UNIVERSITY OF CALIFORNIA
RIVERSIDE

Revisiting Conventional Assumptions in Static and Dynamic Tensor Mining

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Ravdeep Singh Pasricha

September 2022

Dissertation Committee:

Dr. Evangelos E. Papalexakis, Chairperson
Dr. Eamonn Keogh
Dr. Michalis Faloutsos
Dr. Tamar Shinar

The Dissertation of Ravdeep Singh Pasricha is approved:

Committee Chairperson

University of California, Riverside

Acknowledgments

First and foremost I would like to thank my advisor Dr. Evangelos (Vagelis) Papalexakis for his support and guidance, without which this work wouldn't have been possible. I would also like to thank other members of the committee Dr. Michalis faloutsos, Dr. Tamar Shinar and Dr. Eamonn Keogh for being part of my dissertation thesis committee and providing valuable comments and suggestions on how to improve my work. I would also like to thank Dr. Vassilis Tsotras, who provide valuable comments which sparked the inspiration for greedy algorithm for adaptive granularity in tensors work. A huge thanks to all my collaborators for their valuable insights and suggestions.

Next I would like to thank my friends who were always there for me and supported me throughout my PhD journey. Thank you, Ishu Kataria, Payas Rajan, Abhishek Srivastava, Jason Ott, Shelly Ott, Saheli Ghosh, Bashar Romanous, Ekta Gujral, Uday Saini, Pravallika Devineni, Yorgos Tsitsikas, and Kiran Ranganath.

Finally I would like to thank my parents and family for their support and letting me pursue my dreams.

For my loving grandmother.

ABSTRACT OF THE DISSERTATION

Revisiting Conventional Assumptions in Static and Dynamic Tensor Mining

by

Ravdeep Singh Pasricha

Doctor of Philosophy, Graduate Program in Computer Science
University of California, Riverside, September 2022
Dr. Evangelos E. Papalexakis, Chairperson

Tensor methods have been used successfully in modeling multi-aspect data and finding useful latent factors in various applications. These applications range from finding meaningful communities in social networks, detecting fake news, brain data analysis (EEG) and countless other applications in various domains like Chemometrics, Psychometrics, and Signal Processing.

Despite the success of tensor methods in a wide variety of problems, the application of tensor methods typically entails a number of assumptions which, even though they may pertain only to a limited set of applications or to certain algorithms, are so pervasive that they are considered conventional. However, those assumptions may not be generalizable and, in fact, may hurt performance of tensor methods broadly. The main motivation behind this thesis is to revisit some of those conventional assumptions, propose methods to tackle problems arising from relaxing those assumptions, and observing the subsequent advantages of doing so.

Below is the list of works covered in this thesis. Firstly, we focus on a domain specific problem, in which we create a richer feature space for hyperspectral pixel classifications. The next three projects focus on time-evolving graphs, specifically how latent factors evolve over time in streaming tensor decomposition and adaptive granularity in multi-aspect temporal data.

- **Feature Space Explosion**: In this work, we used tensor factorization to generate a richer feature space for pixel classification in hyperspectral images. We propose a feature explosion technique which maps the input space to a higher dimensional space, which is contrary to the typical low-rank factorization to a low-dimensional space. We propose an algorithm called ORION, which exploits the multi-linear structure of the 3-D hyperspectral tensor using tensor decomposition and generates a space which is more expressive than the original input space. Effectiveness of our method was demonstrated against traditional linear and non-linear supervised learning methods such as SVM with kernels, and the Multi-Layer Perceptron model.

- **Concept Drift in Streaming Tensor Decomposition**: Streaming tensor decomposition is usually performed on incoming data in batches to find latent factors and update the existing decomposition results. The number of latent factors are considered to be fixed over time, which is not the case in most scenarios. The number of latent factors might change from batch to batch. For example in time-evolving social graphs, communities don't remain static; they evolve over time. Some communities disappear and some new communities are formed. In this work and to the best of our knowledge, we were the first to introduce the notion of latent concept drift in the

context of streaming tensor decomposition and propose an algorithm called SeekAndDestroy which detects concept drift, discovers new latent factors and updates the existing decomposition result.

- **Greedy Algorithm for Adaptive Granularity:** Time evolving graphs are usually modelled as tensors where each time unit is represented as an adjacency matrix and then whole data over a certain time interval can be modelled as a 3-D tensor. In such scenarios, granularity of collected data can decide on the utility of tensor decomposition algorithms. If data is collected at very frequent intervals, the resulting tensor might be extremely sparse and hence not amenable to tensor decompositions. Usually practitioners combine data points to form fixed time-intervals. For instance, an application may combine milliseconds to form seconds, or seconds to form a minute etc. But there might exist a natural aggregation which can provide important insights in the form of latent factors. In this work, we introduce the problem of temporal aggregation in tensors called *Trapped Under Ice* and we provide a greedy solution called IceBreaker, which locally maximizes some quality function and finds a tensor with meaningful structure.

- **Factorization-based Granularity Estimation:** Lastly we introduce a method called Harvester, in which the problem of aforementioned Adaptive Granularity is framed in terms of a factorization-based optimization problem. To the best of our knowledge, Harvester is the first principled factorization-based approach which seeks to identify the best temporal granularity of a given tensor. Unlike IceBreaker which follows a greedy approach, Harvester leverages multiple aggregated views of

the tensor, and a carefully-designed optimization problem, in order to uncover the best reasonable aggregation. We extensively evaluated HARVESTER on synthetic, semi-synthetic and real datasets, and demonstrated that it consistently produces tensors of very high quality.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Many real world applications deal with data that are multi-dimensional in nature. An example of multi-dimensional data can be interactions between different users in a social network over a period of time. Interactions such as "who messages whom", "who liked whose posts" or "who shared (re-tweeted) whose posts". This can be modeled as a three-mode tensor. Tensors and tensor decomposition algorithms has been successfully used in various applications like social networks [66, 63, 9], detecting misinformation [33], neuron activity analysis [88], chemometrics [7, 6, 20], signal processing [60, 80], health care [91, 39] recommendation system [68], EEG data [58] and many more to extract meaningful latent factors.

Tensor decomposition is an important tool in the unsupervised learning toolbox [44, 65, 79, 26]. The focus of thesis is to revisit and re-examine some of the conventional assumptions in the literature. These assumptions are not always generalizable, even though they might provide great results for certain applications in a domain. In particular, we use

the following three problems as vehicles for understanding the limitations of conventional assumptions and providing alternative solutions: (a) Hyperspectral pixel classification, (b) Streaming tensor decomposition, and (c) Adaptive granualrity in tensors

Hyperspectral pixel classification The first problem we tackle in this thesis is hyperspectral pixel classification, where the image of an object is captured in hundreds of spectral bands. In particular we focus on hyperpspectral images of a land area. These images in multiple spectral band can be modelled as 3-D tensor. We developed a novel framework that uses tensor factorization to generate richer feature spaces for pixel classification in hyperspectral images. We proposed the ORION algorithm [73] which takes as input a hyperspectral image tensor and a rank, and outputs an enhanced feature space from the factor matrices of the decomposed tensor exploiting the multi-linear structure of the tensor. Our method is a feature explosion technique that maps low dimensional input space in $\mathbb{R}^K$ to high dimensional space in $\mathbb{R}^R$, where $R \gg K$, like a kernel, *which is contrary to dimensional reduction techniques like low rank factorization.*

Streaming tensor decomposition The second problem we tackle in thesis is that of streaming tensor decomposition. Many real-world applications are dynamic in nature. To deal with this dynamic nature of data, there exist a variety of online tensor decomposition algorithms. *A central assumption in all such algorithms is that the number of latent concepts remains fixed throughout the entire stream.* However, this need not be the case. Every incoming batch in the stream may have a different number of latent concepts, and the difference in latent concepts from one tensor batch to another can provide insights into how a particular application behaves and deviates over time. In this work, we define "concept"

and "concept drift" in the context of streaming tensor decomposition as the manifestation of the variability of latent concepts throughout the stream. Furthermore, we introduce SeekAndDestroy [71] an algorithm that detects concept drift in streaming tensor decomposition and is able to produce results robust to that drift. To the best of our knowledge, this is the first work that investigates concept drift in streaming tensor decomposition.

Adaptive granualrity in tensors The third problem we tackle in this thesis is that of adaptive granularity in the temporal mode of tensor. *We introduce the problem of finding a tensor of adaptive aggregated granularity that can be decomposed to reveal meaningful latent concepts from datasets that, in their original form, are not amenable to tensor analysis.* Such datasets fall under the broad category of sparse point processes that evolve over space and/or time. We call this problem *Trapped Under Ice*. We provide two solutions to this problem.

(a) *Greedy Algorithm for Adaptive Granularity*: To the best of our knowledge, this is the first work that explores adaptive granularity aggregation in tensors. We developed an efficient and effective greedy algorithm called IceBreaker [72] which follows a number of intuitive decision criteria that locally maximize the "goodness of structure", resulting in high-quality tensors, hence making datasets amenable to tensor analysis. We also developed an recursive algorithm IceBreaker++ which takes output of IceBreaker as input, until the temporal dimension is only a single slice (a matrix) or stopping condition is met. Hence providing multiple tensors of different resolution which can provide useful insights into the data.

(b) *Factorization-Based Granularity Estimation*: In this work we provide a new framework to solve the *Trapped Under Ice* problem. We introduced an factorization based approach called HARVESTER which take multiple aggregated views of the tensor as input and uses those aggregated views to solve an optimization based problem to find a tensor of "good" aggregation which is further more aggregated then the views used. Hence providing an tensor of resolution which better latent factors then the original tensor and the views.

1.1 Thesis Organization

The rest of the thesis is organized as follows. We introduce some of the preliminary definitions on tensors and tensor decomposition required for the thesis in Chapter 2. In Chapter 3, we present our work on Tensorized Feature Spaces for Hyperspectral Pixel Classification. Our work on detecting and alleviating concept drift in streaming tensor decomposition is discussed in Chapter 4. In Chapter 5, we present our work on greed algorithm for adaptive granularity in tensors. In Chapter 6, we present the factorization-based solution to adaptive granularity in tensor problem. Finally, we conclude the thesis in Chapter 7.

Chapter 2

Preliminaries

In this chapter, we present some of the preliminary definitions and concepts which are used in this thesis. These definitions are required to define various problems and their solutions in upcoming chapters in the thesis. Table 2.1 describes the symbols used and their descriptions.

2.1 Preliminary Definitions

Tensor: Tensors are multi-dimensional arrays and are used to model multi-aspect data. Each dimension of a tensor is called a *mode*. For example, a 1-mode tensor is a vector, and a 2-mode tensor is a matrix. A 3-mode tensor is represented by $\underline{\mathbf{X}}$, where $\underline{\mathbf{X}} \in \mathbb{R}^{I \times J \times K}$.

Fibers: The row and column vectors are fibers in a matrix. Given an n-mode tensor, we obtain fibers by fixing the $n-1$ indices. For example, in a 3-mode tensor $\underline{\mathbf{X}}$ with indices I, J, K, $\underline{\mathbf{X}}(:, J, K), \underline{\mathbf{X}}(I, :, K)$ and $\underline{\mathbf{X}}(I, J, :)$ are considered mode-1, mode-2 and mode-3 fibers respectively.

Table 2.1: Symbols and their description

Symbols	Description
$\underline{\mathbf{X}}, \mathbf{X}, \mathbf{x}, x$	Tensor, matrix, column vector, scalar
$\mathbb{R}$	Set of Real Numbers
$\circ$	Outer product
$\underline{\mathbf{X}}(I, J, :)$	Spanning all elements in the 3rd-mode of $\underline{\mathbf{X}}$
$\otimes$	Kronecker product
$\odot$	Khatri-Rao product
$\circledast$	Element wise product
$\|\mathbf{A}\|_F, \|\mathbf{a}\|_2$	Frobenius norm, ℓ_2 norm
$\mathbf{X}(:, r)$	r^{th} column of $\mathbf{X}$

Tensor Slice: A tensor slice for a 3-mode tensor $\underline{\mathbf{X}} \in \mathbb{R}^{I \times J \times K}$ is derived by fixing indices in one mode and varying indices for two other mode. The different various type of slices one can have for 3-tensor are horizontal $\underline{\mathbf{X}}$(I,:,:) , lateral $\underline{\mathbf{X}}$(:,J,:), and frontal $\underline{\mathbf{X}}$(:,:,K) slices.

Tensor Batch: A batch is a (N-1)-mode partition of tensor $\underline{\mathbf{X}} \in \mathbb{R}^{I \times J \times K}$ where size is varied only in one mode and other modes remain unchanged. Here, tensor $\underline{\mathbf{X}}_{new}$ is of dimension $\mathbb{R}^{I \times J \times t_{new}}$ and existing tensor $\underline{\mathbf{X}}_{old}$ is of dimension $\mathbb{R}^{I \times J \times t_{old}}$. The full tensor $\underline{\mathbf{X}} = [\underline{\mathbf{X}}_{old}; \underline{\mathbf{X}}_{new}]$ where its temporal mode $K = t_{old} + t_{new}$.

Outer Product: The outer product of two vectors $\mathbf{a}$ and $\mathbf{b}$ of dimension of $I \times 1$ and $J \times 1$ respectively is given by:

$$\mathbf{a} \circ \mathbf{b} = \mathbf{a}\mathbf{b}^T$$

The resulting $\mathbf{a}\mathbf{b}^T$ is a matrix of size $I \times J$. This definition can be extended to n number of vectors to generalize for n-way outer product.

Kronecker Product: The Kronecker product of two matrices $\mathbf{A} \in \mathbb{R}^{I \times J}$ and $\mathbf{B} \in \mathbb{R}^{M \times N}$ is given by $\mathbf{A} \otimes \mathbf{B} \in \mathbb{R}^{IM \times JN}$

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & a_{12}\mathbf{B} \dots a_{1J}\mathbf{B} \\ a_{21}\mathbf{B} & a_{22}\mathbf{B} \dots a_{2J}\mathbf{B} \\ \vdots & \vdots & \ddots \\ a_{I1}\mathbf{B} & a_{I2}\mathbf{B} \dots a_{IJ}\mathbf{B} \end{bmatrix}$$

where a_{ij} refers to elements in matrix $\mathbf{A}$.

Khatri-Rao Product: Khatri-Rao product of two matrices $\mathbf{A} \in \mathbb{R}^{I \times R}$ and $\mathbf{B} \in \mathbb{R}^{J \times R}$ is the column-wise Kronecker product given by $\mathbf{A} \odot \mathbf{B} \in \mathbb{R}^{IJ \times R}$.

$$\mathbf{A} \odot \mathbf{B} = [\mathbf{a_1} \otimes \mathbf{b_1} \quad \mathbf{a_2} \otimes \mathbf{b_2} \quad \dots \quad \mathbf{a_R} \otimes \mathbf{b_R}]$$

where $[\mathbf{a_1}, \mathbf{a_2} \dots, \mathbf{a_R}]$ and $[\mathbf{b_1}, \mathbf{b_2} \dots, \mathbf{b_R}]$ are columns of $\mathbf{A}$ and $\mathbf{B}$ respectively.

Rank-1 Tensor: A tensor is a rank-1 tensor if it can represented as an n-way outer product of vectors. Figure 2.1 shows a rank-1 tensor for a 3-mode tensor which can be represented as an outer product of three vectors as $\mathbf{a}_1 \circ \mathbf{b}_1 \circ \mathbf{c}_1$

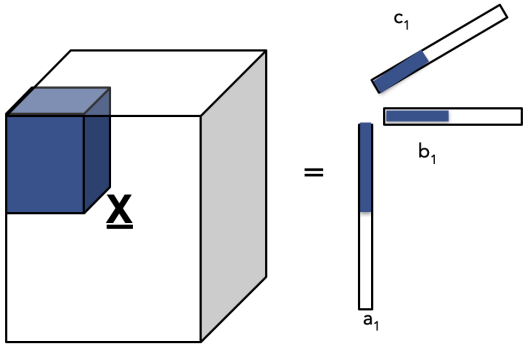


Figure 2.1: 3-mode rank-1 tensor

Low Rank of a Tensor: The rank($\underline{\mathbf{X}}$) is the minimum number of rank-1 tensors computed from its latent components which are required to re-produce $\underline{\mathbf{X}}$ as their sum. Computing the actual rank of a tensor is a hard problem [36, 38].

Matricization: A tensor can be unfolded or flattened into one of its modes to form a matrix. For example, a 3-mode tensor $\underline{\mathbf{X}} \in \mathbb{R}^{I \times J \times K}$ can be matricized in three ways: $\mathbf{X_1} \in \mathbb{R}^{I \times JK}$, $\mathbf{X_2} \in \mathbb{R}^{J \times IK}$ and $\mathbf{X_3} \in \mathbb{R}^{K \times IJ}$, where $\mathbf{X_n}$ represents matricization in n^{th}-mode.

The n-Mode product: The n-mode product of a tensor [65, 44] $\underline{\mathbf{X}} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_m}$ and matrix $\mathbf{W} \in \mathbb{R}^{I_n \times K}$(predicated on matching dimensions in the n-th mode of the tensor and the rows of the matrix) is given as

$$\underline{\mathbf{Y}} = \underline{\mathbf{X}} \times_n \mathbf{W}$$

where, in the general case:

$$\underline{\mathbf{Y}}(i_1, i_2 \ldots, i_{n-1}, k, i_{n+1}, \ldots, i_m) = \sum_{j=1}^{I_n} \underline{\mathbf{X}}(i_1, \ldots, i_{n-1}, j, i_{n+1}, \ldots, i_m)\mathbf{W}(j, k)$$

and tensor $\underline{\mathbf{Y}} \in \mathbb{R}^{I_1 \times \ldots I_{n-1} \times K \times I_{n+1} \ldots \times I_m}$

For instance, a tensor of size $I \times J \times K$ where $n = 3$ and $\mathbf{W}$ of size $K \times K^*$, the product $\underline{\mathbf{X}} \times_n \mathbf{W}$ multiplies all third mode slices of $\underline{\mathbf{X}}$ with $\mathbf{W}$ and results in a $I \times J \times K^*$ tensor.

Canonical Polyadic Decomposition: The most popular and extensively used tensor decompositions is the Canonical Polyadic or CANDECOMP/PARAFAC decompo-

sition, [24, 35] referred to as CP decomposition henceforth. The CP decomposition of a 3-mode tensor $\underline{\mathbf{X}} \in \mathbb{R}^{I \times J \times K}$ for a particular rank R is given by sum of R rank-one tensors:

$$\underline{\mathbf{X}} \approx \sum_{r=1}^{R} \mathbf{A}(:, r) \circ \mathbf{B}(:, r) \circ \mathbf{C}(:, r)$$

where $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ are factor matrices of size $I \times R$, $J \times R$ and $K \times R$ respectively and $\circ$ represents the three way outer product. The above definition of the CP decomposition is usually interpreted as sum of R rank-1 tensor. In case of an n-mode tensor, it represents an n-way outer product. Figure 2.2 shows a CP decompoition of three mode tensor $\underline{\mathbf{X}}$ as sum of R rank-1 tensors.

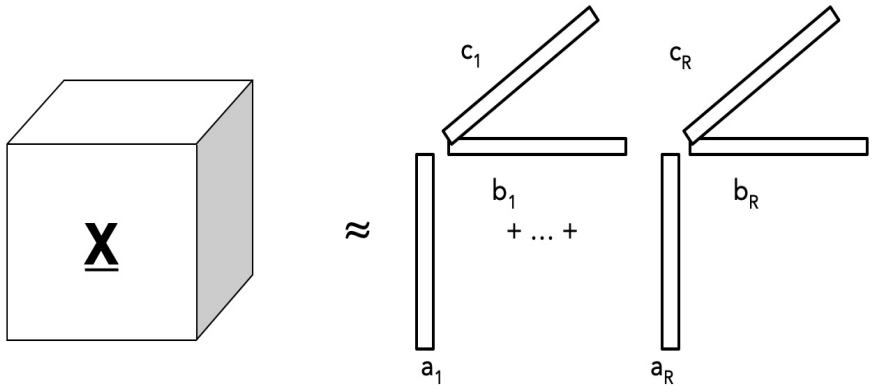


Figure 2.2: CP Decomposition of 3-mode tensor

CP decomposition is essentially unique under mild conditions – which is much different from matrix rank decomposition [78]. For fitting the decomposition, in this thesis we adopt existing work that uses least squares approximation, which is by far the most popular approach [79, 44, 65], and is minimizing the loss $\mathcal{L} \approx \min \frac{1}{2}||\underline{\mathbf{X}} - \mathbf{A}(\mathbf{C} \odot \mathbf{B})^T||_F^2$ where $||\underline{\mathbf{X}}||_F^2$ is the sum of squares of its all elements and $||.||_F$ is *Frobenius* (norm).

Tucker Decomposition [85] of a tensor $\underline{\mathbf{X}} \in \mathbb{R}^{I \times J \times K}$ with multi-linear rank R_1, R_2, R_3 is given by

$$\underline{\mathbf{X}} \approx \sum_{p=1}^{R_1} \sum_{q=1}^{R_2} \sum_{r=1}^{R_3} g_{\mathrm{pqr}} a_p \circ b_q \circ c_r = [\![\underline{\mathbf{G}};\ \mathbf{A},\ \mathbf{B},\ \mathbf{C}]\!]$$

where $\underline{\mathbf{G}} \in \mathbb{R}^{R_1 \times R_2 \times R_3}$ is the so-called core tensor that captures the interactions between the different latent factors of $\mathbf{A} \in \mathbb{R}^{I \times R_1}$, $\mathbf{B} \in \mathbb{R}^{J \times R_2}$ and $\mathbf{C} \in \mathbb{R}^{K \times R_3}$. $\mathbf{A}, \mathbf{B}, \mathbf{C}$ are usually taken to be orthonormal. [44]

Tensor Completion: Tensor completion is the task of predicting missing values in a tensor using tensor factorization. Tensor factorization strives to capture the underlying hidden structure even with the case of missing values [3, 53].

We refer interested reader to [44, 65, 79], which present detailed surveys on tensors, tensor decompositions and their applications. This work uses MATLAB format of matrix and vector indexing, for instance, $\mathbf{A}(i,:)$ spans the i-th row of matrix $\mathbf{A}$ whereas $\mathbf{A}(:,j)$ spans the j-th column, and so on.

Chapter 3

Tensorized Feature Spaces for Feature Explosion

This chapter is based on material published in [73].

Conventional Assumption Revisited:

High-rank tensor decompositions are typically not desirable.

Contribution:

High-rank decomposition, exploiting uniqueness properties of tensor decomposition, can provide a more expressive feature space.

3.1 Introduction

Hyperspectral imaging techniques capture images of objects or materials with hundreds of spectral bands at each pixel [30]. A particularly important use of these techniques is in capturing images of land area on the earth's surface from above using an aircraft or a

satellite fitted with sensors. Since objects under observation reflect different wavelengths of the spectral band, each pixel has a large number of features corresponding to the spectral bands. These features have most popularly been used to accomplish two tasks – identify the class of each given pixel, a classification task [23, 25, 55] or, see what that pixel is made of, an unmixing task [43, 18]. Bioucas *et al.* [17] present a survey of problems often encountered in analyzing hyperspectral remote sensing data. In this work, we focus on the pixel classification task, where labelled data for some of the pixels is available.

Hyperspectral images (HSI) can be considered as a set of images stacked together like a 3D cube. For hyperspectral images with high spatial resolution, the pixel classification task assumes that each pixel is 'pure', i.e. it corresponds to a single class. In contrast with the classification task, the unmixing task assumes that each pixel may be composed of multiple materials or 'endmembers' [17]. The main challenges involved in hyperspectral pixel classification are the large number of spectral bands, leading to high dimensionality and the limited availability of labelled data. Previous work considered the feature space generated using kernel functions for HSI classification [23, 22, 25]. These works treat data as a matrix where each row is a pixel in multi-spectral bands. However, there are three challenges in these kernel spaces (a) choice of kernel and its parameters. For example, tuning the parameter γ in Radial Basis Function (RBF) kernel is non-trivial and impacts the performance of the classifier, (b) the number of features generated by the kernel methods is dependent on the number of pixels, i.e. the kernel function $K(\mathbf{X}, \mathbf{X}) \rightarrow \mathbf{F}$ takes the k spectral bands for xy pixels as $\mathbf{X} \in \mathbb{R}^{xy \times k}$ will yield a feature matrix $\mathbf{F} \in \mathbb{R}^{xy \times xy}$, and (c)

these kernel spaces assume that the pixels are independent and identically distributed (IID) samples and ignore the spatial correlation that exists between the pixels.

In this work we address these challenges by exploring a new feature explosion method called ORION that uses tensor completion to generate a richer feature space by exploiting the multi-dimensional nature of data. ORION allows relaxing the dimension of the obtained feature space $\mathbf{F} \in \mathbb{R}^{xy \times R}$ instead of fixed dimension xy from kernel methods. While we demonstrate the usefulness of ORION for HSI classification, we would like to emphasize that it can be applied to a broader range of problems.

3.2 Problem Definition

This work explores a new feature space using tensor completion and to show its effectiveness, we apply it to hyperspectral pixel classification. Previous literature related to HSI employed methods like feature reduction and kernel methods [22, 25]. In our work, we exploit the multi-linear structure of the hyperspectral image tensor using tensor decomposition to generate a richer space, where the pixels are linearly separable.

More formally we define our problem as follows:

Given a three dimensional hyperspectral image tensor $\underline{\mathbf{X}} \in \mathbb{R}^{I \times J \times K}$, a label matrix $\mathbf{Y} \in \mathbb{R}^{I \times J}$ and rank R, generate a feature space for a classifier such that pixels in the images are classified into one of the given classes.

3.3 Algorithm

In this section, we introduce our method ORION and present the intuition behind it. ORION takes as input a three dimensional tensor $\underline{\mathbf{X}}$ and a tensor rank R and generates a feature space using tensor factorization. The general idea behind the proposed method lies in mapping the input data to some high dimensional space corresponding to the rank decomposition of the tensor.

3.3.1 The Orion algorithm

Algorithm 1 presents the steps involved in ORION, applied to the hyperspectal pixel classification problem. Consider a 3-mode tensor $\underline{\mathbf{X}} \in \mathbb{R}^{I \times J \times K}$, where I and J represent the resolution of the image and K represents the number of spectral bands. For a given test size, we select pixels with non-zero classes using stratified sampling as specified in line 2 of the algorithm 1. We do this in order to ensure that the training and testing data has the same percentage of representation from each class. We mark all spectral values of test pixels as zero, i.e. all the third mode fibers of the test data points are marked as zero, treating the problem of filling missing values as a tensor completion task. We employ the tensor completion algorithm CP-WOPT (Weighted Optimization) [3], implemented in [12] to predict the missing values. This produces three factor matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$. The first two matrices $\mathbf{A}$ and $\mathbf{B}$ correspond to the two modes of the image are used to generate a new feature space.

$$\mathbf{data} = \mathbf{A} \odot \mathbf{B}$$

where $\odot$ represents Khatri-Rao product. To scale up the values, we multiply **data** with diagonalized $\boldsymbol{\lambda}$ matrix as shown in line 6 in algorithm 1. We use initial training and testing indices to generate training and testing data respectively, removing all the data points with class value as 0. Using the newly created feature space, we now train a linear Support Vector Machine (SVM) with 5-fold cross validation.

Algorithm 1 ORION

Input: A 3-mode tensor $\underline{\mathbf{X}}$, a label matrix $\mathbf{Y}$, rank r and testSize

Output: A vector of predicted classes

1: Extract pixel indices $[I, J]$ of all the non-zero classes.

