# Two-Variable Recommendation Models

Fall 2014
6.867 Final Project
Geoffrey Gunow and Neha Patki

## 1 Introduction

In 2014, Yelp published an academic dataset containing over 42,000 businesses that span 5 metropolitan areas. Types of data included the full star rating, and reviews for each business, as well as aggregated statistics for each user's reviews. In this project, we use the dataset to formulate, model, and solve the problem of estimating a user's overall review for a business, given the user's history. The algorithm we present estimates a single rating, but can be applied to multiple business to determine the best one for a given user. Thus, our problem falls in the category of recommendation systems.

### 1.1 Motivation

Yelp does not currently give personal recommendations to its users, but the data it collects about reviews and users make it possible to learn a user's preferences. Typical approaches fall in two categories. Collaborative filtering analyzes similarities between different users, and the items they rate highly. Content-based filtering focuses on a single user's personal history to determine individual likes and dislikes.

For Yelp's data set, we find that a collaborative filter would not take into account an individual's own preferences for cuisines and restaurant sub-categories. However, we cannot use pure content-based filtering because a single user may not have reviewed a significant number of restaurants, making the data points sparse. This motivates us to use a hybrid approach in estimation, where we combine a user's history with a cluster-based approach meant to account for the sparsity of data points. We introduce a hidden variable that models an overall group of similar users, and use this variable to separately learn preferences for each group.

An additional challenge and motivation for the problem is analyzing the given data to determine relevant features. Yelp offers a vast array of data ranging from the ambiance in each restaurant, to the number of times a user was voted funny by their peers. Furthermore, the 5 metropolitan areas are diverse in their offerings of food, and users may show preferences towards particular types of restaurants. We spend a significant time engineering feature vectors to reflect the variety of information. After the learning process, this enables us to comment on significant features to reveal trends in user preferences.

### 1.2 Problem Formulation

The overall goal is to accept, as inputs, a user and business, and to output a floating point value in $[1, 5]$ that predicts what the user will rate the business. We do this using two steps.

1. Categorize the users into groups who have groughly similar interests. This is the hidden variable.

2. Separetely train each category to learn the average business rating for only the users in that group.

For example, assume a particular group we learn in step 1 contains users who prefer Mexican restaurants. In step 2, the expected output is the average rating given only by its members. This means that another group may be trained to rate the same business differently, which shows its users have different broad preferences. Figure 1 shows an overall schematic of the system.

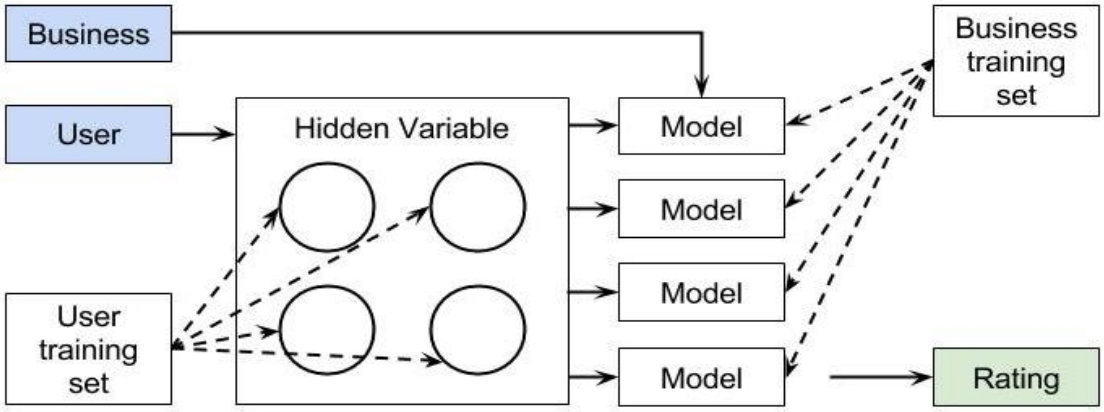


Figure 1: The architecture for the overall system. Inputs are colored blue while the output is shown in green. Dotted lines represents inputs during training, and solid lines represent the flow of information at testing time. The hidden variable is used to select a model, or a mixture of models, that the business data is tested on. The output is the rating.

