Capstone Project Final Report
User Segmentation Models

Siwen Tang
st3101@columbia.edu

Meiqi Chi
mc4394@columbia.edu

Leshen Sui
ls3452@columbia.edu

TaeYoung Choi
tc2777@columbia.edu

Aditya Huddedar
ah3426@columbia.edu

Abstract

We present a diverse set of clustering algorithms to obtain the most optimal user segmentation based on a binary user-traits data provided by Adobe Audience Manager team. We divide the problem into two parts, namely, optimizing the number of clusters and optimizing the clustering method and solve each one independently. We use a modified version of the Calinski-Harabasz Index to evaluate the performance of our methods. The given dataset is extremely sparse, and hence we want to use a denser lower dimension representation of the data for clustering. We design three uniquely distinct models to perform the clustering. We offer a naive Bisecting K-means model for simple but fast clustering needs. Our Auto-encoder based model captures the non-linear structure in the data extremely well, whereas the Matrix Factorization based clustering effectively handles missing entries in the dataset. Based on our evaluation metric values for different numbers of clusters, we conclude that given dataset has around five underlying user segments. We believe, with enough training, our models can be tuned to cluster any general sparse binary dataset.

1 Introduction

Advances in social media technology in the last decade have made it possible to collect a lot of user information based on their online behavior. With a plethora of these user-traits data available, it is now possible to target the audience more efficiently to maximize the reach of the content, be it an advertisement or a social message. For our Data Science Capstone Project, we are working on the problem of user segmentation based on the characteristics data.

Our goal is to design an algorithm that automatically clusters the users into a fixed number of classes. The number of classes can be provided as an input to the algorithm. We also provide visualizations of the clusters for qualitative evaluation. Using the Calinski-Harabasz index and Silhouette score, we find the optimal number of user segments.

In the dataset provided by Adobe Audience Manager team, each user is associated with a certain number of traits. Each trait is a binary variable stating whether a user possesses the trait. The traits are defined by the customers of Adobe, and hence, they are not standardized. Due to security reasons, we cannot access any meta-data associated with the traits. Therefore, we treat these traits as a black box. The traits are large in number which makes the user segmentation job really difficult. On top of that, the zeros in the dataset do not necessarily denote the absence of a trait. In some cases, the status of certain traits is not known for a particular user, and the corresponding entries would be marked as zeros.

Our aim in this project is to alleviate both the **high dimensionality** and **missing data** issues by designing efficient and scalable clustering algorithms.

2 Related Work

In this section, we review some of the existing techniques that can be used for user segmentation. We divide the techniques into three broad categories: naive clustering methods, matrix factorization and lower dimensional encodings.

2.1 Naive Clustering Methods

Naive clustering methods involve very little to no pre-processing of the dataset and can be used when the given dataset is relatively dense.

2.1.1 Centroid-based clustering

The classic example of a centroid-based clustering algorithm is K-means. It is perhaps the simplest, yet elegant way to visualize the underlying segments of a given dataset. The objective is to minimize the within-cluster sum of squares (WCSS). A typical implementation of K-means iteratively assigns the data points to one of the clusters and updates the cluster centers until the assignment reaches a steady state. The resulting clusters may be a local optimum, i.e., not necessarily the best possible outcome.

Fuzzy C-means[2] is another centroid-based algorithm that assigns data points a membership in each cluster center. The fuzzy logic principles can be useful to cluster multidimensional data and implementation is available in python. Clustering large applications (CLARA) [11] is an iterative K-medoid algorithm that deals with clustering of a large dataset and is available in R.

2.1.2 Density-based clustering

In density-based clustering, clusters are defined as areas of higher density than the remainder of the data set. The data points from these sparse areas are usually treated as noise or border points.

One of the most popular density-based clustering methods is DBSCAN[5]. Built on top of the DBSCAN, HDBSCAN[12] method developed by McInnes and Healy (2017) allows clusters with varying density. OPTICS[1] (Ordering Points To Identify the Clustering Structure) successfully deals with the ϵ-neighborhoods problem in DBSCAN and provides solutions of better quality as compared to the default hierarchical methods.

2.1.3 High-dimensional clustering

The algorithm BIRCH[17] (Balanced Iterative Reducing and Clustering using Hierarchies) is a well-known reduction method for high dimensional data. It summarizes the data into a height-balanced tree (Characteristic Feature Tree) to minimize the cost of input-output operations during clustering.

Proposed by Moise, Sander and Ester (2006), P3C [13] is another robust algorithm for projected high-dimensional clustering that can effectively discover clusters in the data while minimizing the number of parameters required as input.

SubClu[10] is able to detect arbitrarily shaped and positioned clusters in sub-spaces and deliver for each subspace the same clusters DBSCAN would have found when applied to this subspace separately.

2.2 Dimensionality Reduction

Dimensionality Reduction facilitates feature selection, noise reduction, data visualization and reduces the complexity of machine learning algorithms.

Principal component analysis (PCA) [8] is a widely used statistical technique for unsupervised dimension reduction. It transforms high dimension data into low dimensions and selects the dimensions with greatest variances, which is equivalent to computing the best low-rank approximation of the data via singular value decomposition (SVD)[4].

However, PCA assumes that the principal components are a linear combination of the original features, which restricts the power of this model. Autoencoder [7] is a nonlinear generalization of PCA that uses an adaptive, multilayer encoder network to transform the high-dimensional data into a low-dimensional code and a similar network to recover the data from the code.

2.3 Matrix Factorization

Non-negative matrix factorization (NMF) is frequently used in clustering, recommendation systems and decomposition. There are many methods for conducting NMF. DD Lee, HS Seung(2001) analyze two different multiplicative algorithms in [3]; one minimizes the conventional least squares error while the other minimizes the generalized Kullback-Leibler divergence.

As mentioned in the Introduction, the dataset might contain some missing data. Therefore, we need to consider some measures in matrix factorization space to handle the missing data. Most matrix factorization models assume that the data is missing at random (MAR), but this is rarely the case. Steck(2010) [6] showed that missing data carried useful information for improving the prediction performance in recommendation systems. Jose, Neil & Zoubin(2014) [9] proposed a probabilistic matrix factorization method that models both the generative process for the data and the missing data and obtained improved performance by learning these two models jointly.

