# Clustering to Forecast Sparse Time-Series Data

Abhay Jha, Shubhankar Ray, Brian Seaman
*Smart Forecasting, @WalmartLabs*
USA
{ajha,sray,brian}@walmartlabs.com

Inderjit S. Dhillon
*CS, University of Texas*
Austin, USA
inderjit@cs.utexas.edu

***Abstract*—Forecasting accurately is essential to successful inventory planning in retail. Unfortunately, there is not always enough historical data to forecast items individually– this is particularly true in e-commerce where there is a long tail of low selling items, and items are introduced and phased out quite frequently, unlike physical stores. In such scenarios, it is preferable to forecast items in well-designed groups of similar items, so that data for different items can be pooled together to fit a single model. In this paper, we first discuss the desiderata for such a grouping and how it differs from the traditional clustering problem. We then describe our approach which is a scalable local search heuristic that can naturally handle the constraints required in this setting, besides being capable of producing solutions competitive with well-known clustering algorithms. We also address the complementary problem of estimating similarity, particularly in the case of new items which have no past sales. Our solution is to regress the *sales profile* of items against their semantic features, so that given just the semantic features of a new item we can predict its relation to other items, in terms of as yet unobserved sales. Our experiments demonstrate both the scalability of our approach and implications for forecast accuracy.**

## I. Introduction

Forecasting is a key problem encountered in inventory planning [1]. In order to buy inventory in advance, retailers need an estimate of the number of units an SKU, or as we informally refer to: an item, is going to sell in the coming weeks. Buying fewer units leads to lost sales opportunity, hence lower revenue; buying too much leads to loss since the cost of buying inventory isn't compensated by income from sales. In general, the loss function of buying higher/lower is not symmetric, and besides forecasts, one also needs its variance. But for this paper, we will ignore such details, and just assume the goal is to forecast *accurately*[1].

Forecasting time-series data is a well-studied field, see [2] for a survey. There are two main components that contribute to an item's forecast: *trend* and *seasonality*. Trend refers to the velocity of sales in the past few weeks, i.e. the rate at which sales have been increasing/decreasing. An item on promotion, for example, may experience increasing sales and in those weeks that trend is an important factor for forecast. Seasonality refers to periodic events that influence the sales, for e.g. Thanksgiving, Back-to-School sales. These events produce a bump in sales for many items.

In brick and mortar stores, the assortment of items is fairly stable, hence there is enough past sales data to model each item separately. In e-commerce, however, the assortment is much more dynamic as there is smaller overhead to adding an item. Many items are short lived as well. In such cases, when an item has not been around for even a year, one cannot estimate its seasonality. Fitting a non-trivial model to such short time-series is also not recommended, since it is difficult to robustly estimate model parameters. Hence, it makes sense to forecast items in groups: that way even if each item has a short or sparse life cycle, the group has enough data to estimate features like seasonality. Also, modeling the group is more robust as any outliers or missing data in one item does not have as much influence over the final model. We illustrate this in Fig. 1: not every time-series has adequate data on its own, but they rise and fall in similar fashion, so one can estimate the influence of a seasonal event on an item even though it was not around last year.

This is not a novel insight, in fact modeling items in groups is the foundation of the entire field of analysis of panel/longitudinal data [3]. Panel data refers to repeated observations of multiple entities over a period of time. They are very commonly used in economics (for instance GDP of countries over time), finance, political science, and other social sciences literature. Panel data models can use correlation across the cross-section of entities in addition to the temporal correlation, which is the only thing univariate models exploit. Of course, we can't model all the items together, since there is too much variation in their behavior. So, we need to decompose the items into groups so that items in a group can be modeled together. There is a lot of work on multivariate models, viz. VAR [4], DLM [5] that can be used after a suitable grouping has been obtained.

Here, we will try to formulate the grouping problem independent of the forecasting model being used. Given a set of time-series, we want to assign them into some groups so that each item is assigned to at least one group and the size of each group obeys a lower and upper bound. The upper bound is necessary because of computational considerations, and a lower bound is necessary since with fewer items the model is more susceptible to outliers and missing data in individual time-series, which tend to be very noisy. Note that the groups do not need to be disjoint; assigning items to multiple groups does increase the computational load of modeling, and hence should be avoided as much as possible. Unlike traditional notions of clustering/segmenting, our goal here is not to identify groups of similar items so that the groups are dissimilar to each other; just to find overlapping groups of *not dissimilar items* respecting the size constraints. For e.g.,

[1]accuracy metric discussed in Sect. II-A

putting two very similar items in different groups would be considered bad by traditional clustering objectives, but in our setting that is OK. But putting two very dissimilar items in the same group is anathema. There is an additional constraint of a cluster having enough time-series data points, for instance having at least $k$ observations per week of the year, to make sure we have a robust estimate of seasonality and the model we fit is reliable. All these constraints make it non-trivial to extend most known clustering algorithms in this setting, and since violating these can lead to bad or no model fit, hence no forecasts, they need to be enforced rigidly. In this paper, we attempt to solve this problem along the two dimensions of estimating similarity between items, which is essential to successful grouping and the grouping algorithm itself.

*a) Estimating Similarity:* There is a rich body of literature to estimate similarity in case of sufficient sales data. But the items newly introduced present a challenge, since they have no past sales and the only way to forecast them is to predict other items that it is going to sell like. A common approach in these cases is to use the semantic data about items, which includes its name, description, brand, etc. Our approach seeks to generalize the successful idea of *Latent Semantic Indexing(LSI)* [6]. LSI summarizes the semantic feature matrix, $X$, by its *principal components*, i.e. the top directions in which this matrix has highest variance. Now suppose, in addition to this, we had the sales matrix $Y$ as well. We could then use the top directions maximizing the covariance between $X$ and $Y$. That way we are able to find the directions in which the semantic features have maximal variance *and* correlation with sales. To put this idea to work though, one needs to find these directions during training on the set of items with enough sales, and then use them to predict on the set of items with little sales. We use Partial Least Squares(PLS) regression which successfully implements this idea in a regression and predictive setting. We refer the reader to [7] for a good survey; our approach is discussed in detail in Sect. III. We demonstrate that the similarity predicted by this approach is indeed a much better predictor of sales correlation than using just semantic features.

*b) Clustering:* We propose a local-search based heuristic that starting from an initial clustering, moves items between clusters to minimize the cost function. One of the advantages of having small local moves is that its easier to maintain feasibility of the constraints. Moreover, as we show, one can actually find the optimal move in just linear time, which means each move is relatively quick, which allows us to perform many more iterations. Local-search methods like this have been used in clustering before as well, see [8], [9]. Our heuristic though similar in spirit differs in technical details on how to choose the moves. [9] demonstrated that local-search based heuristics provided a good alternative, especially in cases when k-means converges to local optima in a few iterations. We observe the same phenomena, with k-means making progress only for first few iterations, but unlike [9], who rely on a combination of k-means and local search, we rely completely on local search. This is based on the

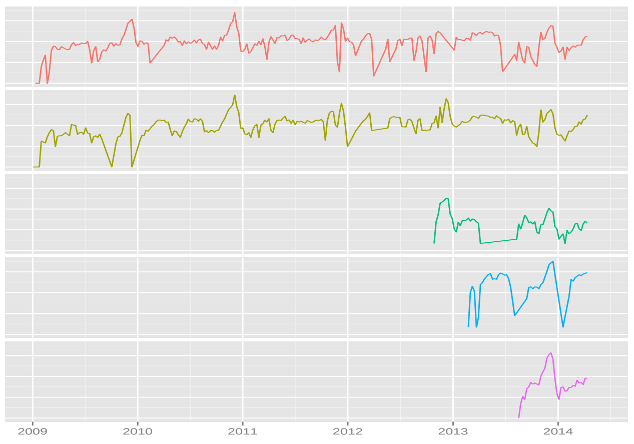

Fig. 1. Sales of similar items across time. Note how there are times of year when sales rise or fall for most items. There is of course some per-item variation. This illustrates the value in exploiting cross-sectional correlation across item time-series.

tradeoffs of extending k-means to deal with all the constraints required in our case, and benefit in performance obtained. The comparisons are discussed and empirically demonstrated in Sect. IV.