Modeling the hidden variable in step 1 is an unsupervised learning problem, because we are constructing groups for which we have no prior labels. However, after learning the hidden variable, the step 2 is fully supervised because we are presented with a full list of ratings for each business. Considering the full system, we see that we can perform an end-to-end test given a user, a business, and an expected rating. This allows us to indirectly supervise the hidden variable to determine the appropriate hyperparameters for step 1. The overall complexity of our system grows, and we perform a train-validate-test strategy recursively for step 2 within the context of the overall system.

In the next sections, we discuss constructing the feature vectors that create the user and business data shown in Figure 1. We describe multiple approaches to implement both variables, and describe our train-validate-testing strategy. Finally, we compare the different methodologies in terms of accuracy and computational time.

## 2 Features

A key to success in any machine learning problem is the selection of relevant features to accurately represent the data of interest. While algorithms are important, the success that can be achieved is often bounded by the ability of the features to accurately represent the problem. In our project, we need to develop two sets of features. The first set of features corresponds to user attributes that

will be used to predict the hidden class associated with each user. The second set corresponds to business attributes that will be used to predict how a user in a given class will rate a restaurant.

### 2.1 User Data

The Yelp dataset is comprised of three relevant files: a user review file, a user attribute file, and a business attribute file. A straightforward approach would be to classify a user based purely off the data presented in the user attribute file. This file contains information about the user along with a user ID. However, the information provided is only basic with attributes such as the total number of reviews a user has written, the average of the user's ratings, and less relevant attributes such the user's full name.

Therefore, we use a different approach to form user features. We still take a couple features from the user attribute file, namely the user's average rating and the total number of reviews, but most features are formed by cross referencing the user's ID with the user reviews file. With our approach, we find the reviews associated with the user's ID and store the rating and business ID associated with each review. Then from the business ID we can cross reference the business file to determine the type of restaurant. Yelp lists categories associated with each business. For instance, every restaurant will have the string 'restaurant' in its categories but might also have the string 'Mexican restaurant.'

This allows us to form features that describe how a user prefers different types of restaurants. In particular, there is a feature associated with the user's average rating for restaurants in each category along with another feature for the frequency a user has reviewed each category. To compensate for users that preferentially give good or bad reviews, the user's average rating is subtracted from the category-averaged rating. Therefore these features provide information on the user's preferences in restaurants, such as liking Italian food but disliking Mexican food.

One major challenge with this approach is missing data. For the vast majority of users, there will be categories in which the user has given no reviews. For most machine learning algorithms, the data needs to be complete. To remedy this, we choose to fill missing data using data imputation. We assign the user's average review to categories in which the user has no reviews. Since we subtract by the user's average review in category ratings, this corresponds to inserting zeros for these missing features.

Typically, data imputation is not the appropriate method to solve the missing data problem for large data sets [**?**]. The preferred method is expectation-maximization (EM) in which the objective function is directly maximized [**?**]. However, EM presents a few problems for our scheme. Fist, there is a large amount of missing data. For almost all users there is more than one missing feature value. In addition, many clustering algorithms do not work well with the probabilistic approach of EM. In EM, no single value is assigned to missing data points. Instead, a probability is assigned. In an iterative process estimates for this probability are updated along with density estimation parameters. Some clustering algorithms such as k-means do not perform density estimation but rather partition the feature space [**?**]. This is problematic since EM requires evaluation of the likelihood of the data.

Overall, the user feature vector contains 33 features, 28 of which are associated with the average rating and number of reviews for 14 categories of restaurants. The remaining 5 features are statistical features associated with the user, such as variance of review ratings and average review rating. All of the features are described in Table 1 and Table 2.