In general, a Matrix Factorization technique can be used to decompose a large matrix into two smaller matrices, which when multiplied together yield a matrix approximately equal to the large matrix. For clustering purposes, reducing the large sparse traits matrix dataset into smaller latent feature matrices for users and traits will be useful in obtaining a denser representation for the users and then clustering based on the latent features.

3 Our Contributions

We visualize the user segmentation problem as an optimization problem where we want to optimize both the Clustering Method and the Number of Clusters in the dataset using a common evaluation metric. Since these two are fairly independent problems, we can evaluate them in parallel.

Based on our literature survey from section 2, we choose the following three candidate algorithms for clustering:

- **Naive K-means**: This is our baseline model. It is the simplest and the fastest way to obtain the user segments.
- **Autoencoder**: We aim to capture the non-linear relationships among the traits using an Autoencoder. The lower dimension output of the encoded layer from the network would then be used to obtain the user segments.
- **Matrix Factorization**: We use matrix factorization to both reduce the dimensions, as well as to capture any missing data patterns in the dataset. The latent vectors from user space are subsequently used for clustering.

We run each of these algorithms for a variety of the values of K (number of clusters) and choose the best value of K based on a quantitative measure that we will describe in later sections.

4 Exploratory Data Analysis

Data exploration is the first step towards drawing inferences from the data. The dataset provided by Adobe Audience Manager team contains information about approximately 137 million user profiles and 5197 traits. It is a random sample of the user-traits data shared by one of the customers of Adobe. Due to data security reasons, we cannot access labels or any meta-data for the users or the traits.

The dataset is presented as a 137 million $\times$ 5197 size matrix, where each row corresponds to a unique user profile. The matrix is extremely sparse. Only 0.18 % of the matrix is populated with ones. We generate a small random sample of the dataset for exploratory analysis.

A quick examination of traits histogram in Fig 1 reveals that the trait '800' occurs most frequently in the sample. Besides, it is the only trait for

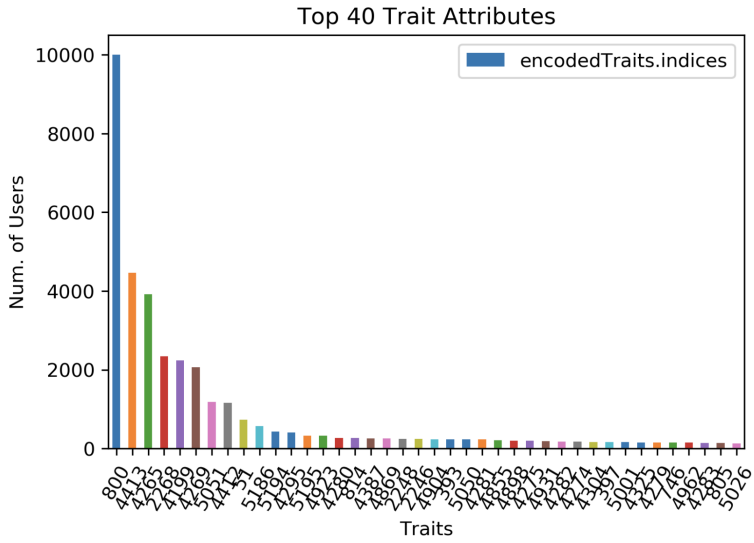

Figure 1: Top 40 Traits Histogram

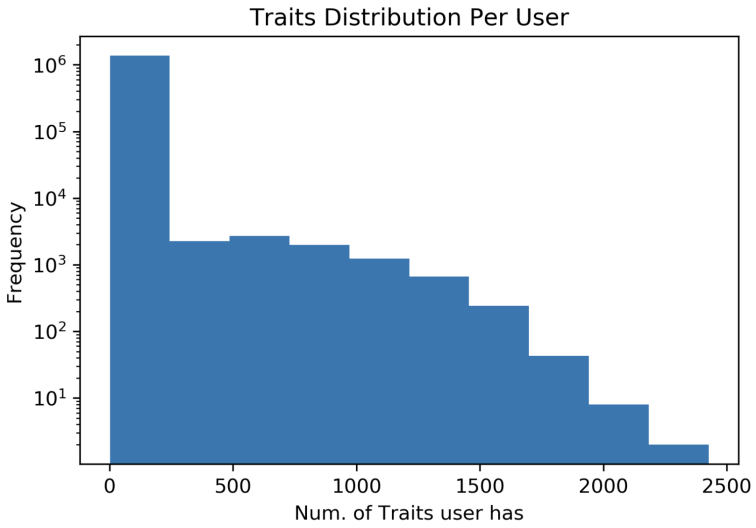

Figure 2: Log Scale Traits Distribution

all single-trait users, which are more than 50 % in the sample data. The trait has a significant contribution to the long tail effect.

The distribution of the number of traits a user has is illustrated by the

histogram in Figure 2. As expected, a few outliers in the user group have numerous traits, whereas most of the users have less than 10 traits. The number of users gradually decreases as the number of traits increases. There are very few users who have more than 2000 traits.

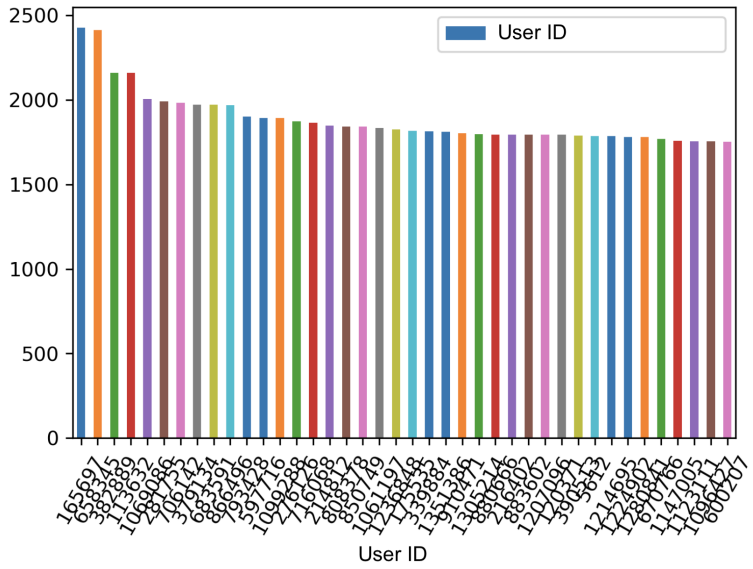

Figure 3: Histogram w.r.t Top Users

On the other hand, the plot in Figure 3 shows the histogram distribution of the top 40 users and their traits frequency. There are no extreme values observed from the number of traits among top users.