2: Split $[I, J]$ into training and testing data in a stratified fashion.

3: Create tensor $\underline{\mathbf{P}}$ of ones with same dimensions as $\underline{\mathbf{X}}$

4: $\underline{\mathbf{P}}\ [testI, testJ, :] = 0$

{% Tensor completion problem}

5: $\mathbf{A}, \mathbf{B}, \mathbf{C}, \boldsymbol{\lambda} = CP\_WOPT(\underline{\mathbf{X}}, \underline{\mathbf{P}}, Rank)$ [3]

6: $\mathbf{data} = (A \odot B) * diag(\boldsymbol{\lambda})$

7: Using indices in Step 2, split **data** into training and testing data

8: Train a linear SVM using training data with 5-fold cross validation

9: Run the model against testing data

10: **return** model predictions

3.3.2 Intuition Behind Orion

The idea behind ORION is to map the input space to higher dimensional space by exploiting multi-linear structure of the tensor. Consider a 3-mode tensor, $\underline{\mathbf{X}} \in \mathbb{R}^{I \times J \times K}$. The CP decomposition with rank R of $\underline{\mathbf{X}}$ yields three factor matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ of size $I \times R$, $J \times R$ & $K \times R$ respectively. The Khatri-Rao product of matrices $\mathbf{A}$ and $\mathbf{B}$ generates the new data space in $\mathbb{R}^{IJ \times R}$. Whereas unfolding of tensor $\underline{\mathbf{X}}$ in third mode would generate data space in $\mathbb{R}^{IJ \times K}$. Since $K \ll IJ$, the matrix rank is bounded by K. However, the upper bound on tensor rank for which CP can still uniquely identify the components within the tensor is $\min(IJ, JK, KI)$ [79, 78], which is considerably larger. Thus, by using a large-enough rank, by virtue of CP's uniqueness [79, 78], we are able to extract a feature space that is more expressive than simple unfolding of the features (or any spectral method in that unfolded matrix).

3.4 Experimental Evaluation

In this section, we describe our experimental setup and present our results. We use Tensor Toolbox [12] in Matlab for our tensor completion task, CP-WOPT [3] is implemented in this toolbox. For classification algorithms we use Python Scikit-Learn [74] and tensorly [47] for tensor operations. In the interest of reproducibility, the implementation of our algorithm and baselines used is publicly available[1].

[1] https://github.com/ravdeep003/ORION

3.4.1 Datasets

To evaluate our method, we use the following publicly available datasets [28].

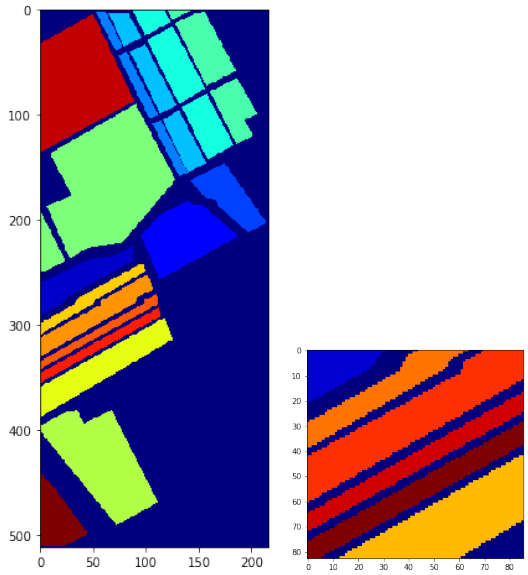
- **Indian Pines:** This dataset was acquired using the AVIRIS[2] sensor [14] and consists of 145×145 pixels and 200 spectral bands. This dataset consist of 10249 labelled pixels spanning over 16. There is high class imbalance in this dataset.

- **University of Pavia:** This dataset was collected using ROSIS sensor over Pavia in Northern Italy. The original image resolution of the dataset is 610×610 but most of the image didn't contain any information so the image resolution is reduced to 610×340 over 103 spectral bands. This dataset consist of 42776 labelled pixels spanning over 9 classes

- **Salinas:** This dataset was gathered using AVIRIS sensor over Salinas Valley, California. The dataset consists is of 512×217 resolution over 204 spectral bands. It has 54219 non-zero pixels, labelled with 16 classes. Figure 3.1(a) shows the ground truth of Salinas dataset.

- **Salinas-A**: This dataset represents a subscene in the Salinas dataset. It consists of 86×83 pixels and 6 classes. Number of nonzero pixels in this dataset are 5348. Figure 3.1(b) presents the ground truth of Salinas-A dataset.

[2] https://aviris.jpl.nasa.gov/



(a) Salinas Full. (b) Salinas Subscene.

Figure 3.1: Ground truth of Salinas and Salinas-A HSI datasets.

3.4.2 Baseline Methods

We evaluate our proposed method ORION against traditional linear and non-linear methods like the Kernel SVM and Multi-Layer Perceptron, and employ grid search to tune the hyperparameters in these methods.

Support Vector Machines (SVMs)

Support vector machines (SVMs) [27] are supervised learning methods that are powerful classifiers, specially when combined with kernels. SVMs discriminate between

between data points by finding a hyperplane which maximizes the margin. Given a dataset (x_i, y_i), where x_i are data points and y_i are labels,

$$\min_{\mathbf{w},b,\xi} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_i \xi_i \tag{3.1}$$

subject to $y_i(\mathbf{w}^T\phi(x_i) + b) \geq 1 - \xi_i$ and $\xi_i \geq 0$ for all $i = 1 \ldots n$

where $\mathbf{w}$ is a normal to the hyperplane separating the data points, C is a penalty term for misclassification, ξ is slack variables, ϕ is mapping from input space to kernel space and b is the bias or the offset for the hyperplane. Equation 3.1 is the primal form and it has dual form:

$$\min_{\alpha_1,\ldots\alpha_n} \frac{1}{2}\sum_{i,j} \alpha_i\alpha_j y_i y_j K_{ij} - \sum_i^n \alpha_i \tag{3.2}$$

such that $0 \leq \alpha_i \leq C$ and $\sum_i^n \alpha_i y_i = 0$

where α_i are Lagrange multipliers and K_{ij} is inner product of data points in kernel space. We use the following kernels in our experiments:

- Linear Kernel: We tune the parameter **C** in the objective function. We do a grid search from 10^{-3} to 10^3 in multiples of 10.

$$K(x_i, x_j) = {x_i}^T x_j$$

- Polynomial Kernel: We tune the parameters **C** and **degree** of the polynomial. **C** is tuned in the range 10^{-3} to 10^3 and **degree** is tuned in the range from 2 to 5. We perform a grid search to tune these parameters.

$$K(x_i, x_j) = ({x_i}^T x_j + 1)^d$$

where d is the degree of the polynomial.

- RBF Kernel: We tune the parameters **C** and γ. C is tuned from the same range as before and γ is tuned from 10^{-3} to 10 in multiples of 10.

$$K(x_i, x_j) = \exp\left(-\gamma||x_i - x_j||^2\right)$$

Multi-Layer Perceptron (MLP)

We use a Multi Layer Perceptron model (also known as artificial neural network) as one of our baselines. Various parameters involved in training the model were tuned using grid search. These parameters include hidden layer sizes $[(50, 100, 50), (100, 100, 100), (150, 100, 150)]$, alpha - $L2$ regularization term $[10^{-4}, 10^{-3}, 10^{-2}]$, initial learning rate $[10^{-4}, 10^{-3}, 10^{-2}]$ and learning rate (constant or adaptive). We used ReLu (rectified linear unit) activation function, Adam optimization solver, and an iteration count of 500.

All of the models were trained with 5-fold cross validation on training data and using weighted $F1$ as scoring function for 10 different runs. We chose the weighted $F1$ as our scoring function since the datasets are multi-class and have imbalanced classes. To evaluate the performance of different models, we use overall accuracy and $F1$ score of the classifier against test data. We also employ one against one scheme for multi-class classifiers.

3.4.3 Results

We performed an $80 - 20$ stratified split on the datasets, where 80% of the data was used for training and the rest 20% for testing, and ran experiments using ORION and baseline methods. Table 3.1 presents the results of that experiment. In case of Indian Pines

and Salinas, we see that ORION with rank 1000 and 2000 performs better than the baselines. In case of University of Pavia, ORION with 2000 rank performs better than all baselines, with the Multi-Layer Perceptron being a close second. We discuss the results of Salinas-A in subsection 3.4.4.

One of the challenges in HSI classification is that the amount of labelled data available is limited. To demonstrate this, we run all experiments using 30-70 stratified split, where 30% is training data and 70% is testing data. For the most part, our results remain similar as shown in table 3.2. For Indian Pines and Salinas datasets, ORION with 1000 and 2000 rank provides better overall classification accuracy. In case of University of Pavia, two best performing methods are SVM with RBF kernel and MLP, but ORION with rank 2000 performs at par with the baseline and has similar classification accuracy. This depicts that ORION is effective even with limited labelled data. For both of the scenarios (80-20 and 30-70 split), tables 3.3 and 3.4 report mean $F1$ scores of our method and baselines. They follow the same trend as the overall accuracy.

We explore the effect of rank on the overall accuracy for Indian Pines and Salinas-A datasets. Figure 3.2 shows the plot of mean accuracy vs. rank for Indian Pines dataset. We observed that as the rank increases, classification accuracy improves as well until a certain point, where the change in rank does not provide any significant improvements and the accuracy stabilizes.

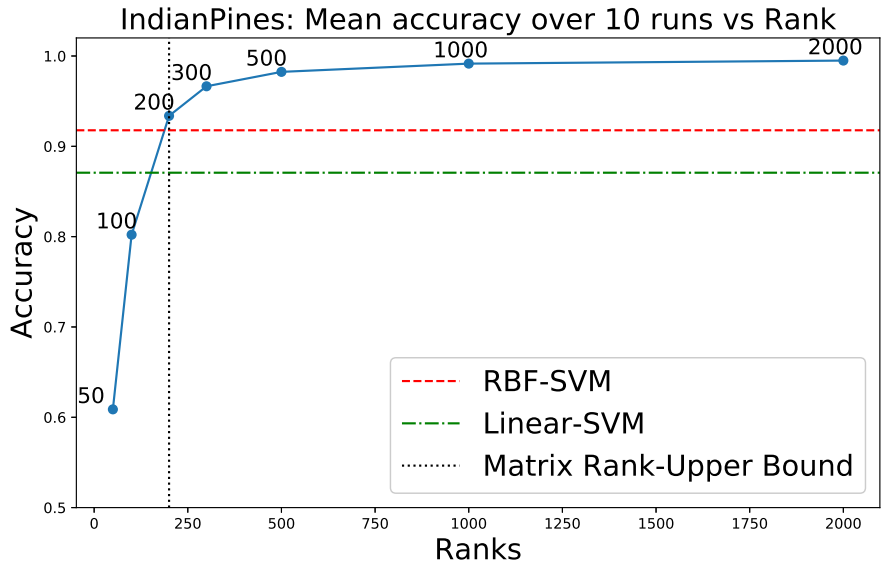


Figure 3.2: Mean accuracy vs. Rank of the tensor observed over 10 runs. As rank increases, the classification accuracy increases and stabilizes after a certain point.

3.4.4 Discussion about Salinas-A and Salinas

The Salinas-A (Figure 3.1(b)) dataset has 5348 labelled pixels with 6 classes, and is a subscene of the full Salinas (Figure 3.1)(a) dataset which has 54129 labelled pixels and 16 classes. For Salinas-A, all baselines outperform ORION with rank 1000 and 2000, however, in the case of Salinas, ORION with same ranks outperforms all the baselines. This trend is similar in both 80-20 and 30-70 splits. Upon visual inspection, Salinas-A appears linearly separable whereas Salinas is not. We conjecture that Salinas appears to have more concrete and uniform blocks, potentially better trilinear structure that is exploited via CP decomposition. Judging the trilinearity of a dataset is a difficult problem and while there exist heuristics for this [21], we reserve further investigation for our future work.

3.5 Related Work

Hyperspectral image classification takes as input a set of observations and assigns a unique label to each pixel [18]. Supervised linear methods in HSI classification are prone to the curse of dimensionality due to the lack of large number of training samples [40]. Support vector machines (SVM) have been employed to deal with this phenomenon [22]. SVMs allow classification of data points in a higher dimensional space using a nonlinear transformation.

Tensor methods like CP decomposition [55, 42] have been used to represent high-order feature data in low-dimensional space with good accuracy. [56] presented a deep learning-based classification method that hierarchically constructs high-level features automatically. In particular, their model exploits a convolutional neural network (CNN) to encode the spectral and spatial information of pixels and a multilayer perceptron to conduct the classification task. [59] use recurrent neural network (RNN) to characterize the sequential property of a hyperspectral pixel vector for the classification task. Our proposed method ORION employs factors obtained from tensor factorization to generate a feature space that maps the given feature space to a higher-dimensional space in order to improve classification accuracy.

3.6 Contributions

Our contributions in this work are as follows:

- **Tensorized Feature Space:** We introduce a new feature space based on factors generated using tensor factorization. This works better or on par with traditional

state-of-the-art classification methods. To the best of our knowledge, this is the first work that presents a formal study of feature space explosion with defined number of features R, unlike kernel methods that fix the length of the dimension for the number of available samples.

- **Experimental Evaluation:** We demonstrate the effectiveness of our proposed method ORION by evaluating it on publicly available hyperspectral datasets and compare it against traditional state-of-the-art baselines, linear and nonlinear supervised learning methods like Linear, Polynomial and RBF Support Vector Machines, and Multi-Layer Perceptrons.

	Indian Pines	Pavia University	Salinas-A	Salinas
Linear SVM	0.8708± 0.0035	0.9176 ± 0.0017	**0.9986 ± 0.0016**	0.9339 ± 0.0014
Polynomial SVM	0.8979± 0.0054	0.9481 ± 0.0015	0.9978 ± 0.0015	0.9463 ± 0.0014
RBF SVM	0.9178± 0.0050	0.9622 ± 0.0020	**0.9985 ± 0.0017**	0.9620 ± 0.0024
MLP	0.9182± 0.0057	**0.9635 ± 0.0041**	0.9982 ± 0.0010	0.9629 ± 0.0045
ORION-1000	**0.9916± 0.0022**	0.9502 ± 0.0032	0.9690 ± 0.0067	**0.9927 ± 0.0010**
ORION-2000	**0.9949± 0.0022**	**0.9828 ± 0.0030**	0.9680 ± 0.0063	**0.9954 ± 0.0006**

Table 3.1: Classification accuracy of all the methods for 80-20 split

	Indian Pines	Pavia University	Salinas-A	Salinas
Linear SVM	0.8371 ± 0.0034	0.9134 ± 0.0015	**0.9965 ± 0.0010**	0.9322 ± 0.0007
Polynomial SVM	0.8511 ± 0.0042	0.9367 ± 0.0010	0.9941 ± 0.0017	0.9406 ± 0.0009
RBF SVM	0.8739 ± 0.0041	**0.9546 ± 0.0007**	**0.9966 ± 0.0011**	0.9515 ± 0.0012
MLP	0.8693 ± 0.0098	**0.9556 ± 0.0029**	0.9931 ± 0.0029	0.9475 ± 0.0041
ORION-1000	**0.9725 ± 0.0032**	0.9119 ± 0.0015	0.8607 ± 0.0146	**0.9662 ± 0.0013**
ORION-2000	**0.9806 ± 0.0031**	0.9544 ± 0.0021	0.8982 ± 0.0073	**0.9832 ± 0.0013**

Table 3.2: Classification accuracy of all the methods for 30-70 split

	Indian Pines	Pavia University	Salinas-A	Salinas
Linear SVM	0.8700 ± 0.0036	0.9162 ± 0.0019	**0.9986 ± 0.0016**	0.9326 ± 0.0016
Polynomial SVM	0.8977 ± 0.0054	0.9477 ± 0.0016	0.9978 ± 0.0015	0.9454 ± 0.0014
RBF SVM	0.9175 ± 0.0050	0.9620 ± 0.0020	**0.9985 ± 0.0017**	0.9620 ± 0.0024
MLP	0.9180 ± 0.0056	**0.9634 ± 0.0040**	0.9982 ± 0.0010	0.9628 ± 0.0045
ORION-1000	**0.9915 ± 0.0022**	0.9484 ± 0.0038	0.9687 ± 0.0068	**0.9927 ± 0.0010**
ORION-2000	**0.9949 ± 0.0022**	**0.9823 ± 0.0032**	0.9675 ± 0.0066	**0.9954 ± 0.0006**

Table 3.3: Mean F1-Score of all the methods for 80-20 split over 10 runs

	Indian Pines	Pavia University	Salinas-A	Salinas
Linear SVM	0.8358 ± 0.0033	0.9118 ± 0.0016	**0.9965 ± 0.0010**	0.9310 ± 0.0006
Polynomial SVM	0.8503 ± 0.0042	0.9361 ± 0.0010	0.9941 ± 0.0017	0.9396 ± 0.0009
RBF SVM	0.8734 ± 0.0041	**0.9544 ± 0.0007**	**0.9966 ± 0.0011**	0.9512 ± 0.0012
MLP	0.8690 ± 0.0095	**0.9555 ± 0.0029**	0.9931 ± 0.0029	0.9469 ± 0.0041
ORION-1000	**0.9725 ± 0.0032**	0.9068 ± 0.0018	0.8583 ± 0.0151	**0.9661 ± 0.0013**
ORION-2000	**0.9804 ± 0.0031**	0.9528 ± 0.0025	0.8961 ± 0.0074	**0.9832 ± 0.0012**

Table 3.4: Mean F1-Score of all the methods for 30-70 split over 10 runs

Chapter 4

Concept Drift in Streaming Tensor Decomposition

This chapter is based on material published in [71].

Conventional Assumption Revisited:

Rank remains constant in streaming tensor decomposition.

Contribution:

Not assuming constant rank gives us qualitatively and quantitatively better results.

4.1 Introduction

In today's world data is not static, data keeps on evolving over time. In real world applications like stock market and e-commerce websites hundred of transaction (if not thousands) takes place every second, or in applications like social media where every second, thousands of new interactions take place forming new communities of users who

interact with each other. In this example, we consider each *community* of people within the graph as a *concept*.

There has been a considerable amount of work in dealing with online or streaming CP decomposition [93, 34, 61], where the goal is to absorb the updates to the tensor in the already computed decomposition, as they arrive, and avoid recomputing the decomposition every time new data arrives. However, despite the already existing work in the literature, a central issue has been left, to the best of our knowledge, entirely unexplored. All of the existing online/streaming tensor decomposition literature assumes that the concepts in the data (whose number is equal to the rank of the decomposition) remains *fixed* throughout the lifetime of the application. What happens if the number of components changes? What if a new component is introduced, or an existing component splits into two or more new components? This is an instance of *concept drift* in unsupervised tensor analysis, and this chapter is a look at this problem from first principles.

4.2 Problem Formulation

Let us consider a social media network like Facebook, where a large number of users ($\approx 684K$) update information every single minute, and Twitter, where about $\approx 100K$ users tweet every minute[1]. Here, we have interactions arriving continuously at high velocity, where each interaction consists of User Id, Tag Ids , Device, and Location information etc. How can we capture such dynamic user interactions? How to identify concepts which can signify a potential newly emerging community, complete disappearance of interactions, or a

[1] https://mashable.com/2012/06/22/data-created-every-minute/

merging of one or more communities to a single one? When using tensors to represent such dynamically evolving data, our problem falls under "streaming" or "online" tensor analysis. Decomposing streaming or online tensors is challenging task, and concept drift in incoming data makes the problem significantly more difficult, especially in applications where we care about characterizing the concepts in the data, in addition to merely approximating the streaming tensor adequately.

Before we conceptualize the problem that this chapter deals with, we define certain terms which are necessary to set up the problem. Consider $\underline{\mathbf{X}}$ and $\underline{\mathbf{Y}}$ be two incremental batches of a streaming tensors of rank R and F respectively. Let $\underline{\mathbf{X}}$ be the initial tensor at time t_0 and $\underline{\mathbf{Y}}$ be the batch of the streaming tensor which arrives at time t_1 such as $t_1 > t_0$. The CP decomposition for these two tensors is given as follows:

$$\underline{\mathbf{X}} \approx \sum_{r=1}^{R} \mathbf{A}(:, r) \circ \mathbf{B}(:, r) \circ \mathbf{C}(:, r) \tag{4.1}$$

$$\underline{\mathbf{Y}} \approx \sum_{r=1}^{F} \mathbf{A}(:, r) \circ \mathbf{B}(:, r) \circ \mathbf{C}(:, r) \tag{4.2}$$

Concept: In case of tensors, we define *concept* as one latent component; a sum of R such components make up the tensor. In above equations tensor $\underline{\mathbf{X}}$ and $\underline{\mathbf{Y}}$ has R and F concepts respectively.

Concept Overlap: We define *concept overlap* as the set of latent concepts that are common or shared between two streaming CP decompositions. Consider Figure 4.1 where R and F both are equal to three, which means both tensors $\underline{\mathbf{X}}$ and $\underline{\mathbf{Y}}$ have three concepts. Each concept of $\underline{\mathbf{X}}$ corresponds to each concept of $\underline{\mathbf{Y}}$. This means that there are three concepts that overlap between $\underline{\mathbf{X}}$ and $\underline{\mathbf{Y}}$. The minimum and maximum number of concept overlaps

between two tensors can be zero and $\min(R, F)$ respectively. Thus, the value of concept overlap lies between 0 and $\min(R, F)$.

In Section 4.3 we propose an algorithm for detecting such overlap.

$$0 \leq \text{Concept Overlap} \leq \min(R, F) \tag{4.3}$$

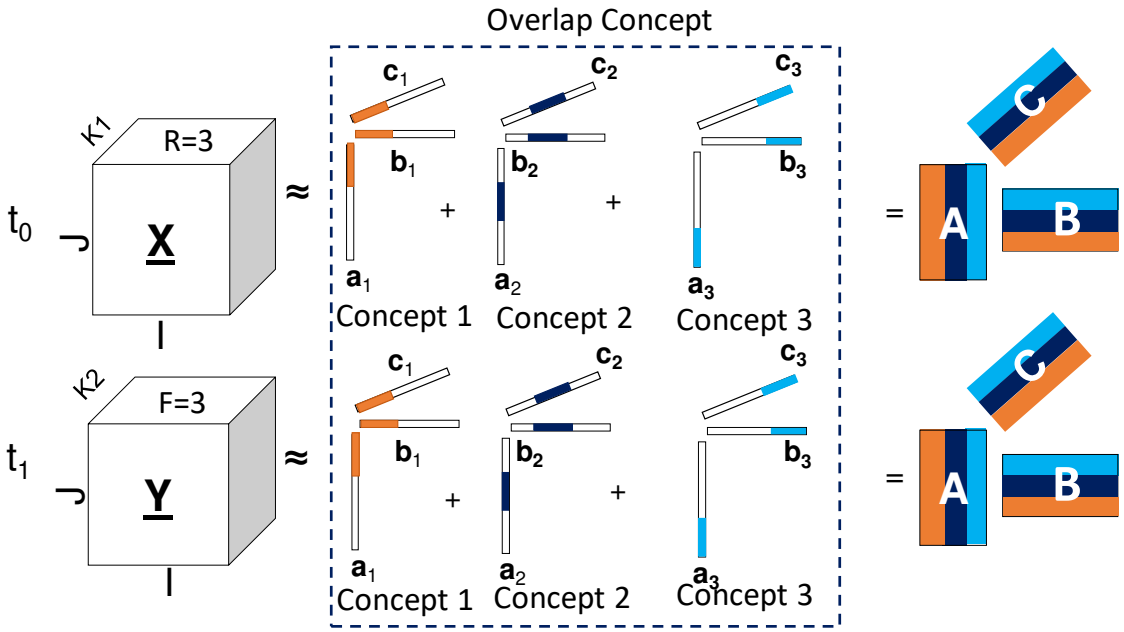


Figure 4.1: Complete overlap of concepts

New Concept: If there exists a set of concepts which are not similar to any of the concepts already present in the most recent tensor batch, we call all such concepts in that set as *new concepts*. Consider Figure 4.2, where $\underline{\mathbf{X}}$ has two concepts ($R = 2$) and $\underline{\mathbf{Y}}$ has three concepts ($F = 3$). We see that at time t_1 tensor $\underline{\mathbf{Y}}$ batch has three concepts, out of which, two

match with tensor $\underline{\mathbf{X}}$ concepts and one concept(namely concept 3) does not match with any concept of $\underline{\mathbf{X}}$. In this scenario we say that concept 1 and 2 are *overlapping concepts* and concept 3 is a *new concept*.

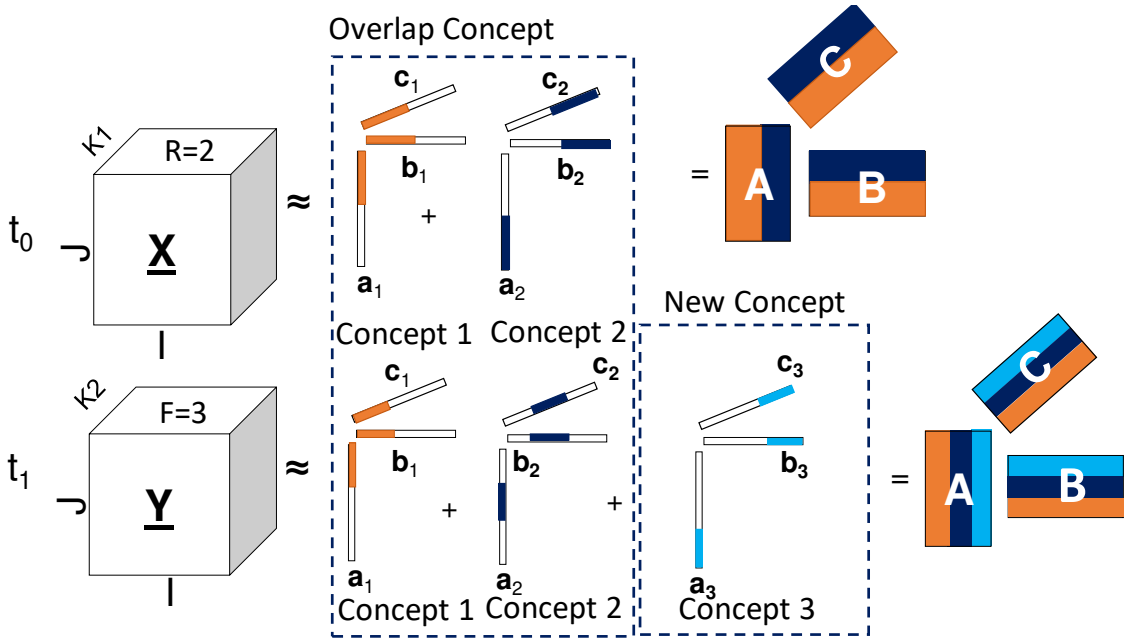


Figure 4.2: Concept appears

Missing Concept: If there exists a set of concepts which was present at time t_0, but was missing at future time t_1, we call the concepts in the set *missing concepts*. For example, consider Figure 4.3, at time t_0, the CP decomposition of $\underline{\mathbf{X}}$ has three concepts, and at time t_1 CP decomposition of $\underline{\mathbf{Y}}$ has two concepts. Two concepts of $\underline{\mathbf{X}}$ and $\underline{\mathbf{Y}}$ match with each other and one concept, present at t_0, is missing at t_1; we label that concept, as *missing concept*.