Table 1: User Feature Descriptions (Restaurant Categories)

| Restaurant Category | Feature Numbers | |
|---|---|---|
| | Review Frequency | Scaled Review Average |
| Mexican | 0 | 14 |
| American (Traditional) | 1 | 15 |
| Fast Food | 2 | 16 |
| Pizza | 3 | 17 |
| Sandwiches | 4 | 18 |
| Nightlife | 5 | 19 |
| Bars | 6 | 20 |
| Food | 7 | 21 |
| American (New) | 8 | 22 |
| Italian | 9 | 23 |
| Chinese | 10 | 24 |
| Burgers | 11 | 25 |
| Breakfast/Brunch | 12 | 26 |
| Japanese | 13 | 27 |

Table 2: User Feature Descriptions (Restaurant Statistics)

| Feature | Feature Number |
|---|---|
| Number of present user reviews | 28 |
| Variance of present user ratings | 29 |
| Average of present user ratings | 30 |
| Total number of user reviews | 31 |
| Average of total user ratings | 32 |

### 2.2 Business Data

Once users are partitioned into clusters, a model must be developed for each cluster that predicts the business rating. To accomplish this, features related to the businesses are required. For this set, most features are pulled directly from the list of binary attributes. The business attributes are primarily binary features such as whether a restaurant allows smoking. However, again missing data is common in the business features.

For business features, missing data is treated differently than it was for user features. This is because missing business data directly corresponds to missing data in the Yelp dataset and might actually be representative of the business. For instance if the business is new or not very popular, many attributes corresponding to the business may be missing. For established businesses, we expect that more attributes will be present.

Therefore, missing binary features are treated as categorical features. More specifically, we split each binary feature into three features. Only one feature is 1 and the other two are 0. If there is a present and positive binary feature, the first feature is marked with a 1. If there is a present and negative binary feature, the second feature is marked with a 1. In the case of missing data, the last feature in the set is marked with a 1.

A similar approach is taken with actual categorical features. For instance, the attire attribute

may be listed as casual, dressy, formal, or the data might be missing. This feature is therefore split into four features with an approach similar to the binary features wherein only one feature is marked with a 1.

Additional features are engineered from the data available. The Yelp dataset is based off data from 5 metro areas: Phoenix, Las Vegas, Madison, Waterloo, and Edinburgh. The business attributes lists the city in which the restaurant is located, but not the metro area. This is problematic since intuitively the metro area should have an impact on the rating since the five cities are very different in nature. However, the latitude and longitude of the locations is provided. Using these coordinates along with the coordinates of the centers of the major cities, as provided by Wikipedia, the metro area can easily be determined by finding the closest center. Therefore a categorical feature can be developed for the metro area much like the attire example presented previously. Additionally, distance to the center of the city might be important so a feature is added that gives the distance to the center of the city based off the coordinates.

# 3 Methodology

After constructing the feature vectors, our data now consists of a set $B$ of business vectors, and a set $U$ of user vectors. Assume the vectors are constructed as described in the previous section. We also have access a set of ratings, $R$, where a single rating is determined by a user and a business. Let $r_{ub} \in R$ denote the rating given by user $u$ to business $b$.

We first train the user vectors, $U$ with the hidden variable. This will return multiple sets of users, $U_s \subseteq U$. We then define a function that takes in a set of users and a business, and outputs the average rating given to the business only by the users in the set. Mathematically, this is:

$$f(U_s, b) = \frac{\sum_{u \in U_i} r_{ub}}{|U_i|} \tag{1}$$

Where $U_i \subseteq U_s$ is the set of users who have given a rating to business $b$. If $U_i = \emptyset$, then this function is not defined.