Having obtained a basic overview of the dataset, we now proceed towards the user segmentation models in the next few sections.

5 Evaluation Metric

The first thing we need to define, in order to evaluate our clustering approaches, is an evaluation metric. We choose a variant of the Calinski-Harabasz Index, customized to handle to binary data, as our evaluation metric. Following is a mathematical formulation of our evaluation metric.

$$s(k) = \frac{B_k}{W_k} \times \frac{N-k}{k-1}$$
$$B_k = \sum_{q=1}^{k} n_q \cdot d(c, c_q)$$
$$W_k = \sum_{q=1}^{k} \sum_{x \in C_q} d(x, c_q)$$

Here, c_q is the center of cluster q and c is the global center of dataset. n_q is the number of points within cluster q. Lastly, $d(x, y)$ is the Hamming distance between x and y.

Note that, in place of the Euclidean distance, we use the Hamming distance as our distance metric. Consequently, we also compute the center of the cluster by binarizing the mean of the cluster points. We discovered that using the mode leads to identical cluster centers because a majority of columns are zeros. Therefore, we use a threshold of 0.003 (which is the average proportion of 1's in our dataset) to binarize the mean. This approach is more robust to biases than computing the mode.

Also, note that the evaluation metric is always applied to the final clusters on the original dataset, irrespective of the underlying model. This technique makes our evaluation method consistent across the models and enables us to compare different clustering methods.

In the following three sections we will go over our models in detail.

6 Naive K-means

Naive K-means is our baseline model for simple and fast evaluation. As the name suggests, we simply run a Bisecting K-means algorithm on the traits data. More details about the implementation of bisecting K-means can be found at `http://mlwiki.org/index.php/K-Means#Bisecting_K-Means`.

7 Autoencoder

As mentioned in Section 3, we would like to create a lower dimensional dense representation of the given dataset before applying a clustering method to it. There are two ways to achieve this. We can treat Dimensionality Reduction and Clustering as independent problems or combine them into a single optimization problem, as in [15] or [16]

We choose the former model for its ease of implementation and training. We use an autoencoder for dimensionality reduction and apply a Bisecting K-means clustering on the encodings. Figure 4 shows our implementation of the network in PyTorch. Our autoencoder model consists of five non-linear encoding layers, with sizes (1024, 512, 256, 128, 64) respectively and five symmetric decoding layers. We use the ReLU function to introduce non-linearity. Noticeably, we do not use any Convolutional layers, mainly because we do not believe the local, regional combinations of the traits will add any significant value to the model.

Since our data is binary, we use a Binary Cross Entropy loss function to preserve most of the information in the input. Figure 5 depicts the

```
class autoencoder(nn.Module):
    def __init__(self):
        super(autoencoder, self).__init__()
        self.encoder = nn.Sequential(
            nn.Linear(data.shape[1], 1024),
            nn.ReLU(True),
            nn.Linear(1024, 512),
            nn.ReLU(True),
            nn.Linear(512, 256),
            nn.ReLU(True),
            nn.Linear(256, 128),
            nn.ReLU(True),
            nn.Linear(128, 64),
            nn.ReLU(True))
        self.decoder = nn.Sequential(
            nn.Linear(64, 128),
            nn.ReLU(True),
            nn.Linear(128, 256),
            nn.ReLU(True),
            nn.Linear(256, 512),
            nn.ReLU(True),
            nn.Linear(512, 1024),
            nn.ReLU(True),
            nn.Linear(1024, data.shape[1]),
            nn.Sigmoid())

    def forward(self, x):
        encoded = self.encoder(x)
        decoded = self.decoder(encoded)
        return encoded, decoded
```

Figure 4: Autoencoder Network

20 epochs of training on the sample dataset. Notice that the model loss quickly converges within 10 epochs. Once we have the encodings from the autoencoder, we run a Bisecting K-means algorithm on them and generate the clusters.

As mentioned in Section 4, the trait "800" seems a little problematic. It is present as the only trait for many users and thus skews the clustering

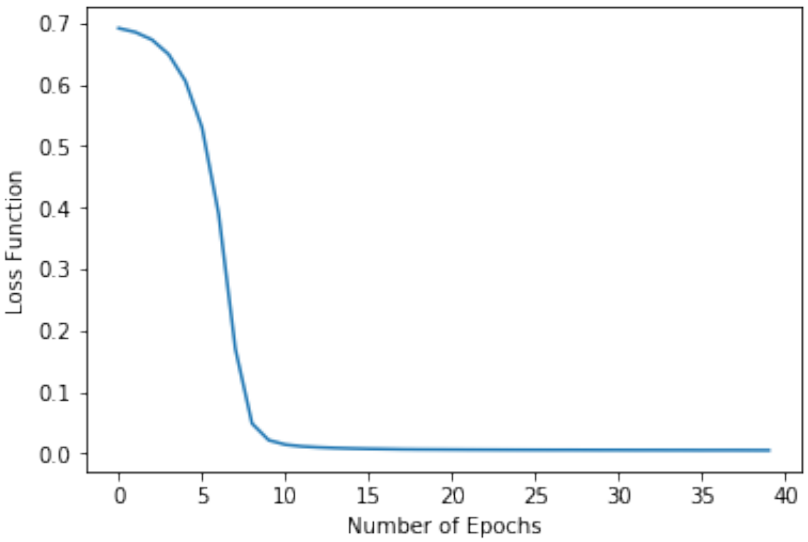

Figure 5: Encoder Training

heavily towards it. Therefore, we create separate models, with and without the trait "800".

7.1 Results

Figure 6 shows the evaluation metric Plots for different K's, for different models discussed in sections 6 and 7. The plot shows that $K = 5$ is the first non-trivial value of K with the best result for the evaluation metric.

As we can clearly see in this plot, the autoencoder adds significant value to the naive K-means model. Also, removal of the trait "800" helps improve the clustering even further.

7.2 Stability

When we train an autoencoder, the only constraint that we impose is between the input, which is the original dataset, and the output of the decoding layer. There is no explicit guarantee that the encoded layer efficiently captures the signals from the input. Since we heavily rely on the embeddings for the clustering step, we need to make sure the variance in the encodings does not affect the clustering significantly.