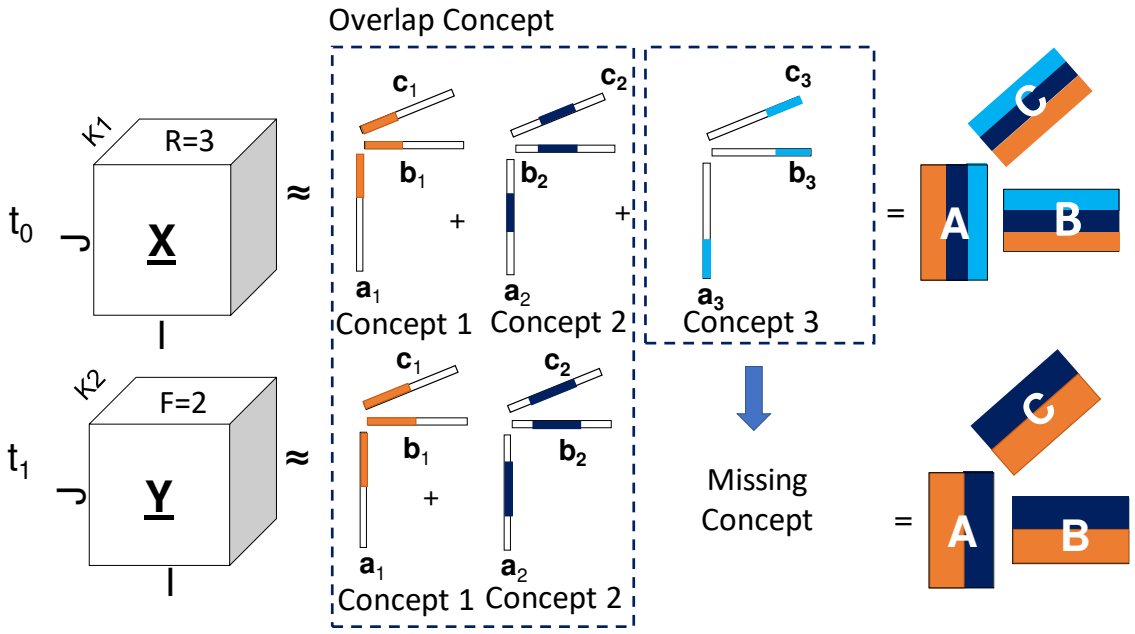


Figure 4.3: Concept disappears

Running Rank: Running Rank (runningRank) at time t is defined as the total number of unique concepts (or latent components) seen until time t. Running Rank is different from tensor rank of a tensor batch. It may or may not be equal to rank of the current tensor batch. Consider Figure 4.1, runningRank at time t_1 is three, since the total unique number of concepts seen until t_1 is three. Similarly runningRank of Figure 4.3 at time t_1 is three, even though rank of $\underline{\mathbf{Y}}$ is two, since the number unique concepts seen until t_1 is three.

Let us assume rank of the initial tensor batch $\underline{\mathbf{X}}$ at time t_0 is R and rank of the subsequent tensor batch $\underline{\mathbf{Y}}$ at time t_1 is F. Then runningRank at time t_1 is sum of running rank at t_0 and number of new concepts discovered from t_0 to t_1. At time t_0 running rank is equal to initial rank of the tensor batch in this case R.

$$\text{runningRank}_{t_1} = \text{runningRank}_{t_0} + num(\text{newConcept})_{t_1 - t_0} \tag{4.4}$$

Concept Drift: Concept drift is usually defined in terms of supervised learning [16, 86, 87, 29]. In [86], authors define concept drift in unsupervised learning as the change in probability distribution of a random variable over time. We define concept drift in the context of latent concepts, which is based on rank of the tensor batch. We first give an intuitive description of concept in terms of running rank, and then define concept drift.

Intuition: Consider running rank at time t_1 be runningRank_{t_1} and running at time t_2 be runningRank_{t_2}. If runningRank_{t_1} is not equal to runningRank_{t_2}, then there is a concept drift i.e. either a new concept has appeared, or a concept has disappeared. However, this definition does not capture every single case. Assume if runningRank_{t_1} is equal to runningRank_{t_2}. In this case, there is no drift only when there is a complete overlap. However there may be concept drift present even if runningRank_{t_1} is equal to runningRank_{t_2}, since a concept might disappear while runningRank remains the same.

Definition: Whenever a new concept appears, a concept disappears, or both from time $t1$ to $t2$, this phenomenon is defined as *concept drift*.

In a streaming tensor application, a tensor batch arrives at regular intervals of time. Before we decompose a tensor batch to get latent concepts, we need to know the rank of the tensor. Finding tensor rank is a hard problem [36, 38] and it is beyond the scope of this work. There has been considerable amount of work which approximates rank of a tensor [62, 57]. In this work we employ AutoTen [62] to compute a low rank of a tensor. As new advances in tensor rank estimation happen, our proposed method will also benefit.

Given (a) tensor $\underline{\mathbf{X}}$ of dimensions $I \times J \times K_1$ and rank R, (b) $\underline{\mathbf{Y}}$ of dimensions $I \times J \times K_2$ of rank F at time t_0 and t_1 respectively as shown in Figure 4.4. Compute $\underline{\mathbf{X}}_{new}$ of dimension $I \times J \times (K_1 + K_2)$ of rank equal to runningRank at time t_1 as shown in equation (5) using factor matrices of $\underline{\mathbf{X}}$ and $\underline{\mathbf{Y}}$.

$$\underline{\mathbf{X}}_{new_{t_1}} \approx \sum_{r=1}^{\text{runningRank}} \mathbf{A}(:, r) \circ \mathbf{B}(:, r) \circ \mathbf{C}(:, r) \tag{4.5}$$

4.3 Method

Consider a social media application where thousands of connections are formed every second, for example, who follows whom or who interacts with whom. These connections formed can be viewed as forming communities. Over a period of time communities disappear, new communities appear or some communities re-appear after sometime. Number of communities at any given point of time is dynamic. There is no way of knowing what communities will appear or disappear in future. When this data stream is captured as a tensor, communities refer to latent concepts and appearing and disappearing of communities over a period of a time is referred to as concept drift. Here we need a dynamic way of figuring out number of communities in a tensor batch rather than assuming constant number of communities in all tensor batches.

To the best of our knowledge, there is no algorithmic approach that detects concept drift in streaming tensor decomposition. As we mentioned in Section 4.1, there has been considerable amount of work [34, 93, 61] which deals with streaming tensor data and applies

batch decomposition on incoming slices and combine the results. But these methods don't take change of rank in consideration, which could reveal new latent concept in the data sets. Even if we know the rank(latent concept) of the complete tensor, the tensor batches of that tensor might not have same rank as the complete tensor.

Here we propose SeekAndDestroy, a streaming CP decomposition algorithm that does not assume rank is fixed. SeekAndDestroy detects the rank of every incoming batch in order to decompose it, and finally, updates the existing decomposition after detecting and alleviating concept drift, as defined in Section 4.2.

An integral part of SeekAndDestroy is detecting different concepts and identifying concept drift in streaming tensor. In order to do this successfully, we need to solve following problems:

P1: Finding the rank of a tensor batch.

P2: Finding New Concept, Concept Overlap and Missing Concept between two consecutive tensor batch decomposition.

P3: Updating the factor matrices to incorporate the new and missing concepts along with concept overlaps.

Finding Number of Latent Concepts: Finding the rank of the tensor is beyond the scope of this work, thus we employ AutoTen [62]. Furthermore, in Section 4.4, we perform our experiments on synthetic data where we know the rank (and use that information as given to us by an "oracle") and repeat those experiments using AutoTen, comparing the error between them; the gap in quality signifies room for improvement that SeekAndDestroy will reap, if rank estimation is solved more accurately in the future.

Finding Concept Overlap: Given a rank of tensor batch, we compute its latent components using CP decomposition. Consider Figure 4.4 as an example. At time t_1, the number of latent concepts we computed is represented by F, and we already had R components before new batch $\underline{\mathbf{Y}}$ arrived. In this scenario, there could be three possible cases: (1) $R = F$ (2) $R > F$ (3) $R < F$.

For each one of the cases mentioned above, there may be new concepts appear at t_1, or concepts disappear from t_0 to t_1, or there could be shared concepts between two decompositions. In Figure 4.4. we see that, even though R is equal to F, we have one new concept, one missing concept and two shared/overlapping concepts. Now, at time t_1, we have four unique concepts, which means our runningRank at t_1 is four.

In order to discover which concepts are shared, new, or missing we use the *Cauchy-Schwarz inequality* which states for two vectors $\mathbf{a}$ and $\mathbf{b}$ we have $\mathbf{a}^T\mathbf{b} \leq ||\mathbf{a}||_2||\mathbf{b}||_2$. Algorithm 3 provides the general outline of technique used in finding concepts. It takes a column-normalized matrices $\mathbf{A_{old}}$ and $\mathbf{A_{batch}}$ of size $I \times R$ and $I \times batchRank$ respectively as input. We compute the dot product for all permutations of columns between two matrices, as shown below

$$\mathbf{A}^T_{\mathbf{old}}(:, col_i) \cdot \mathbf{A_{batch}}(:, col_j)$$

col_i and col_j are the respective columns. If the computed dot product is higher than the threshold value, the two concepts match, and we consider them as shared/overlapping between $\mathbf{A_{old}}$ and $\mathbf{A_{batch}}$. If the dot product between a column in $\mathbf{A_{batch}}$ and with all the columns in $\mathbf{A_{old}}$ has a value less than the threshold, we consider it as a new concept.

Algorithm 2 SeekAndDestroy for Detecting & Alleviating Concept Drift

Require: Tensor $\underline{\mathbf{X}}_{new}$ of size $I \times J \times K_{new}$,

Factor matrices $\mathbf{A}_{old}, \mathbf{B}_{old}, \mathbf{C}_{old}$ of size $I \times R$, $J \times R$ and $K_{old} \times R$ respectively,

runningRank, mode.

Ensure: $\mathbf{A}_{updated}, \mathbf{B}_{updated}, \mathbf{C}_{updated}$ of size $I \times$ runningRank, $J \times$ runningRank & $(K_{new} + K_{old}) \times$ runningRank, $\boldsymbol{\rho}$, runningRank.

$batchRank \leftarrow getRankAutoten(\underline{\mathbf{X}}_{new}, runningRank)$

$[\mathbf{A}, \mathbf{B}, \mathbf{C}, \boldsymbol{\lambda}] =$ CP$(\underline{\mathbf{X}}_{new}, batchRank)$.

colA, **colB**, **colC** $\leftarrow$ Compute Column Normalization of $\mathbf{A}, \mathbf{B}, \mathbf{C}$.

normMatA, **normMatB**, **normMatC** $\leftarrow$ Absorb $\boldsymbol{\lambda}$ and Normalize $\mathbf{A}, \mathbf{B}, \mathbf{C}$.

$rhoVal \leftarrow colA .* colB .* colC$

[newConcept, conceptOverlap, $overlapConceptOld$] $\leftarrow findConceptOverlap(\mathbf{A}_{old}$, **normMatA**$)$

if newConcept **then**

  runningRank $\leftarrow$ runningRank $+ len$(newConcept)

  Aupdated $\leftarrow \left[\mathbf{A}_{old}\ \textbf{normMatA}(:, \text{newConcept})\right]$

  Bupdated $\leftarrow \left[\mathbf{B}_{old}\ \textbf{normMatB}(:, \text{newConcept})\right]$

  Cupdated $\leftarrow$ update $\mathbf{C}$ depending on the New Concept,

  Concept Overlap, overlapConceptOld indices and runningRank

else

  Aupdated $\leftarrow \mathbf{A}_{old}$

  Bupdated $\leftarrow \mathbf{B}_{old}$

  Cupdated $\leftarrow$ update $\mathbf{C}$ depending on the Concept Overlap, overlapConceptOld indices and runningRank

end if

Update ρ depending on the New Concept and Concept Overlap indices

if newConcept or $(len$(newConcept$) + len$(conceptOverlap$) <$ runningRank**) then**

  Concept Drift Detected

end if

This solves problem **P2**. In the experimental evaluation, we demonstrate the behavior of SeekAndDestroy with respect to that threshold.

SeekAndDestroy: This is our overall proposed algorithm, which detects concept drift between the two consecutive tensor batch decompositions, as illustrated in Algorithm 2 and updates the decomposition in a fashion robust to the drift. SeekAndDestroy takes factor matrices($\mathbf{A_{old}}$, $\mathbf{B_{old}}$, $\mathbf{C_{old}}$) of previous tensor batch (say at time t_0), running rank at t_0($\mathbf{runningRank}_{t_0}$) and new tensor batch($\underline{\mathbf{X}}_{new}$) (say at time t_1) as inputs. Subsequently,

SeekAndDestroy computes the tensor rank for the batch (**batchRank**) for $\underline{\mathbf{X}}_{new}$ using AutoTen.

Using the estimated rank **batchRank**, SeekAndDestroy computes the CP decomposition of $\underline{\mathbf{X}}_{new}$, which returns factor matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$. We normalize the columns of A, B, C to unit ℓ_2 norm and we store the normalized matrices into **normMatA**, **normMatB**, and **normMatC**, as shown by lines 3-4 of Algorithm 2. Both $\mathbf{A_{old}}$ and normalized matrix $\mathbf{A}$ are passed to $findConceptOverlap$ function as described above. This returns the indexes of new concept and indexes of overlapping concepts from both matrices. Those indexes inform SeekAndDestroy, while updating the factor matrices, where to append the overlapped concepts. If there are new concepts, we update A and B factor matrices simply by adding new columns from normalized factor matrices of $\underline{\mathbf{X}}_{new}$ as shown in lines 9-10 of Algorithm 2. Furthermore, we update the running rank by adding number of new concept discovered to the previous running rank. If there is only overlapping concepts and no new concepts, then $\mathbf{A}$ and $\mathbf{B}$ factor matrices does not change.

Updating Factor Matrix C: In this work, for simplicity of exposition, we are focusing on streaming data that are increasing only on one mode. However, our proposed method readily generalizes to cases where more than one modes grow over time.

In order to update the "evolving" factor matrix ($\mathbf{C}$ in our case), we use a different technique from the one used to update $\mathbf{A}$ and $\mathbf{B}$. If there is a new concept discovered in **normMatC** then

$$\mathbf{C}_{updated} = \begin{bmatrix} \mathbf{C}_{old} & zeroCol \\ zerosM & \mathbf{normMatC}(:, newConcept) \end{bmatrix} \tag{4.6}$$

where $zeroCol$ is of size $K_{old} \times len(newConcept)$, $zerosM$ is of size $K_{new} \times R$ and $\mathbf{C}_{updated}$ is of size $(K_{old} + K_{new}) \times$ runningRank.

If there are overlapping concepts, then we update $\mathbf{C}$ accordingly as shown below; in this case $\mathbf{C}_{updated}$ is again of size $(K_{old} + K_{new}) \times$ runningRank.

$$\mathbf{C}_{updated} = \begin{bmatrix} \mathbf{C}_{old}(:, overlapConceptOld) \\ \mathbf{normMatC}(:, conceptOverlap) \end{bmatrix} \tag{4.7}$$

If there are missing concepts we append an all-zeros matrix (column vector) to those indexes.

The Scaling Factor ρ: When we reconstruct the tensor from updated factor (normalized) matrices, we need a way to re-scale the columns of those factor matrices. In our approach we compute element wise product on normalized columns of factor matrices ($\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$) of $\underline{\mathbf{X}}_{new}$ as shown in line 5 of Algorithm 2. We use the same technique as the one used in updating C matrix, in order to match the values between two consecutive intervals, and we add this value to previously computed values. If it is a missing concept, we simply add zero to it. While reconstructing the tensor we take the average of vector over the number of batches received and we re-scale the components as follows

$$\underline{\mathbf{X}}_r = \sum_{r=1}^{\text{runningRank}} \rho_r \mathbf{A}_{\text{upd.}}(:, r) \circ \mathbf{B}_{\text{upd.}}(:, r) \circ \mathbf{C}_{\text{upd.}}(:, r).$$

Algorithm 3 Find Concept Overlap

Require: Factor matrices $\mathbf{A}_{old}$, **normMatA** of size $I \times R$, $I \times batchRank$ respectively.

Ensure: newConcept, conceptOverlap, overlapConceptOld

$THRESHOLD \leftarrow 0.6$

if $R == batchRank$ **then**

    Generate all the permutations for [1:R]

    for all permutation **do**

        Compute dot product of $\mathbf{A}_{old}$ *and* **normMatA**$(:, \mathbf{permutation})$

    end for

else if $R > batchRank$ **then**

    Generate all the permutations(1:R, batchRank)

    for all permutation **do**

        Compute dot product of $\mathbf{A}_{old}(:, permutation)$ *and* **normMatA**

    end for

else if $R < batchRank$ **then**

    Generate all the permutations (1:batchRank, R)

    for all permutation **do**

        Compute dot product of $\mathbf{A}_{old}$ *and* **normMatA**$(:, \mathbf{permutation})$

    end for

end if

Select the best permutation based on the maximum sum.

If dot product value of a column is less than threshold its a New Concept

If dot product value of a column is more than threshold then its a Concept Overlap.

Return column index's of New Concept and Concept Overlap for both matrices

4.4 Experimental Evaluation

We evaluate our algorithm on the following criteria:

Q1: Approximation Quality: We compare SeekAndDestroy's reconstruction accuracy against state-of-the-art streaming baselines, in data that we generate synthetically so that we observe different instances of concept drift. In cases where SeekAndDestroy outperforms the baselines, we argue that this is due to the detection and alleviation of concept drift.

Q2: Concept Drift Detection Accuracy: We evaluate how effectively SeekAndDestroy is able to detect concept drift in synthetic cases, where we control the drift patterns.

Q3: Sensitivity Analysis: As shown in Section 4.3, SeekAndDestroy expects the matching threshold as a user input. Furthermore, its performance may depend on the selection of the batch size. Here, we experimentally evaluate SeekAndDestroy's sensitivity along those axes.

Q4: Effectiveness on Real Data: In addition to measuring SeekAndDestroy's performance in real data, we also evaluate its ability to identify useful and interpretable latent concepts in real data, which elude other streaming baselines.

4.4.1 Experimental Setup

We implemented our algorithm in Matlab using tensor toolbox library [12] and we evaluate our algorithm on both synthetic and real data.We use [62] method available in literature to find rank of incoming batch.

In order to have full control of the drift phenomena, we generate synthetic tensors with different ranks for every tensor batch, we control the batch rank of the tensor with factor matrix **C**. Table 4.1 shows the specification of the datasets created. For instance dataset **SDS2** has an initial tensor batch whose tensor rank is 2 and last tensor batch whose tensor rank is 10(full rank). The batches in between the initial and final tensor batch can have any rank between initial and final rank(in this case 2-10). The reason we assign the final batch rank as the full rank is to make sure the tensor created is not rank deficient. We make the synthetic tensor generator available as part of our code release.

DataSet	Dimension	Initial Rank	Full Rank	Batch Size	Matching Threshold
SDS1 SDS2	100 x 100 x 100	2	5 10	10	0.6
SDS3 SDS4	300 x 300 x 300	2	5 10	50	0.6
SDS5 SDS6	500 x 500 x 500	2	5 10	100	0.6

Table 4.1: Table of Datasets analyzed

In order for us to obtain robust estimates of performance, we require all experiments to either 1) run for 1000 iterations, or 2) the standard deviation converges to a second significant digit (whichever occurs first). For all reported results, we use the median and the standard deviation.

4.4.2 Evaluation Metrics

We evaluate SeekAndDestroy and the baselines methods using *relative error*. Relative Error provides the measure of effectiveness of the computed tensor with respect to the original tensor and is defined as follows (lower is better):

$$RelativeError = \Big(\frac{||\underline{\mathbf{X}}_{original} - \underline{\mathbf{X}}_{computed}||_F}{||\underline{\mathbf{X}}_{original}||_F}\Big) \tag{4.8}$$

4.4.3 Baselines for Comparison

To evaluate our method, we compare SeekAndDestroy with two state-of-the-art streaming baselines: OnlineCP [93] and SamBaTen [34]. Both baselines assume that the rank remains fixed throughout the entire stream. When we evaluate the approximation accuracy of the baselines, we run two different versions of each method, with different input ranks: 1) *Initial Rank*, which is the rank of the initial batch, same as the one that SeekAndDestroy uses, and 2) *Full Rank*, which is the "oracle" rank of the full tensor, if we assume we could compute that in the beginning of the stream. Clearly, *Full Rank* offers a great advantage to the baselines since it provides information from the future.

4.4.4 Q1: Approximation Quality

The first dimension that we evaluate is the approximation quality. More specifically, we evaluate whether SeekAndDestroy is able to achieve good approximation of the original tensor (in the form of low error) in case where concept drift is occurring in the stream. Table 4.2 contains the general results of SeekAndDestroy's accuracy, as compared to the baselines. We observe that SeekAndDestroy outperforms the two baselines,

in the pragmatic scenario where they are given the same starting rank as SeekAndDestroy (Initial Rank). In the non-realistic, "oracle" case, OnlineCP performs better than SamBaTen and SeekAndDestroy, however this case is a very advantageous lower bound on the error for OnlineCP.

Through extensive experimentation we made the following interesting observation: in the cases where most of the concepts in the stream appear in the beginning of the stream (e.g., in batches 2 and 3), SeekAndDestroy was able to further outperform the baselines. This is due to the fact that, if SeekAndDestroy has already "seen" most of the possible concepts early-on in the stream, it is more likely to correctly match concepts in later batches of the stream, since there already exists an almost-complete set of concepts to compare against. Indicatively,in this case SeekAndDestroy achieved 0.1176 ± 0.0305 where as OnlineCP achieved 0.1617 ± 0.0702.

4.4.5 Q2: Concept Drift Detection Accuracy

The second dimension along which we evaluate SeekAndDestroy is its ability to successfully detect concept drift. Figure 4.5 shows the rank discovered by SeekAndDestroy at every point of the stream, plotted against the actual rank. We observe that SeekAndDestroy is able to successfully identify changes in rank, which, as we have already argued, signify concept drift. Furthermore, Table 4.4 shows three example runs that demonstrate the concept drift detection accuracy.

4.4.6 Q3: Sensitivity Analysis

The results we have presented so far for SeekAndDestroy have used a matching threshold of 0.6. The threshold was chosen because it is intuitively larger than a 50% match, which is a reasonable matching threshold. In this experiment, we investigate the sensitivity of SeekAndDestroy to the matching threshold parameter. Table 4.3 shows exemplary approximation errors for thresholds of 0.4, 0.6, and 0.8. We observe that 1) the choice of threshold is fairly robust for values around 50%, and 2) the higher the threshold, the better the approximation, with threshold of 0.8 achieving the best performance.

4.4.7 Q4: Effectiveness on Real Data

To evaluate effectiveness of our method on real data, we use the Enron time-evolving communication graph dataset [10]. Our hypothesis is that in such complex real data, there should exists concept drift in streaming tensor decomposition. In order to validate that hypothesis, we compare the approximation error incurred by SeekAndDestroy against the one incurred by the baselines, shown in Table 4.5. We observe that the approximation error of SeekAndDestroy is lower than the two baselines. Since the main difference between SeekAndDestroy and the baselines is that SeekAndDestroy takes concept drift into consideration, and strives to alleviate its effects, this result 1) provides further evidence that there exists concept drift in the Enron data, and 2) demonstrates SeekAndDestroy's effectiveness on real data.

The final rank for Enron as computed by SeekAndDestroy was 7, indicating the existence of 7 time-evolving communities in the dataset. This number of communities is higher than what previous tensor-based analysis has uncovered [10, 64]. However, analyzing the (static) graph using a highly-cited non-tensor based method [19], we were able to detect 7 communities, therefore SeekAndDestroy may be discovering subtle communities that have eluded previous tensor analysis. In order to verify that, we delved deeper into the communities and we plot their temporal evolution (taken from matrix $\mathbf{C}$) along with their annotations (when inspecting the top-5 senders and receivers within each community) as shown in Figure 4.6. Indeed, a subset of the communities discovered matches with the ones already known in the literature [10, 64]. Additionally, SeekAndDestroy was able to discover community #3, which refers to a group of executives, including the CEO. This community appears to be active up until the point that the CEO transition begins, after which point it dies out. This behavior is indicative of concept drift, and SeekAndDestroy was able to successfully discover and extract it.

4.5 Related Work

Tensor decomposition: Tensor decomposition techniques are widely used for static data. With the explosion of big data, data grows at a rapid speed and an extensive study required on the online tensor decomposition problem. Sidiropoulos [61] introduced two well-known PARAFAC based methods namely RLST (recursive least squares) and SDT (simultaneous diagonalization tracking) to address the online 3-mode tensor decomposition. Zhou et al. [93] proposed OnlineCP for accelerating online factorization that can track the

decompositions when new updates arrived for N-mode tensors. Gujral et al. [34] proposed Sampling-based Batch Incremental Tensor Decomposition algorithm which updates online computation of CP/PARAFAC and performs all computations in the reduced summary space. However, no prior work addresses concept drift.

Concept Drift: The survey paper [86] provides the qualitative definitions of characterizing the drifts on data stream models. To the best of our knowledge, however, this is the first work to discuss concept drift in tensor decomposition.

4.6 Contributions

Our contributions in this work are the following:

- **Characterizing concept drift in streaming tensors**: We define concept and concept drift in time evolving tensors and provide a quantitative method to measure the concept drift.
- **Algorithm for detecting and alleviating concept drift in streaming tensor decomposition**: We provide an algorithm which detects drift in the streaming data and also updates the previous decomposition without any assumption on the rank of the tensor.
- **Experimental evaluation on real & synthetic data**: We extensively evaluate our method on both synthetic and real datasets and out-perform state of the art methods in cases where the rank is not known a priori and perform on par in other cases.