*Roadmap.* The paper is organized as follows. We describe our problem and overall system design along with some background in Sect. II. Sect. III describes our approach to estimating similarity along with empirical results from some experiments. Sect. IV contains the details of our clustering algorithm and experimental results and comparisons of it with other alternatives. Related work is summarized in Sect. V. We conclude in Sect. VI. We use terms like proximity, similarity, distance interchangeably since it is trivial to transform one to another. Same goes for terms like grouping and clustering.

## II. Grouping for Forecasting: Basics

In this section, we discuss some background to get the reader acquainted with the problem setting, and also get a sense of the challenges involved in solving them. Wherever deemed relevant, we have described a related methodology; for a more detailed related work though, we refer the reader to Sect. V.

### A. Problem Statement

We will denote our observed time-series by an $n \times m$ matrix $Y$: there are $n$ time-series $Y_i$ each of length $m$. Each data point $Y_{i,j}$ can be a real number or $NA$, which stands for *missing value*. Note that in general, the time-series are of unequal length, but we just pad each series with $NA$ to make them of equal length. We should mention that even in the absence of padding, our time-series have a lot of $NA$. Also, it is possible for some rows of $Y$ to be completely $NA$: that would mean it is a new item.

Besides the sales, we have an $n \times l$ semantic feature matrix $X$ which can be composed of features like n-grams from item descriptions, and indicator matrix of other attributes like brand, category, etc. Unlike $Y$, $X$ is a complete matrix with no missing values, but it does have a lot of zeroes: in that sense it is also a very sparse matrix.

*Forecasting Problem*: Given $Y$, and a natural number $h$, the forecasting problem is to predict $h$-step ahead values of $Y$. In other words we need to output $n \times h$ matrix $\hat{Y}$ so as to minimize the error w.r.t to the actual values we observe in future: $\bar{Y}$. The error metric we use is defined as follows:

$$\text{Error} = \frac{\sum_{i,j} |\hat{Y}_{i,j} - \bar{Y}_{i,j}|}{\sum_{i,j} \bar{Y}_{i,j}} \tag{1}$$

For a particular item $p$, the error is given by $\frac{\sum_j |\hat{Y}_{p,j} - \bar{Y}_{p,j}|}{\sum_j \bar{Y}_{p,j}}$. There are many other definitions in use, most of which aggregate the error per item. Taking mean of item errors is too susceptible to outliers, while median ignores too many errors. The above definition is preferred in our setting because while it includes the contribution of everything, it is robust to a few big errors. .

*Grouping Problem*: Given a distance matrix $D(n \times n)$ and a candidate $k$ for the number of groups, the goal is to decompose the items into groups $\mathbf{C}$ s.t. that every item is in at least one group and items in a group are not too *dissimilar* with *enough* data point between them as defined by a function $f$. Mathematically, we can write the problem as:

$$min_{\mathbf{C}} \sum_{C \in \mathbf{C}} \sum_{p,q \in C} D(p,q) + \lambda \left( \sum_{C \in \mathbf{C}} |C| - n \right) \; s.t. \tag{2}$$

$$\forall C \in \mathbf{C} \;\; L \leq |C| \leq U, \; f(C) \geq \delta, \text{ and} \tag{3}$$

$$\bigcup_{C \in \mathbf{C}} C = [n] \tag{4}$$

$L, U, \delta, \lambda$ are constants chosen based on experiments. $f$ can be a function which says we need at least some data for every part of the year, say week/month. This is to ensure a good seasonality estimate per group. The only restriction we place upon $f$ is that given $f(C)$ for a group $C$, $f(C \pm v)$ for any item $v$ can be computed in constant time. For our example $f$ this is true, assuming constant size of the time-series involved. Note that there may not always be a feasible solution to the above problem. But we will assume that the constants are chosen judiciously so that there is one, which is true in practice as well.

One could imagine replacing distance in the above objective with a similarity function and then maximize the objective. The two are very distinct! The above objective doesn't care if two very similar items are place in different groups– it just seeks to minimize the dissimilarity within a group so that a model that assumes items in a group are similar is not affected adversely. One can verify mathematically as well by replacing distance with similarity subtracted by a constant in the above objective function, we do not get an objective that maximizes similarity.

There are two things we have glossed over. First is defining distance metric $D$; this is discussed in Section II-B1. Another is picking the number of clusters $k$. There are two opposing forces that push the estimate of $k$. First note that, as we increase $k$ to $n$, within group dissimilarity is completely minimized. But there is going to be very little data per group to model. On the other hand, putting every item in one group, $k = 1$, means a lot of data for the group but high dissimilarity within it. The success of the model depends on both of these factors. One could define a function $g$ which takes any $\mathbf{C}$ and outputs a number proportional to the amount of useful data in its groups. For instance, a group where every item has data for only past six months has effectively fewer data than a group that has an assortment of items: some for past few years, some for past few months, even if the total number of non-missing data points in both are the same. If we treat each $X_i$ as a set of time-indices where it has non-missing value, then we could define

$$g(\mathbf{C}) = \log \left( \sum_{C \in \mathbf{C}} | \cup_{i \in C} X_i | \right)$$

This is one way of defining $g$, chosen based on our experiments, but depending on one's need, we envisage other definitions as well. If the value of objective from Eq. 2 is $O(\mathbf{C})$, we choose $k$ to minimize $O(\mathbf{C})/g(\mathbf{C})$.

## *B. Background*

Below, we provide some background on the two problems we focus on in this paper: estimation of similarity and clustering.

### *1) Computing Time-Series Similarity:*

*a) Sufficient Sales Data:* There is a large literature on proximity/distance estimation between time-series, see [10]. There are two important factors in choosing one for our purpose. First, it has to be robust to outliers. Outliers, both in forms of very low and very high sales are common in our setting, and a distance function like say Euclidean distance, that is easily influenced by an outlier is not a good choice. In fact, due to the size of outliers, even $L_1$-norm is easily influenced. The other consideration is dealing with missing values and unequal length: we need a function which does not get easily disrupted by missing values. For e.g., consider a sophisticated approach that computes various characteristics of the time-series data, which may involve fitting a model, as in [11]: due to the sparsity of the data, the model may not fit well, and even other parameters computed are likely to be unreliable. This limits the possible approaches considerably in our setting.

For modeling, we consider two time-series to be similar if they rise and fall in unison. The concept of *pearson correlation* captures whether two vectors are linearly related, and is defined as the cosine angle between the two vectors, after *centering* :

$$\texttt{Corr}(v, w) = \frac{(v - \bar{v}) \cdot (w - \bar{w})}{\| (v - \bar{v}) \| \| (w - \bar{w}) \|}$$

where $\bar{v}, \bar{w}$ denote the mean of $v, w$ respectively. Note that the above definition is invariant to shifts in $v, w$, i.e. $\texttt{Corr}(v, w) = \texttt{Corr}(v + c, w + c)$ for any constant $c$. This is accomplished by centering. The definition of cosine angle is invariant to vector magnitudes by itself, and adding centering makes it invariant to shifts as well– both are desirable characteristics for a similarity function in our case. In case of missing values, we only keep those time points where both vectors are non-missing.

One of the limitations of any similarity function when applied to a sparse time-series, as in our case, is that the absolute values may be very noisy, so its useful to have a mechanism that tests whether the similarity is significant or not. This gives us an alternative binary similarity function that just records two items as similar or not. This definition is much more immune to noise. The general idea to test significance is to fix a threshold, say 10%, then the value of correlation is significant if it is in the top 10%(positive correlation) or bottom 10%(negative correlation) of its distribution. For long enough vectors, one can approximate the distribution of pearson correlation asymptotically: $r\sqrt{\frac{s-2}{1-r^2}}$, where $r$ is the correlation and $s$ is the length of vectors, is distributed as Student's t-distribution with $s-2$ degrees of freedom. For smaller vectors, one can use *permutation test*: sample many permutations of $w$ and find their correlation with $v$ to approximate the distribution, and then test the significance of their correlation value using the provided threshold. For more, we refer the reader to standard statistics books like [12].

Pearson correlation however is not as robust to outliers. To alleviate that, we use *spearman* correlation: pearson correlation between the *ranked vectors*. For e.g. if $v = \{10, 35, 11, 20\}$, then ranked $v$ would be $\{1, 4, 2, 3\}$. An additional advantage of using ranked vectors is that it captures the intuitive definition of $v$ and $w$ rising and falling in unison better. Since this is still a pearson correlation, significance can be tested as discussed above.