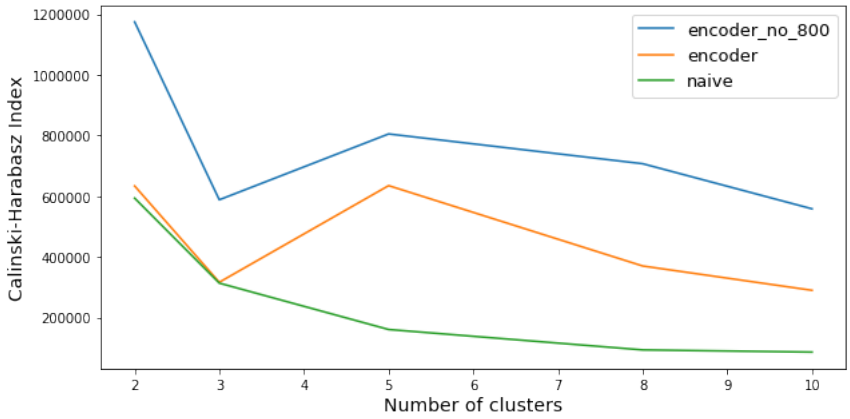

Figure 6: Calinski-Harabasz Index for $1 < k \leq 10$

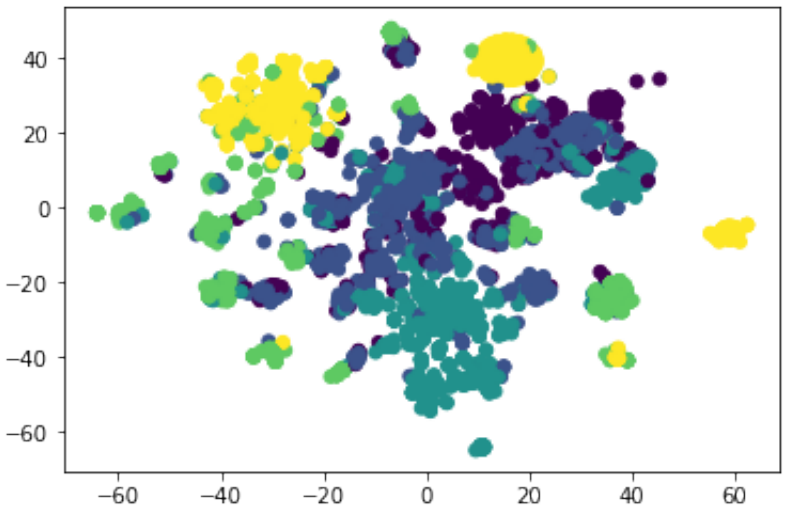

Figure 7: T-SNE with 5 Clusters

As seen in Figure 8, the embeddings vary depending on the random initialization of autoencoder. The columns in the figure represent ten different trials and rows represent the embeddings for those trials. This result aligns our suspicion about the instability of the encodings. However, the impact of this instability on the final clusters seems to minimal. In the table below, we present the evaluation metric values with $K = 5$, for the ten different trials.

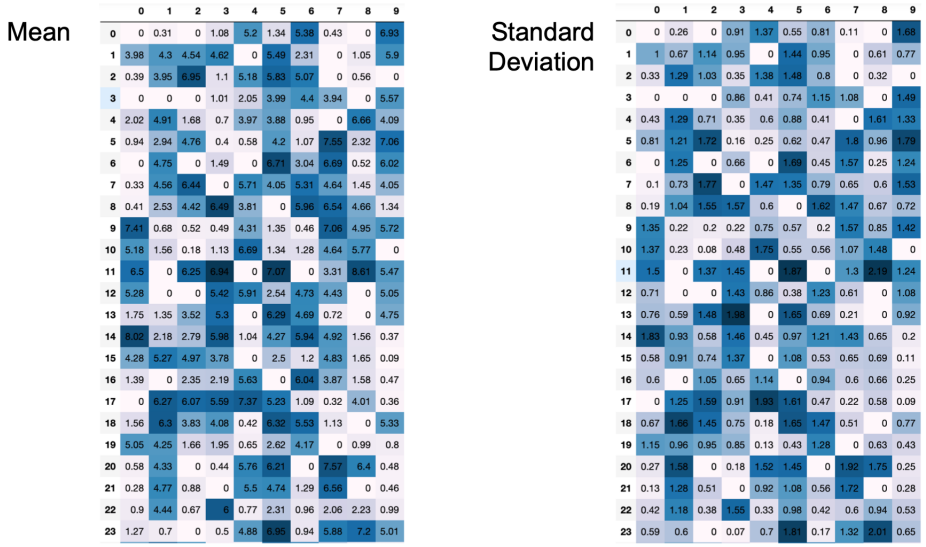

Figure 8: Distribution of Embeddings for 10 test runs

Trial	Calinski-Harabasz Index
1	154,488
2	158,717
3	152,616
4	155,738
5	159,042
6	159,631
7	157,952
8	155,647
9	155,676
10	154,438

The scores are very stable even though the embeddings look completely different for each autoencoder initialization. The mean of ten scores is 156,391 with a standard deviation of 2325.5, which is only 1.4% of the mean. This tells us that the construction of embeddings is largely affected by the order in which we feed the dataset during training and the initialization of autoencoder. However, that does not affect the final clusters significantly. Thus, we can conclude that these encodings are all local extrema that can map original features down to a lower dimension effectively. This is a fascinating characteristic of our model that makes an additional explicit constraint over

the embedding layer unnecessary.

7.3 Scalability

Autoencoder requires careful training like any other deep neural networks. It often takes a large amount of data to achieve an accurate network, which makes the training job hard. Considering we are dealing with a relatively simpler data structure, a binary matrix, we try to find the optimal sample of the data which would be sufficient for training our model. We can then use the trained model to generate encodings from the rest of the dataset and then apply a K-means model on top of those encodings.

We design a test to train the same autoencoder using a 10% random sample of the given dataset. We further obtain 10 %, 20 %, 40 %, 60 % and 80 % random samples from this sample. Our goal is to train the autoencoder on these subsamples and use the trained model to encode the entire 10% sample.

As seen in Figure 9, the quality of the clusters ceases to improve once the subsample size reaches 40 % of the sample. We need to further investigate on why clustering quality decreases from 40 % to 60 %, but it is clear that the scalability of autoencoder can be achieved by using a smaller subset of the given data. We can reduce the training time by 65% by using a 40% random sample while achieving better clustering quality.