We can then perform an inference method for the non-hidden variable. We define its inputs and outputs in terms of the business vectors, $B$. As inputs, we have the set of businesses that some user in the set has rated. If the set is $U_s$, we denote the relevant set of businesses as $B_s$, where for every $b \in B_s$, $r_{ub}$ exists for at least one $u \in U_s$. Then the inference is trained with input $B_s$ and expected output $\{f(U_s, b)\}$, $\forall b \in B_s$.

We now present the three separate ways we formulated the problem to discover the sets of users, $U_s$. We also describe the methods of inference we use for the second variable.

## 3.1 K-Means Clustering

Perhaps the most naive way to calculate the hidden variable is to perform k-means clustering analysis using the set $U$ of training user vectors, and to use the cluster as the hidden variable. The different clusters are then trained separately. If we let $k$ be the number of clusters, then the k-means clustering will return sets of users $U_0, U_1, \ldots, U_{k-1}$ such that a user only belongs to one of the sets ($U_i \cap U_j = \emptyset$ for any $U_i \neq U_j$). Each of the clusters can then be trained independently using any method of inference.

It is important to note that for two different clusters of users, the businesses the users have rated are not necessarily identical. That is, for $U_i, U_j$ where $U_i \neq U_j$, it is not necessarily the case that $B_i = B_j$. Furthermore, if there does exist a $b \in B_i$ and $b \in B_j$, then it is not necessarily the case that $f(U_i, b) = f(U_j, b)$. This means that the same business vectors can be trained with different expected output based on the clusters. It represents our belief that the users in different clusters have different preferences, so they may rate the same business differently.

Then, we can predict the rating given a user $u$ and a business $b$ by first finding its cluster, and then running the result of the inference method on the business vector.

1: **function** PREDICT($u, b$)
2: $i \leftarrow$ cluster($u$) ▷ A cluster number.
3: method $\leftarrow$ inference(cluster= $i$) ▷ The trained inference predictor for the cluster.
4: **return** method($b$)
5: **end function**

Note that the complexity of this grows with the number of clusters we choose. With k clusters, finding the cluster in line 2 and the appropriate cluster's trained inference method in line 3 take $O(k)$ time.

### 3.2 K-Neighbors

A different approach determines the hidden clusters dynamically at run time. In this approach, we do not hard-code any clusters but decide them based on the input variables when testing. We perform a nearest-neighbors analysis, and let that particular set of users be the cluster used to perform the inference. Let $k$ be an integer that represents the number of neighbors we will use. Then at testing time, we find the $k$ nearest neighors and peform an inference on these neighbors dynamically.

1: **function** PREDICT($u, b$)
2: $U_s \leftarrow kNeighbors(u)$, $Y \leftarrow []$
3: $B_i \leftarrow businesses(U_s)$ ▷ Businesses users in $U_s$ have rated.
4: **for** $b \in B_i$ **do**
5: $r = f(U_s, b)$ ▷ Returns average rating within $U_s$.
6: $Y \leftarrow Y + r$
7: **end for**
8: $inference = train(input = B_i, output = Y)$ ▷ Prediction method.
9: **return** $inference(b)$
10: **end function**

Here, $U_s$ is the set of k nearest neighbors to $u$. In step 8, we train to learn the ratings that only users in set $U_s$ have given. This enables us to find the most similar users to the queried user, $u$, in real-time to dynamically train a model fitted to the user. Similar to the k-means algorithm, every set $U_s$ of users may be trained to learn the business ratings differently. Even though this method requires almost no work during training, it becomes expensive at test time because it is necessary to perform inference.

Note that the total number of unique sets $U_s$ of size $k$ is $\binom{|U|}{k}$. In theory, we could complete an inference algorithm on each of these sets during testing time, but we found this to be infeasible with

a growing size of $U$ and $k$. Furthermore, dynamically performing the inference allows the model to be more flexible in that we can add more users to $U$ at any point, or change the parameter $k$. The downside is that each prediction takes a longer time to run because there is a complete inference problem that must be solved for every queried $u, b$ pair.