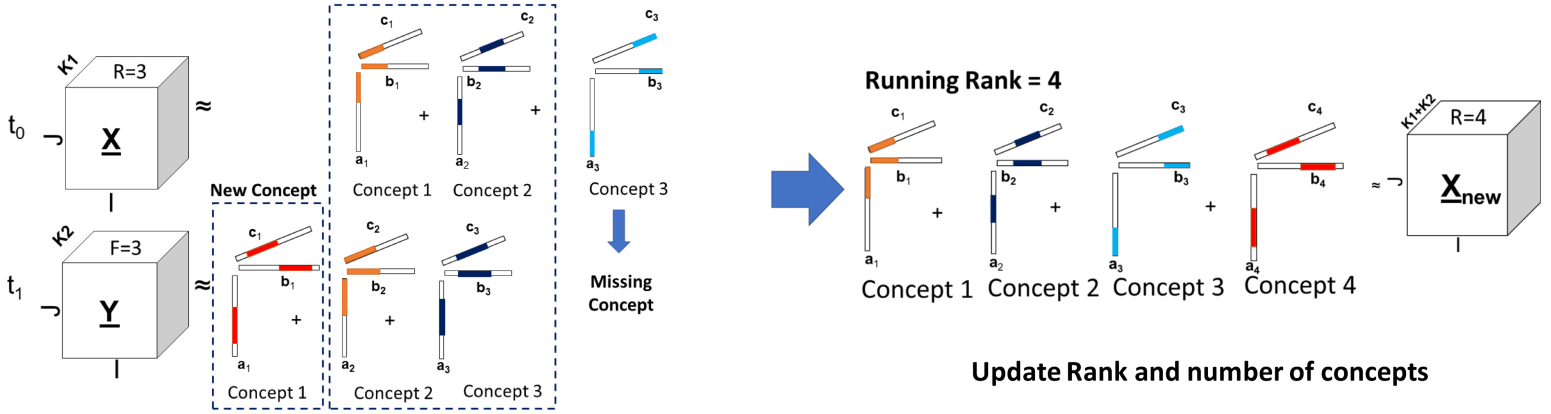
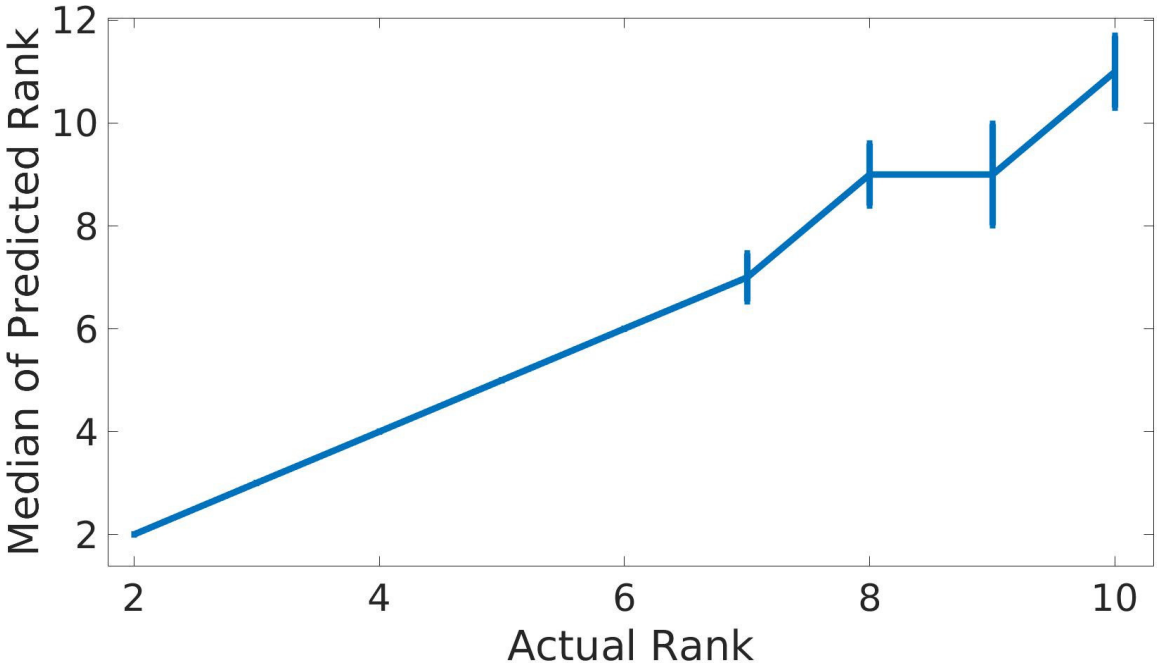


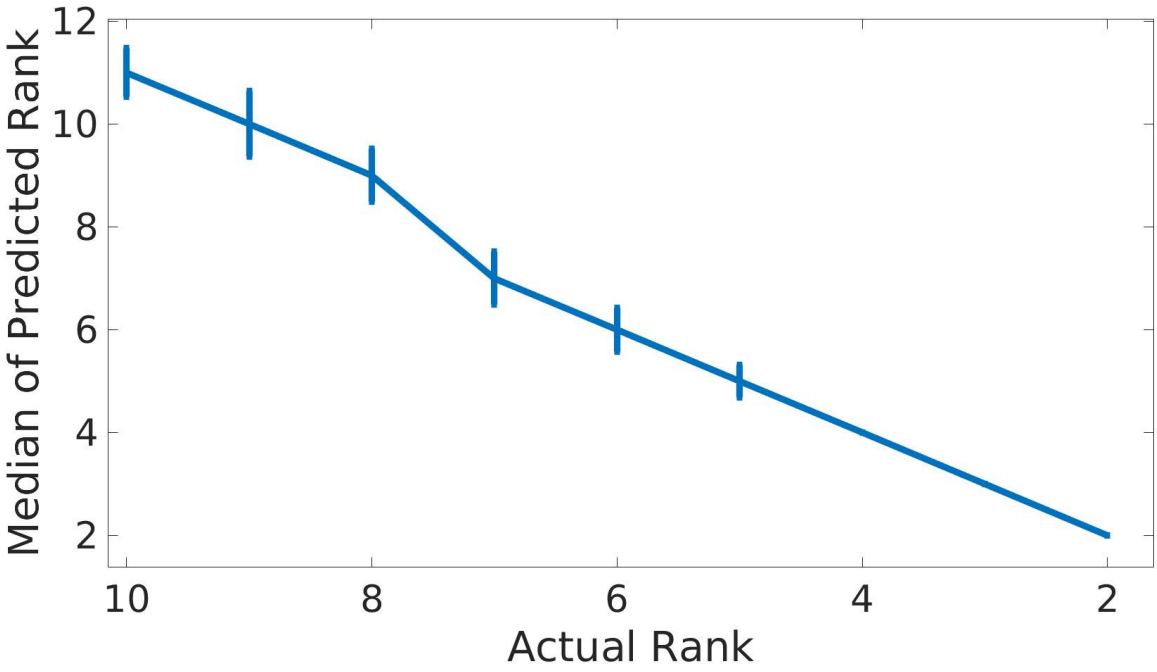
Figure 4.4: Problem formulation

DataSet	OnlineCP (Initial Rank)	OnlineCP (Full Rank)	SamBaTen (Initial Rank)	SamBaTen (Full Rank)	SeekAndDestroy
SDS1	0.2782±0.0221	0.197±0.086	**0.261±0.048**	0.317±0.058	0.283±0.075
SDS2	0.2537±0.0125	0.168±0.507	**0.244±0.028**	0.480±0.051	0.253±0.0412
SDS3	0.2731±0.0207	0.205±0.164	0.385±0.021	0.445±0.164	**0.266±0.081**
SDS4	0.245±0.013	0.171±0.537	0.299±0.045	0.402±0.049	**0.221±0.0423**
SDS5	0.2719±0.0198	0.206±0.022	0.559±0.046	0.519±0.0219	**0.256±0.105**
SDS6	0.238±0.013	0.171±0.374	0.510±0.036	0.547±0.0276‘	**0.208±0.0433**

Table 4.2: Approximation error for SeekAndDestroy and the baselines. SeekAndDestroy outperforms the baselines in the realistic case where all methods start with the same rank



(a) Increasing rank



(b) Decreasing rank

Figure 4.5: SeekAndDestroy is able to successfully detect concept drift, which is manifested as changes in the rank throughout the stream

Threshold	SDS2	SDS4
0.4	0.253±0.041	0.221 ± 0.042
0.6	0.253±0.041	0.221 ± 0.042
0.8	0.101 ±0.040	0.033 ± 0.011

Table 4.3: Experimental results for error of approximation of incoming batch with different matching threshold values. Dataset SDS2 and SDS4 are of dimension $\mathbb{R}^{100\times100\times100}$ and $\mathbb{R}^{300\times300\times300}$, respectively. We see that the threshold is fairly robust around 0.5, and a threshold of 0.8 achieves the highest accuracy.

Running Rank	Actual Rank	Predicted Rank	**Approx. Error**	
			Actual Rank	Predicted Rank
6	[2,4,3,4,3,3,5,3,3,5]	[2,4,3,4,3,3,5,3,3,6]	0.185	0.194
6	[2,4,3,4,3,3,5,3,3,5]	[2,4,3,4,3,3,5,3,3,6]	0.185	0.197
7	[2,4,3,4,3,3,5,3,3,5]	[2,4,3,5,3,3,6,3,3,6]	0.185	0.278

Table 4.4: Experimental results on SDS1 for error of approximation of incoming slices with known and predicted rank.

Running Rank	Predicted Full Rank	Batch Size	**Approximation Error**		
			SeekAndDestroy	SambaTen	OnlineCP
7±0.88	4±0.57	22	**0.68 ± 0.002**	0.759± 0.059	0.941± 0.001

Table 4.5: Evaluation on Real dataset

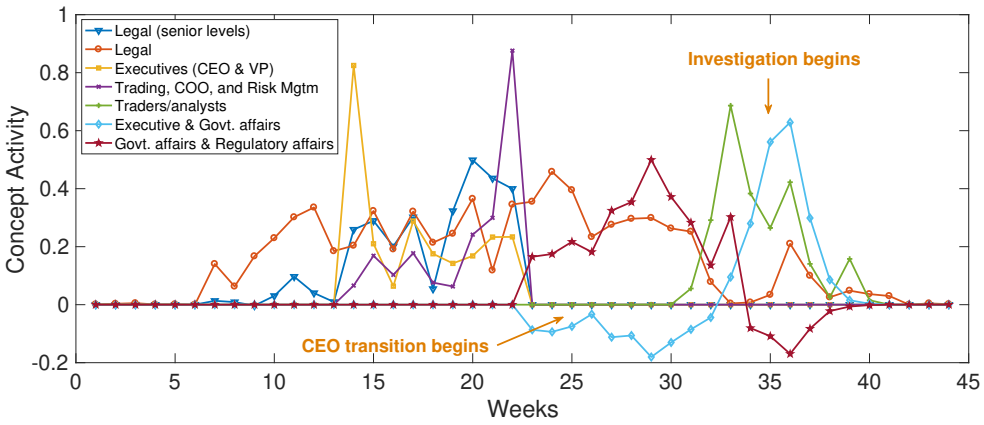


Figure 4.6: Timeline of concepts discovered in Enron

Chapter 5

Adaptive Granularity in Tensors: Problem Formulation and a Greedy Algorithm

This chapter is based on material presented in [72, 70].

Conventional Assumption Revisited:

Tensors created using raw data/fixed window aggregation have exploitable structure for analysis.

Contribution:

Using raw granularities or fixed window aggregation yields poor quality. IceBreaker++ provides multiple tensors with exploitable structure of different resolution.

5.1 Introduction

In the age of big data, applications deal with data collected at very fine-grained time intervals. In many real world applications, the data collected spans long periods of time and can be extremely sparse. For instance, a time-evolving social network that records interactions of users every second results in a very sparse adjacency matrix if observed at that granularity. Similarly, in spatio-temporal data, if one considers GPS data over time, discretizing GPS coordinates based on the observed granularity can lead to very sparse data which may not contain any visible and useful structure. How can we find meaningful and actionable structure in these types of data? A great deal of such datasets are multi-aspect in nature and hence can be modeled using tensors. For instance, a three-mode tensor can represent a time-evolving graph capturing user-user interactions over a period of time, measuring crime incidents in a city community area over a period of time [81], or measuring traffic patterns [92]. Tensor decomposition has been used in order to extract hidden patterns from such multi-aspect data [79, 65, 44]. However the degree of sparsity in the tensor, which is a function of the granularity in which the tensor is formed, significantly affects the ability of the decomposition to discover "meaningful" structure in the data.

Consider a dataset which can be modeled as three-mode tensor, where the third mode is temporal as shown in Figure 5.1. If the granularity of the temporal mode is too fine (in milliseconds or seconds), one might end up with a tensor that is extremely long on the time mode and where each instance of time has very small number of entries. This results in a extremely sparse tensor, which typically is of very high rank, and which usually has no underlying exploitable structure for widely popular and successful tensor

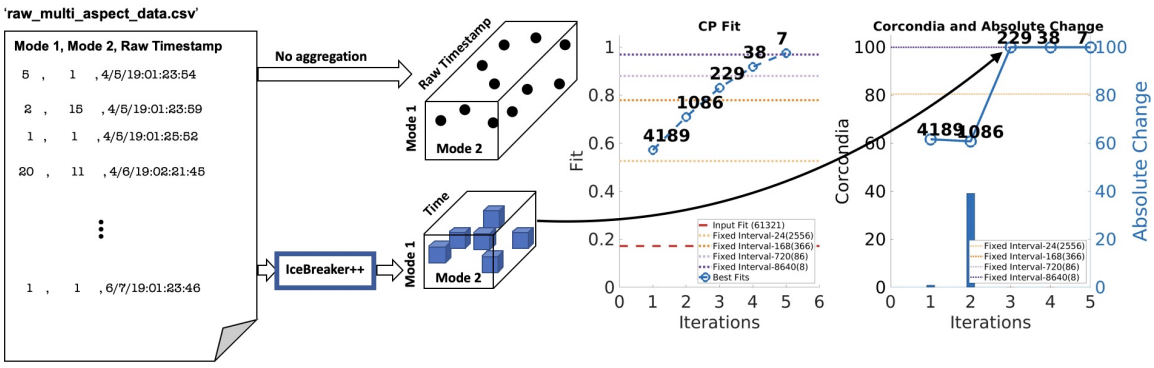


Figure 5.1: Starting from raw CSV files, IceBreaker++ discovers a tensor that has good structure (under various measures of quality, including interpretability and predictive quality), outperforming traditional fixed aggregation heuristics. Furthermore, IceBreaker++ using various notions of locally optimal structure, discovers different resolutions in the data.

decomposition algorithms [79, 65, 44]. However, as we aggregate data points over time, exploitable structure starts to appear (where-by "exploitable" we define the kind of low-rank structure that a tensor decomposition can successfully model and extract). In this chapter we set out to explore what is the best such data-driven aggregation of a tensor which leads to better, exploitable, and interpretable structure, and how this fares against the traditional alternative of selecting a fixed interval for aggregation.

As far as tackling the problem above, there is considerable amount of work that focuses on a special case, that of aggregating edges of a time evolving graph into "mature" adjacency matrices based on certain graph properties [82, 83, 84]. In our work, however, we address the problem in more general terms, where the underlying data can be any point process that is observed over time and/or space, and where the aggregation/discretization of the corresponding dimensions directly affects our ability to extract interpretable patterns via

tensor decomposition. Effectively, as shown in Figure 5.1, in this chapter we work towards automating the data aggregation starting from raw data into a well-structured tensor.

5.2 Problem Formulation

5.2.1 Tensor decomposition quality

Unsupervised tensor decomposition, albeit very popular, poses a significant challenge: how can we tell whether a computed decomposition is of "high quality", and how can we go about defining "quality" in a meaningful way? Unfortunately, this happens to be a very hard problem to solve [62], and defining a new measure of quality is beyond the scope of this work. However, there has been significant amount of work in that direction, which basically boils down to 1) model-based measures, where the quality is measured by how well a given decomposition represents the intrinsic hidden structure of the data, and 2) extrinsic measures, where the quality is measured by how well the computed decomposition factors perform in a predictive task. However, extrinsic measures do not generalize, as they specialize to a particular labeled task, and in general we cannot assume that labels will be available for the data at hand. Thus, in this work we focus on model-based measures, which can provide a general solution.

In model-based measures, the most straightforward one is the fit, i.e., how well does the decomposition approximate the data under the chosen loss function, in a *low rank*. Low rank is key, because the number of components (rank) has to be as small and compact as possible in order to lend itself to human evaluation and exploratory analysis. However, fit has been shown to be unstable and prone to errors especially in real and noisy data, thus

the community has collectively turned its attention to more robust measures such as the Core Consistency Diagnostic (CORCONDIA for short) [21], which measures how well the computed factors obey the CP model.

Both types of quality measure capture different elements of what an end-user would deem good in a set of decomposition factors. In this chapter, we are going to use such popular measures of quality in order to characterize the quality of a given tensor dataset $\underline{\mathbf{X}}$. In order to do so, we assume that we have a function $\mathcal{Q}(\underline{\mathbf{X}})$ which, optimizes the quality measure $q()$ for a given tensor over all possible decomposition ranks R [1], i.e.,

$$\mathcal{Q}(\underline{\mathbf{X}}) = \max_{R} q(\underline{\mathbf{X}}, \mathbf{A}, \mathbf{B}, \mathbf{C})$$

where $\mathbf{A}, \mathbf{B}, \mathbf{C}$ are the R-column factor matrices for $\underline{\mathbf{X}}$. Finally, a useful operation is the n-mode product, where a matrix $\mathbf{W}$ is multiplied by the n-th mode of a tensor (predicated on matching dimensions in the n-th mode of the tensor and the rows of the matrix), denoted as $\underline{\mathbf{X}} \times_n \mathbf{W}$. For instance, an $I \times J \times K$ tensor where $n = 3$ and $\mathbf{W}$ of size $K \times K^*$, the product $\underline{\mathbf{X}} \times_n \mathbf{W}$ multiplies all third mode slices of $\underline{\mathbf{X}}$ with $\mathbf{W}$ and results in a $I \times J \times K^*$ tensor.

5.2.2 The *Trapped Under Ice* problem

To give reader an intuition of the problem, consider an example of time-evolving graph which captures social activity over the span of some time. This example can be modeled as three-mode tensor $\underline{\mathbf{X}}$ of dimensions $I \times J \times K$ where "sender" and "receiver" are the first two modes, "time" being the third mode, and non-zero entry in the tensor

[1] In practice, this is done over a small number of low ranks, since low-rank structure is desirable.

represents communications between user at a particular time. If the time granularity is extremely fine-grained (milliseconds or seconds), there might be only handful of data points at a particular time stamp causing resulting tensor to be extremely sparse and to have a high tensor-rank as a result. In that case, $\underline{\mathbf{X}}$ might not have any interpretable low-rank structure that can be exploited by CP. In this example we assume that the third mode (time mode) is too fine-grained but in reality any mode (one or more) can be extremely fine grained. For example, in spatio-temporal data, where the first two modes are latitude and longitude and the third mode is time, all three modes can suffer from the same problem.

Given tensor $\underline{\mathbf{X}}$ which is created using the "raw" granularities, how does one find a tensor (say $\underline{\mathbf{Y}}$) which has better exploitable structure and hence can be decomposed into meaningful latent structure. This, is informally the *Trapped Under Ice* problem that we define here (which draws an analogy between the good structure that may exist within the data as being trapped under the ice and not visible by mere inspection). *Trapped Under Ice* has an inherent assumption that the mode in which we aggregate is ordered (e.g., representing time or space), thus permuting the third mode will lead to a different instance of the problem.

More formally we define our problem as follows:

Given a tensor $\underline{\mathbf{X}}$ of dimensions $I \times J \times K$ Find:

A tensor $\underline{\mathbf{Y}}$ of dimensions $I \times J \times K^*$ with $K^* \leq K$ such that

$$\max_{\mathbf{W}} \mathcal{Q}\left(\underline{\mathbf{X}} \times_3 \mathbf{W}\right)$$

where $\mathcal{Q}$ is a measure of goodness and $\mathbf{W}(i,j) = 1$ if slice i in tensor $\underline{\mathbf{X}}$ is aggregated into slice j in the resulting tensor, otherwise $\mathbf{W}(i,j) = 0$.

At first glance, *Trapped Under Ice* might look like a problem amenable to dynamic programming, since it exhibits the optimal substructure property. However, it lacks the overlapping subproblems property: there are overlapping subproblems across the set of different $\mathbf{W}$ matrices (e.g., two different matrices may have overlapping subproblems) but not within any single $\mathbf{W}$. Thus, we still have to iterate over 2^{K-1} $\mathbf{W}$'s refer subsection 5.2.3 for more details.

Structure of W: The matrix $\mathbf{W}$ has a special structure. Here we provide an example. Consider a three-mode tensor $\underline{\mathbf{X}}$ of dimensions $10 \times 10 \times 10$, with the third mode being the time mode. Suppose that the optimal level of aggregation for $\underline{\mathbf{Y}}$ is $K^* = 3$.

In this case, $\mathbf{W}$ is of size 3×10 and an example of such matrix is:

$$\mathbf{W} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

This $\mathbf{W}$ aggregates first three slice of $\underline{\mathbf{X}}$ to form first slice of $\underline{\mathbf{Y}}$, next three to form the second slice and last four to form the third slice. No two $\mathbf{W}$ matrices will produce the same aggregation. They can have the same K^* but order of aggregation of slices will be different.

5.2.3 Solving *Trapped Under Ice* optimally is hard

Solving *Trapped Under Ice* optimally poses a number of hurdles. First and foremost, the hardness of the problem depends on the definition of function $\mathcal{Q}$, and most reasonable and intuitive such definitions are very hard to optimize since they are non-differentiable,

non-continuous, and not concave. So far, in the literature, to the best of our knowledge, there are only heuristics for this quality function. Even so, those heuristic functions can only be evaluated on a single already fully-aggregated tensor, not a partially aggregated version thereof. Thus, *Trapped Under Ice* can be only solved optimally via enumerating all admissible solutions and choosing the best. In order to conduct this enumeration, we need to calculate the cardinality of the set of all $\mathbf{W}$ for a given instance of the problem.

Lemma 1 *For an instance of a problem with K initial slices, the cardinality of the set of all $\mathbf{W}$ is 2^{K-1}*

Proof. To get K^* aggregated slices there are $\binom{K-1}{K^*-1}$ ways to choose each of them leading to a different $\mathbf{W}$. This is a number of ways that $K-1$ partition slots can be filled partitioned by K^*-1 blocks. In order to get the final number, we need to sum up over all potential K^*:

$$\sum_{K^*=0}^{K-1} \binom{K-1}{K^*} = 2^{K-1}$$

■

Direct corollary of the above lemma is that solving optimally *Trapped Under Ice* requires calling the function $\mathcal{Q}$ $O\left(2^K\right)$ times, which is computationally intractable. There may be small room for improvement by exploiting special structure in the set of all $\mathbf{W}$, however, given discontinuities in our objective function $\mathcal{Q}$, this is not be a feasible alternative either. In this chapter we define proxy quality functions $\mathcal{Q}$ that lend themselves to partial evaluation on a partially aggregated solution, thus allowing for efficient algorithms Thus, in the next section we propose a greedy approach which locally optimizes different criteria quality.

5.3 Algorithms

In this section, we propose our efficient and effective greedy algorithm called IceBreaker which takes a tensor $\underline{\mathbf{X}}$ as an input, which has been created directly from raw data, and has no exploitable structure. and returns a tensor $\underline{\mathbf{Y}}$ which maximizes the interpretable and exploitable structure. The basic idea behind IceBreaker is to make a linear pass on the mode for which the granularity is suboptimal, and using a number of intuitive and locally optimal criteria for goodness of structure (henceforth referred to as *utility functions*), we greedily decide whether a particular slice across that mode needs to be aggregated[2] into an existing slice or contains good-enough structure to stand on its own. IceBreaker can choose from a number of intuitive utility functions which are based on different definitions of good quality in matrices.

5.3.1 The IceBreaker algorithm

Algorithm 4 gives a high level overview of IceBreaker. More specifically, the algorithm takes a three-mode tensor $\underline{\mathbf{X}}$ of dimension $I \times J \times K$ as an input and loops over all the K slices of tensor $\underline{\mathbf{X}}$. Two slices next to each other get aggregated into a single slice if a certain utility function *has stabilized*, i.e., if aggregating the two slices does not offer any additional utility (larger than a particular threshold), then the second slice should not be aggregated with the first, and should mark the beginning of a new slice.

Consider a three-mode tensor $\underline{\mathbf{X}}$ with time as third mode of dimension $I \times J \times K$ is ran through IceBreaker with a particular utility function. Our algorithm iterates over the

[2]For the purposes of our work, we use matrix addition as aggregation of slice but this might not be the case and would depend on the problem domain. Other aggregation functions that can be used are OR, min, max, depending on the application domain (e.g., binary data).

time mode (K slices) and aggregates slices as decided by the utility function. IceBreaker is agnostic to utility function used. Let us consider a slice that has been aggregated into a single slice from indices i to $j-1$ called previous slice and another aggregated slice from indices i to j called a candidate slice. Both previous and candidate slice are passed to utility function separately to obtain a value each called previous and current value respectively. These values are compared (line 5 in algorithm 4) to decide whether j^{th} slice is absorbed(line 6 in algorithm 4) into previous slice or previous slice has stabilized and entry is added in W to indicate which indices of tensor $\underline{\mathbf{X}}$ are aggregated together(line $8-9$ in algorithm 4). Now j^{th} slice becomes the previous slice and aggregated slice of j and $j+1$ become the candidate slice, the whole process is repeated until all the slices are exhausted.

Note that IceBreaker's complexity is *linear* in terms of the slices K of the original tensor, and its overall complexity depends on the specific utility function used (which is called $O(K)$ times).

5.3.2 Utility functions

In this subsection, we summarize a number of intuitive utility functions that we are using in this work. This list is by no means exhaustive, and can be augmented by different functions (or function combinations) that capture different elements of what is good structure and can be informed by domain-specific insights.

1. **Norm:** We use multiple norm types to find adaptive granularity of a tensor. For a given threshold, if rate of change of norm between previous and candidate slice is less

than the threshold, candidate slice is not selected. Our assumption in this case is no significant amount of information is being added to previous slice and is considered to have been stabilized. Matrix $\mathbf{W}$ is updated accordingly with indices of the previous slice (aggregated slices in previous slice). Otherwise the candidate slice is selected and the process continues until all the slices are exhausted. Different norms demonstrated in this work are Frobenius, 2-norm, and Infinity norm.

2. **Matrix Rank:** In case of matrix rank, we focus on the 95% reconstruction rank, which is typically much lower than the full rank of the data, but captures the essence of the number of components within the slice. In this case, we consider previous slice to be stabilized if the matrix-rank of previous slice decreases by addition of new slice, no more slices are added and an entry in matrix $\mathbf{W}$ is added. We keep aggregating slices if the matrix-rank of the slice is increasing or remains constant.

3. **Missing Value Prediction:** If a piece of data has good structure, when we hide a small random subset of the data, the remaining data can successfully reconstruct the hidden values, under a particular model that we have chosen. To this end, we employ a variant of matrix factorization based collaborative filtering [46] as a utility function to see how good is the aggregated matrix in predicting certain percent of missing values. This utility function takes percent of missing value as a parameter, hides those percent of non zeros values in the matrix. Our implementation of matrix factorization with Stochastic Gradient Descent tries to minimize the loss function: $\min_{\mathbf{U},\mathbf{V}} \sum_{i,j\in\Omega} \mathcal{RMSE}\left(\mathbf{A_{ij}} - \mathbf{U_{i,:}} \cdot \mathbf{V_{:,j}}\right)$ where $\mathbf{A}$ is a given slice, $\mathbf{U}, \mathbf{V}$ are factor matrices for a given rank (typically chosen using the same criterion as the matrix

rank above), and Ω is the set of *observed* (i.e., non-missing) values. In order to create a balanced problem, since we are dealing with very sparse slices, we conduct *negative sampling* where we randomly sample as many zero entries as there are non-zeros in the slice, and this ends up being the Ω set of observed values.

5.3.3 The IceBreaker++ algorithm

IceBreaker algorithm returns a tensor $\underline{\mathbf{Y}}$ as an output which is considered to have an exploitable and better structure than the input tensor $\underline{\mathbf{X}}$. The idea behind IceBreaker++ is to recursively feed the output back to IceBreaker until the third mode is reduced to a single slice (matrix) or the dimenison of third mode does not change. IceBreaker algorithm returns a tensor associated with each utility function. So if we used 5 utility functions, we would get 5 tensors associated with each of them. Now we select the tensor with highest CP Fit (see 5.4.1), use that as input for IceBreaker and we repeat this process until the stopping condition is met. The output of each iteration is a candidate tensor. At the end we have multiple tensors (one for each iteration) which has different temporal resolutions, which can help us get a tensor with optimal resolution based on the evaluation measures used. Algorithm 5 describes the process discussed in this section.

5.4 Experimental Evaluation

In this section we present a thorough evaluation of IceBreaker++ using variety of data, including synthetic, semi-synthetic and real data. We empirically evaluate our

analysis using a number of criteria described in detail below. We implement our method in Matlab using tensor toolbox library [12].