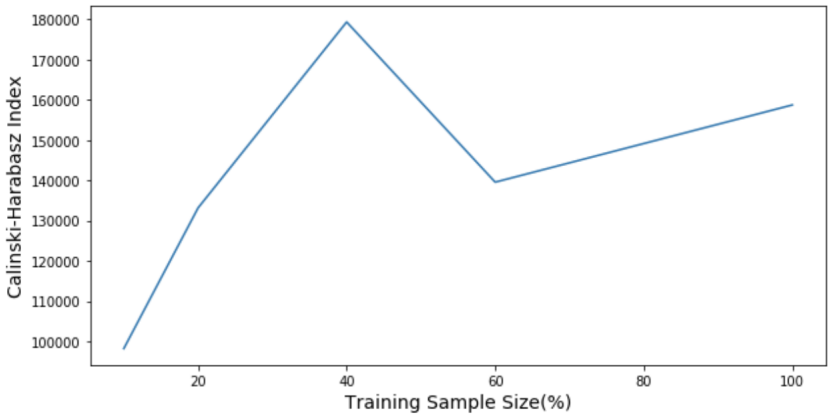

Figure 9: Calinski-Harabasz Index by training size

Thus, autoencoder indeed seems to be a viable option to obtain a lower dimensional abstraction of the given dataset.

8 Matrix Factorization

In this section, we explore the Matrix Factorization approach. As stated in Section 1, we suspect a high number of <user, trait> entries are missing and are populated as zeros in the dataset. One of the goals of our Matrix Factorization models is to impute the score for an unknown user-trait pair, which indicates the probability that the given user possesses the trait. Matrix Factorization approach is very effective for decomposing large sparse matrix into low dimensional user factors and trait factors, which is a simple yet effective way of representing users for the objective of user segmentation.

8.1 Matrix Factorization with ALS

The model represents the probability of a user i possessing a trait j, p_{ij}, by the product of latent user factor and trait factor $p_{ij} = u_i^T v_j$, where u_i, v_j represents the i's user factor and j's trait factor. For entries which have already been filled with a 1, we interpret the probability to be 1, and after the imputation, the higher score of p_{ij}, the more likely that user i possesses the trait j. In order to find the best low-rank matrices, we want to minimize the reconstruction error on observed entries(also known as partial fitting), which can be expressed as:

$$\min_{U,V} \sum_{p_{ij}==1} (1 - u_i^T v_j)^2 + \lambda(\sum_i \|u_i\|_2^2) + \sum_j \|v_j\|_2^2) \tag{1}$$

8.1.1 Experiment

We run our model on a 1 % sample of the entire dataset. The sample contains 1.3M users, which leads to a 1.3M $\times$ 5197 sparse matrix. Running ALS on such a large dataset without a GPU takes an acceptable amount of time for one pass, but it takes too long for hyper-parameter tuning. So in this experiment, we sub-sample 100k data points for cross-validation and visualization usage, and we perform clustering based on our best imputation to the whole sample and report the clustering result.

To construct a test dataset, we mask 20 % of the observed entries and evaluate Mean Squared Error(MSE) and precision at k (denoted by P@k) on masked entries via cross-validation, and we grid search on the number of factors k and regularization parameter λ. Here we define P@k to be the fraction of known positives in the first k positions of the ranked list of results. The result is as follow:

	factors	**10**	**20**	**40**	**50**
$\lambda = 0.1$	P@k	0.424 25	0.090 95	0.0819	0.001 35
	MSE	0.001 17	0.001 24	0.001 32	0.001 21
$\lambda = 10$	P@k	0.4252	0.446 05	0.3935	0.392 45
	MSE	0.001 09	0.001 15	0.001 19	0.001 21
$\lambda = 100$	P@k	0.4252	0.5245	0.524 65	0.524 65
	MSE	0.000 57	0.000 57	0.000 57	0.000 57

Based on these results, we pick factors, $k = 40$ and $\lambda = 100$ as our best model for clustering.

8.1.2 K-means Clustering on Users' Factors

We apply our initial K-means model on user's factors, and use T-SNE to project the factors to 2-D space for visualization. We grid search on the number of clusters K based on Silhouette Score:

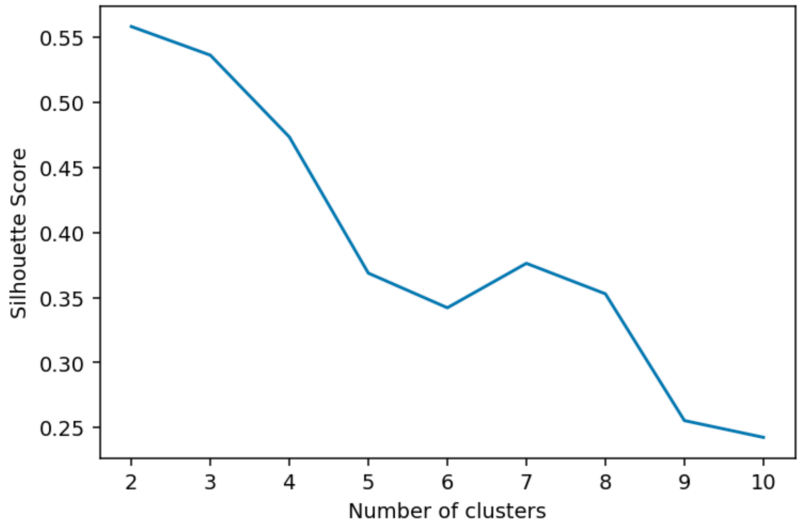

Figure 10: Silhouette Score for $1 < \mathrm{K} \leq 10$

It is a little surprising that $K = 2$ achieves the best Silhouette Score(Figure 10), which indicates a smaller number of clusters is reasonable here. We explore some more choices of K, and visualize the clusters projected to 2-D via T-SNE.