### 3.3 Mixture of Gaussians

Our third model treats clustering using a probabilistic approach. Assuming that the users in $U$ are a mixture of several gaussian distributions allows us to treat a single user, $u$, as probabilistically belonging to different clusters. If we choose $k$ as the number of mixtures (our hyperparameter), this means that asking to cluster a user $u$ will return a list of size $k$, where each element $i$ is in $[0, 1]$. This respresents the probability of user $u$ belonging to mixture $i$.

During training time we perfom $k$ inference algorithms, one for each mixture, similar to the k-means method. However, instead of completely partitioning the users into clusters, we use all users and weight them based on how likely they are to belong there. Let $p_i(u)$ be the probability that user $u$ belongs to cluster $i$. Our calculation of the rating then takes a different form.

$$g(i, b) = \frac{\sum_{u \in U} r_{ub} p_i(u)}{\sum_{u \in U} p_i(u)} \tag{2}$$

Here $i$ is the index assigned to a particular mixture. For each mixture, we consider all the users in $U$, but the ratings of these users are weighted according to how likely it is the user belongs to $i$. We can think of this as a more general form of Equation (1), which treats the weights as a continuous probability in $[0, 1]$ instead of a discrete value $\{0, 1\}$.

During testing time, we are given a user $u$ and can similarly calculate the probability that the user belongs to the different mixtures. We can use this to either choose the maximum liklihood estimate, or weight the different clusters to convey our uncertainty.

#### 3.3.1 Maximum Liklihood Estimate

In the case of the maximum liklihood estimate, this strategy works similarly to k-means clustering. Though the underlying gaussian mixture model is different, we perform the same analysis at testing time. Given a user $u$, we find the cluster, $i$, that has the maximum probability $p_i(u)$. We can then use a trained inference model for the $i^{th}$ cluster with a business, $b$, to compute our prediction of the rating.

#### 3.3.2 Probabilistic Weights

In this case, we must perform the trained inference for each mixture and output the weighted result of all mixtures, according to the probabilities of the user belong to that mixture.

```
1: function PREDICT(u, b)
2:     p ← mixture(u)                          ▷ List of probabilities.
3:     total ← 0
4:     for p_i ∈ p do                          ▷ Weight of mixture i.
5:         method ← inference(cluster = i)     ▷ Trained inference for mixture i.
```

6: $total \leftarrow total + p_i(method(b))$
7: **end for**
8: **return** *total*
9: **end function**

Similar to the k-means algorithm, this algorithm also grows linearly with the number of mixtures, $k$, because we enter the loop in line 4 exactly $k$ times. Note that the sum of all entries in $p$ must equal to 1, so it is not necessary to normalize the value stored in the *total* variable.

### 3.4 Inference Algorithms

In each of the above cases, we have mentioned that after discovering an individual cluster, or mixture, we are able to perform an inference algorithm. Any inference algorithm we use must be able to accept, as inputs, a list of business features, $B$, along with the expected output $Y$, which are the corresponding ratings. It must output a function that is able to predict a rating, $r$ given a business vector $b$. This part can be directly supervised because we are provided with all the ratings each user has given to a business.

We consider 5 different algorithms when performing the inference.

- MLE: The maximum liklihood estimate just returns the average rating from all the given ratings. This is used as a benchmark for testing how the other algorithms compare.
- Ridge: This is the ridge regression, as implemented in class.
- Lasso: This is the lasso regression, as implemented in class.
- Bayesian Ridge: This is a ridge-type regression that assumes all the input data is Bayesian. We were motivated to try this because we expected a user's ratings patterns to follow roughly a bell-shaped distribution.
- Random Forests: This constructs rules to use for classification. We were motivated to try this because many of our features were discrete valued, and the output star ratings all lied in the range $[1, 5]$.

For each of the three hidden-variable methods (K-Means, K-Neighbors, and Mixture of Gaussians), we test the inference algorithms above after we determine the appropriate cluster, or mixture of clusters.