5.4.1 Evaluation measures

When formulating the problem, we did not specify a quality function $\mathcal{Q}$ to be maximized, nor did we use such a function in our proposed method. The reason for that is because we reserve the use of different quality functions as a form of evaluation. In particular, we use the two following notions of quality:

- **CP Fit:** To evaluate effectiveness of our method, we compute CP fit of the computed tensor for a particular rank with respect to the Input tensor.

$$Relative\ Fit = 1 - \left(\frac{||\underline{\mathbf{X}}_{Input} - \underline{\mathbf{X}}_{computed}||_F}{||\underline{\mathbf{X}}_{Input}||_F}\right) \tag{5.1}$$

- **CORCONDIA**: To evaluate the *interpretability* of the resulting tensor we employ Autoten [62] that given a tensor and some estimated tensor rank, returns a CORCONDIA [21] score and low rank that provides best attainable tensor decomposition quality in a user-defined search space.

We should note at this point that the two quality measures above are far from continuous and monotonic functions, thus we do not expect that our method progresses the quality will monotonically increase. Thus, we calculate the quality for the final solution of IceBreaker++, and we reserve investigating whether monotonic and well-behaved quality functions exist for future work.

In our experiments we used 5 utility functions (see 5.3.2) namely Frobenius norm, 2-norm, Infinity norm, Matrix Rank and Missing Value Prediction. In case of synthetic datasets we ran all the utility functions once except for Missing Value Prediction which we ran for 10 times. In case of both semi-synthetic and real datasets, in the interest of computational efficiency, we ran all the utility functions once.

5.4.2 Baseline methods

A naive way to find tensor $\underline{\mathbf{Y}}$ can be by aggregating time mode based on some fixed intervals. If time granularity was in milliseconds, then combining one thousand slices to form slices of seconds granularity reducing the third dimension of tensor $\underline{\mathbf{X}}$ from K to $K/1000$. This can be applied incrementally from seconds to minutes and so on to find a tensor which has some exploitable structure. We compare the resulting tensor $\underline{\mathbf{Y}}$ determined by IceBreaker against tensors constructed with fixed aggregations. For fixed aggregation we aggregate the temporal with window size of 10, 100 and 1000 for synthetic data. For semi-synthetic and real datasets we use appropriate time windows accordingly.

5.4.3 Performance for synthetic data

Creating synthetic data: In order to create synthetic dataset, we follow a two-step process,

1. We create a random sparse tensor of specific sparsity.

2. Subsequently, we randomly distribute (drawn from uniform distribution) non zero entries in each slice over some fixed number of slice as explained in below example.

Example: Consider a three-mode tensor $\underline{\mathbf{X}}$ of dimension $I \times J \times K$, for purpose of this example consider $K = 4$ as shown in Figure 5.2. Now for each slice of size $I \times J$, distribute randomly (drawn from Uniform distribution) all the non-zeros entries across W slices preserving the I and J indices, creating a tensor of size $I \times J \times W$. Now append all the tensors in the same order as they appeared in the original tensor, we get a resulting tensor of size $I \times J \times 4W$, which is used as an input for IceBreaker. So if the original tensor if of size $I \times J \times K$ and bucket size W, the resulting tensor is of size $I \times J \times KW$ approximately[3]. Table 5.1 shows the synthetic data used for experiments.

Dataset	Original Dimension	Window size(W)	Approximate Final Dimension	Number of datasets
SD1	$100 \times 100 \times 10$	50	$100 \times 100 \times 500$	10
SD2	$100 \times 100 \times 100$	50	$100 \times 100 \times 5000$	10

Table 5.1: Table of Synthetic Datasets analyzed

Dataset	Original Dimension	Window size(W)	Approximate Final Dimension
Enron Weekly	$184 \times 184 \times 44$	4	$184 \times 184 \times 176$
Enron Daily	$184 \times 184 \times 44$	30	$184 \times 184 \times 1320$
Enron Hourly	$184 \times 184 \times 44$	720	$184 \times 184 \times 31680$

Table 5.2: Table of Semi-synthetic Datasets analyzed

Results for synthetic data: In order to evaluate the performance of IceBreaker++, we measure CORCONDIA and fit on 10 synthetic datasets for both type of datasets as

[3]The number of slice can be less than KW, since slice for each non-zero value is selected randomly, there can be a case where a slice is not selected

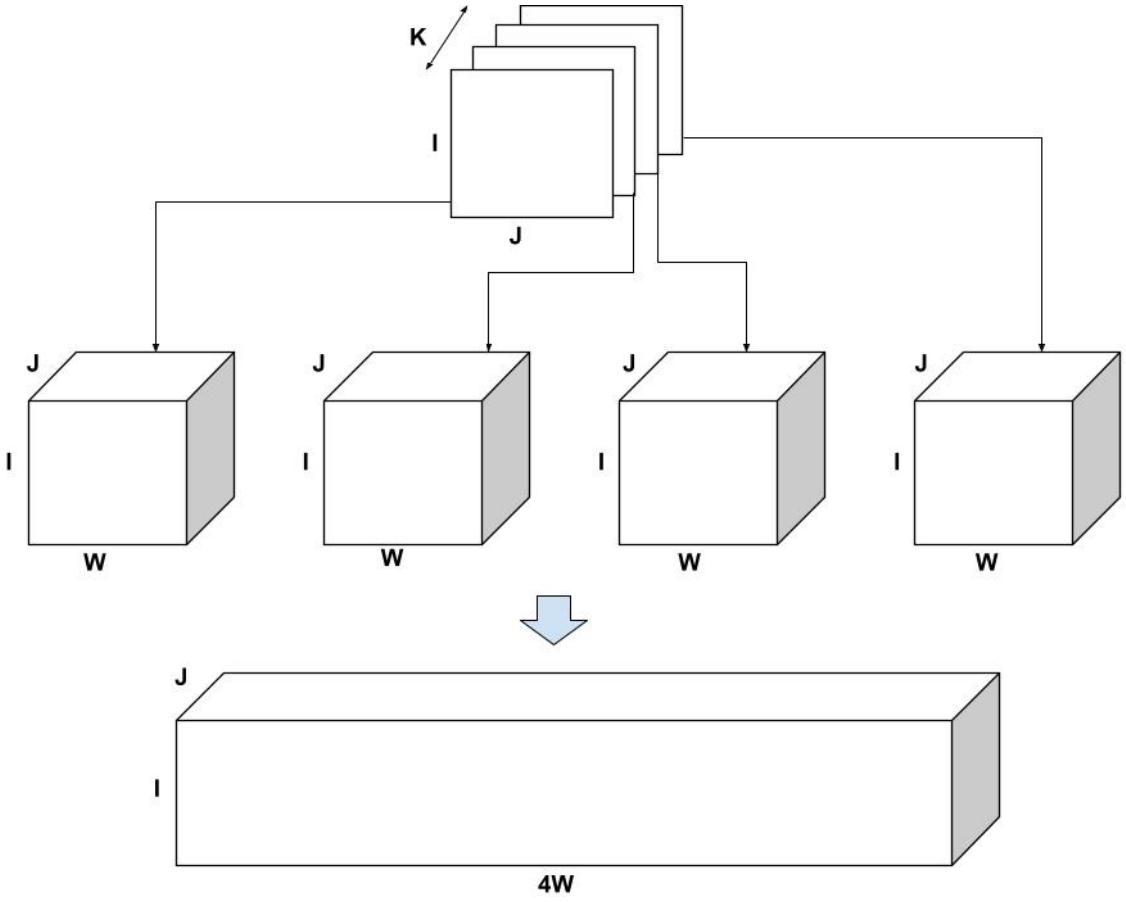


Figure 5.2: Creating Synthetic Data

mentioned in Table 5.1. In interest of conserving space we only show one set of results for both synthetic datasets. The leftmost part of the Figure 5.3 and 5.4 shows the best fit at end of each iteration. The number on top of the dots represent the dimension of the third mode after each iteration. The dotted line in the plot show the fit of the input tensor and fixed intervals tensor.[4] The rightmost part of the Figure 5.3 and 5.4 shows the

[4]The number in the parenthesis represents the dimension of the third mode for that tensor.

CORCONDIA computed at the end of each iteration and absolute change of CORCONDIA. Absolute change of CORCONDIA is computed as shown below:

$$abs(corcondia(j+1) - corcondia(j))$$

The dotted line in the plot represent CORCONDIA value for the fixed intervals tensor. When there is sudden drop in the value of CORCONDIA we consider the iteration before as an suitable candidate for tensor analysis. In the case of SD1 that would be iteration number 2 and resulting tensor of size $100 \times 100 \times 8$. In the case of SD2 that would also be iteration number 2 and resulting tensor of size $100 \times 100 \times 57$

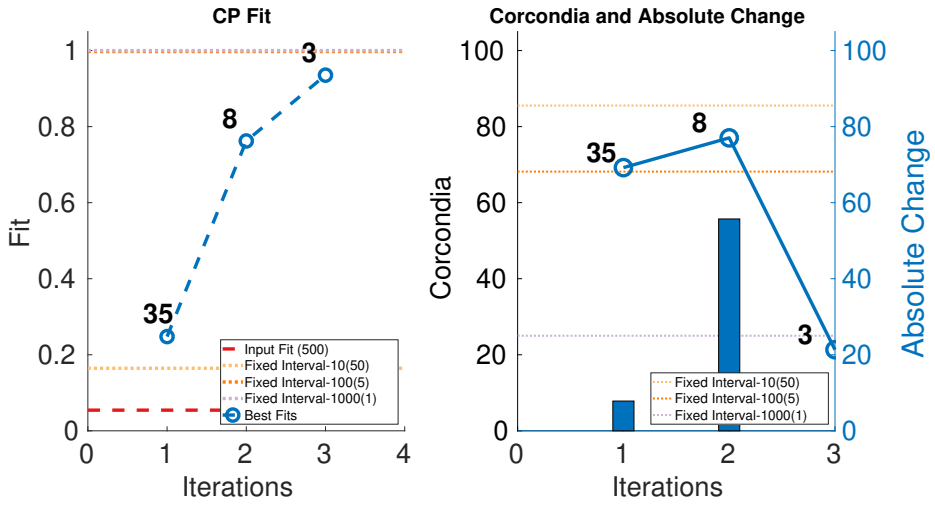


Figure 5.3: CP fit and Corcondia of best fit tensor & its absolute change at each iteration for SD1.

5.4.4 Performance for semi-synthetic data

Creating semi-synthetic data: Here we used Enron dataset [77, 11], which is dataset of number of email exchanges between employees spread over 44 months. Each month is

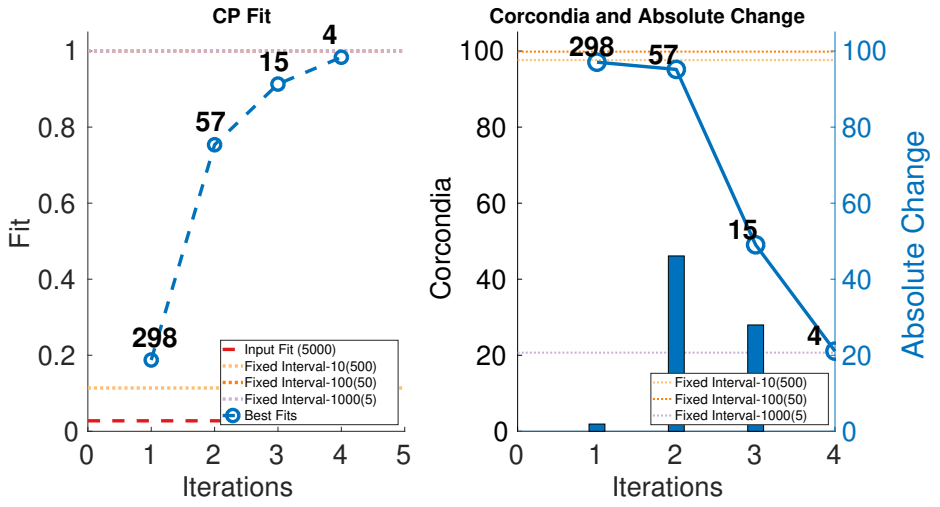


Figure 5.4: CP fit and CORCONDIA of best fit tensor & its absolute change at each iteration for SD2.

represented by a matrix. To create the semi-synthetic data, we use the step 2 as described in the generation of synthetic case. We take the non-zero elements and randomly distribute non zero entries in each slice over some fixed number of slice. For this dataset we converted the monthly data into weekly, daily and hourly data. Non-zero entries in each slice was distributed over 4 different candidate slices of monthly (roughly aprroximating 4 weeks as a month). In the case of daily each slice of monthly data was distributed over 30 different slices as mentioned in Table 5.2 and finally in the case of hourly each non zero entry in the monthly slice was distributed over 720 slices (24×30).

Results for semi-synthetic data: The leftmost parts of Figures 5.5, 5.6 and 5.7 show the fit of different iterations and rightmost part of the figures Figures 5.5, 5.6 and 5.7 show the CORCONDIA computed at different iterations. In the case of Enron Weekly we see a sudden drop in CORCONDIA after iteration 1 as shown in Figure 5.5 and corresponding tensor is of size $184 \times 184 \times 17$. In the case of Enron Daily we don't see a significant change in

CORCONDIA values in two iterations and corresponding tensors are of size $184 \times 184 \times 78$ and $184 \times 184 \times 5$ giving us tensors of different granularity.

In the case of Enron Hourly we see a drop in CORCONDIA after iteration 1 and 2 as shown in Figure 5.7. In this case practitioner can make choice between a tensor of resolution $184 \times 184 \times 469$ or $184 \times 184 \times 34$ depending on what evaluation metric they value more, fit, CORCONDIA or both. Tensor after iteration 2 ($184 \times 184 \times 34$) seems to have good score for both fit and CORCONDIA whereas Tensor after iteration 1 has good COROCONIA score but not a good CP fit.

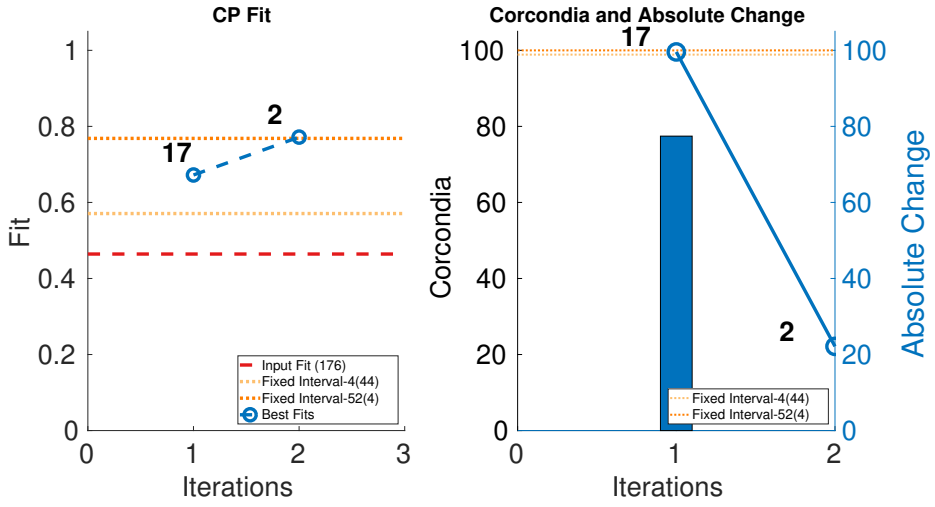


Figure 5.5: CP fit and CORCONDIA of best fit tensor & its absolute change at each iteration for Enron Weekly.

5.4.5 Data mining case study

Chicago crime dataset: For our case study we use a dataset provided by the city of

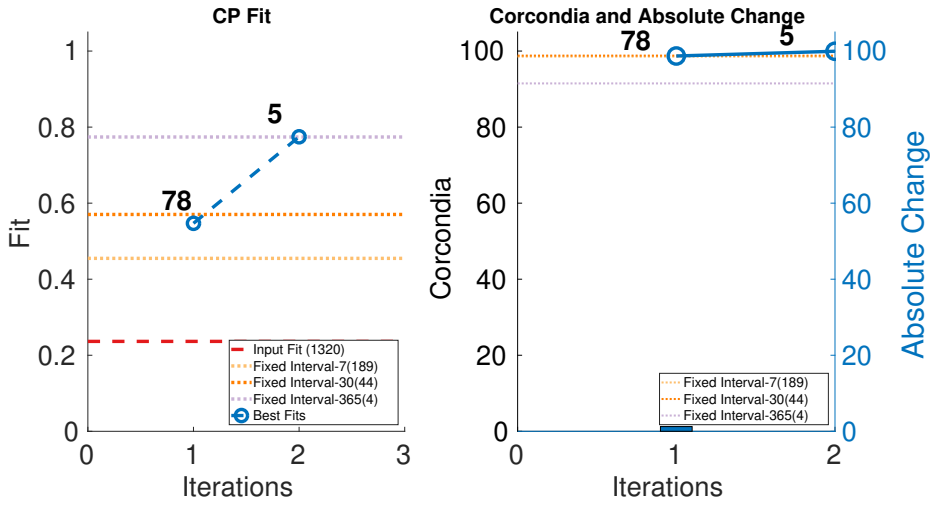


Figure 5.6: CP fit and CORCONDIA of best fit tensor & its absolute change at each iteration for Enron Daily.

Chicago[5] that records different types of crime committed in different areas of the city over a period of time[81]. The tensor we create has modes (area, crime, timestamp), where "community area" and "crime" are discretized by the city of Chicago and "timestamp" is the coarsely aggregated (hourly) timestamp. The dates that we focused on span a period of 7 years, between December 13, 2010 to December 11, 2017.

We ran IceBreaker++ on this dataset which of size $77 \times 32 \times 61321$, and in right most part of the Figure 5.8 we show its CORCONDIA for each iterations and we observe that iterations 3, 4, 5 has high value of CORCONDIA, which would suggest they offer a resolution with an exploitable structure. Iteration 1 and 2 also have decent CORCONDIA value. Given these two range of CORCONDIA values, we decided to drill down and look into the actual tensor components that can be extracted from those different tensors. In the interest of space, we took the tensor returned by iteration 2 as $\underline{\mathbf{X}}_1$, the tensor $\underline{\mathbf{X}}_2$ and

[5] https://data.cityofchicago.org/Public-Safety/Crimes-2001-to-Present/ijzp-q8t2

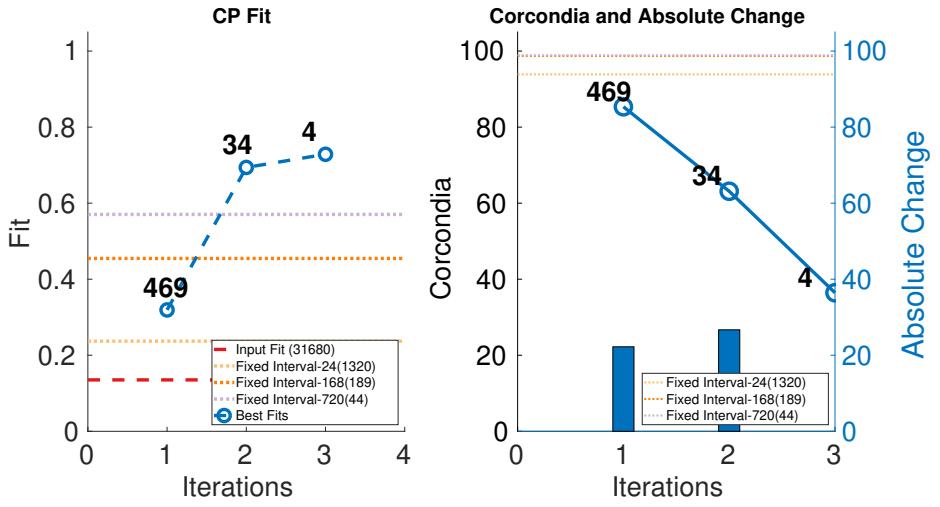


Figure 5.7: CP fit and CORCONDIA of best fit tensor & its absolute change at each iteration for Enron Hourly.

tensor $\underline{\mathbf{X}}_3$ returned by iteration 3 and 4 respectively. Tensor $\underline{\mathbf{X}}_1$ contains three high-quality components, whereas $\underline{\mathbf{X}}_2$ and $\underline{\mathbf{X}}_3$ contains two.

Figure 5.9, 5.10 and 5.11 shows sets of patterns[6] for $\underline{\mathbf{X}}_1$, $\underline{\mathbf{X}}_2$ and $\underline{\mathbf{X}}_3$ respectively: interestingly, factor 1 of $\underline{\mathbf{X}}_1$ and factor 1 of $\underline{\mathbf{X}}_2$ pertain to the similar spatial and criminal pattern. As shown in figure 5.10 and 5.11 we observed that both factors of tensor $\underline{\mathbf{X}}_2$ and $\underline{\mathbf{X}}_3$ pertain to the similar spatial and criminal patterns. In summary, tensors $\underline{\mathbf{X}}_1$, $\underline{\mathbf{X}}_2$ and $\underline{\mathbf{X}}_3$ capture similar interpretable patterns over different temporal resolutions.

[6]We omit plotting the temporal mode since we lack external information that we can potentially correlate it with, however, an analyst with such side information can find the different time resolutions of $\underline{\mathbf{X}}_1$, $\underline{\mathbf{X}}_2$ and $\underline{\mathbf{X}}_3$ useful.

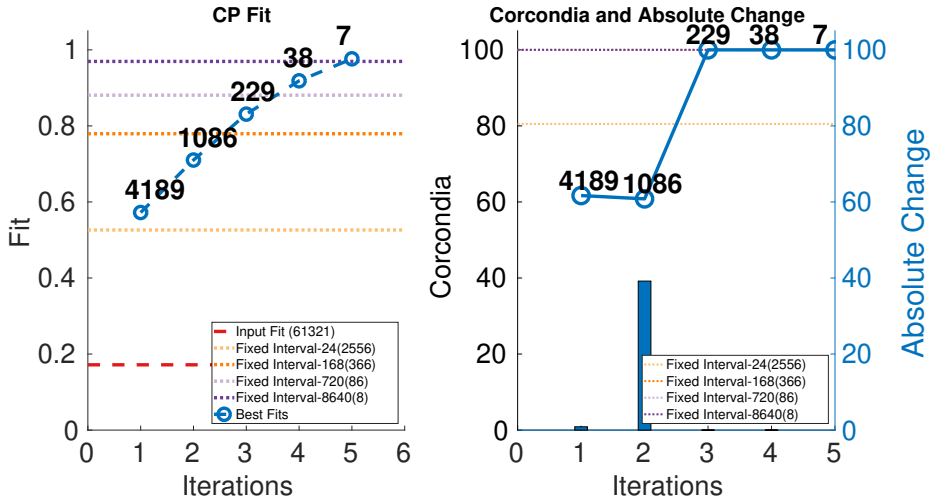


Figure 5.8: CP fit and CORCONDIA of best fit tensor & its absolute change at each iteration for Chicago Crime Dataset.

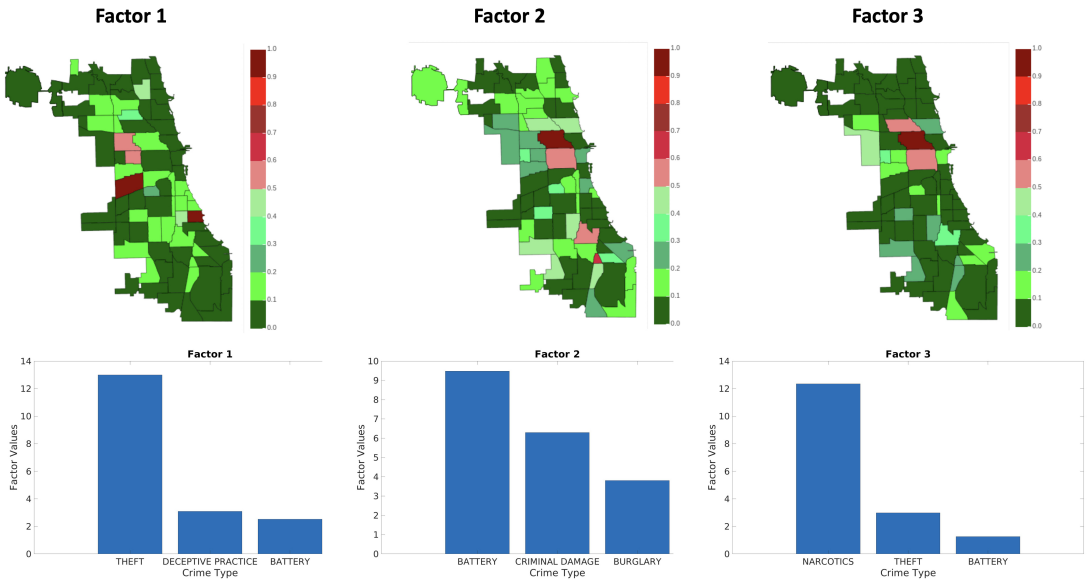


Figure 5.9: Analyzing the Chicago data from Iteration-2 ($\underline{\mathbf{X}}_1$)

.Chicago heatmap value ranges from 0.0 to 1.0

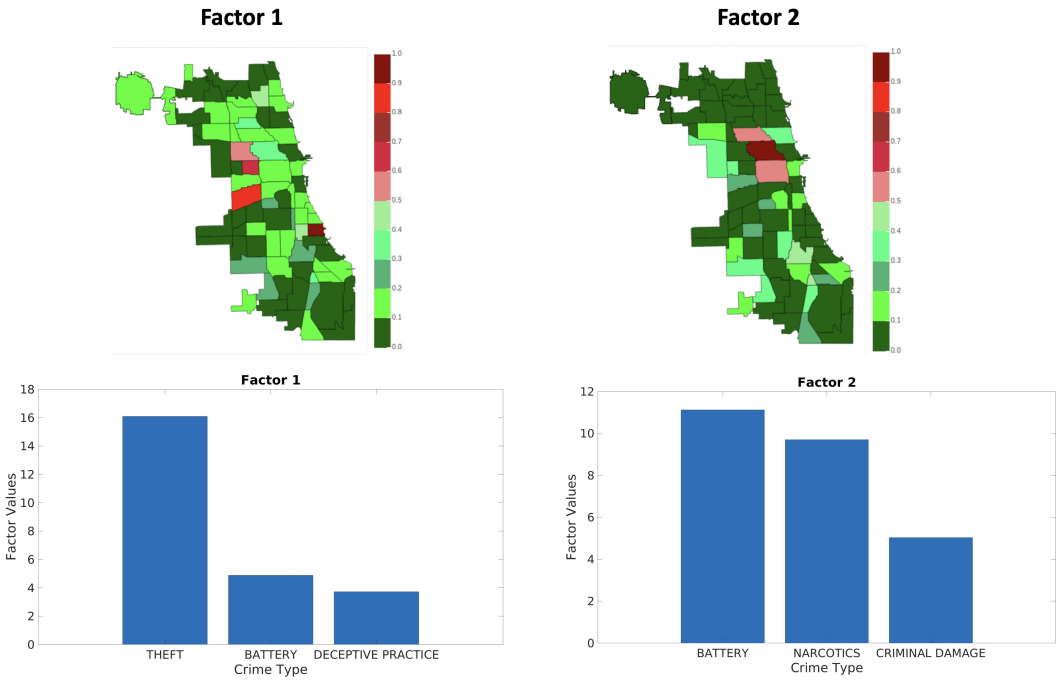


Figure 5.10: Analyzing the Chicago data from Iteration-3 ($\underline{\mathbf{X}}_2$). Chicago heatmap value ranges from 0.0 to 1.0

Comparison against fixed aggregation: A natural question is whether the results are qualitatively "better" than the ones by a fixed aggregation. Answering this question heavily depends on the application at hand, however, here we attempt to quantify this in the following way: intuitively, a good set of components offers more *diversity* in how much of the data it covers. For instance, a practitioner would prefer a set of results for the Chicago crime dataset where the components span most of the regions of the city and uncover diverse patterns of crime, over a set of components that seem to uncover a particular type of crime. Even though there may be a number of confounding factors, aggregating on a regular time interval may be very good in capturing periodic activity (in this example, crime that exhibits

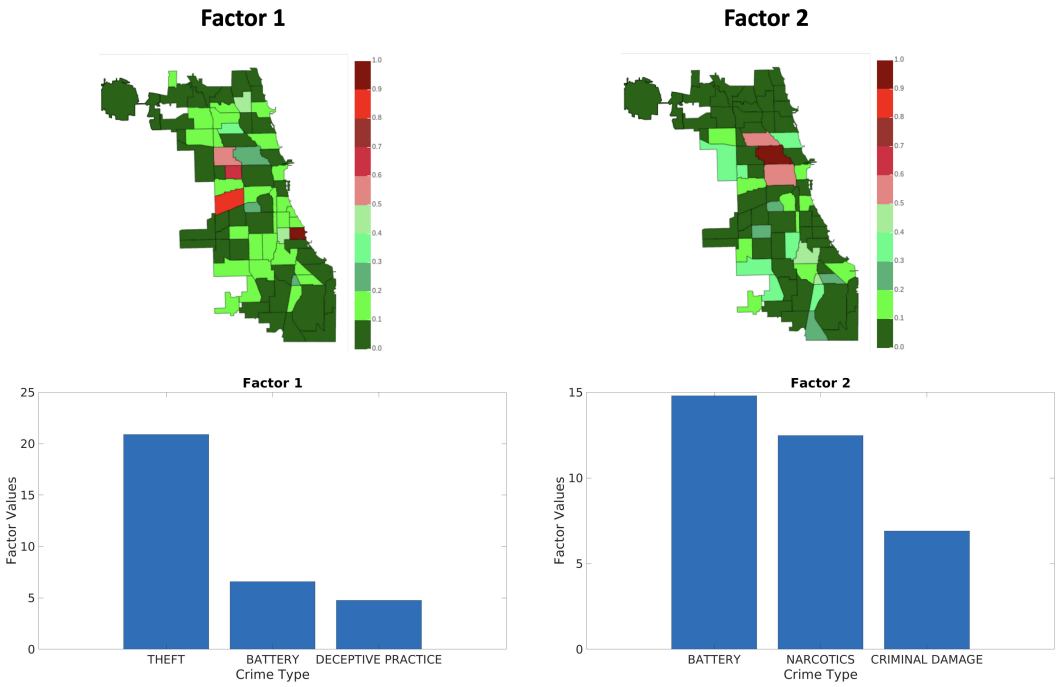


Figure 5.11: Analyzing the Chicago data from Iteration-4 ($\underline{\mathbf{X}}_3$). Chicago heatmap value ranges from 0.0 to 1.0

normal periodicity that happens to coincide with the aggregation resolution we have chosen), whereas aggregating adaptively may help discover structure that is more erratic and more surprising. In order to capture this and test this hypothesis, we compute the coverage of entities for the first and second mode of the tensor (i.e., areas of Chicago and crime types in this example) in all the discovered components: for each component, we measure the top-k entities, and through that we compute the empirical probability distribution of all entities in the results. A more preferable set of results will have a higher coverage, resulting in a distribution with higher entropy. In Table 5.3 we show the entropy for both modes 1 and 2 for IceBreaker++ and for the different fixed aggregations (averaged over 10 different

runs), where IceBreaker++ overall offers more diverse patterns in both space and criminal activity.