The clustering results in Figure 11 look reasonable because there are clear borderlines among clusters. As K increases to 6, there is a fair amount

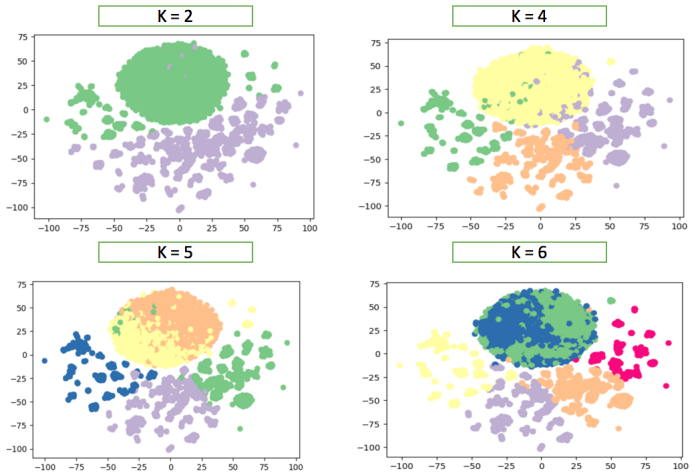

Figure 11: Clustering projected by T-SNE

of overlap between the green and blue clusters. In fact, we find out that the eclipse cluster at the top of each plot is related to those users with single trait "800", hence the surrounding clusters may be more interesting to look at, depending on the context of their traits.

8.1.3 Clustering After Removing Trait 800

The next experiment is to remove trait "800" and repeat the above process. It is worth to note that after we remove trait "800", around 600K users do not have a single trait, so we have to remove them as well. The number of users reduces from 1.3M to 677K for this experiment, and the clusters projected to 2-D are plotted in Figure 12:

As Figure 12 shows, there is minimal overlap among the clusters even when K increases to 8, and the clusters are relatively equal in sizes. From the Silhouette Score in Figure 13, we can argue that the best number of clusters could be around 5 to 8, because we probably want as many clusters as possible without the Silhouette Score dropping too much. Here $K = 5$ achieves the best Silhouette Score and after $K >= 8$ the Silhouette Score drops fast.

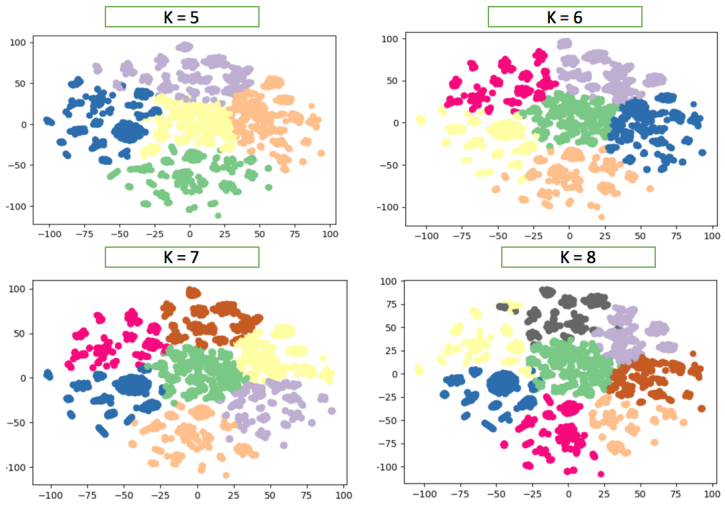

Figure 12: Clustering Without Traits 800

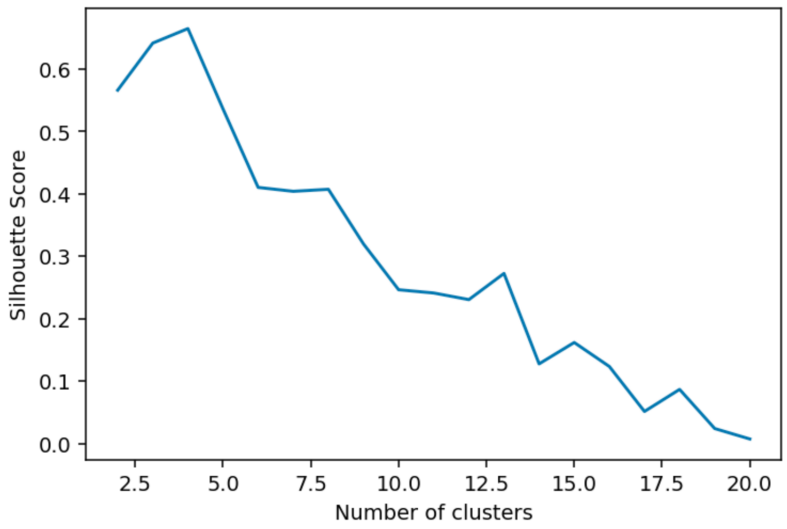

Figure 13: Silhouette Score Without Traits 800

8.1.4 Scaling Up

We migrate our best parameter setting to the whole dataset(137M users), and run ALS on a 16-core, 104 GB memory virtual machine. The training

process takes around 7 minutes/epoch, and the MSE loss curve becomes flat after 5 epochs and stabilizes at 0.00241. The Silhouette Score in Figure 14 achieves maximum at $K = 4$ and drops fast when K increases from 5 to 6, indicating that there are 4 or 5 underlying user segments in the dataset.

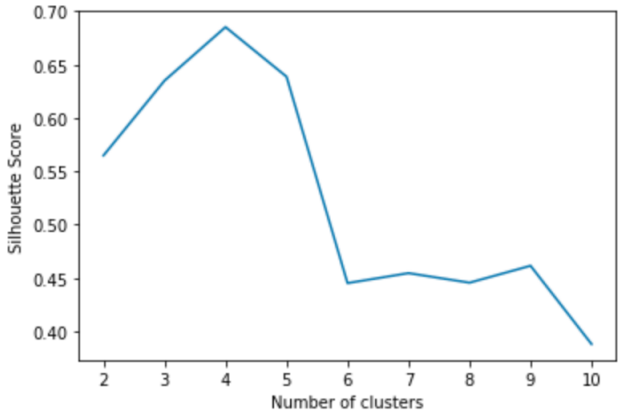

Figure 14: Silhouette Score on all users Without Traits 800

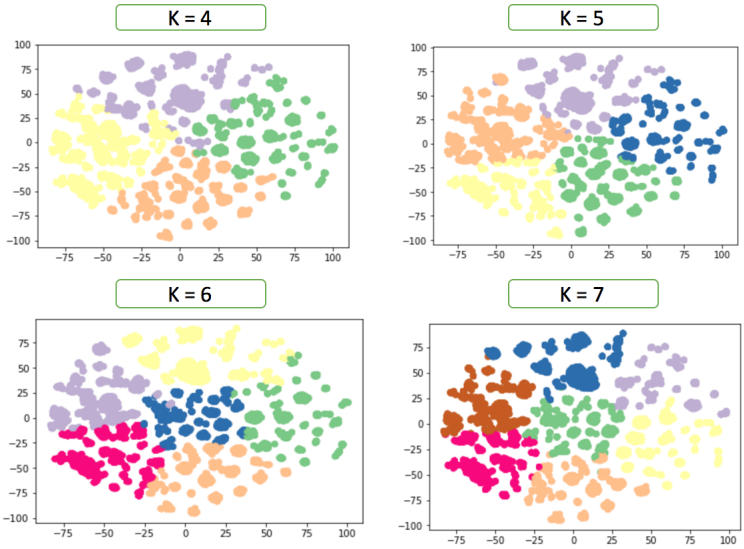

Figure 15: Clustering Without Traits 800 on Whole dataset

The visualization of clusters projected to 2-D by T-SNE in Figure 15 looks similar to our experiments on the 1 percent sample dataset. Thus, scaling up our algorithm doesn't deteriorate the quality of the user segments.