*b) Insufficient Sales Data:* In case of insufficient sales data, it is customary to work with the semantic feature matrix $X$. Generally $X$ is constructed using Information Retrieval(IR) techniques and hence to get similarity, the popular IR methods of *cosine similarity* and LSI are used ( [13] is a good reference on IR). Cosine similarity, as the name suggests, treats items as vectors based on matrix $X$ and the similarity is computed as cosine angle between the vectors. However, $X$, almost always, is high dimensional and sparse. So, it is customary to work with a low-rank decomposition of $X$ along the top few singular vectors. These vectors can thought of as latent directions which are most informative, hence this is referred to as Latent Semantic Indexing(LSI). For a good explanation of its efficacy, see [14].

LSI is a special case of dimension reduction techniques based on PCA wherein one embeds a feature space in a low-dimensional space so that the distance between the objects in low-dimensional space approximates the distance in original feature space well. The rationale for this is not just computational; often this embedding removes a lot of noise and the distances in this space turn out to be more informative and useful. Similar to LSI for $X$, one could also construct an embedding $Y_e$ for sales data $Y$ for the items with sufficient sales data. For the other items, we seek to find the coordinates in this space based on $X$. This can be formulated in a regression framework : $Y_e \sim X$. The two challenges in solving this regression is that $Y_e$ is multi-dimensional and $X$ is sparse and possibly collinear. So, Ordinary Least Squares(OLS), the simple linear regression model fails because of singularity of $X^TX$. This can be circumvented by regularized least squares, like *ridge*, *lasso*, etc., but these were not designed with multidimensional response in mind, and hence fail to take the structure of $Y_e$ into account. Also, high sparsity of $X$ doesn't help. Building on the success of LSI, one would think about using an approach like Principal Component Regression(PCR), where the regression is using the top principal components of $X$. A limitation of this approach is that we are using the directions that explain the variance in $X$ successfully, but don't take into account $Y_e$. Our approach is to use *Partial Least Squares*(PLS), which is akin to PCR, but instead of using principal components of $X$, it uses directions that explain the covariance of $X, Y_e$ well. This is discussed in detail in Sect. III.

**Algorithm 1:** K-MEANS WITH DISTANCE MATRIX

```
Input: Distance Matrix $D(n \times n)$, initial clusters
       $C_i^0, 1 \le i \le k$, threshold $\delta$
Output: Clusters $C_i, 1 \le i \le k$
1 $C \leftarrow C^0$, $\rho \leftarrow \infty$, $\rho_0 \leftarrow \infty$
2 while $|\rho - \rho_0| \le \delta\rho_0$ do
      // Find the medoids
3     for $i \leftarrow 1$ to $k$ do
4         $M_i \leftarrow \mathbf{argmin}_j \sum_{p \in C_i} D[j,p]$
      // Update Clusters
5     for $i \leftarrow 1$ to $k$ do
6         $C_i \leftarrow \{j \mid \mathbf{argmin}_l D(j, M_l) == i\}$
      // Quality of New Clusters
7     $\rho_0 \leftarrow \rho$
8     $\rho \leftarrow \sum_{i=1}^{k} \sum_{j \in C_i} D[M_i, j]$
9 return $C$
```

*2) Clustering:* Clustering is a well-studied field and it wouldn't be possible to do justice to all the proposed methods in a limited space. We will instead describe three approaches, we feel are relevant to our problem, see Sect. V for more. It is also helpful to know these algorithms to get the motivation for our final algorithm. At this point, its helpful to remind ourselves of the clustering objective from Eq. 2. A simplified version of this objective, if we assume there are no overlaps would be: $\sum_{C \in \mathbf{C}} \sum_{p,q \in C} D(p,q)$, the sum of within group dissimilarities.

The most popular clustering algorithm by far is K-Means. It can be described in different ways: since our input is a distance matrix we have described it with an input distance matrix in Algorithm 1. In this setting, K-Means is often referred to as K-Medoids, but we refer it to as K-Means because as we will shortly show they are essentially the same approach manifested in different settings. K-Means, like most clustering algorithms, takes as input the number of clusters required and an initial clustering. The algorithm is iterative and in each iteration, we perform two steps. First, we find a medoid for each cluster– which is the point closest to all the points in the cluster, and then we create new clusters around each medoid, each point now assigned to its closest medoid. The iterations can be stopped when the clusters don't change or after a pre-specified limit. Each iteration has a time complexity of $O(nk)$. K-Means is known to be susceptible to local minima, but generally with multiple trials, one can find a good quality clustering. In our setting, we have found k-means to get stuck at a local minima very quickly, and multiple trials don't help. Another thing to note is that the cost function being minimized by K-Means is different: $\sum_{C \in \mathbf{C}} \sum_{p \in C} D(p, m_C)$, where $m_C$ is the medoid of cluster $C$. This is only useful when the clusters are *spherical*, i.e. there is a central point, a centroid/medoid around which most points lie. This is circumvented in practice by using *spectral clustering* [15]: projecting the points into an Euclidean space more amenable to clustering. Spectral transformation, however, is not scalable enough to used in our setting; another way to achieve the same without the overhead is to use *kernel k-means* [16], which takes $O(n^2)$ time per iteration. We will show comparisons with kernel k-means in Sect. IV. At this point, we would like to point out that if our input was in say a Euclidean space, then the point closest to a group of points would be their *centroid*. This is the version commonly reported as K-Means, but note that it is just a special case of the above: when we are only given the distance matrix, we find the medoid as the center of the cluster, while in general in a euclidean space one can compute the optimal center– the centroid.

Partitioning around Medoids(PAM) [17] proposed by Kaufman & Rousseeuw, is a simple gradient descent procedure to directly minimize the within-group dissimilarity, our metric of choice, see Algorithm 2. In PAM, a clustering is represented by medoids: given a set of mediods, each point is assigned to the closest medoid cluster. Given initial mediods, in each iteration, PAM tries to replace a medoid with a non-medoid, to maximize the decrease in cost function. The time complexity of each iteration is $O(nk(n-k))$. PAM is known to be very reliable and robust due to the nature of its cost function, however it is very slow when compared with k-means. Our goal is to simulate this behavior without the expensive time complexity.

Kernighan and Lin proposed a local-search heuristic for graph partitioning in [8]. It is described in Algorithm 3 for the simple case of partitioning a graph on $2n$ vertices into two parts of $n$ vertices each. The cost function being minimized is once again within-group dissimilarity. In each iteration, one vertex from each group is selected and then swapped: this pair is chosen to minimize the decrease in cost function. Note that after $n$ iterations the change is zero, as we get the same partitioning. Hence, one chooses some prefix of changes between 1 to $n$ that leads to the best net positive decrease in the cost function. As the authors explain in their original paper, one of the main advantages of this approach is that its fast: each iteration takes $O(n)$ time and its easy to maintain constraints on the sizes of clusters with this approach.

**Algorithm 2:** PARTITIONING AROUND MEDOIDS(PAM)

```
Input: Distance Matrix $D(n \times n)$, initial clusters
       $C_i^0, 1 \le i \le k$, threshold $\delta$
Output: Clusters $C_i, 1 \le i \le k$
1  $C \leftarrow C^0$, $\rho \leftarrow \infty$, $\rho_0 \leftarrow \infty$
2  while $|\rho - \rho_0| \le \delta\rho_0$ do
3      $\rho_0 \leftarrow \rho$
       // Find the medoids
4      for $i \leftarrow 1$ to $k$ do
5          $M_i \leftarrow \mathbf{argmin}_j \sum_{p \in C_i} D[j, p]$
       // Swap mediods with non-medoids
6      for $i \leftarrow 1$ to $k$, $j \leftarrow 1$ to $n$ do
7          $M' \leftarrow M$, $M'[i] \leftarrow j$
8          for $l \leftarrow 1$ to $k$ do
9              $C'_l \leftarrow \{m \mid \mathbf{argmin}_p D(m, M_p) == i\}$
           // Test new cluster quality
10         $\rho_{curr} \leftarrow \sum_{p=1}^{k} \sum_{q,r \in C'_p} D(q, r)$
11         if $\rho_{curr} < \rho$ then
12             $\rho \leftarrow \rho_{curr}$, $C \leftarrow C'$
13 return $C$
```

In our setting, we have observed bad convergence properties for k-means, while found PAM to be expensive. Kernel K-Means, or spectral clustering based on normalized cut lead to very big or very small clusters. Also, we find that much of the change happens in the first few steps, while after that there is very little change per iteration, even though we still spend $O(n^2)$ time per step. Based on these observations, our motivation is to have an algorithm which makes small incremental changes per iteration, but each iteration be very fast, like the local-search heuristic described above. Such an approach can naturally control for the constraints required on clustering as well, without the need for a post processing step. This is discussed in Sect. IV.

