampening factor $\lambda$, the difference between the two functions becomes minor. Furthermore, `GaussianDiff` is considerably more computationally expensive. For these reasons, we chose to use `SimpleDiff`.

## 4 Consumer Clustering by Genre Preferences

The source data set of the consumer clustering portion of our research is from the Netflix Prize Competition previously hosted on Kaggle (https://www.kaggle.com/netflix-inc/netflix-prize-data). This data set provides four files containing user ratings (1-5) for unique movies or shows and one file that maps movie identification codes to specific titles and years of release. In total, this data contains information for over 450,000 users and 17,000 titles, resulting in roughly 100,000,000[10] individual entries.

### 4.1 Pre-Processing

In our pre-processing step for clustering consumers by genre preferences, we must develop a metric for each consumer's aggregate score of each genre.

#### 4.1.1 Mapping Genres to MovieIDs

First, we had to determine the genres of each movie in the data set. The Netflix data contains no information on genres of each movie, so these had to be mapped from our previous data set used for content clustering.

We began by performing an inner join on the two data sets, whereby the movie titles of the Netflix data was used as the primary key and the movie titles of the themoviedb.org data was used as the foreign key. Next, release year, present in both data sets, was leveraged to confirm accurate matching. Finally, duplicate entries and entries that contained no information on genres were removed. This resulted in a master data set containing unique mappings for Netflix movie identification codes and their corresponding genres[11]. Only these movies would be considered for further analysis.

[9] This can be derived by setting $E[x^2] = \frac{n^2}{k}$ where n is the number of data points

[10] These are approximations.

[11] Results in 8,429 movies

#### 4.1.2 Consumer Scoring of Genres

For the next step of scoring genes on an individual consumer basis, we developed a Python script to read in the data and determine all of the movies a consumer had seen and their corresponding ratings. For improved computational time and data quality, only users with more than 150 movies seen and rated were considered for further analysis[12]. The following procedure was then applied to transform the consumer ratings on an individual basis:

1. **Normalize.** This adjusts for consumers having different relative rating systems and puts all of the ratings on the same scale. Outliers[13] were then removed.

2. **Exponentiate.** This not only transforms the ratings to all be positive, but also means that higher ratings have increased variance. When applying KMeans clustering, this results in consumers being more likely to be clustered by their interests than their dislikes.

3. **Scale.** Divide ratings by number of movies consumer has rated. When the scores are later summed, this puts the final sums on the same scale for every consumer.

Finally, we iterated through every consumer and every movie and added the transformed rating to each of the corresponding genres. This ultimately results in a data set containing genre scores for each consumer that can then be used to cluster consumers by their genre preferences.

### 4.2 Distance Functions

We explored two additional distance functions for our K-Means clustering of consumers because of the different structure of its data. These are elaborated upon below:

**SquareDiff** Takes a scalar parameter $\lambda$. Compares the input `GenreScoreVectors` of each data point bitwise and returns $\lambda \times \sum_i (\texttt{GenreScoreVector}_0[i] - \texttt{GenreScoreVector}_1[i])^2$.

**CubicDiff** Takes a scalar parameter $\lambda$. Compares the input `GenreScoreVectors` of each data point bitwise and returns $\lambda \times \sum_i |\texttt{GenreScoreVector}_0[i] - \texttt{GenreScoreVector}_1[i]|^3$.

We found `CubicDiff` to over penalize points that were far away from each other, resulting in clusters that did not generalize well. Therefore, we chose to continue with our results from using `SquareDiff`.

[12] Results in 158,979 consumers

[13] Corresponds to normalized ratings with values greater than 2

### 4.3 Mean Functions

The `ArithMean`[14] was, likewise, the only function that we deemed appropriate for the computation of the mean of a subset of data points in the consumer data set.

### 4.4 Error Functions

`IntraErrorOnly` was deemed appropriate for the optimization of the individual clusters.

Penalizing the differences in cluster sizes did not prove to be necessary as the clusters were fairly distributed for each k. In a real world scenario, this plays a critical role. The distribution of content, whether it be through targeted servers or taken to the extreme with peer-to-peer sharing, is constrained by congestion and cost. On the upper end, large consumer clusters results in higher traffic to the distributed servers, thus re-generating the same problems that are currently facing the industry. On the lower end, small consumer clusters become costly to serve on a per capita basis. Thus, we penalize cluster size extremes in an attempt to mitigate these issues.