5.5 Related Work

To the best of our knowledge, this is the first attempt at formalizing and solving this problem, especially as it pertains in the tensor and multi-aspect data mining domain. Nevertheless, there has been significant amount of work on temporal aggregations in graphs [82, 83, 84] and in finding communities in temporal graphs [32]. In the graph literature, the closest work to ours is [82], in which the authors looks at aggregating stream of temporal edges to produce sequence of structurally mature graphs based on a variety of network properties.

In the tensor literature, [5] are solving the inverse of this problem, where the goal is to disaggregate a tensor. Concurrently to our work, Kwon et al. [49] develop a streaming CP decomposition that works on the original granularity of the data, instead of preprocessing the tensor in order to identify one or more optimal aggregations. We reserve a full investigation of connections between our problem formulation and Kwon et al. [49] for future work.

5.6 Contributions

Our contributions in this work are as follows:

- **Novel Problem Formulation**: We formally define the problem of optimally aggregating a tensor, which is formed from raw sparse data in their original level of

aggregation, into a tensor with exploitable and interpretable structure. We further show that solving this problem optimally is computationally intractable.

- **Practical Algorithm**: We propose a practical, efficient, and effective algorithm that is able to produce tensors with exploitable structure from raw data without incurring the combinatorial cost of the optimal solution. Our proposed method follows a greedy approach, where at each step we decide whether different "slices" of the tensor are aggregated based on a variety of intuitive functions that characterize the "goodness of structure" locally.

- **Experimental Evaluation**: We extensively evaluate our proposed method on synthetic, semi-synthetic data, and in real data where we use popular heuristic measures of structure goodness to measure success. Furthermore, we conduct a data mining case study on a large real dataset of crime over time in Chicago, where we identify interpretable hidden patterns in multiple time resolutions.

Algorithm 4 IceBreaker

Require: Tensor $\underline{\mathbf{X}}$ of dimension $I \times J \times K$

Ensure: Tensor $\underline{\mathbf{Y}}$ of dimension $I \times J \times K_1$and matrix $\mathbf{W}$ of size $K_1 \times K$

1: $i = 1; j = 2$

2: $previousValue = UtilityFunction(X(:,:,i))$

3: **while** $j \leq K$ **do**

4: $currentValue = UtilityFunction(sum(X(:,:,i:j),3)$

5: **if** $previousValue \gtreqless currentValue$ **then**

6: j = j+1 {Aggregate Slice}

7: **else**

8: {Create a New Slice}

Add a row in $\mathbf{W}$ with value as 1 for indices i to $j-1$.

{Update indices for next candidate slice}

9: $i = j; j = j + 1;$

10: previousValue = UtilityFunction(X(:,:,i));

11: **end if**

12: **end while**

13: $\underline{\mathbf{Y}} = \underline{\mathbf{X}} \times_3 \mathbf{W}$

14: return $\underline{\mathbf{Y}}$ and $\mathbf{W}$

Algorithm 5 IceBreaker++

Require: Tensor $\underline{\mathbf{Y}}$ of dimension $I \times J \times K$

Ensure: One Tensor for each iteration

while $K \leq 1$ **do**
  for all Uitlity Functions **do**
    $[\underline{\mathbf{Z}}, \mathbf{W}]$ = IceBreaker $(\underline{\mathbf{Y}})$
  end for
  Select $\underline{\mathbf{Z}}$ with the best Realtive fit
  {Third mode dimension}
  $K_1 = size(\underline{\mathbf{X}}, 3)$
  if $K_1 == K$ **then**
    break;
  else
    $K = K1$
    $\underline{\mathbf{Y}} = \underline{\mathbf{Z}}$
  end if
end while
return one Tensor for each iteration

	Iteration 1	Iteration 2	Iteration 3	Iteration 4	Iteration 5	Fixed Interval-24	Fixed Interval-168	Fixed Interval-720	Fixed Interval-8640
Area	**2.8554**	2.6810	2.5850	2.5850	2.5850	2.7255	2.5850	2.5850	2.5850
Crime	**2.8783**	2.7255	2.2516	2.2516	2.0850	2.4362	2.2516	2.2516	2.1183

Table 5.3: Entropy of top-3 components in factors for area and crime type

Chapter 6

Harvester: Principled Factorization-based Tensor Temporal Granularity Estimation

Conventional Assumption:

Tensors created using raw data or fixed window aggregation have exploitable structure for analysis.

Contribution:

HARVESTER provides multiple tensors of aggregated resolution with high quality of latent structure.

6.1 Introduction

Tensor decomposition methods have been used to find latent structures in multi-modal data in a wide variety of applications like web link analysis [45], social network analysis [8, 63], brain data analysis [2], health care data analysis [39, 75, 90] and many more. When dealing with data that is temporal in nature, the granularity of data plays a crucial role in determining its usability in any data mining or machine learning algorithms. In applications like web mining, social networks or computer network analysis, if the data is collected at very fine temporal granularity the resulting tensor can be extremely sparse and noisy, in addition to being very high-dimensional. Unfortunately, tensors of that form are typically not amenable to popular and widely-used tensor decompositions[1], which seek to identify low-rank latent structure in the data.

There is a considerable amount of work that deals with the temporal nature of data in graphs [82, 83, 84] and tensors [32, 50]. In [32] authors try to learn overlapping communities across times by imposing smoothness on the temporal mode motivated by the fact that granularity may not be perfect; while in [50] the authors propose a data model and family of online tensor decomposition (CP) algorithms which updates the factor matrices of the CP decomposition in response to the new data point in the original granularity. However these works are different from the problem we are trying to tackle in this work. In this work, we seek the optimal aggregation of the temporal mode, which results in a tensor with the highest "quality" of structure that can be extracted by tensor decomposition. Note that extreme aggregation hides away the temporal information (and the model identifiability

[1] We use the terms "decomposition" and "factorization" interchangeably.

boost that comes with it), while no aggregation leaves us with very sparse and possibly noisy data, which is not amenable to low-rank modeling. Hence the optimal level of aggregation lies somewhere in between these two extremes, and our goal is to find it.

To circumvent the problem of having a long tensor in temporal mode and no exploitable structure, the most intuitive and widely-used approach is to aggregate the temporal mode using certain *fixed window sizes*. For example converting milliseconds to seconds, seconds to minutes, minutes to hours, hours to days, and so on. This solution can work, and has been serving tensor analysis of temporal data well for many years; however, there are a few issues with it, which we seek to mitigate in our work. First, it requires a copious amount of trial-and-error experimentation to evaluate what aggregation level works the best for the given data and tensor decomposition method. Second, and more importantly, tensor decomposition on these fixed size aggregated tensors may result in finding "good" latent structures however there might exist more "natural" aggregation which does not follow any fixed or arbitrary window size aggregations and might provide latent structures that are missed because of fixed window aggregation. To tackle this problem, [69, 72] introduced the problem *Trapped Under Ice* and provided a greedy solution called IceBreaker, which iterates over the temporal mode and decides to aggregate two timestamps together or not based on certain utility functions. [72] also introduced an IceBreaker++, which is recursive algorithm based on IceBreaker, where output of one iteration of IceBreaker is an input to the next iteration. So IceBreaker++ provides multiple candidate tensors of different granularity which can be used for analysis or downstream tasks. This does find a

tensor which provides "good" quality structure however it takes multiple runs and is greedy in nature. So the solution found might be a suboptimal one.

In this work, we introduce a principled way of finding an optimal aggregation of the temporal mode. We take inspiration from [4, 5] which essentially tackles the inverse problem to ours: given multiple views of an aggregated tensor in different modes (temporal and non-temporal), estimate the disaggregated tensor. Motivated by that approach, we propose HARVESTER where we leverage multiple aggregated views of a tensor, derived from the tensor in the original temporal granularity in the temporal mode, and define a decomposition-based approach for recovering the best granularity.

6.2 Problem Formulation

6.2.1 Measures of low rank tensor decomposition quality

We take inspiration from [72, 69] and use the Core Consistency Diagnostic (CORCONDIA) [21] to measure the quality of the factors generated by CP decomposition. In this work we also employ tensor completion [3, 53, 5] as an alternative way to measure the quality of factors of tensor decomposition for a downstream task.

CORCONDIA: The Core Consistency Diagnostic was first introduced in [21] as a new efficient way to discover the number of latent factors in the CP model. A CP decomposition of a tensor $\underline{\mathbf{X}}$ can be modelled as a restricted tucker3 model, if $R_1 = R_2 = R_3 = F$ and factor matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$ are not orthogonal. The core tensor of the Tucker model in that case is a super diagonal tensor. So if CP is a good model of the given data,

then the Tucker model for the same (now fixed) $\mathbf{A}, \mathbf{B}, \mathbf{C}$ should yield a core tensor $\underline{\mathbf{G}}$ that should be close to superdiagonal. The CORCONDIA criterion is computed as follows:

$$CORCONDIA = 100 * (1 - \frac{\sum_{i=1}^{F}\sum_{j=1}^{F}\sum_{i=1}^{F}(\underline{\mathbf{G}}(i,j,k) - \underline{\mathbf{I}}(i,j,k))^2}{F})$$

where $\underline{\mathbf{G}}$ is the Tucker core tensor corresponding to the given $\mathbf{A}, \mathbf{B}, \mathbf{C}$, and $\underline{\mathbf{I}}$ is a tensor with ones on its super diagonal.

Like matrix completion, **Tensor Completion** [3, 5, 53] is the task of predicting missing values in data modelled as tensors. Tensor completion has been successfully used in applications like recommendation systems, social networks, etc. In this work, we employ tensor completion as one of downstream tasks to determine the usability of the aggregated data model.

6.2.2 Problem Definition

Consider a tensor $\underline{\mathbf{X}} \in \mathbb{R}^{I \times J \times K}$ which has extremely fine raw temporal granularity, and is very long and sparse/noisy in the temporal mode (in this example we consider the third mode as the temporal mode) which makes it ill-suited for compact modeling via CP decomposition. Using the original granularity tensor $\underline{\mathbf{X}}$, we create multiple views which are aggregated in the temporal mode given by $\underline{\mathbf{Y}}^1 \in \mathbb{R}^{I \times J \times K_1}$ and $\underline{\mathbf{Y}}^2 \in \mathbb{R}^{I \times J \times K_2}$ such that $K_2 < K_1 < K$. Given only these views, we try to find an aggregated tensor $\underline{\mathbf{Y}} \in \mathbb{R}^{I \times J \times K^*}$ which has good "quality" tensor decomposition such that $K^* < K_2 < K_1 < K$. Towards this end, we need to make a reasonable assumption – see section 6.3 for details.

Given a tensor $\underline{\mathbf{X}} \in \mathbb{R}^{I \times J \times K}$ and multiple view of the tensor $\underline{\mathbf{X}}$, $\underline{\mathbf{Y}}^1 \in \mathbb{R}^{I \times J \times K_1}$ and $\underline{\mathbf{Y}}^2 \in \mathbb{R}^{I \times J \times K_2}$, where $K_2 < K_1 < K$. Find an aggregated tensor $\underline{\mathbf{Y}} \in \mathbb{R}^{I \times J \times K^*}$ where $K^* < K_2 < K_1 < K$, such that its temporal mode latent factor matrix is an optimal low-rank compression of the original temporal mode latent factor, while respecting the temporal sequence of indices in that mode.

Creating Views: In this work, we focus on temporal data so the views are created using some fixed window aggregation. For example if the original tensor $\underline{\mathbf{X}}$ granularity is seconds, some possible views for $\underline{\mathbf{Y}}^1$ and $\underline{\mathbf{Y}}^2$ can be of minute, hour, day, etc granularity. We can create these views by using 3-mode product of tensor $\underline{\mathbf{X}}$ with a matrix $\mathbf{W}$. To give a more concrete example, lets assume $\underline{\mathbf{X}}$ is of size $I \times J \times 9$ and if we want to aggregate the third mode on some fixed window size, say 3, so $\mathbf{W}$ is of size 3×9 and takes the form:

$$\mathbf{W} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

The resulting tensor will be of size $I \times J \times 3$, where the first 3 slices (1 to 3) are aggregated into one slice, the next 3 slices (4 to 6) into a second slice and last three slices (7 to 9) into the third slice. In this work, when we aggregate slices we add the values corresponding to the same (i, j) indices, but, depending on the application, logical or can also be used as the aggregation operator.

6.3 Proposed Method

In this section we present our optimization based solution to the transformed version of the Trapped Under Ice problem introduced in [72]. Consider a three mode tensor $\underline{\mathbf{X}} \in \mathbb{R}^{I \times J \times K}$ where I, J denote the dimensions of the first two modes and K that of the third mode which is temporal in nature and has fine granularity, making tensor $\underline{\mathbf{X}}$ extremely sparse and having no exploitable latent structure. The CP decomposition of tensor $\underline{\mathbf{X}}$ for a rank F is given by:

$$\underline{\mathbf{X}} \approx [\![\mathbf{A},\ \mathbf{B},\ \mathbf{C}]\!] \tag{6.1}$$

Where $\mathbf{A} \in \mathbb{R}^{I \times F}$, $\mathbf{B} \in \mathbb{R}^{J \times F}$ and $\mathbf{C} \in \mathbb{R}^{K \times F}$.

Let us assume there exists an oracle tensor $\underline{\mathbf{Y}}$, which has perfect latent structure, which is created by the third mode-product of the tensor $\underline{\mathbf{X}}$ with a matrix W_{opt} which has special structure as specified in Section 6.2.2

$$\underline{\mathbf{Y}} = \underline{\mathbf{X}} \times_3 \boldsymbol{W_{opt}} \tag{6.2}$$

W_{opt} is of size $K_{\text{opt}} \times K$ so the dimension of $\underline{\mathbf{Y}}$ is $I \times J \times K_{\text{opt}}$, such that $K_{\text{opt}} << K$. The CP decompositon of oracle tensor $\underline{\mathbf{Y}}$ is given by

$$\underline{\mathbf{Y}} \approx [\![\mathbf{A},\ \mathbf{B},\ \mathbf{C_{opt}}]\!] \tag{6.3}$$

where matrix $\mathbf{C_{opt}}$ is of size $K_{\text{opt}} \times F$

The key assumption we make is that $\mathbf{C}$ can be obtained by some dimensionality-expanding linear transformation of the unknown (latent) $\mathbf{C}_{\text{opt}}$. This is tantamount to assuming that there exists low rank matrix structure in the temporal mode.

$$\mathbf{C} = \mathbf{M}\mathbf{C}_{\text{opt}} \tag{6.4}$$

where $\mathbf{M} \in \mathbb{R}^{K \times K_{\text{opt}}}$ and $\mathbf{M}$ is a tall matrix.

We create $m \geq 2$ views of tensor $\underline{\mathbf{X}}$ as follows:

$$\underline{\mathbf{Y}}^1 = \underline{\mathbf{X}} \times_3 \boldsymbol{W_1} = [\![\mathbf{A},\ \mathbf{B},\ \mathbf{C_1}]\!];\ \mathbf{C_1} = \mathbf{W_1}\mathbf{C} \tag{6.5}$$

$$\underline{\mathbf{Y}}^2 = \underline{\mathbf{X}} \times_3 \boldsymbol{W_2} = [\![\mathbf{A},\ \mathbf{B},\ \mathbf{C_2}]\!];\ \mathbf{C_2} = \mathbf{W_2}\mathbf{C} \tag{6.6}$$

$$\vdots$$

$$\underline{\mathbf{Y}}^m = \underline{\mathbf{X}} \times_3 \boldsymbol{W_m} = [\![\mathbf{A},\ \mathbf{B},\ \mathbf{C_m}]\!];\ \mathbf{C_m} = \mathbf{W_m}\mathbf{C} \tag{6.7}$$

Before we formalize the problem, we need to address some issues regarding the problem

1. Since we don't have access to the oracle tensor $\underline{\mathbf{Y}}$ we cannot directly estimate $\mathbf{C_{opt}}$.
2. Most importantly, we don't know the size K_{opt} for the matrix $\mathbf{C_{opt}}$. Essentially we don't know what the perfect aggregation of the third mode is supposed to be. In [4, 5], authors work on a problem where given different views of aggregated tensors, they find the underlying disaggregated tensor, which is an inverse of our problem. But they go from coarser to finer resolution, whereas we go the other way around – and we do not know what is the "right" K_{opt} beforehand.

Thus, we seek to estimate $\widetilde{\mathbf{C}}$ as shown in the below equation 6.9, which is suppose to be good approximation of $\mathbf{C_{opt}}$. Using equation 6.4 and the fact that $\mathbf{C}_i = \mathbf{W}_i\mathbf{C}$ per equations 6.5-6.7, we have,

$$\mathbf{C_i} = \mathbf{W}_i\mathbf{M}\mathbf{C}_{opt} \tag{6.8}$$

In the above equation, we know $\mathbf{C_i}$ and $\mathbf{W_i}$, but we do not know $\mathbf{M}$ and $\mathbf{C}_{opt}$ – not even K_{opt}. As a result, we do not know the product $\mathbf{W}_i\mathbf{M}$. How can we bypass this seemingly insurmountable challenge? We propose to bring in ideas from sparse regression. The idea is to over-parameterize the product $\mathbf{W}_i\mathbf{M}$ as a new matrix variable $\mathbf{P_i}$, and use a sparse diagonal row-selection matrix to pick up the essential (reduced-dimension) row span of $\mathbf{C}_{opt}$. In more detail, we approximate

$$\mathbf{C_i} \approx \mathbf{P_i}\mathbf{\Lambda}\widetilde{\mathbf{C}} \tag{6.9}$$

where $\mathbf{P_i}$ is of size $K_i \times K$, $\widetilde{\mathbf{C}}$ is of size $K \times F$, and $\mathbf{\Lambda}$ is a sparse diagonal matrix of size $K \times K$, whose zero entries are meant to strike out columns of $\mathbf{P_i}$ and corresponding rows of $\widetilde{\mathbf{C}}$. The product $\mathbf{P_i}\mathbf{\Lambda}$ is a proxy for $\mathbf{W_i}\mathbf{M}$, with the diagonal $\mathbf{\Lambda}$ effectively providing us with a tall matrix of *a priori* unknown number of columns K_{opt}.

Using equation 6.9, we define our optimization problem as follows:

$$\mathcal{L} = \min_{\widetilde{\mathbf{C}},\mathbf{P_i},\mathbf{\Lambda}} \sum_{i=1}^{m} ||\boldsymbol{C_i} - \boldsymbol{P_i}\mathbf{\Lambda}\widetilde{\boldsymbol{C}}||_F^2 + \alpha||\widetilde{\boldsymbol{C}}^{\boldsymbol{T}}||_F^2 + \beta||\mathbf{\Lambda}||_1 + \gamma||\boldsymbol{P_i}||_F^2 \tag{6.10}$$

We optimize equation 6.10 for matrices $\widetilde{\mathbf{C}}$, $\mathbf{\Lambda}$ and $\mathbf{P_i}$. the sparsity or L-1 regularization constraint on $\mathbf{\Lambda}$ is there to help reduce the granularity of the the $\widetilde{\mathbf{C}}$. The non-zero entries in the diagonal of $\mathbf{\Lambda}$ control the final granularity of the resulting tensor.

Discussion: Since we are trying to find the best aggregation in the third mode and we assume that there exists a low dimensional structure in that mode, one may ask why not

just perform singular value decomposition (SVD) on the third mode matricization of the tensor and use the top-k right singular vectors as the basis for the aggregation, which would yield the best compression in the least squares sense.

$$[\mathbf{U}, \mathbf{S}, \mathbf{V}] = \text{svd}(\mathbf{X_3})$$

$$\underline{\mathbf{Y}} = \underline{\mathbf{X}} \times_3 \mathbf{V}_k$$

$$\underline{\mathbf{Y}} \approx [\![\mathbf{A},\ \mathbf{B},\ \mathbf{C}_{opt}]\!]$$

In our case, this does not solve the problem we set out to tackle, because even though there may be low-rank structure in the temporal mode, captured by the aggregation matrix $\mathbf{V}_k$, this aggregation does not respect the temporal sequence of slices, therefore is not appropriate for recovering the type of aggregation that solves our problem.