8.2 Reconstruction Error Weighted by Traits Frequency

The intuition for this experiment is that focusing on traits that are possessed less by the users would help us discover more similarity among users' attributes. For example, trait "800" is possessed by almost every user; hence it does not play a big role in terms of user representation. In order to achieve this, we normalize each observed entry by the total number of traits that appear in the dataset. For example, trait 800 appears 100K times in the sample dataset. Therefore each entry becomes 10^{-5} in that column.

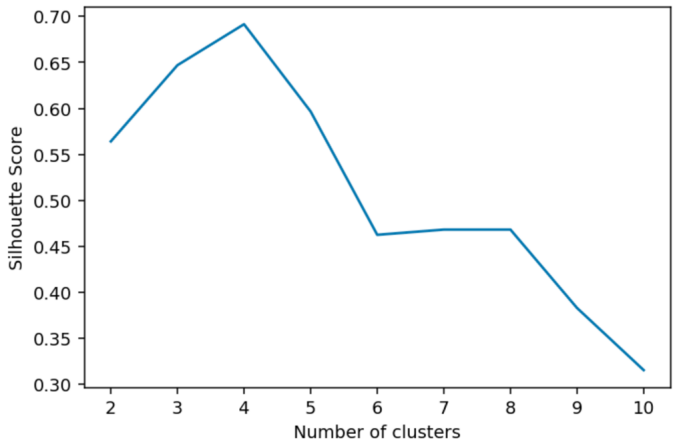

Figure 16: Silhouette Score on Weighted Reconstruction Error ALS Factors

From Figure 16 and 17, we conclude that $K = 4$ achieves the best number of clusters by Silhouette Score, and the quality of the clusters are similar to the previous experiment in 8.1.2. We believe this approach should work for traits with even more skewed distribution.

8.3 Factorization Machine

As stated in the previous section, Implicit ALS matrix factorization is powerful in representing users in low dimensional space and provides an insightful clustering result for user segmentation. However, the dataset is huge,

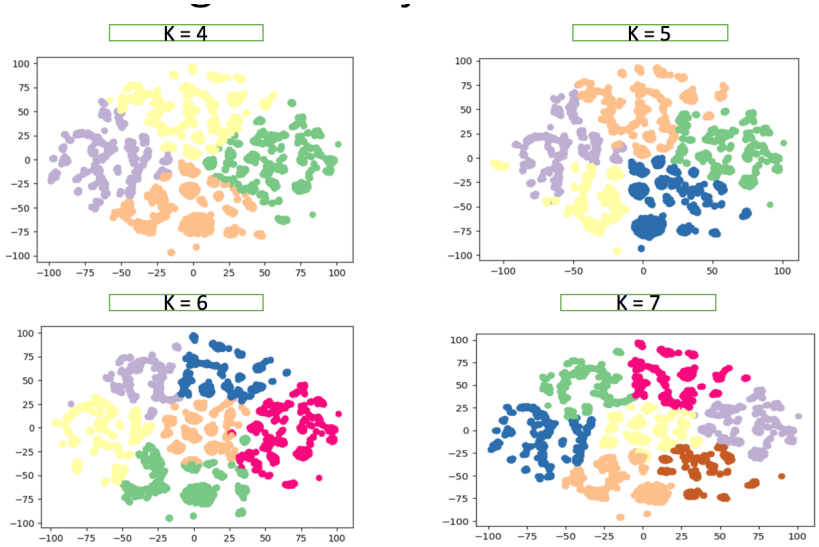

Figure 17: Clustering on Weighted Reconstruction Error ALS Factors

making predictions for such datasets becomes challenging with limited computation resources. We find that the matrix factorization takes over an hour training on the entire dataset. This motivates us to look for a model that can reduce the training time and achieve a meaningful user representation at large scale. We explore the Factorization Machine (FM) model to reduce the time of dimensionality reduction step.

This technique models the user-trait interactions using factorized parameters. It computes all pairwise interactions in linear time complexity using a simple mathematical manipulation. FM[14] is not only effective in scaling up the model on large sparse datasets but also extremely powerful in expressive capacity to generalize methods such as Matrix Factorization.

On a sparse dataset, where there is little information to estimate very complex interactions, FM with second-order interaction is effective enough to extract most important latent features.

A second order FM framework is shown below.

$$\hat{y}(x) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle v_i, v_j \rangle x_i x_j$$

.

Where the v's represent K-dimensional latent vectors associated with each variable (i.e., users and traits) and the $\langle\rangle$ operator represents the inner

product. Although FM's could be especially useful for incorporating external data as features, the FM without user/trait side information fits better for our data. The model works as follows: if we assume each x_j is the only non-zero vector at positions u and i, where $u \in U$ for all users and $i \in I$ for all traits, we can drop all other biases and interactions, and the equation becomes the following:

$$\hat{y}(x) = w_0 + w_u + w_i + \langle v_i, v_j \rangle$$

It contains a global bias w_0, user/trait specific biases w_u, w_i and user-traits interactions $\langle v_i, v_j \rangle$. This is the classic Matrix Factorization model.

The plots below depict the results of Factorization Machine method.

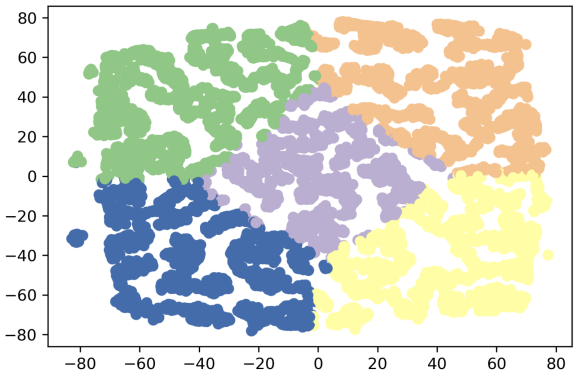

Figure 18: FM with 5 Cluster on Entire dataset

9 Conclusions and Limitations

We present a diverse set of models to solve the sparse binary feature data clustering problem.