## 5 Mapping of Consumer Clusters to Content Clusters

Once all of the clustering results have been generated both for customer genre preferences and movie genres, the final step in our analysis is to find the optimal mapping of customer clusters to movie clusters for some pair of clustering results.

Given the relatively small size of the result set for each clustering simulation[15] and the relatively small size of clusters per result [16], we decided to take a brute-force approach to the evaluation of mappings.

Namely, for each clustering result for customers and each clustering result for movies (pairwise), we found the movie cluster that minimized a cost function that evaluates the "distance" between the given movie cluster and each customer cluster. The normalized[17] sum of the "best" mappings of each customer cluster to movie cluster is classified as the total error for the given pair of clustering results. The minimal total error across all possible pairs of clustering results is what we define as the optimal mapping.

### 5.1 Mapping Cost Functions

Given the fact that both customer and movie cluster means are of the same format, namely a vector of real values in $[0,1]$ for each of the predefined genres, we determined that a simple

[14] Defined in section 3.3

[15] Determined by the range of k values used.

[16] Also determined by the range of k values used.

[17] Normalization is based on number of customer clusters. Otherwise, the sum would favor smaller amounts of clusters.

difference function would suffice. This function is described below:

**DirectDiff** Takes a scalar parameter $\lambda$. Compares the input customer and movie cluster `MeanVectors` bitwise and returns $\lambda \times \sum_i |\texttt{MeanVector}_{consumer}[i] - \texttt{MeanVector}_{movie}[i]|$.

For a given pair of clustering results (one consumer, one movie), each consumer cluster is mapped to its best movie cluster by finding the movie cluster whose mean vector, when paired with the given consumer cluster mean vector, minimizes the `DirectDiff` cost function.

Once the best movie cluster has been identified for each customer cluster, the overall cost of the given clustering result pair is defined as the sum of the outputs of `DirectDiff` for each best mapping. To ensure that clustering result pair costs are comparable, this aggregate cost is normalized by dividing out the number of consumer clusters (k-value) for the given pair.

The optimal clustering result pair is defined as the pair whose normalized cost is minimal.

# 6 Results

The sections below outline the results we obtained through our initial experiments. It is worth noting that several more experiments are planned to be conducted in the future.

## 6.1 Movie Clustering Results

For movie clustering, two primary experiments were conducted:

- K-Means clustering for k=$[2,20]$ where the distance function used was `DirectDiff` with $\lambda = 1$, the mean function used was `ArithMean` and the error function used was `IntraErrorOnly`.
- K-Means clustering for k=$[2,20]$ where the distance function used was `DirectDiff` with $\lambda = 1$, the mean function used was `ArithMean` and the error function used was `IntraClusterSize` with $\lambda = 1$.

See https://drive.google.com/drive/u/0/folders/1IaQjcQrhl9kTY-bfPQ1j_YiilQMwE4bu for the raw result .json files.

## 6.2 Consumer Clustering Results

For consumer clustering, only one primary experiment was conducted:

- K-Means clustering for k=$[4,20]$ where the distance function used was `SquareDiff` with $\lambda = 1$, the mean function used was `ArithMean` and the error function used was `IntraErrorOnly`.

| Ranking | Optimal Customer K-Value | Optimal Movie K-Value | Cluster Error (Normalized) |
|---|---|---|---|
| 1 | 20 | 17 | 2.073 |
| 2 | 8 | 17 | 2.136 |
| 3 | 4 | 17 | 2.138 |
| 4 | 5 | 17 | 2.139 |
| 5 | 7 | 17 | 2.157 |

| Ranking | Optimal Customer K-Value | Optimal Movie K-Value | Cluster Error (Normalized) |
|---|---|---|---|
| 1 | 20 | 4 | 2.198 |
| 2 | 5 | 4 | 2.245 |
| 3 | 4 | 4 | 2.254 |
| 4 | 7 | 4 | 2.266 |
| 5 | 20 | 13 | 2.268 |

Figure 1: Experiment 1 results on top, Experiment 2 results on bottom

## 6.3 Cluster Mapping Results

For cluster mapping, two primary experiments were conducted:

- Using customer clustering results of experiment 1 (see 6.2) and using the the movie clustering results of experiment 1 (see 6.1). Using the mapping cost function `DirectDiff` with $\lambda = 1$.
- Using customer clustering results of experiment 1 (see 6.2) and using the the movie clustering results of experiment 2 (see 6.1). Using the mapping cost function `DirectDiff` with $\lambda = 1$.