## III. Learning token weights to predict sales correlation

In this section, we describe our approach to predict sales correlation for items with little or no sales. We assume that we have a semantic feature matrix $X$: rows corresponds to items and features are on the columns. Generally it is constructed using IR techniques like *n-grams* based on some semantic information about items like brand, description, etc. $Y$ is the matrix of item sales, where once again items are on the rows. $Y$ will have fewer rows than $X$, since we will assume it only has items with enough sales history. A major component of our approach is the concept of *Principal Components*: the popular idea of LSI is also based on the same applied to $X$. Our goal though is to do it in a way that predicts

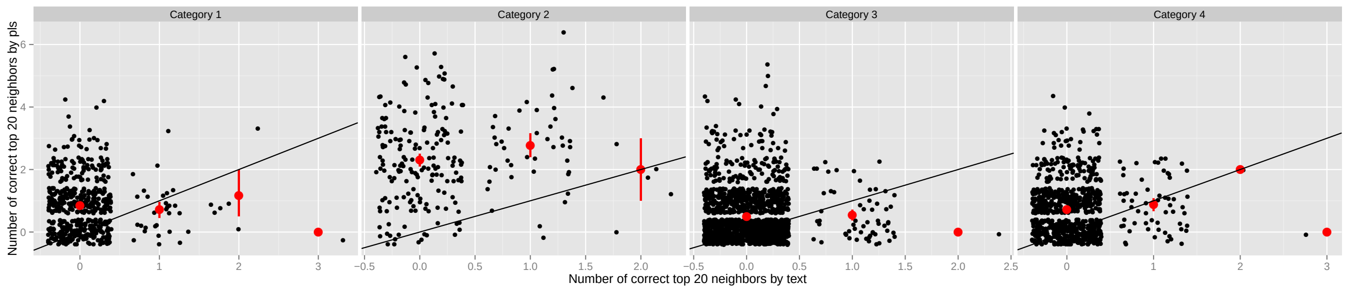

Fig. 2. Intersection of top 20 neighbors predicted by text/pls with the actual top 20 neighbors based on sales. Just using the semantic features almost always gives none of the top 20 neighbors. Using a supervised approach, like we outlined, to train these features lifts the number of correct neighbors for a lot of items.

**Algorithm 3:** KERNIGHAN-LIN HEURISTIC(KL)

```
Input: Distance Matrix $D(2n \times 2n)$, initial clusters
       $A^0, B^0$ each of size $n$, threshold $\delta$
Output: Clusters $A, B$
1  function $D(i,S) \leftarrow \sum_{j \in S} D[i,j]$
2  function $f(p,S_1,S_2) \leftarrow D(p,S_2) - D(p,S_1)$
3  $g \leftarrow 1 + \delta$
4  while $g > \delta$ do
5      $U \leftarrow [2n]$
6      for $it \leftarrow 1$ to $n$ do
7          function $\Delta(p,q) \leftarrow$
           $f(p, A^{it-1}, B^{it-1}) + f(q, B^{it-1}, A^{it-1}) - 2D[p,q]$
8          $(m,n) \leftarrow \mathbf{argmax}_{p,q \in U} \Delta(p,q)$
9          $g^{it} \leftarrow \Delta(m,n)$
10         $A^{it} \leftarrow A^{it-1} - \{n\} + \{m\}$
11         $B^{it} \leftarrow B^{it-1} - \{m\} + \{n\}$
12         $U \leftarrow U - \{m,n\}$
13     $p \leftarrow \mathbf{argmax}_j \sum_i^j g_i$
14     $A \leftarrow A^p, B \leftarrow B^p, g \leftarrow \sum_i^p g_i$
15 return $A$,$B$
```

correlations amongst sales matrix $Y$. As a side-effect, one also gets a weighting of the tokens, the columns of $X$, which are better correlated with sales than traditional methods like Term-Frequency(TF)/Inverse Document Frequency(IDF).

## A. Formulation

Before formulating our approach, we review the concept of PCA using sales matrix $Y$. To perform PCA, we center the rows of input matrix $Y$, so that each row sums to zero. The reason for this will become clear shortly, but in brief this enables us to compute covariance matrix of sales with a simple matrix multiplication. Given $r$, we want to decompose $Y$ as

$$Y = \sum_{i=1}^{r} t_i p_i^T + E \tag{5}$$

$$= TP^T + E \tag{6}$$

where $P$ is orthonormal, i.e. $\forall i,j \; p_i^T p_j = 1$ if $i = j$ else $0$. One way to interpret the above is as a sum of $r$ rank 1 matrices and a residual $E$; clearly $r$ can't be more than the rank of $Y$. $T$ is commonly referred to as the scores and $P$ the loadings. The decomposition is actually unique given $r$: $P$ are the top $r$ unit eigenvectors of $Y^TY$ and $T$ are the *projections* of $Y$ along those directions, i.e $T = YP$. In other words, $P$ gives us the $r$-dimensional space, while $T$ gives the projections of $Y$ in that space. In general, one is only interested in the scores $T$. Formally, they can be computed iteratively as follows :

$$t_m = \mathbf{max}_v Var\,(Yv) \tag{7}$$

$$\texttt{s.t.} \|v\| = 1, \; \texttt{Cov}\,(t_m, t_i) = 0, \; 1 \leq i \leq m-1 \tag{8}$$

$t_1$ is the variance along the vector of maximum sales covariance. And $t_i$ gives us the maximum possible variance along any vector, while being uncorrelated with $t_1..t_{i-1}$. Together, $P$ gives us a space that explains as much variance as possible in $r$ dimensions.

Our goal is to use the co-ordinates of $Y$ in this space: $T$, as a low-dimensional embedding. The property of this embedding that is most relevant for us is the following: For any rank $r$ matrix $\Lambda$: $\|XX^T - TT^T\| \leq \|XX^T - \Lambda\Lambda^T\|$. Note that $XX^T$ is the covariance matrix of the item sales, hence $T$ gives us an embedding in dimension $r$ that approximates this as closely as possible.

With this insight, we formulate our problem as follows. Divide $X$ into $X_{train}, X_{predict}$, where $X_{train}$ is the items for which we have sales in $Y$ and $X_{predict}$ are the rest. Fix $r$, and find the principal component scores for $Y$: $T$. Since distance in this space approximates sales well, it is sufficient if we can predict for the items with no sales, their coordinates in this space. Hence our problem can be formulated as a regression problem of $T$ against $X_{train}$. $r$ can be picked using cross-validation. We discuss how to perform that regression below.

## B. Solving the regression

We will focus on linear regression, and assume a relationship of the form $Y = XB + F$. In most common scenarios, $Y$ the response is a vector. But what distinguishes our problem is the fact that it is multidimensional, and the number of dimensions is not necessarily small either. The *Ordinary Least*

*Square(OLS)* estimate for $B$ is $\hat{B} = \left(X^T X\right)^{-1} X^T Y$. Note that in our case $X$, the semantic matrix, is high-dimensional and sparse. Hence OLS generally fails because of the presence of collinear columns in $X$, which render $X^T X$ non-invertible.

There are many regularization techniques to deal with such problems. The two common ones that we investigated are *lasso* and *ridge*. In these two, the linear regression is solved subject to an upper bound on the $L_1$ and $L_2$ norm of $B$ respectively. Lasso gives a very sparse estimate of $B$, hence a lot of the tokens may have no influence at all. Ridge, on the other hand, shrinks the coefficients, so that the correlated tokens will have similar weights. Since $X$ in our case is very sparse semantic matrix, which in practice, is generally summarized by its principal components, it also makes sense to try *Principal Component Regression(PCR)*, which is the regression of $Y$ against principal components of $X$. In general, we didn't find much difference in their performance, Sect. III-C has detailed comparison. However, all of these approaches were designed with a univariate $Y$ in mind, and hence there is a lost opportunity here in terms of exploiting the structure of $Y$. The approach that we finally use for regression because of its reliable performance, especially as dimension of $Y$ increases, is *Partial Least Squares(PLS)*.