## 4 Experiments

From the Yelp data set, we pulled out businesses that were categorized as 'Restaurants' to narrow the scope. After doing so, we had a total of 14,304 business data points, and 12,596 user data points. The rest of this section describes our strategy for training, validating, and testing this data on the methods described previously. We then present our experimental results from running the data, both in terms of accuracy and time efficiency.

### 4.1 Setup

Our overall setup implemented a recursive train-validation-test strategy. Note that our method implements the overall problem of predicting a rating given a user, business pair. It does so in two steps:

1. Using the user as the input, determine the hidden variable. This is either a single cluster, or a mixture of clusters.

2. Using the cluster and the business as inputs, predict the overall rating.

Step 2 can be directly supervised because we are given the actual average rating for a business. Thus, we can tune hyperparameters to the inference problem using the same train-validation-test strategy used in class by splitting the business data into 3 categories.

The problem is that step 1 cannot be supervised because it is a hidden variable, even though we need to tune its hyperparameters (the number of clusters, number of nearest neighbors, or number of mixtures). To solve this problem, we note that we can perform end-to-end testing on the entire system. Given a user and a business, we know what the overall rating should be. This means that we can split the user data into 3 categories as well, training the entire system, and validating the system end-to-end to determine the hyperpameters of the hidden variable. Figure 2 shows a visual schematic of our testing system.

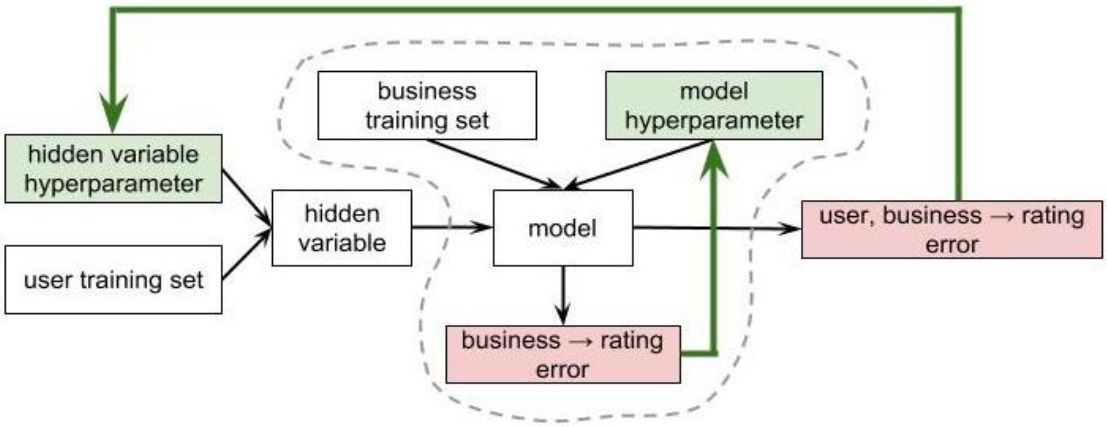


Figure 2: The overall testing strategy shown for a single model. The model is trained to take in a business and compute the expected rating. As shown by the dashed line, it is a sub-problem within the overall system. When tested end-to-end, it outputs the expected rating for a user, business pair. Red boxes represent the two types of error that we can measure. The green arrows represent which hyperparameters the validation data is affecting.

The strategy works so that we recursively train-validate-test the business data for every model within a single iteration of training the user data. When we perform validation within the model, represented by the dashed grey area in Figure 2, we are testing predictions on businesses that the model has not seen before, but for the same users. In contrast, when we perform end-to-end testing, we are making predictions on users and businesses that the overall system has not seen. We chose a scheme where we used 50% of the user data and business data to train, 25% of the data to validate, and 25% to test.

### 4.2 Results

We now present the results we achieved for each of the variables, as well as the overall end-to-end testing. While the hidden variable method was not supervised, we still present significant trends and comment on the perceived effiency. In the overall system, we present validation accuracies for different hyperparameters.