- Naive K-means is a simple but fast way to obtain the clusters
- Autoencoder + K-means captures the non-linear structure in the data very well, and once trained on enough data can be used to draw inferences on a large dataset
- Matrix Factorization + K-means is the only approach, we tried, which handles the missing data aspect of the problem

Based on our evaluation metric values on clustering for different K's, we can conclude that the ideal number of clusters in the dataset is around 5. As far as the scalability of the models is concerned, Naive K-means and Matrix Factorization are ideal candidates, given pyspark implementations for the both of them already exist. With no built-in implementation, autoencoder seems a little difficult to train on the entire dataset.

Finally, we run the autoencoder model, the matrix factorization (MF) based model and the factorization machine (FM) based model on the same sample from the dataset. As seen in Figure 19, the matrix factorization (ALS + K-means) model seems to perform the best, followed by the autoencoder model and the factorization machine model. On the other hand, FM model is the fastest to train while the autoencoder model takes forever to train on the entire dataset.

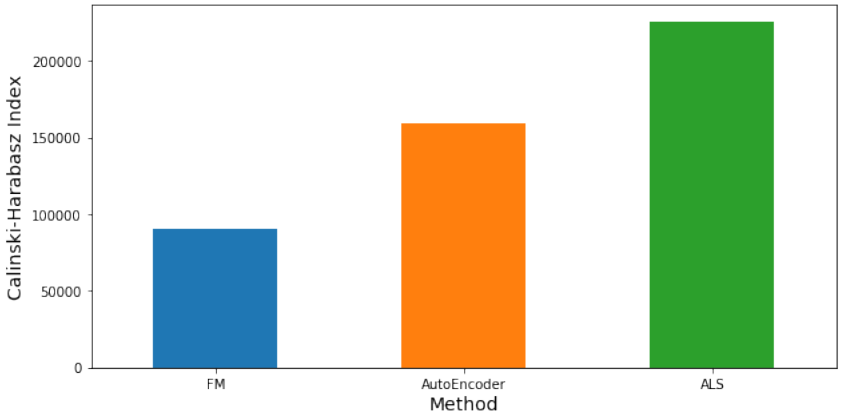

Figure 19: Performance Comparison

In terms of clustering quality, running K-means on ALS latent vectors generates the most compact and well-separated clusters. However, we have to keep in mind that ALS cannot create a latent vector for a new user. For a system like Adobe Audience Manager that experiences constant in-flow of new users, computing matrix decomposition in a timely manner could be too costly. Autoencoder, on the other hand, can compute embeddings for a new user without retraining the model. For the changing trait space, both methods require retraining, which becomes more problematic for autoencoder as it requires more time and resources to train.

Another drawback of the dataset itself is that it contains no meta-data for the traits. This limits our ability to interpret the clustering results

effectively. We can only rely on the evaluation metric to tune our models.

10 Acknowledgements

We would like to thank Prof. Eleni Drinea for giving us the opportunity to work on this project. We would also like to thank Charles and Handong for their valuable inputs.

11 Miscellaneous

Our codebase is hosted on GitHub at `https://github.com/selinatang95/capstone`. TaeYoung and Aditya worked on Naive K-means and Autoencoder approaches. Leshen, Meiqi and Siwen contributed to the Matrix Factorization based models.

References

[1] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. Optics: Ordering points to identify the clustering structure. *SIGMOD Rec.*, 28(2):49–60, June 1999.

[2] N. Bharill, A. Tiwari, and A. Malviya. Fuzzy based scalable clustering algorithms for handling big data using apache spark. *IEEE Transactions on Big Data*, 2(4):339–352, Dec 2016.

[3] H.Sebastian Seung Daniel D.Lee. *Algorithms for Non-negative Matrix Factorization*. Neural Information Processing Systems, 2000.

[4] Young G. Eckart, C. *The approximation of one matrix by another of lower rank*. Psychometrika, 1936.

[5] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, pages 226–231. AAAI Press, 1996.

[6] Alcatel Lucent Harald Steck. *Training and testing of recommender systems on data missing not at random*. Proceedings of the 16th ACM SIGKDD International conference on Knowledge discovery and data mining, 2010.

[7] Geoffrey E. Hinton and Ruslan R. Salakhutdinov. *Reducing the dimensionality of data with neural networks.* Science, 2006.

[8] I. Jolliffe. *Principal component analysis.* Springer, 2002.

[9] Zoubin Ghahramani José Miguel Hernández-Lobato, Neil Houlsby. *Probabilistic matrix factorization with non-random missing data.* ICML'14 Proceedings of the 31st International Conference on International Conference on Machine Learning, 2014.

[10] K. Kailing, H. Kriegel, and P. Kröger. *Density-Connected Subspace Clustering for High-Dimensional Data*, pages 246–256. 2004.

[11] Martin Maechler, Peter Rousseeuw, Anja Struyf, Mia Hubert, and Kurt Hornik. *cluster: Cluster Analysis Basics and Extensions*, 2018. R package version 2.0.7-1 — For new features, see the 'Changelog' file (in the package source).

[12] L. McInnes and J. Healy. Accelerated hierarchical density based clustering. In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 33–42, Nov 2017.

[13] G. Moise, J. Sander, and M. Ester. P3c: A robust projected clustering algorithm. In *Sixth International Conference on Data Mining (ICDM'06)*, pages 414–425, Dec 2006.

[14] Steffen Rendle. Factorization machines. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 995–1000. IEEE, 2010.

[15] Junyuan Xie, Ross B. Girshick, and Ali Farhadi. Unsupervised deep embedding for clustering analysis. *CoRR*, abs/1511.06335, 2015.

[16] Bo Yang, Xiao Fu, Nicholas D. Sidiropoulos, and Mingyi Hong. Towards k-means-friendly spaces: Simultaneous deep learning and clustering. *CoRR*, abs/1610.04794, 2016.

[17] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: an efficient data clustering method for very large databases. In *In Proc. of the ACM SIGMOD Intl. Conference on Management of Data (SIGMOD*, pages 103–114, 1996.