PLS is similar to PCR, except that the principal components are chosen not just to maximize the variance of $X$, but also the correlation with $Y$. For the sake of exposition, we will assume that $Y$=$y$ is univariate: the univariate case can illustrate the concept just as well, without going into details. The algorithm can be described iteratively in a similar manner to PCA from Eq. 6. We decompose $X$ similarly as

$$X = TP^T + E$$

However the components $T$ are now chosen to maximize both the variance of $X$ and its correlation with $y$. Hence the analogue of Eq. 7,8 would be

$$t_m = \mathbf{max}_v Corr^2\left(y, Xv\right) Var\left(Xv\right)$$
$$\texttt{s.t.} \|v\| = 1,\ Cov\left(t_m, t_i\right) = 0,\ 1 \leq i \leq m-1$$

Finally, we do the same thing as PCR, regress $y$ on $T$ to get $y = TC + F$. Note that since $T$ is in the span of $X$, one can express $T = XW$, which gives $y = XWC + F$: this lets us express the result in terms of $X$. The weight of different columns in $X$ which correspond to tokens can be useful for other interpretation purposes as well.

One might wonder, why instead of predicting the principal components, we didn't just predict the sales matrix, since our response is already multi-dimensional. The main hindrance to that is the inordinate number of $NA$ in the sale matrix. Even though the NIPALS algorithm [18], used for solving PLS, can work with $NA$, its performance would severely degrade in the presence of so many $NA$; see [19] for more on PCA on missing data. Thats why we instead use *Probabilistic PCA(PPCA)* [20] to summarize the sales matrix before employing PLS. PPCA is much more robust to NA.

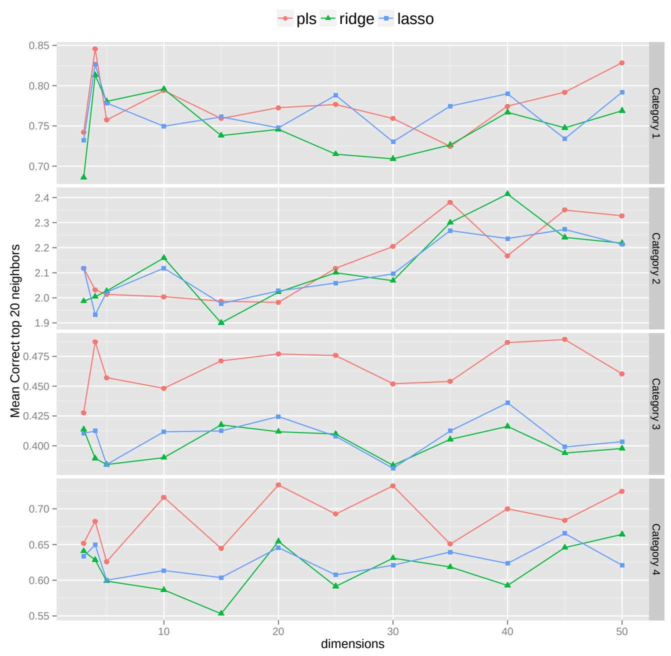

Fig. 3. Comparison of regression methods: we compute the top 20 neighbors predicted by each method and evaluate the number of them which are actually among the top 20. At higher dimensions, in particular the highest: 50, we find PLS always dominates. This is because PLS can exploit the structure of a multi-dimensional response better. From these figure it is clear that though there is no clear winner, overall one would prefer PLS

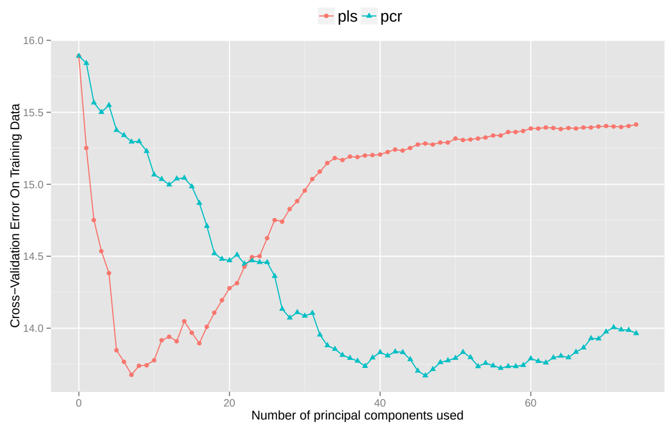

Fig. 4. Cross-Validation Error on Training Data versus number of components used by PCR vs PLS. PCR almost always needs far more components than that needed by PLS. For both approaches, coefficients over the components on training data need to be transformed to those over feature matrix for use on test data. Because of the high number of components required, which get increasingly inaccurate, PCR is not as reliable on test data.

## C. *Empirical Comparisons*

In this section, we report on the results from empirical evaluation of this approach over the sales data from Walmart.com transactions. We picked 4 representative categories of items on which we report the results. Our implementation is done in R, and makes use of the following R packages– **pls** [21] for

PLS and PCR, **pcaMethods** [19] for PPCA and **glmnet** [22] for ridge, lasso.

**Overall Improvement:** Our goal in this section is to predict sales distance/correlation between items. But for evaluation, we don't want to compare directly the distance predicted by an algorithm to the actual, since this would be too ambitious a problem, and the results would be hard to rely on as the correlation computed is itself not very accurate, and changes with addition of one time-point of data. A more robust evaluation measure would be to compare the top neighbors of an item. There are many ways of comparing rankings proposed in the literature, but we opt for the following simple measure: size of intersection of top 20 neighbors predicted by an algorithm to the top 20 neighbors based on sales. The main benefit is that the results are easily interpretable. We have also made sure to run our experiments on only the subset of the items which are consistent-sellers and hence have long sales history which ensures that the correlation/distance computed are reliable.

The overall improvement our method provides is illustrated in Fig. 2. Note that the points in the plot have been *jittered*, since our metric is discrete: this way instead of seeing just one point at say (0,0), we see a cloud of points around it that gives a sense of number of points at that location. For a category, we have evaluated both PLS and text-based neighbors based on a 10-fold cross-validation. As one would expect, generally getting neighbors purely based on semantic data gives us none of the top 20 neighbors. Our method, on the other hand, sometimes provides as many as 6 of the top 20! The average improvement along with standard error is also demonstrated in the plot.

**Regression:** Recall that we construct an embedding of items in some dimension $r$, before regressing it on semantic data. Fig. 3 illustrates the performance of various regression solvers vs $r$. First, observe that for the maximum dimension 50, pls performs best for every category. For Categories 1,2, even though in small dimensions PLS is not very different from ridge, lasso, but as the dimensions increase it starts to dominate. This is to demonstrate that PLS is reliable when higher dimensional embedding is chosen, while ridge, lasso may not be. For two categories, 3 and 4, PLS outperforms ridge, lasso consistently. These two also happen to have the largest number of items, as can be seen from the point clouds from Fig. 2 and worst results. This is not a coincidence; in bigger categories, we often have many items with similar names, or common brand, but not all of them are correlated. While this experiment was done on a small sample of consistent sellers, in general this problem is exacerbated when all items are included. PLS does work better in this case; even though the average improvement may not be much a lot of items go from no good neighbors to around 2-3 which can make a big difference in the clustering results.
We didn't compare against PCR, because strictly speaking PLS is a generalization of PCR. Often on training data PCR can perform just as well as PLS, except that it would need many more dimensions, as illustrated in Fig. 4. Note though that in both PCR and PLS, regression coefficients on the components are converted to coefficients over the semantic feature data. Because PCR uses so many components, and latter components can get very inaccurate, its performance on test data is not as consistent as PLS. Another useful thing that this plot demonstrates is that picking the optimal number of components for PLS is easier because its error curve forms a sharp valley unlike PCR.