See Figure (1) for the "best" mappings computed in each experiment.

# 7 Analysis

## 7.1 Movie Clustering

The goal in movie clustering, independent of the optimal mapping to consumer clusters, was to assess the feasibility of grouping quantifiably "similar" media content such that each group is similar enough that it would be reasonable to expect that only a subset of consumers would be actively requesting its content while at the same time being specific enough that the physical storage required for the given cluster can be reasonably constrained.

The logical metric to assess the similarity of content within a cluster is the final aggregate error upon convergence of K-Means. The smaller this aggregate error is, the better the mean of the given cluster characterizes its members and it can thus be inferred that the members are similar to the mean and, by

transitivity, each other. Per the results of both clustering experiments, as one would expect, the nominal aggregate errors and variances within clusters decreased as k increased. It is worth noting that average error within clusters was consistently higher in the `IntraAndClusterSize` experiment, as might be expected when penalizing clusters of certain sizes. This is a positive trend for the hypothesis, but, without further experimentation and labeling, it is difficult to qualify the nominal error values in terms of a true similarity rating.

The logical metric to assess the specificity of content within a cluster is to look at the average "distance" between the means of clusters. The bigger this difference is, the more dissimilar the content in each cluster is - which effectively implies that clusters are increasingly specific. Once again, per the results of both clustering experiments and as expected by the construction of K-Means, the average distance and variance across clusters increased as k increased. It is worth noting that average error across cluster means was consistently lower in the `IntraAndClusterSize` experiment, as might be expected when penalizing clusters of certain sizes. This is again a positive trend for the hypothesis.

Finally, to help understand the physical storage constraints that can be imposed on a given cluster, the distribution of clusters sizes can be analyzed. Particularly large or small clusters, assuming the movie data set is representative of the entire population of content, would not permit very tight bounds on the physical storage required per cluster. In contrast, uniform cluster sizes would yield the optimal upper bound on storage. In each experiment, as expected, the average cluster size decreased as k increased. However, the `IntraAndClusterSize` experiment, given its penalty imposed on large and small clusters, yielded lower cluster size variances for all k.

In general, the most troubling trend is the existence of outliers in cluster sizes across all k in both experiments. There were always at least some clusters significantly smaller and significantly larger than the average cluster size. It is thus difficult to confidently constrain the physical storage required for a given cluster without further analysis.

## 7.2 Consumer Clustering

The goal with the consumer clustering portion of this research was to determine whether or not there was enough variability and balance in consumer interests to form distinct "communities" of consumers that in the extreme may operate as a peer-to-peer network to share content with one another. To function in the real world, these communities must be large enough to locally host a significant amount of content, but small enough to be practically managed by a controller to avoid congestion and security concerns.

Following the same reasoning discussed in the analysis of movie clustering, the final aggregate error was used to assess the clustering specificity. As would be expected, the total aggregate error decreased as the number of clusters, k, increased. It is worth mentioning that, traditionally, the optimal k for K-means clustering is chosen where the error begins decreasing at a decreasing rate[18]. While we were sure to include this "optimal" point in the ranges of both clustering experiments, taking the corresponding k values may not be the optimal mapping for this research. This is because increased specificity in clustering may reduce the mapping cost. Therefore, the entire range of k's were analyzed during the mapping.

Next, the distribution of cluster sizes for each k value can be analyzed to assess the feasibility of real world application. Cluster sizes deviating from the expected cluster size[19] were not penalized for consumer clustering because it was deemed unnecessary given the balance in the results of using `IntraErrorOnly`. Only at k=20 was a single outlier detected among cluster sizes. This consistent uniformity is reassuring and suggests a certain amount of homogeneity in consumer preferences. These bounds ensure that the potential peer-to-peer networks have enough collective storage capacity, but are not so large that they are unmanageable.