6.3.1 Proposed method: Harvester

In this section we present our method HARVESTER to solve the problem in equation 6.10. We solve it using non negative multiplicative update (NMU) [51, 48] in an alternating fashion. In this section we derive the update steps only using two views but this can easily be extended to more views. Using tensor $\underline{\mathbf{X}}$ we create two views for that tensor namely $\underline{\mathbf{Y}}^1$ and $\underline{\mathbf{Y}}^2$ which have less resolution then the original tensor. To create these views we multiply the third mode of the tensor with a $\mathbf{W_1}$ and $\mathbf{W_2}$ matrix.

$$\underline{\mathbf{Y}}^1 = \underline{\mathbf{X}} \times_3 \boldsymbol{W_1} \tag{6.11}$$

$$\underline{\mathbf{Y}}^2 = \underline{\mathbf{X}} \times_3 \boldsymbol{W_2} \tag{6.12}$$

We create a tensor $\underline{\mathbf{Z}}$ by stacking tensor $\underline{\mathbf{Y1}}$ and $\underline{\mathbf{Y2}}$ onto one another hence making them coupled in first two mode. The dimension of tensor $\underline{\mathbf{Z}}$ is $I \times J \times (K_1 + K_2)$. Then we per-

form CP decomposition on tensor $\underline{\mathbf{Z}}$ which yields factor matrices need for our optimization algorithm. We split factor matrix $\mathbf{D}$ into $\mathbf{C_1}$ and $\mathbf{C_2}$ based on their respective dimensions as shown below:

$$\begin{aligned} \underline{\mathbf{Z}} &= [\underline{\mathbf{Y}}^1; \underline{\mathbf{Y}}^2] \\ \underline{\mathbf{Z}} &\approx [\![\mathbf{A},\ \mathbf{B},\ \mathbf{D}]\!] \\ \mathbf{C_1} &= \mathbf{D}(\mathbf{1} : \mathbf{K_1}, :) \\ \mathbf{C_2} &= \mathbf{D}(\mathbf{K_1} + \mathbf{1} : \mathbf{K_2}, :) \end{aligned} \tag{6.13}$$

We use equation 6.9 and 6.13 to create our optimization problem that follows from equation 6.10. The optimization steps to solve equation 6.14 are derived along the lines of [76].

$$\begin{aligned} \mathcal{L} = \min_{\widetilde{\mathbf{C}}, \mathbf{P_1}, \mathbf{P_2}\mathbf{\Lambda}} & \underbrace{||\boldsymbol{C_1} - \boldsymbol{P_1}\mathbf{\Lambda}\widetilde{\boldsymbol{C}}||_F^2}_{f} + \underbrace{||\boldsymbol{C_2} - \boldsymbol{P_2}\mathbf{\Lambda}\widetilde{\boldsymbol{C}}||_F^2}_{g} + \\ & \underbrace{\alpha||\widetilde{\boldsymbol{C}}^{\boldsymbol{T}}||_F^2 + \beta||\mathbf{\Lambda}||_1 + \gamma||\boldsymbol{P_1}||_F^2 + \gamma||\boldsymbol{P_2}||_F^2}_{h} \end{aligned} \tag{6.14}$$

The gradient of the loss function $\mathcal{L}$ in equation 6.14 with respect to $\widetilde{\mathbf{C}}$:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \widetilde{\mathbf{C}}} &= \frac{\partial f}{\partial \widetilde{\mathbf{C}}} + \frac{\partial g}{\partial \widetilde{\mathbf{C}}} + \frac{\partial h}{\partial \widetilde{\mathbf{C}}} \\ \frac{\partial f}{\partial \widetilde{\mathbf{C}}} &= \frac{\partial}{\partial \widetilde{\mathbf{C}}}(||\boldsymbol{C_1} - \boldsymbol{P_1}\mathbf{\Lambda}\widetilde{\boldsymbol{C}}||_F^2) \\ \frac{\partial f}{\partial \widetilde{\mathbf{C}}} &= \frac{\partial}{\partial \widetilde{\mathbf{C}}} \operatorname{Tr}((\boldsymbol{C_1} - \boldsymbol{P_1}\mathbf{\Lambda}\widetilde{\boldsymbol{C}})(\boldsymbol{C_1} - \boldsymbol{P_1}\mathbf{\Lambda}\widetilde{\boldsymbol{C}})^T) \\ \frac{\partial f}{\partial \widetilde{\mathbf{C}}} &= \frac{\partial}{\partial \widetilde{\mathbf{C}}} \operatorname{Tr}(\mathbf{C_1}\mathbf{C_1^T} - \mathbf{C_1}\widetilde{\mathbf{C}}\mathbf{\Lambda}^\mathbf{T}\mathbf{P_1^T} - \mathbf{P_1}\mathbf{\Lambda}\widetilde{\mathbf{C}}\mathbf{C_1^T} + \mathbf{P_1}\mathbf{\Lambda}\widetilde{\mathbf{C}}\widetilde{\mathbf{C}}^\mathbf{T}\mathbf{\Lambda}\mathbf{P_1^T}) \\ \frac{\partial f}{\partial \widetilde{\mathbf{C}}} &= -\mathbf{\Lambda}^\mathbf{T}\mathbf{P_1^T}\mathbf{C_1} - \mathbf{\Lambda}^\mathbf{T}\mathbf{P_1^T}\mathbf{C_1} + \mathbf{\Lambda}^\mathbf{T}\mathbf{P_1^T}\mathbf{P_1}\mathbf{\Lambda}\widetilde{\mathbf{C}} + \mathbf{\Lambda}^\mathbf{T}\mathbf{P_1^T}\mathbf{P_1}\mathbf{\Lambda}\widetilde{\mathbf{C}} \\ \frac{\partial f}{\partial \widetilde{\mathbf{C}}} &= \mathbf{2\Lambda}^\mathbf{T}\mathbf{P_1^T}\mathbf{P_1}\mathbf{\Lambda}\widetilde{\mathbf{C}} - \mathbf{2\Lambda}^\mathbf{T}\mathbf{P_1^T}\mathbf{C_1} \end{aligned} \tag{6.15}$$

We can similarly derive $\frac{\partial g}{\partial \widetilde{\mathbf{C}}}$,

$$\frac{\partial g}{\partial \widetilde{\mathbf{C}}} = \mathbf{2\Lambda}^\mathbf{T}\mathbf{P_2^T}\mathbf{P_2}\mathbf{\Lambda}\widetilde{\mathbf{C}} - \mathbf{2\Lambda}^\mathbf{T}\mathbf{P_2^T} \tag{6.16}$$

Gradient of h w.r.t $\widetilde{\mathbf{C}}$

$$\begin{aligned}\frac{\partial h}{\partial \widetilde{\mathbf{C}}} &= \frac{\partial}{\partial \widetilde{\mathbf{C}}} \operatorname{Tr}(\alpha \widetilde{\mathbf{C}}^{\mathbf{T}} \widetilde{\mathbf{C}}) \\ &= \mathbf{2}\alpha \widetilde{\mathbf{C}}\end{aligned} \tag{6.17}$$

Combining equation 6.15, 6.16 and 6.17

$$\frac{\partial \mathcal{L}}{\partial \widetilde{\mathbf{C}}} = 2(\mathbf{\Lambda}^T \mathbf{P_1}^T \mathbf{P_1} \mathbf{\Lambda} \widetilde{\mathbf{C}} + \mathbf{\Lambda}^T \mathbf{P_2}^T \mathbf{P_2} \mathbf{\Lambda} \widetilde{\mathbf{C}} - \mathbf{\Lambda}^T \mathbf{P_1}^T \mathbf{C_1} - \mathbf{\Lambda}^T \mathbf{P_2}^T \mathbf{C_2} + \alpha \widetilde{\mathbf{C}}) \tag{6.18}$$

Setting $\frac{\partial \mathcal{L}}{\partial \widetilde{\mathbf{C}}}$ to zero and using non-negative multiplicative update method to compute update step. $\circledast$ here represents element wise product and fraction equation below represents element wise division.

$$\boxed{\widetilde{\mathbf{C}} \Leftarrow \widetilde{\mathbf{C}} \circledast \frac{\mathbf{\Lambda}^T \mathbf{P_1}^T \mathbf{C_1} + \mathbf{\Lambda}^T \mathbf{P_2}^T \mathbf{C_2}}{(\mathbf{\Lambda}^T \mathbf{P_1}^T \mathbf{P_1} \mathbf{\Lambda} + \mathbf{\Lambda}^T \mathbf{P_2}^T \mathbf{P_2} \mathbf{\Lambda} + \alpha I)\widetilde{\mathbf{C}}}} \tag{6.19}$$

The gradient of the loss function $\mathcal{L}$ in equation 6.14 with respect to **P1** (or **P2**):

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \mathbf{P_1}} &= \frac{\partial f}{\partial \mathbf{P_1}} + \frac{\partial g}{\partial \mathbf{P_1}} + \frac{\partial h}{\partial \mathbf{P_1}} \\ \frac{\partial f}{\partial \mathbf{P_1}} &= \frac{\partial}{\partial \mathbf{P_1}}(||\boldsymbol{C_1} - \boldsymbol{P_1}\mathbf{\Lambda}\widetilde{\boldsymbol{C}}||_F^2) \\ \frac{\partial f}{\partial \mathbf{P_1}} &= \frac{\partial}{\partial \mathbf{P_1}} \operatorname{Tr}((\boldsymbol{C_1} - \boldsymbol{P_1}\mathbf{\Lambda}\widetilde{\boldsymbol{C}})(\boldsymbol{C_1} - \boldsymbol{P_1}\mathbf{\Lambda}\widetilde{\boldsymbol{C}})^T) \\ &= \frac{\partial}{\partial \mathbf{P_1}} \operatorname{Tr}(\mathbf{C_1}\mathbf{C_1^T} - \mathbf{C_1}\widetilde{\mathbf{C}}\mathbf{\Lambda}^{\mathbf{T}}\mathbf{P_1^T} - \mathbf{P_1}\mathbf{\Lambda}\widetilde{\mathbf{C}}\mathbf{C_1^T} + \mathbf{P_1}\mathbf{\Lambda}\widetilde{\mathbf{C}}\widetilde{\mathbf{C}}^{\mathbf{T}}\mathbf{\Lambda}\mathbf{P_1^T}) \\ &= -\mathbf{C_1}\widetilde{\mathbf{C}}^{\mathbf{T}}\mathbf{\Lambda}^{\mathbf{T}} - \mathbf{C_1}\widetilde{\mathbf{C}}^{\mathbf{T}}\mathbf{\Lambda}^{\mathbf{T}} + \mathbf{P_1}\mathbf{\Lambda}\widetilde{\mathbf{C}}\widetilde{\mathbf{C}}^{\mathbf{T}}\mathbf{\Lambda}^{\mathbf{T}} + \mathbf{P_1}\mathbf{\Lambda}\widetilde{\mathbf{C}}\widetilde{\mathbf{C}}^{\mathbf{T}}\mathbf{\Lambda}^{\mathbf{T}} \\ &= 2(\mathbf{P_1}\mathbf{\Lambda}\widetilde{\mathbf{C}}\widetilde{\mathbf{C}}^{\mathbf{T}}\mathbf{\Lambda}^{\mathbf{T}} - \mathbf{C_1}\widetilde{\mathbf{C}}^{\mathbf{T}}\mathbf{\Lambda}^{\mathbf{T}})\end{aligned} \tag{6.20}$$

Gradient of g with respect to P_1 is zero. Now computing gradient of h with respect to P_1

$$\begin{aligned}\frac{\partial h}{\partial \mathbf{P_1}} &= \frac{\partial}{\partial \mathbf{P_1}} \operatorname{Tr}(\gamma \mathbf{P_1}\mathbf{P_1^T}) \\ &= \mathbf{2}\gamma \mathbf{P_1}\end{aligned} \tag{6.21}$$

Combining equation 6.20 and 6.21

$$\frac{\partial \mathcal{L}}{\partial \mathbf{P_1}} = 2(\mathbf{P_1}\mathbf{\Lambda}\widetilde{\mathbf{C}}\widetilde{\mathbf{C}}^{\mathbf{T}}\mathbf{\Lambda}^{\mathbf{T}} - \mathbf{2}\mathbf{C_1}\widetilde{\mathbf{C}}^{\mathbf{T}}\mathbf{\Lambda}^{\mathbf{T}} + \mathbf{2}\gamma \mathbf{P_1}) \tag{6.22}$$

Setting $\frac{\partial \mathcal{L}}{\partial \mathbf{P_1}}$ to zero and using non-negative multiplicative update method to compute update step. Again $\circledast$ here represents element wise product and fraction equation below represents element wise division.

$$\boxed{\mathbf{P_1} \Leftarrow \mathbf{P_1} \circledast \frac{\mathbf{C_1}\widetilde{\mathbf{C}}^T\mathbf{\Lambda}^T}{\mathbf{P_1}(\mathbf{\Lambda}\widetilde{\mathbf{C}}\widetilde{\mathbf{C}}^T\mathbf{\Lambda}^T + \gamma\mathbf{I})}} \tag{6.23}$$

We can compute update step for P_2 similarly,

$$\boxed{\mathbf{P_2} \Leftarrow \mathbf{P_2} \circledast \frac{\mathbf{C_2}\widetilde{\mathbf{C}}^T\mathbf{\Lambda}^T}{\mathbf{P_2}(\mathbf{\Lambda}\widetilde{\mathbf{C}}\widetilde{\mathbf{C}}^T\mathbf{\Lambda}^T + \gamma\mathbf{I})}} \tag{6.24}$$

The gradient of the loss function $\mathcal{L}$ in equation 6.14 with respect to $\mathbf{\Lambda}$:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \mathbf{\Lambda}} &= \frac{\partial f}{\partial \mathbf{\Lambda}} + \frac{\partial g}{\partial \mathbf{\Lambda}} + \frac{\partial h}{\partial \mathbf{\Lambda}} \\
\frac{\partial f}{\partial \mathbf{\Lambda}} &= \frac{\partial}{\partial \mathbf{\Lambda}}(||\boldsymbol{C_1} - \boldsymbol{P_1}\mathbf{\Lambda}\widetilde{\boldsymbol{C}}||_F^2) \\
\frac{\partial f}{\partial \mathbf{\Lambda}} &= \frac{\partial}{\partial \mathbf{\Lambda}} \operatorname{Tr}\left((\boldsymbol{C_1} - \boldsymbol{P_1}\mathbf{\Lambda}\widetilde{\boldsymbol{C}})(\boldsymbol{C_1} - \boldsymbol{P_1}\mathbf{\Lambda}\widetilde{\boldsymbol{C}})^T\right) \\
&= \frac{\partial}{\partial \mathbf{\Lambda}} \operatorname{Tr}\left(\mathbf{C_1}\mathbf{C_1^T} - \mathbf{C_1}\widetilde{\mathbf{C}}\mathbf{\Lambda}^\mathbf{T}\mathbf{P_1^T} - \mathbf{P_1}\mathbf{\Lambda}\widetilde{\mathbf{C}}\mathbf{C_1^T} + \mathbf{P_1}\mathbf{\Lambda}\widetilde{\mathbf{C}}\widetilde{\mathbf{C}}^\mathbf{T}\mathbf{\Lambda}\mathbf{P_1^T}\right) \\
&= -\mathbf{P_1^T}\mathbf{C_1}\widetilde{\mathbf{C}}^\mathbf{T} - -\mathbf{P_1^T}\mathbf{C_1}\widetilde{\mathbf{C}}^\mathbf{T} + \mathbf{P_1^T}\mathbf{P_1}\mathbf{\Lambda}\widetilde{\mathbf{C}}\widetilde{\mathbf{C}}^\mathbf{T} + \mathbf{P_1^T}\mathbf{P_1}\mathbf{\Lambda}\widetilde{\mathbf{C}}\widetilde{\mathbf{C}}^\mathbf{T} \\
&= 2(\mathbf{P_1^T}\mathbf{P_1}\mathbf{\Lambda}\widetilde{\mathbf{C}}\widetilde{\mathbf{C}}^\mathbf{T} - \mathbf{P_1^T}\mathbf{C_1}\widetilde{\mathbf{C}}^\mathbf{T})
\end{aligned} \tag{6.25}$$

We can similarly derive the derivative of g w.r.t. $\mathbf{\Lambda}$,

$$\frac{\partial g}{\partial \mathbf{\Lambda}} = 2(\mathbf{P_2^T}\mathbf{P_2}\mathbf{\Lambda}\widetilde{\mathbf{C}}\widetilde{\mathbf{C}}^\mathbf{T} - \mathbf{P_2^T}\mathbf{C_2}\widetilde{\mathbf{C}}^\mathbf{T}) \tag{6.26}$$

Next, Based on update steps in Section 2 of [48], we compute the Gradient of h w.r.t. $\mathbf{\Lambda}$

$$\begin{aligned}
\frac{\partial h}{\partial \mathbf{\Lambda}} &= \frac{\partial}{\partial \mathbf{\Lambda}}||\beta\mathbf{\Lambda}||_1 \\
&= \beta\mathbf{E}
\end{aligned} \tag{6.27}$$

Where $\mathbf{E}$ is an matrix of all ones. We would like to reiterate that, although L-1 Norm is not a differentiable function in the entirety of its domain and thus we must instead compute

its sub-gradient, the constraint of Non-Negativity on $\mathbf{\Lambda}$ allows us to circumvent that issue and compute the gradient as long as values of $\mathbf{\Lambda}$ are not negative, which we guarantee by initializing $\mathbf{\Lambda}$ as Non-Negative and following the non-negative multiplicative procedure to ensure that each iterate turns out as non-negative, thereby fullfilling the criterion for differentiablity.

The imposition of L-1 Penalty leads to sparsity [37] and the framework of Multiplicative updates ensures non-negative iterates conditioned on non-negative initial values, thereby yielding a sparse and non-negative $\mathbf{\Lambda}$ which accounts for interactions between latent dimensions of adjacent factor matrices.

Combining equation 6.25, 6.26 and 6.27

$$\frac{\partial \mathcal{L}}{\partial \mathbf{\Lambda}} = \mathbf{2P_1^T P_1 \Lambda \widetilde{C} \widetilde{C}^T} - \mathbf{2P_1^T C_1 \widetilde{C}^T} + \mathbf{2P_2^T P_2 \Lambda \widetilde{C} \widetilde{C}^T} - \mathbf{2P_2^T C_2 \widetilde{C}^T} + \beta \mathbf{E} \tag{6.28}$$

Setting $\frac{\partial \mathcal{L}}{\partial \mathbf{\Lambda}}$ to zero and using non-negative multiplicative update method to compute update step. $\circledast$ represents element wise product and fraction equation below represents element wise division.

$$\boxed{\mathbf{\Lambda} \Leftarrow \mathbf{\Lambda} \circledast \frac{\mathbf{2P_1}^T \mathbf{C_1} \widetilde{\mathbf{C}}^T + \mathbf{2P_2}^T \mathbf{C_2} \widetilde{\mathbf{C}}^T}{\mathbf{2P_1}^T \mathbf{P_1 \Lambda} \widetilde{\mathbf{C}} \widetilde{\mathbf{C}}^T + \mathbf{2P_2}^T \mathbf{P_2 \Lambda} \widetilde{\mathbf{C}} \widetilde{\mathbf{C}}^T + \beta \mathbf{E}}} \tag{6.29}$$

Using the update steps that we derived in the equations 6.19, 6.23, 6.24 and 6.29, we present our algorithm of HARVESTER in Algorithm 6.

Algorithm 6 return the matrices $\mathbf{\Lambda}$, $\widetilde{\mathbf{C}}$, $\mathbf{P_1}$ and $\mathbf{P_2}$ which are needed to construct the aggregated tensor. We can construct the aggregated tensor in the following ways:

$$\mathbf{C_o} = \mathbf{\Lambda} \widetilde{\mathbf{C}}$$

Algorithm 6 HARVESTER

Input: $\mathbf{C_1}$, $\mathbf{C_2}$

Output: $\mathbf{\Lambda}, \widetilde{\mathbf{C}}, \mathbf{P_1}, \mathbf{P_2}$

while Not Converged **AND** Max iteration not reached **do**

  Update $\widetilde{C}$ using equation 6.19

  {making negative entries zero: Matlab notation}

  $\widetilde{C} = max(\widetilde{C}, 0)$

  Update P_1 using equation 6.23

  Update P_2 using equation 6.24 {making negative entries zero: Matlab notation}

  $P_1 = max(P_1, 0)$

  $P_2 = max(P_2, 0)$

  Update Λ using equation 6.29

  {making negative entries zero: Matlab notation}

  $\Lambda = max(\Lambda, 0)$

end while

- We use one of the techniques from [72], namely norm aggregation and apply it to matrix instead of tensor. Essentially we iterate over the rows of $\mathbf{C_o}$, add a candidate row to previous row, if rate of change of norm between the previous row and sum of previous and candidate row is more than a certain threshold. If not candidate row becomes the previous row and process continues until we reach the end. We use that norm aggregated matrix $\mathbf{C_{norm}}$ to construct a tensor.

$$\mathbf{W_{norm}} = aggreagateOnNorm(\mathbf{C_o}, normThreshold)$$

$$\mathbf{C_{norm}} = \mathbf{W_{norm}}\mathbf{C_o}$$

$$\underline{\mathbf{Y}}^{norm} \approx [\![\mathbf{A},\ \mathbf{B},\ \mathbf{C_{norm}}]\!]$$

- Since Λ is not only a diagonal matrix but it's also a diagonally sparse matrix, which is used in update steps to turn off (zero out) the rows of $\widetilde{C}$. We can just use those non zero rows as anchor points to create a matrix $\mathbf{C_{spar}}$ which contains only the non-zero rows of $\mathbf{C_o}$.

$$\mathbf{W_{spar}} = aggreagateZeroRows(\mathbf{C_o})$$

$$\mathbf{C_{spar}} = \mathbf{W_{spar}}\mathbf{C_o}$$

$$\underline{\mathbf{Y}}^{spar} \approx [\![\mathbf{A},\ \mathbf{B},\ \mathbf{C_{spar}}]\!]$$

We use tensors $\underline{\mathbf{Y}}^{norm}$ and $\underline{\mathbf{Y}}^{spar}$ for evaluation.

6.4 Experimental Evaluation

In this section, we demonstrate the effectiveness of our method HARVESTER by performing evaluation on variety of datasets: synthetic, semi-synthetic and real. We implemented our method in Matlab using tensor toolbox [13] and for using CP-WOPT [3] algorithm in tensor toolbox for tensor completion we make use of L-BFGS-B Matlab wrapper [15] which is based on L-BFGS [52].

Hyperparameter Tuning: Our problem formulation has three hyperparameter which need to be tuned namely α, β and γ as in equation 6.14. We use different ranges for the hyperparameters search space used in equation 6.14 based on the dataset, we specify those ranges while discussing the results for those datasets in their respective sub sections. We perform grid search over these values and choose two point from this grid search results based on following conditions:

- **Sparsity threshold:** Number of non zero entries in diagonal of lambda($\mathbf{\Lambda}$) matrix is less then some threshold. For synthetic dataset we set that threshold to be 1/10 of the third dimension of tensor $\underline{\mathbf{X}}$. For example if input tensor of the size $100 \times 100 \times 500$, then the threshold is set at $500/10 = 50$. This threshold can vary based on the dataset.

- **Error threshold:** From all the points that meets the above threshold condition, we find median error and only consider points which have error less than that threshold.

After applying the above two thresholds, we choose two points one which has smallest error on the loss function (min error point) and other one which has the lowest number of non-zero entries in the diagonal of the lambda ($\mathbf{\Lambda}$) matrix (min sparsity point) as shown in

figure 6.1 (two points which are marked as black). We use these two points for our analysis. So for each of this point we create two tensors namely $\underline{\mathbf{Y}}_{norm}$ and $\underline{\mathbf{Y}}_{spar}$ as mentioned in previous section. In total HARVESTER produces 4 candidate tensor for evaluation namely min error $\underline{\mathbf{Y}}_{norm}$, min error $\underline{\mathbf{Y}}_{spar}$, min sparsity $\underline{\mathbf{Y}}_{norm}$, and min sparsity $\underline{\mathbf{Y}}_{spar}$ The result presented in this work for our method HARVESTER will be of one or more of those points unless specified otherwise.

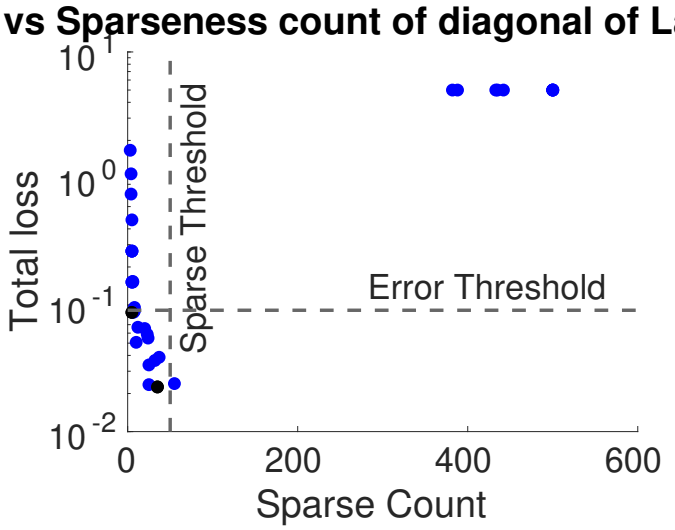


Figure 6.1: Total Loss vs. Sparseness Count of diagonal of Λ matrix for various hyperparameter settings.

6.4.1 Evaluation Metrics and Baseline

We evaluate our method HARVESTER on two model-based measure namely relative fit of the model and CORCONDIA [21, 62] and also on a task-based measure on tensor completion [3].

- **Relative Fit of the model**: We evaluate fit of a tensor for a particular rank given by AUTOTEN [62] using CP_NMU

$$Relative\ Fit = 1 - \Big(\frac{||\underline{\mathbf{X}}_{Input} - \underline{\mathbf{X}}_{computed}||_F}{||\underline{\mathbf{X}}_{Input}||_F}\Big) \tag{6.30}$$

- **CORCONDIA:** As explained in section 6.2.1, we use CORCONDIA to evaluate quality tensor decomposition for given range of ranks. In this work we use CORCONDIA implemented in AUTOTEN [62] only using the frobenius norm version and also we adapt it to use CP_NMU instead of CP_ALS implemented in tensor toolbox [13] as we use non-negative factorization in our work. Higher the CORCONDIA, better the "quality" of the model.

- **Tensor Completion:** This can be viewed as task oriented evaluation, in which we hide 20% of non-zero entries and use CP_WOPT [3] for predicting the missing values. And we report RMSE between the predicted and actual hidden values for the 20% of entries which were marked as missing. Lower the RMSE better the performance on the tensor completion task. For finding the Pareto boundary we instead use $1/RMSE$.

Baseline: We compare our optimization based method HARVESTER against the greedy based method ICEBREAKER++ [72], which introduced the problem of *Trapped Under Ice* and provided a greedy approach to tackle the problem.

We also compare our method against fixed aggregations views generated from the input tensor, which just aggregates the slices based on certain fixed window intervals.

6.4.2 Performance on synthetic datasets

We created the synthetic datasets in an similar fashion as mentioned in the [72]. We first create a sparse tensor of certain sparsity and then we distribute the non-zero entries in each frontal slices $\underline{\mathbf{X}}(:,:,k)$ over certain fixed number of slices(B). We repeat this for every frontal slices in original tensor, every time distributing the non-zero entries to new fixed bucket of slices(B) and concatenate all the buckets to create the dataset. Table 6.1 shows the different synthetic datasets used in the experiments. For example take SD1 (synthetic dataset 1), the original data is of size $100 \times 100 \times 10$, all the non-zero entries in the first slice $\underline{\mathbf{X}}(i,j,1)$ is randomly distributed over 50 slices in the third mode with the same i,j indices. And we repeat this process of other 9 slices in the original tensor where each slice distributed over new set of 50 slices. So finally we end up with a tensor of size $100 \times 100 \times 500$ with high granularity and sparsity, which is then used for creating different views of tensors $\underline{\mathbf{Y}}^1$ and $\underline{\mathbf{Y}}^2$.

We create 4 types of dataset as shown in table 6.1, for each type of dataset we create 10 datasets and we report our findings for these 40 datasets. Hyperparameter ranges used are : for α we use the values $[10^{-4}, 10^{-3}, 10^{-2}]$, for β we use the values $[10^{-2}, 10^{-1}, 10^{0}, 10]$ and for γ we use the values $[10^{-4}, 10^{-3}, 10^{-2}]$.

Behavior of the Loss function

In figure 6.2, we show how does the loss function of our method HARVESTER behave with respect to iterations of the algorithms. Relative Error C1 and Relative Error C2 tracks how does $||\boldsymbol{C_i} - \boldsymbol{P_i}\boldsymbol{\Lambda}\widetilde{\boldsymbol{C}}||_F^2$ changes with iterations for both views. Total Relative Error is

Dataset	Original Dimension ($\underline{\mathbf{X}}_{\text{og}}$)	Rank (R)	Bucket size (B)	Approximate Final Dimension($\underline{\mathbf{X}}$)	View 1 ($\underline{\mathbf{Y}}^1$)	View 2 ($\underline{\mathbf{Y}}^2$)	Number of datasets
SD1	$100 \times 100 \times 10$	5	50	$100 \times 100 \times 500$	$100 \times 100 \times 100$	$100 \times 100 \times 50$	10
SD2	$100 \times 100 \times 10$	10	50	$100 \times 100 \times 500$	$100 \times 100 \times 100$	$100 \times 100 \times 50$	10
SD3	$100 \times 100 \times 10$	30	50	$100 \times 100 \times 500$	$100 \times 100 \times 100$	$100 \times 100 \times 50$	10
SD4	$100 \times 100 \times 100$	20	20	$100 \times 100 \times 2000$	$100 \times 100 \times 400$	$100 \times 100 \times 200$	10

Table 6.1: Table of Synthetic Datasets analyzed

the sum of Relative Error C1 and C2. Total Error keep tracks of the total loss function as defined in equation 6.14. To conserve space, we show results for only one dataset but we do observe the similar behaviour with other datasets as well, depending on the values of the hyperparameters.