**Algorithm 4:** OPTMOVE CLUSTERING ALGORITHM

```
   Input: Distance Matrix $D(n \times n)$, initial clusters
          $C^0[i], 1 \leq i \leq k$, params: $gap, L, U, \delta$,
   Output: Clusters $C[i], 1 \leq i \leq k$
1  function $D(i,C) \leftarrow \sum_{j \in C} D[i,j]$
   // Initialize data structures
2  $C = C^0$
3  item-cluster distance matrix $M\ (n \times k)$:
   $M[i,j] \leftarrow D(i, C[j])$
4  $H$ : $n$ size array of fibonacci min-heaps
   $H[i] \leftarrow \{M[i,j], 1 \leq j \leq k\}$
5  Cluster Membership: $Cm[i] \leftarrow \{j \mid i \in C[j]\}$
6  Cluster Quality : $F[i] \leftarrow f(C[i]),\ 1 \leq i \leq k$
7  Vertices visited : $visited[i] \leftarrow 0, 1 \leq i \leq n$
8  while Convergence do
9      for $i \leftarrow 1$ to $n$ do
10         pick $c_{old}$ randomly from $Cm[i]$
11         $c_{new} \leftarrow H[i].top()$
12         if $|c_{old}| > L \wedge f(c_{old} - \{i\}) \geq \delta \wedge |c_{new}| < U$
           then
13             $wt[i] \leftarrow M[i, c_{old}] - M[i, c_{new}]$
14         else
15             $wt[i] \leftarrow 0$
16     Pick $u$ probabilistically according to $wt$
17     Let $c_{old}, c_{new}$ define move for $u$ as above
18     if $|C[c_{new}]| == U \ || \ visited[u] > 0$ then
19         continue // next iteration
20     decrement all positive $visited$ by 1
21     $visited(u) \leftarrow gap$
22     Update $M, H, C, F, Cm$ : move $i$ from $c_{old}$ to $c_{new}$
23 return C
```

## IV. OPTMOVE CLUSTERING ALGORITHM

In this section, we present our clustering algorithm, and empirical results for comparison with existing approaches.

### A. The Algorithm

We will reiterate the grouping problem formulated in Sect. II-A briefly here. Given a distance matrix $D(n \times n)$, number of groups desired: $k$, and some user-defined params $L, U, \delta, \lambda$, we want to solve the following optimization problem:

$$min_{\mathbf{C}} \sum_{C \in \mathbf{C}} \sum_{p,q \in C} D(p,q) + \lambda \left( \sum_{C \in \mathbf{C}} |C| - n \right) \textit{ s.t.}$$
$$\forall C \in \mathbf{C} \;\; L \le |C| \le U, \;\; f(C) \ge \delta, \text{ and}$$
$$\bigcup_{C \in \mathbf{C}} C = [n]$$

We assume the the params have been chosen in a way that the problem is feasible. $f$ is a function to measure the *quality* of a cluster: we assume $f$ is monotonic, i.e $f(C + \{v\}) \ge f(C)$ and given $f(C)$, one can compute $f(C \pm \{v\})$ in constant time. The former is just an intuitive condition for simplification, and isn't necessary, while the latter property is used in the complexity analysis of our algorithm. Algorithm 4 summarizes our approach, which we call *OPTMOVE*. We first describe the algorithm below, then analyze its complexity, and finally prove its soundness.

**OPTMOVE**, like the Kernighan-Lin(KL) heuristic, seeks to modify the clusters, with small local changes. Unlike the former though, OPTMOVE doesn't swap the vertices, but instead *moves* them. Also, it is not necessarily the optimal vertex move that is picked, but instead a move is picked with a probability proportional to the improvement it would make to the objective function: moves with no change to objective aren't picked since their probability is 0. One of the issues faced by any local heuristic, like OPTMOVE, is that a few vertices, which may be outliers, may always be moved around since they have a higher influence on the objective function. This hinders a better exploration of the moves possible. KL handles this by not moving a vertex once moved. We achieve this in two ways: first, we make a randomized pick which makes sure any vertex that can make an improvement by being moved, can be moved. Besides this, we also stop a vertex once picked from being picked for the next *gap* iterations. We will see the effect of *gap* more in the empirical evaluation section, Sect. IV-B. We also experimented with choosing a random vertex at each step with a small probability, but it had no significant effect either on the time to terminate or the quality of the minima reached.

A good initializing of the clusters is important in our case, since it leads to convergence in fewer steps, hence smaller execution time. Also, initialization has to bear the burden of producing a feasible solution, which involves allowing overlapping clusters. Our strategy is borrowed from the popular *farthest-first* heuristic: pick the cluster mediods to be as far apart from each other as possible. See [23], [24] for instances where this leads to provable guarantees on cluster quality. Our initialization is done as follows. First, we initialize mediods according to farthest-first. Then we assign the closest vertex medoid pair till clusters are of size less than $L$. Once all clusters have reached size $L$, we need to enforce cluster quality : $f(C) \ge \delta$. In this phase, we take clusters where cluster quality is violated and add closest cluster vertex pairs till all clusters satisfy the quality constraint or the cluster vertex distance exceed $\lambda$. At this stage, vertices are too far from these clusters and overlap is preferred, so we assign closest vertices to these clusters, even if they may be present in other clusters until quality constraint is satisfied. Finally, we just assign the unassigned vertices in the closest first fashion.

**Theorem IV.1** (Complexity)**.** *Running OPTMOVE for $M$ iterations takes time $O(nk + nM)$. In particular each iteration takes $O(n)$ time.*

*Proof:* Initializing the data structures takes time $O(nk)$. It suffices to show that each iteration of *While* loop runs in $O(n)$ time. The *For* loop from 1 to $n$ consists of simple manipulations except for accessing the min. element of the heap which is in $O(1)$ time amortized; hence overall the loop takes time $O(n)$. Sampling from $wt$ involves cumulatively summing the elements and then picking a uniform random number in the range. This can be accomplished in $O(n)$ time as well. Finally, since only two clusters are affected by a move, of the $nk$ points in the item cluster distance matrix, only $2n$ are affected. Hence updating $M, H$ also takes time $O(n)$. ■

**Theorem IV.2** (Soundness)**.** *Given a feasible initial solution, OPTMOVE returns feasible solution. Also, the objective value from Eq. 2 never increases in any iteration.*

*Proof:* A feasible solution never becomes infeasible because of the explicit check which ensures that such a move has $wt = 0$, hence it can't be picked. A move can't have negative weight, since it is always moved to the optimal cluster, which in the worst case would be itself. So, no move can increase the objective. ■

**Discussion** One might have noticed that we did not use $\lambda$ during the optimization, but only initialization. This is because overlap in our case is only introduced for feasibility, generally $\lambda$ is high enough that there is not a lot of gains to be made considering the space of overlapping clusters. Also, note that while the space of partitions is of the order $k^n/k!$, an asymptotic approximation for sterling numbers of second kind, overlapping clusters occupy a space of size $(2^n - 1)^k$. An efficient way to explore the former space is preferable especially when the gains possible from the latter space are not believed to be high. Technically, our algorithm can be extended with moves of 2 vertices at a time, which would allow us to explore the space of overlapping clusters, but we believe this increased complexity does not offer any benefit, if the initial clusters are good enough.

## *B. Empirical Results*

Our goal in this section is to evaluate OPTMOVE and compare it against existing graph clustering algorithms. We will evaluate its convergence properties and its performance w.r.t a multivariate forecasting model. All the clustering algorithms compared in this section were implemented in C++ by us.