## 7.3 Cluster Mapping

The goal in cluster mapping is to assess how well requests across consumer clusters would be distributed across content clusters. In the ideal scenario, fixing the numbers of both consumer and content clusters, there would be a one-to-one mapping (or as close as the numbers of clusters can get).

In experiment one, the optimal cluster mapping was k=20 for consumers and k=17 for content. In experiment two, the optimal cluster mapping was k=20 for consumers and k=4 for content.

In both experiments, the trend across consumer cluster numbers was that more clusters yielded lesser (normalized) mapping error. This result is promising as the more consumer clusters that exist, fixing the average cluster size across consumer clusters, the more constrained the number of expected requests per cluster per unit time can be. Tighter upper bounds favor our flavor of a distributed network.

The optimal cluster size for content clusters across experiments varied greatly and there was no visible trend. It cannot be stated with confidence that a one-to-one mapping is favored, as there was only one experiment in which a (close to) one-to-one mapping was optimal. Further experiments would need to be conducted to create a mapping data set that can be properly analyzed.

[18] Commonly referred to as the "Elbow" point

[19] Total Samples / k

## 8 Conclusions

Content delivery is exponentially growing as both the quantity of content and resolution that it is developed in continue to grow at increasing rates. The initial goal of the Content Delivery Network was to distribute the data storage and delivery away from a central location to reduce the number of requests and avoid having a single point of failure. The CDN's themselves, however, are beginning to face a future where the demand for content outweighs their distribution.

In this paper, we presented a solution to combat this imbalance by taking the distribution to the extreme with a peer-to-peer content sharing network or similar architecture. We first demonstrated that content can be clustered by genre categories and consumers can be clustered by genre preferences. We then mapped the consumer clusters to content clusters to find the pairings with the least distance between them.

Increased distribution of CDN's is still a relatively new area of research, but is one that is growing more necessary each day. While we addressed this problem through the lens of Netflix user preferences and movie genres, it could easily generalize to other forms of content delivery as well, be it music or other streaming services. This paper proposed a singular solution, but we recognize the opportunity and encourage the exploration for novel approaches and points of optimization.

In the future, we hope to continue to improve our framework in terms of cost, mean, and error functions as well as their optimal parameters. The most important next steps are to continue to generate and compare results under various conditions to better understand the high-level structure of both media content and consumer content preferences.

## References


[1] Akamai. Live streaming solution brief. 2018.

[2] S. Craciun, G. Wang, A. D. George, H. Lam, and J. C. Principe. A scalable rc architecture for mean-shift clustering. In *2013 IEEE 24th International Conference on Application-Specific Systems, Architectures and Processors*, pages 370–374, June 2013.

[3] Levent Ertöz, Michael Steinbach, and Vipin Kumar. Finding clusters of different sizes, shapes, and densities in noisy, high dimensional data. In *Proceedings of the 2003 SIAM International Conference on Data Mining*, pages 47–58, May 2003.

[4] J. Handl and J. Knowles. Improvements to the scalability of multiobjective clustering. In *2005 IEEE Congress on Evolutionary Computation*, volume 3, pages 2372–2379, September 2005.

[5] Ling Huang, Donghui Yan, Nina Taft, and Michael I. Jordan. Spectral clustering with perturbed data. In *Advances in Neural Information Processing Systems 21*, pages 705–712. December 2009.

[6] Zhexue Huang. Extensions to the k-means algorithm for clustering large data sets with categorical values. In *Data Mining and Knowledge Discovery*, volume 2, pages 283–304. September 1998.

[7] Oscar Luaces, Jorge Díez, Thorsten Joachims, and Antonio Bahamonde. Mapping preferences into euclidean space. In *Expert Systems with Applications*, volume 42, pages 8588 – 8596. December 2015.

[8] Netflix. Open connect overview. 2016.

[9] Ting Su and Jennifer G. Dy. A deterministic method for initializing k-means clustering. In *16th IEEE International Conference on Tools with Artificial Intelligence*, pages 784–786. November 2004.

[10] Lyle H. Ungar and Dean P. Foster. Clustering methods for collaborative filtering. In *AAAI Technical Report*, pages 114 – 129. August 1998.

[11] Xiangliang Zhang, Cyril Furtlehner, and Michèle Sebag. Data streaming with affinity propagation. In *Machine Learning and Knowledge Discovery in Databases*, pages 628–643, September 2008.