#### 4.2.1 Hidden Variable

The first step in our scheme is to cluster users based on the user feature vectors $U$. Included in this data is information about how often users have reviewed several categories of restaurants along with their rating for each category relative to their average rating. Additional features include the total number of reviews.

If we use this raw data in a K-Means clustering algorithm, we observe very poor results. This is due to some features having much greater values and deviation, therefore receiving more weight. In particular, the total number of businesses a user has reviewed can have a large value whereas the average rating of a category relative to the user average is real valued in the range (0,4). This leads to average category ratings having little impact on the clustering. To avoid this, each feature is scaled relative to the mean and standard deviation of that feature in the dataset [**?**]. For every feature $\phi_i^d$ corresponding to the $d^{\text{th}}$ feature for user $i$, the scaled feature value $\tilde{\phi_i^d}$ is calculated as

$$\tilde{\phi_i^d} = \frac{\phi_i^d - \mu^d}{\sigma^d} \tag{3}$$

where $\mu^d$ is the average value for feature $\phi^d$ in the dataset with standard deviation $\sigma^d$. This limits the effect a single feature can have on influencing the cluster boundaries.

After applying this correction we observe clusters indicated in Figure **??**. This shows that many features are influencing the choice of cluster boundaries, not just one. However, if we delve deeper into the results, we see that the dominant features are the frequency of which users attend restaurants of each category. In this first clustering attempt, the frequency features before scaling are simply the number of reviews a user has written for restaurants in each category. Notice that the clusters seem to be grouped by the number of users have written. This is not ideal since we would like to discriminate users based on preference, not the total number of reviews. The total number of reviews is a relevant feature and is captured in our feature set by the total number of reviews feature but we would prefer that it not dominate the cluster boundaries.

To solve this issue, we re-engineer the category review frequency features. Instead of the features simply being the user's total number of reviews for each category, these features are normalized by the number of user reviews. Therefore, the bias of choosing clusters based on the number of reviews is reduced. The resulting clusters are shown in Figure 3.

## 5 References

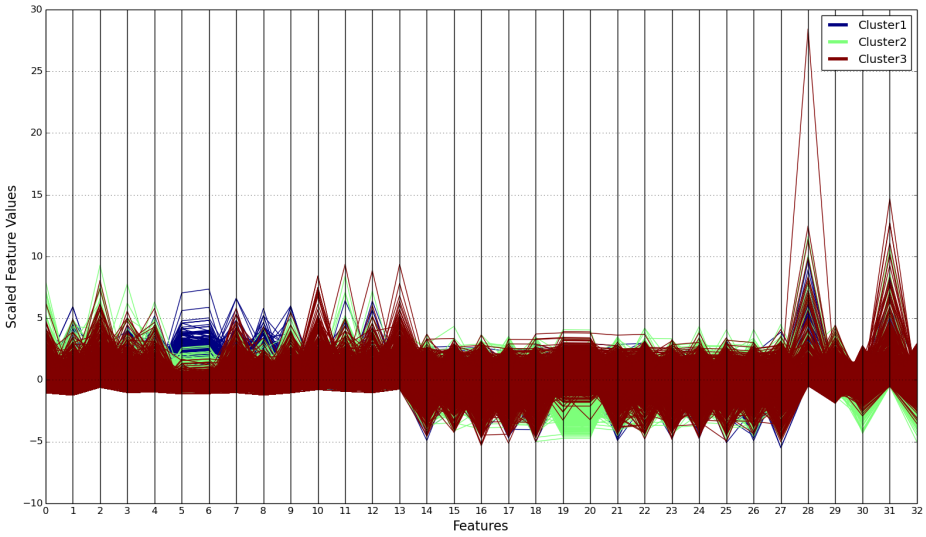
Cluster1
Cluster2
Cluster3
Scaled Feature Values
Features
30
25
20
15
10
5
0
−5
−10
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32

Figure 3: .