By far the most popular clustering algorithm is k-means, but in our setting it converges after very few iterations as illustrated in Fig. 6. It actually seems to get worse than the initialization, but thats because k-means optimizes a different metric than the within-group dissimilarity metric we use. The

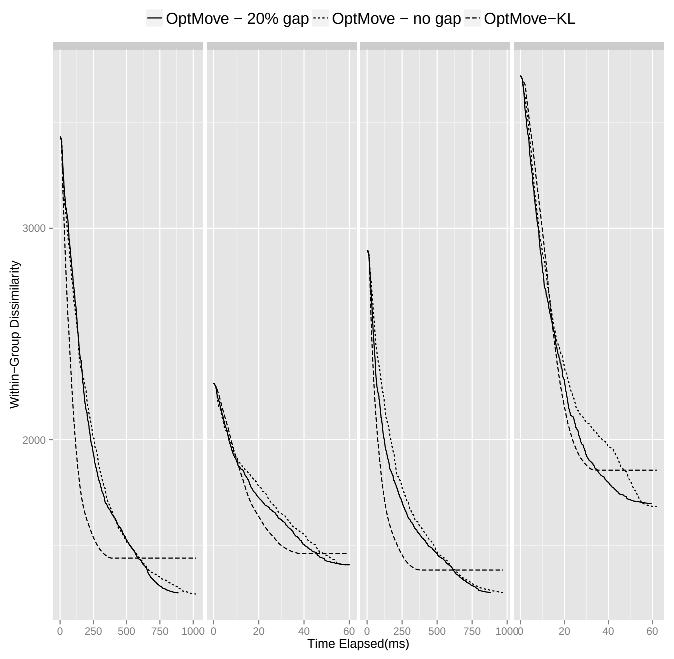

Fig. 5. Convergence Comparison of OPTMOVE with alternatives. OPTMOVE-KL uses moves like OPTMOVE but composes them deterministically like KL heuristic, i.e., the optimal every time. This is why its slope is steeper, but because of a lack in variety of moves, it settles at a minima before OPTMOVE, which picks moves probabilistically. gap, which is the amount of time a vertex once moved has to wait before moved again, has clearly some advantage but not very pronounced in our setting because of the probabilistic choice of moves.

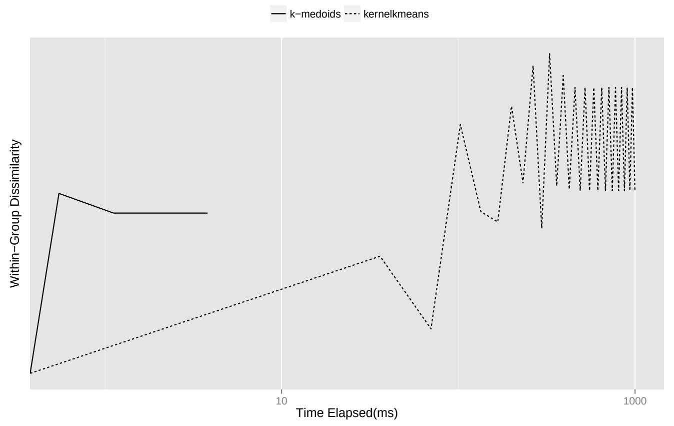

Fig. 6. Convergence properties of k-means/k-medoids and kernel k-means according to our dissimilarity metric. Both of them only make progress for a few moves before converging. Moreover, kernel k-means leads to some very big clusters which is punished by our metric.

direction it goes is not as significant as the fact that it only took 3 steps before converging. Spectral Clustering is very popular in community detection and image segmentation, but because they involved computation of eigenvectors, we haven't found them to scale well for larger assortment of items. A scalable alternative to spectral clustering is kernel k-means. Fig. 6 also illustrates kernel k-means where the kernel is chosen to

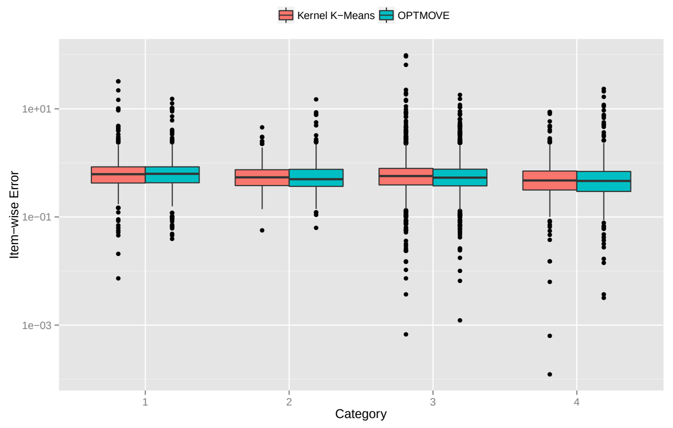

Fig. 7. Item Wise Model Error( Eq. 1) vs clustering scheme used. Note the fewer number of big outlier items in case of OTPMOVE, which is one of the benefits of choosing the right cluster for an item.

mimic spectral clustering with normalized laplacian: it tries to maximize the ratio of similarity in clusters to dissimilarity across them. It does worse than k-means, for the simple reason that in our setting, it has the tendency to form very large or very small clusters and note that our metric discourages big clusters as it leads to higher dissimilarity in the group. But choice of metric aside, it also means that using such an approach would necessitate a post processing step to split big clusters and glue small clusters to existing clusters, which would make a lot of changes to the cluster output. Ideally, one would like the clustering algorithm to be cognizant of these size constraints.

Even though KL heuristic is closely related to OPTMOVE, their approach isn't directly applicable in our setting. However, one key difference between our approach and theirs is that while our moves are probabilistic, theirs is deterministic. We implemented OPTMOVE with the same idea of combining moves as KL, this is similar to the local-search approach proposed in [9], which we call OPTMOVE-KL. Fig. 5 shows that our probabilistic approach always finds a better minima of the objective. While KL strategy, since it picks the optimal move every time, has a better slope; because of the lack of variety in the moves, it settles at minima earlier as well.

A tunable parameter of our approach is *gap*, which specifies how long does a vertex once moved has to wait for the next move. This is to avoid giving few outlier vertices too much influence. This was accomplished in KL by not moving a vertex once moved till every other vertex had been moved. In our heuristic, this is less of a problem since the moves are probabilistic. Still, as Fig. 5 demonstrates letting gap be 0, does seem slightly suboptimal as the curve for that case is consistently higher or at the same levels as when gap is 20%. But clearly even letting gap be 0 does better than the deterministic KL strategy.

Finally, we also evaluated the forecast accuracy of a multivariate forecasting model on clusters produced by OPTMOVE vs Kernel K-Means. We only experimented with consistent sellers with long sales history to make sure the results are reliable. One of the challenges with these comparisons is that most of the error is contributed by *bad* points in the data, that the model can't catch because of missing information to predict them. These can't be alleviated by smart grouping. What a good grouping can do is bring the error on *good* points down, which may be small when compared to the error contributed by the bad points, but is still observable. We summarize the results in Table I. The improvements due to OPTMOVE is around 4-8%. This is about as much improvement as one can expect in aggregate, since as we explained its hard to move the aggregate error numbers too much by just different groupings. Note that the number of items modeled using OPTMOVE is consistently more since kernel k-means produces many small clusters which don't have enough data, hence are discarded during modeling. Fig. 7 has the plot for item-wise error. The two distributions look very similar since the plot had to be done on log scale due to high variation in data. As we can see, there are items with about 100% error. However, as we see in spite of the fact that kernel k-means models fewer items and has already discarded most outliers in small clusters which could not be modeled, OPTMOVE has fewer big outliers. It accomplishes this by bringing some points from bad to good set, which don't show up in aggregate numbers.

## V. Related Work

The general approach of panel data models all the entities together. However, modeling dissimilar entities together has obvious disadvantages, hence many tests have been proposed on whether to model all the entities together, or to find subset of them that can be, see [25]–[27]. The approach we follow, generalizes this idea by allowing for decomposition of entities into multiple groups, successfully demonstrated in some prior work such as [27]–[30]. A major departure of our approach from these is that those clustering algorithms have been developed with a model in mind, to make sure different entities are homogeneous w.r.t their model fits. Indeed one of the motivations of our work was that it isn't even possible to successfully fit a model to individual items in our case. One could conceive of a bayesian model-based clustering framework wherein we would find the groups that lead to best model fit. There are two reasons we don't pursue model-based clustering: (i) it is very expensive and almost impractical in our setting at least when done naively, and (ii) a good model fit doesn't necessarily lead to good forecasts. We instead try to impose intuitive constraints on our clusters that we know to be necessary for reliable forecasts.

There is a big literature on time-series clustering, both on computing similarity and clustering algorithm. Most of the similarity metrics treat the time-series as a vector or a distribution. Another class of metrics try to summarize the time-series using some model fit, or other characteristics [11], [31]–[33]. Since seasonality is frequently the most important part, there are also approaches which measure the similarity just based on seasonality. For e.g., [34] defines similarity based on a $\chi^2$-test to determine if the two seasonal patterns follow the same distribution. Dynamic Time Warping(DTW) [35], [36] tries to align two series to minimize their distance: this is particularly useful if one wanted to compare seasonal profile of time-series, because annual events don't always fall on the same day every year. Many of these metrics could be potentially useful in different scenarios for us, but they need to be extended to be robust against missing values. Also, a lot of work in this area has been focused on scaling these operations to deal with high-dimensionality of the time-series involved: this is not relevant for us since, because of some pre-processing and aggregation, our time-series are of a bounded size.

After the similarity matrix has been computed, clustering time-series is no different from graph clustering which is a very well-studied field in itself. We are not aware of any prior work that handles all the constraints we do, in particular the cluster quality constraint. There is a renewed interest in overlapping clustering thanks to community detection in social networks, see [37]–[39]. These approaches if modified to satisfy the constraints in our setting could be useful.

## VI. Conclusion

In this paper, we discussed, formalized and listed our approaches to grouping SKU for forecasting in retail. We have defined the problem independent of the forecasting model being used. While most of the forecasting literature has tended to side-step the grouping problem, our goal in this paper has been to introduce this as an interesting problem on its own. We presented our heuristic which is scalable and yields solutions with quality as good as the contemporary approaches, while still satisfying the myriad of constraints imposed in our setting. We also presented an approach to computing the similarity metric for a new item with no sales: new items are introduced very frequently in e-commerce and finding items similar to them has typically been done using semantic information with some ad hoc manual manipulation. Our approach seeks to design a more principled approach to this as well. We hope this paper provides readers with a better understanding of some of the problems in e-commerce/retail setting.

## References

[1] P. H. Zipkin, *Foundations of inventory management*, 2000, vol. 2.

[2] J. G. De Gooijer and R. J. Hyndman, "25 years of time series forecasting," *International journal of forecasting*, vol. 22, no. 3, pp. 443–473, 2006.

[3] C. Hsiao, *Analysis of panel data*. Cambridge university press, 2003, vol. 34.

[4] C. A. Sims, "Macroeconomics and reality," *Econometrica: Journal of the Econometric Society*, pp. 1–48, 1980.

[5] P. J. Harrison and C. F. Stevens, "Bayesian forecasting," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 205–247, 1976.

[6] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman, "Indexing by latent semantic analysis," *JASIS*, vol. 41, no. 6, pp. 391–407, 1990.

[7] R. Rosipal and N. Krämer, "Overview and recent advances in partial least squares," in *Subspace, Latent Structure and Feature Selection*. Springer, 2006, pp. 34–51.

| Category | #items-OPTMOVE | #items-k-kmeans | Error-OPTMOVE | Error-k-kmeans |
|---|---|---|---|---|
| 1 | 504 | 486 | 63.64 | 66.34 |
| 2 | 283 | 258 | 53.22 | 55.27 |
| 3 | 1754 | 1703 | 54.31 | 55.60 |
| 4 | 573 | 558 | 45.94 | 49.66 |

TABLE I
FORECAST ACCURACY( EQ. 1) OBTAINED BY CLUSTERING ACCORDING TO OPTMOVE VS KERNEL K-MEANS

[8] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell system technical journal*, vol. 49, no. 2, pp. 291–307, 1970.

[9] I. S. Dhillon, Y. Guan, and J. Kogan, "Iterative clustering of high dimensional text data augmented by local search," in *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on*. IEEE, 2002, pp. 131–138.

[10] T. Warren Liao, "Clustering of time series data–a survey," *Pattern recognition*, vol. 38, no. 11, pp. 1857–1874, 2005.

[11] X. Wang, K. Smith, and R. Hyndman, "Characteristic-based clustering for time series data," *Data Mining and Knowledge Discovery*, vol. 13, no. 3, pp. 335–364, 2006.

[12] M. Kendall and A. Stuart, "The advanced theory of statistics. vol. 1: Distribution theory," *London: Griffin, 1977, 4th ed.*, vol. 1, 1977.

[13] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to information retrieval*. Cambridge university press Cambridge, 2008, vol. 1.

[14] C. H. Papadimitriou, H. Tamaki, P. Raghavan, and S. Vempala, "Latent semantic indexing: A probabilistic analysis," in *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*. ACM, 1998, pp. 159–168.

[15] U. Von Luxburg, "A tutorial on spectral clustering," *Statistics and computing*, vol. 17, no. 4, pp. 395–416, 2007.

[16] I. S. Dhillon, Y. Guan, and B. Kulis, "Kernel k-means: spectral clustering and normalized cuts," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2004, pp. 551–556.

[17] L. Kaufman and P. J. Rousseeuw, *Finding groups in data: an introduction to cluster analysis*. John Wiley & Sons, 2009, vol. 344.

[18] H. Wold *et al.*, "Estimation of principal components and related models by iterative least squares," *Multivariate analysis*, vol. 1, pp. 391–420, 1966.

[19] W. Stacklies, H. Redestig, M. Scholz, D. Walther, and J. Selbig, "pcamethods–a bioconductor package providing pca methods for incomplete data," *Bioinformatics*, vol. 23, no. 9, pp. 1164–1167, 2007.

[20] M. E. Tipping and C. M. Bishop, "Probabilistic principal component analysis," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 61, no. 3, pp. 611–622, 1999.

[21] B.-H. Mevik and R. Wehrens, "The pls package: principal component and partial least squares regression in r," *Journal of Statistical Software*, vol. 18, no. 2, pp. 1–24, 2007.

[22] J. Friedman, T. Hastie, and R. Tibshirani, "Regularization paths for generalized linear models via coordinate descent," *Journal of statistical software*, vol. 33, no. 1, p. 1, 2010.

[23] T. F. Gonzalez, "Clustering to minimize the maximum intercluster distance," *Theoretical Computer Science*, vol. 38, pp. 293–306, 1985.

[24] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," in *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035.

[25] B. H. Baltagi and J. M. Griffin, "Pooled estimators vs. their heterogeneous counterparts in the context of dynamic demand for gasoline," *Journal of Econometrics*, vol. 77, no. 2, pp. 303–327, 1997.

[26] B. H. Baltagi, J. M. Griffin, and W. Xiong, "To pool or not to pool: Homogeneous versus heterogeneous estimators applied to cigarette demand," *Review of Economics and Statistics*, vol. 82, no. 1, pp. 117–126, 2000.

[27] G. Kapetanios, "Cluster analysis of panel data sets using non-standard optimisation of information criteria," *Journal of Economic Dynamics and Control*, vol. 30, no. 8, pp. 1389–1408, 2006.

[28] F. Vahid, "Clustering regression functions in a panel," in *Econometric Society world congress*, 2000.

[29] H. Lu and S. Huang, "Clustering panel data," *Data Mining for Marketing*, p. 1, 2011.

[30] D. C. Bonzo and A. Y. Hermosilla, "Clustering panel data via perturbed adaptive simulated annealing and genetic algorithms," *Advances in Complex Systems*, vol. 5, no. 04, pp. 339–360, 2002.

[31] D. Piccolo, "A distance measure for classifying arima models," *Journal of Time Series Analysis*, vol. 11, no. 2, pp. 153–164, 1990.

[32] E. A. Maharaj, "Cluster of time series," *Journal of Classification*, vol. 17, no. 2, pp. 297–314, 2000.

[33] M. Vlachos, J. Lin, E. Keogh, and D. Gunopulos, "A wavelet-based anytime algorithm for k-means clustering of time series," in *In Proc. Workshop on Clustering High Dimensionality Data and Its Applications*. Citeseer, 2003.

[34] M. Kumar, N. R. Patel, and J. Woo, "Clustering seasonality patterns in the presence of errors," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2002, pp. 557–563.

[35] D. J. Berndt and J. Clifford, "Using dynamic time warping to find patterns in time series." in *KDD workshop*, vol. 10, no. 16. Seattle, WA, 1994, pp. 359–370.

[36] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh, "Searching and mining trillions of time series subsequences under dynamic time warping," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2012, pp. 262–270.

[37] J. J. Whang, D. F. Gleich, and I. S. Dhillon, "Overlapping community detection using seed set expansion," in *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*. ACM, 2013, pp. 2099–2108.

[38] M. Coscia, G. Rossetti, F. Giannotti, and D. Pedreschi, "Demon: a local-first discovery method for overlapping communities," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2012, pp. 615–623.

[39] J. Yang and J. Leskovec, "Overlapping community detection at scale: a nonnegative matrix factorization approach," in *Proceedings of the sixth ACM international conference on Web search and data mining*. ACM, 2013, pp. 587–596.