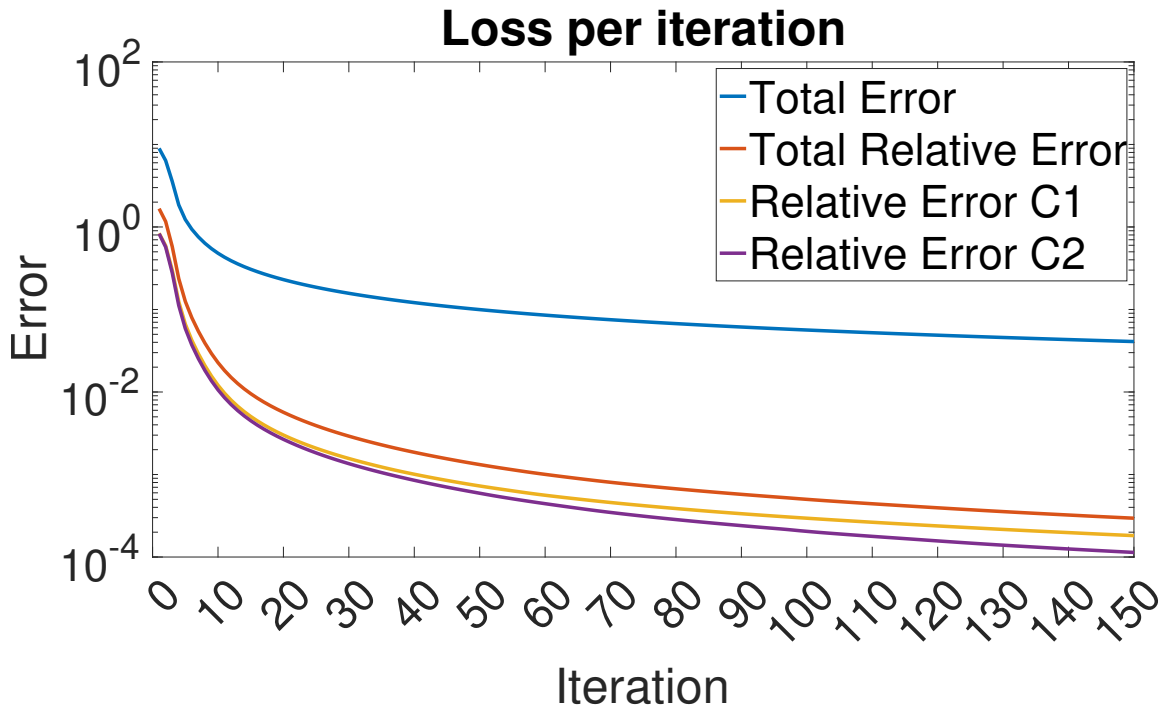


Figure 6.2: SD2: Different Loss vs. iterations of the algorithm for a particular set of hyperparameter values

Effectiveness of Harvester on Synthetic Datasets

We evaluate the quality of our method HARVESTER and baseline method on the three evaluation criteria as mentioned in section 6.4.1. We are seeking a tensor which maximizes CP_FIT, CORCONDIA and minimizes the RMSE for tensor completion (or

maximize 1/RMSE). This is essentially a multi-objective optimization problem, and a widely accepted means of measuring optimality in this case is to identify points in that 3D space which lie in the so-called *Pareto frontier* [54, 41], i.e., points which dominate the rest of the points with respect to the three dimensions/objectives. To that effect, we find the tensors which lie on the Pareto frontier of these evaluation metrics, and we count how frequently a given method produces such tensors. The more frequently a method yields tensors lying on the Pareto frontier, the better the quality of those tensors, thus, the higher performing the method. We perform Pareto finding operation on all the data points obtained using method HARVESTER, ICEBREAKER++, fixed aggregation views of the tensor which we used with our method. Note that we find points lying on Pareto boundary using all three dimensions of the data (i.e., CORCONDIA, Relative Fit and $1/RMSE$), however, for exhibition purposes, we show two scatterplots of pairs of dimensions at a time: In figure 6.3, we plot the CORCONDIA and CP_FIT and in figure 6.4 we plot 1/RMSE and CP_FIT for all the methods for one dataset of type SD4. We highlight the Pareto frontier points with diamond shape and red star is the original tensor($\underline{\mathbf{X}}_{og}$). We do see that all the 4 points found using HARVESTER lie on the Pareto frontier and 2 out of 5 points of ICEBREAKER++ lie on the Pareto frontier. Furthermore, none of the fixed aggregation point lie on the Pareto frontier. In finding points that lie on the Pareto we didn't include the original tensor point since that was used to generate that fine granular dataset.

To measure the effectiveness of our method HARVESTER against baselines we count the number of times each method is on the pareto frontier for all the synthetic datasets we evaluated on and we present that information in table 6.2. We observe that points from

our method HARVESTER are always on the pareto boundary with variety of the synthetic datasets. As shown in table 6.2, HARVESTER performs far better than the baseline methods. In all of the synthetic datasets, we see that tensor generated by HARVESTER was found on Pareto boundary. In SD4 ICEBREAKER++ did reasonably well, as it was on Pareto boundary six times as opposed to HARVESTER which was on Pareto boundary for all ten datasets.

Dataset	Original Dimension ($\underline{\mathbf{X}}_{\text{og}}$)	Approximate Final Dimension($\underline{\mathbf{X}}$)	Harvester Count	IceBreaker Count	Views Count
SD1	$100 \times 100 \times 10$	$100 \times 100 \times 500$	**10**	0	0
SD2	$100 \times 100 \times 10$	$100 \times 100 \times 500$	**10**	0	0
SD3	$100 \times 100 \times 10$	$100 \times 100 \times 500$	**10**	1	0
SD4	$100 \times 100 \times 100$	$100 \times 100 \times 2000$	**10**	6	0

Table 6.2: Table showing how many times each method was on the Pareto Boundary using CORCONDIA, Relative CP fit and RMSE for tensor completion.

6.4.3 Performance on semi-synthetic data

We evaluate our method HARVESTER and baseline methods on Enron email dataset [77, 11] which consist of email sent between the 184 employees over 44 months. The size of the dataset is $184 \times 184 \times 44$, similarly as we did in creation of synthetic dataset we distribute each matrix slice which represents a month over 30 days. The final input is of size $184 \times 184 \times 1320$, which is of daily aggregation. We create two views from this tensor, which are weekly and biweekly of dimensions $184 \times 184 \times 189$ and $184 \times 184 \times 95$ respectively. Hyperparameter range used are : for α we use the values $[10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}]$, for β we use the values $[10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10^{0}]$ and for γ we use the values $[10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}]$.

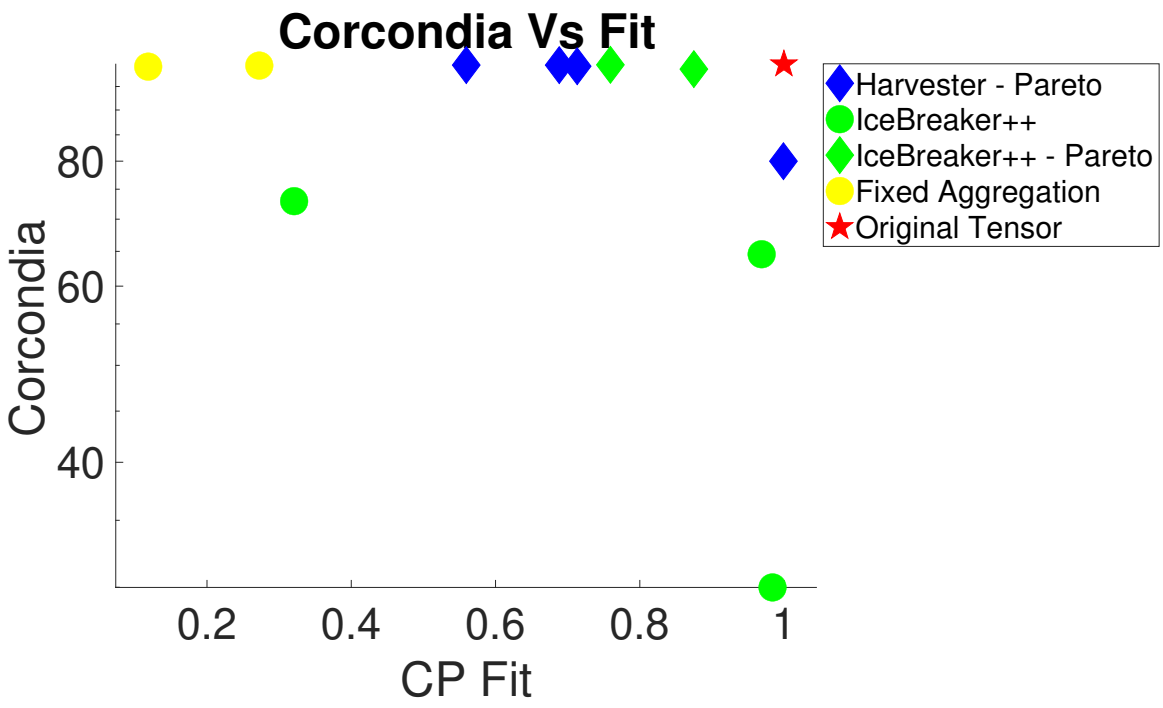


Figure 6.3: SD4: CP_FIT vs. CORCONDIA

We compute CORCONDIA, CP_FIT and RMSE for tensor completion for HARVESTER and baseline methods. Figure 6.5 and 6.6 shows the Pareto points on the CORCONDIA vs Fit plot and 1/RMSE vs Fit plot respectively. In this case we do observe that HARVESTER lie on Pareto boundary. Both ICEBREAKER++ and fixed view aggregation don't lie on the Pareto frontier.

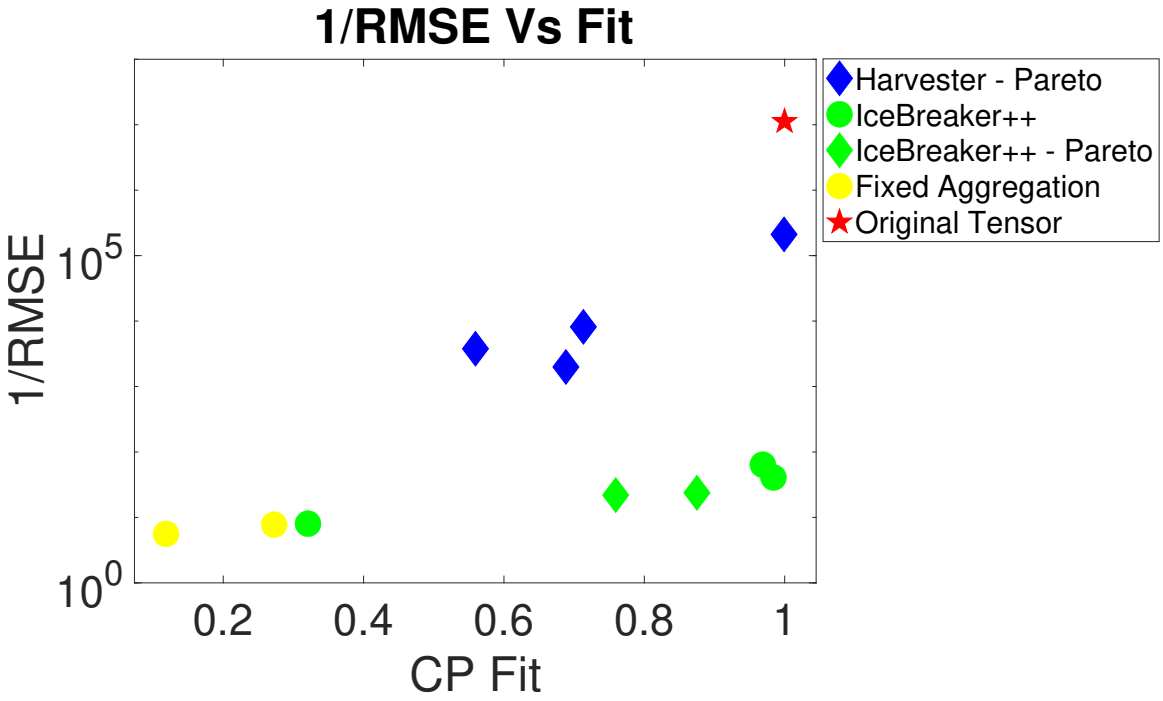


Figure 6.4: SD4: CP_FIT vs. (1/RMSE)

6.4.4 Ablation and Sensitivity Analysis

Are two compressed views enough?

In this section, we explore whether multiple views are more useful or the initial fixed aggregation window used to create views. To explore this we use a single dataset of type SD2 where the final dimension of a tensor $\underline{\mathbf{X}}$ is $100 \times 100 \times 500$, and then we create three different initial conditions:

1. Two views $\underline{\mathbf{Y}}^1$ and $\underline{\mathbf{Y}}^2$ of dimensions $100 \times 100 \times 250$ and $100 \times 100 \times 100$ respectively.

2. Two views $\underline{\mathbf{Y}}^1$ and $\underline{\mathbf{Y}}^2$ of dimensions $100 \times 100 \times 100$ and $100 \times 100 \times 50$ respectively.

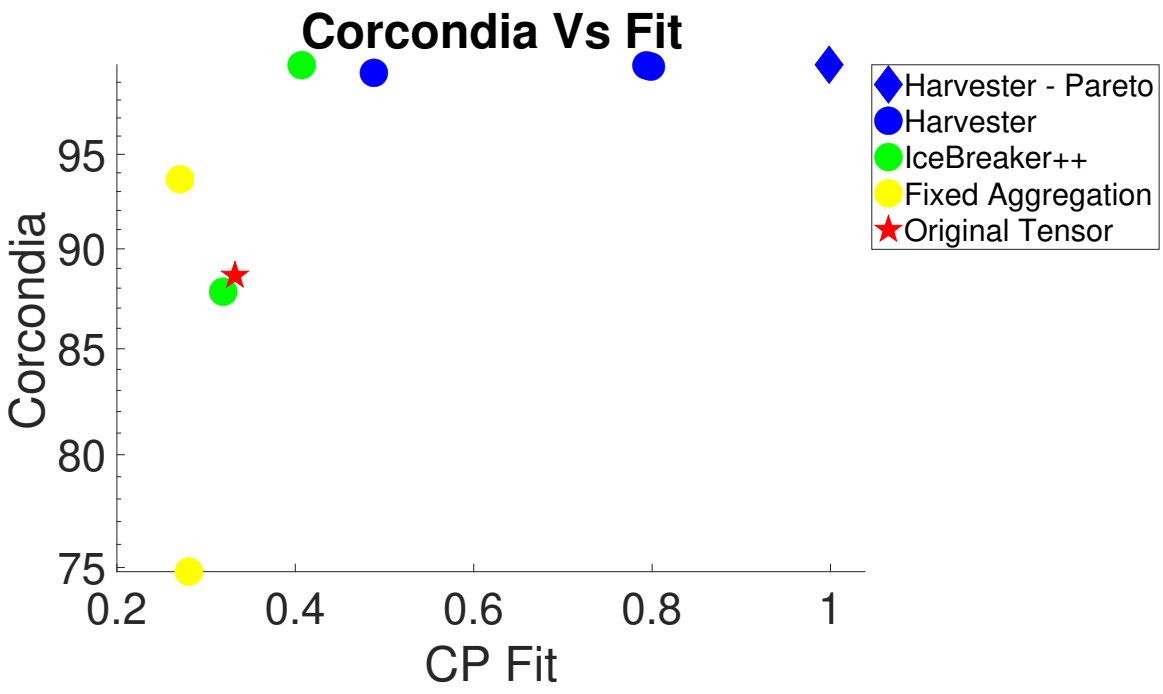


Figure 6.5: Enron: CP_FIT vs. CORCONDIA

3. Three views $\underline{\mathbf{Y}}^1$, $\underline{\mathbf{Y}}^2$ and $\underline{\mathbf{Y}}^3$ of dimensions $100 \times 100 \times 250$, $100 \times 100 \times 100$ and $100 \times 100 \times 50$ respectively.

We run HARVESTER method on these three initial condition 5 times and we present those results: CORCONDIA, Relative fit, RMSE for the tensor completion and third mode aggregation in figure 6.7, 6.8, 6.9 and 6.10 respectively. All those results are average of 5 runs with standard deviations.

Few observations on the results of this experiment:

- All of these tensors have good CORCONDIA score, the only tensors which have lower CORCONDIA are tensors created using norm aggregation of min sparsity point.

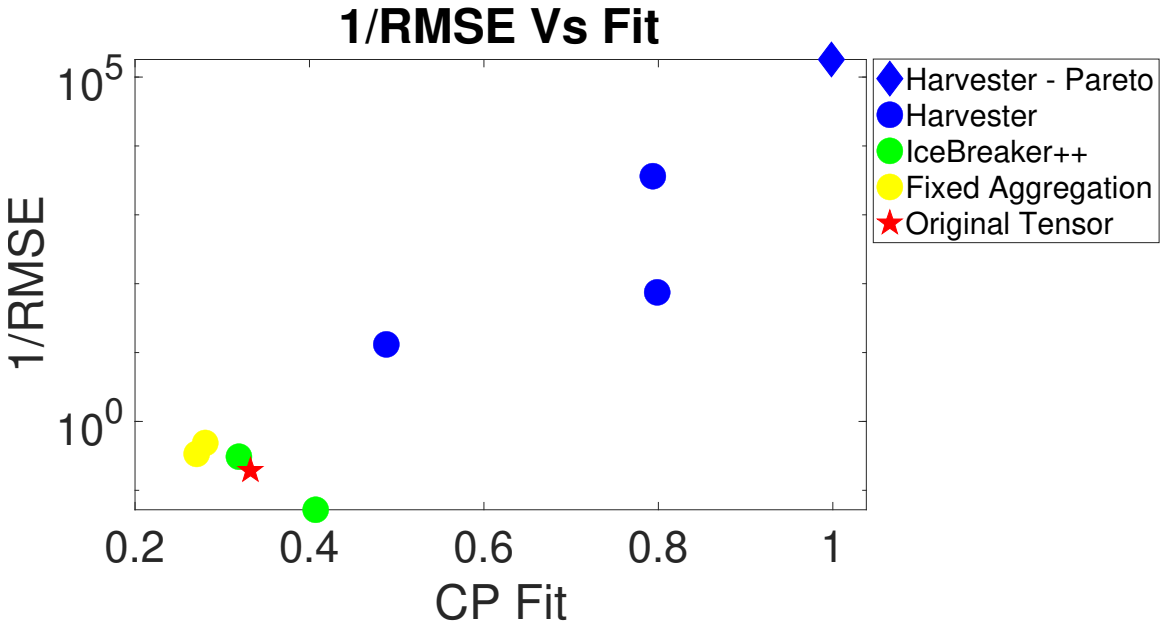


Figure 6.6: Enron: CP_FIT vs (1/RMSE)

- We see the similar behaviour with relative fit and RMSE as well. But realtive fit and RMSE are reasonably good for all the tensors.

- Looking at figure 6.10, which shows the third mode dimension of the aggregated tensors created by HARVESTER, we notice that all three points which are norm aggregation of min sparsity point have third mode aggregation lower the original data third mode (less than 10). Which is on par with our assumption that after some point more aggregation doesn't provide good solution.

- Lastly we observe tensors of different aggregation range have good CORCONDIA score, fit and RMSE. Which is an indication of multiple candidate tensors with different aggregation level, which could be useful for analysis purposes.

This experiment does not provide supporting evidence for three views offering a significant advantage over two views. However, we observe empirical evidence for the importance of the initial condition (i.e., coarser 2-view vs. finer 2-view vs 3-view) coupled with the aggregation criterion (i.e., min error or min sparsity), where it appears that different initial conditions provide essentially different but valuable and high-quality resolutions.

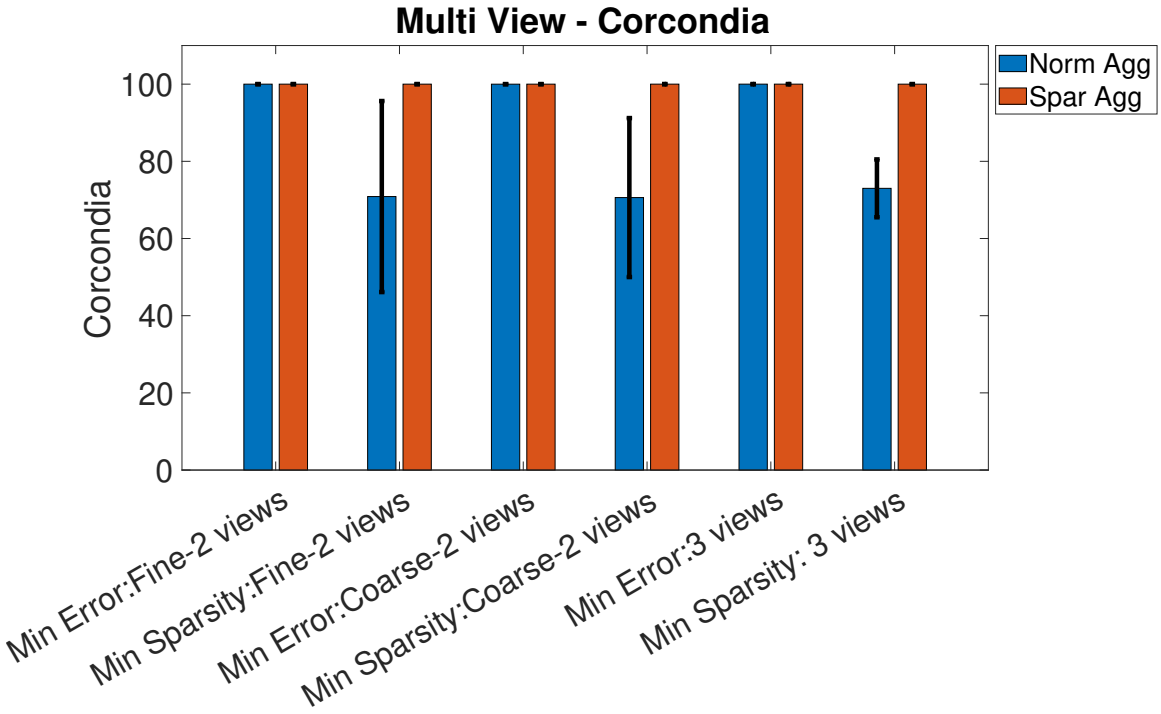


Figure 6.7: Harvester: MultiView CORCONDIA

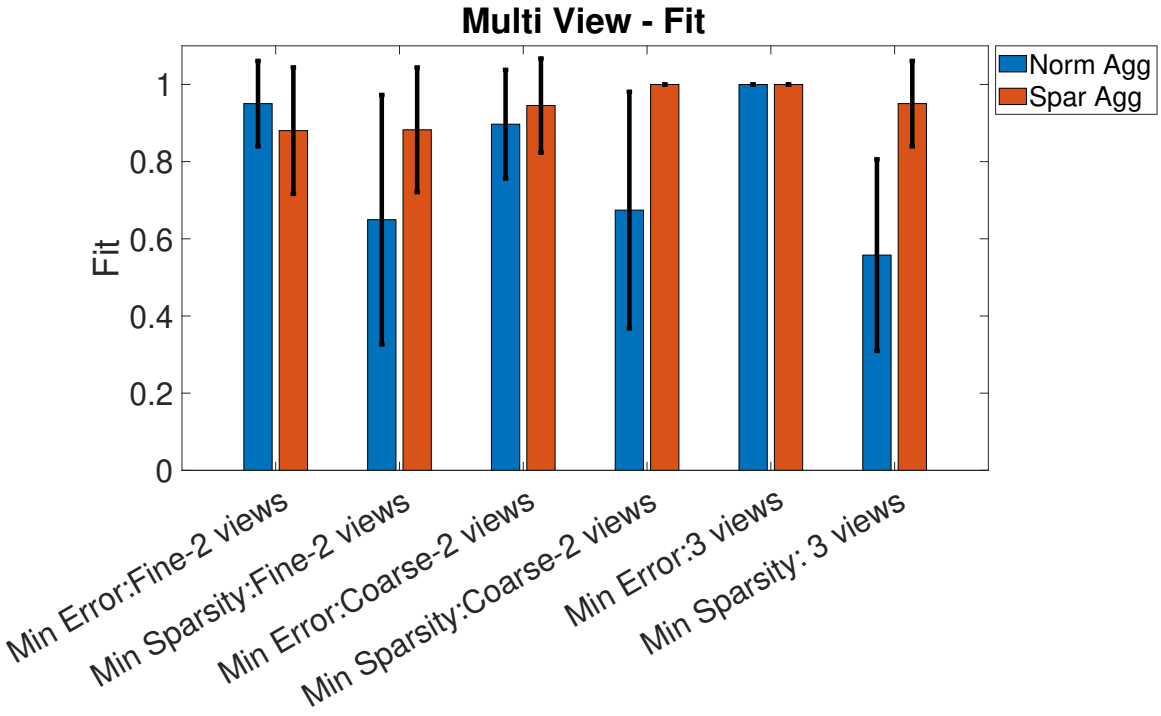


Figure 6.8: Harvester: MultiView Fit

Rank Sensitivity

In this section, we explore the sensitivity of our algorithm HARVESTER with respect to different ranks. We run this experiment using a dataset of size $100 \times 100 \times 10$ generated using rank 10, which was then used to generate a fine grained tensor of size $100 \times 100 \times 500$, using the same method as described in section 6.4.2. In figure 6.11, 6.12 and 6.13 we report mean and standard deviation of Corcondia, Relative fit and RMSE for the dataset over 5 runs. We do observe that HARVESTER has a steady pefromance for the initial rank increase but further increase in the rank deteriorate some of the evaluation metrics. One more thing we observe here is the tensor chosen based on minimum sparsity in the diagonal lambda

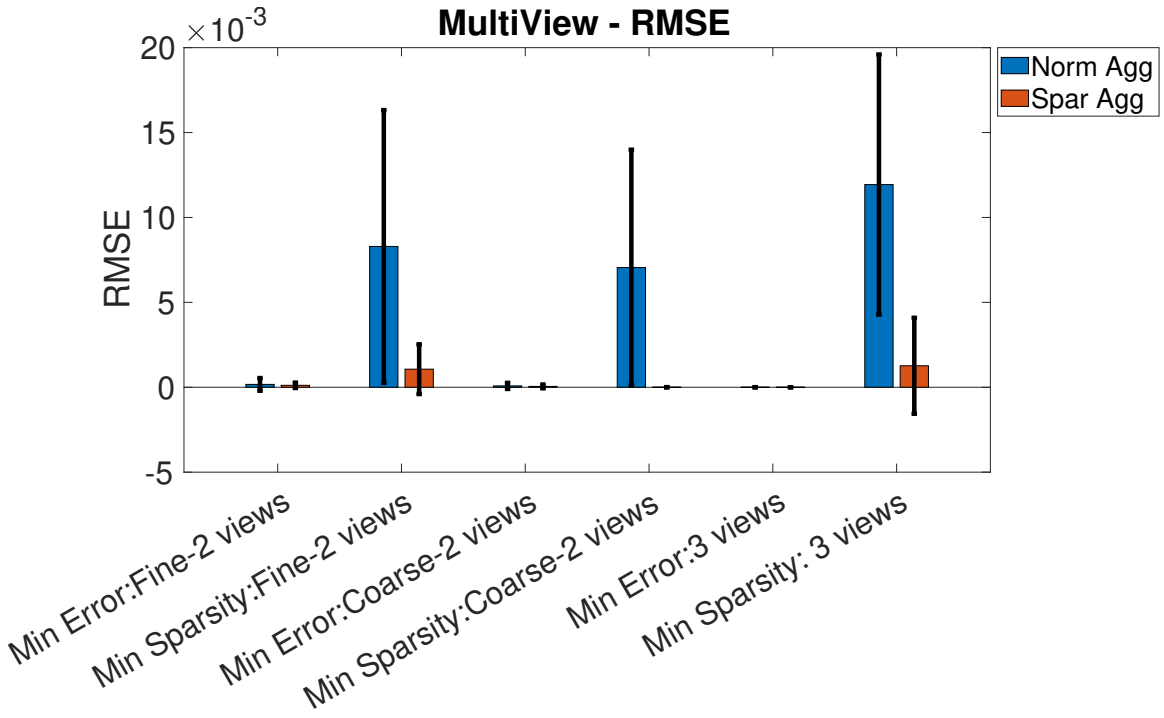


Figure 6.9: Harvester: MultiView RMSE

and when we aggregate that point based on norm (min Sparsity $\underline{\mathbf{Y}}_{norm}$). It performs worst of all the points chosen by harvester in this scenario.

6.4.5 Scalability Analysis

Scalability as the tensor dimensions grow

In this section we show the scalability analysis for HARVESTER as the third mode grows, since HARVESTER depends on the views of the original tensor. We run HARVESTER on the different settings one with coarser views and other one with finer views. In figure

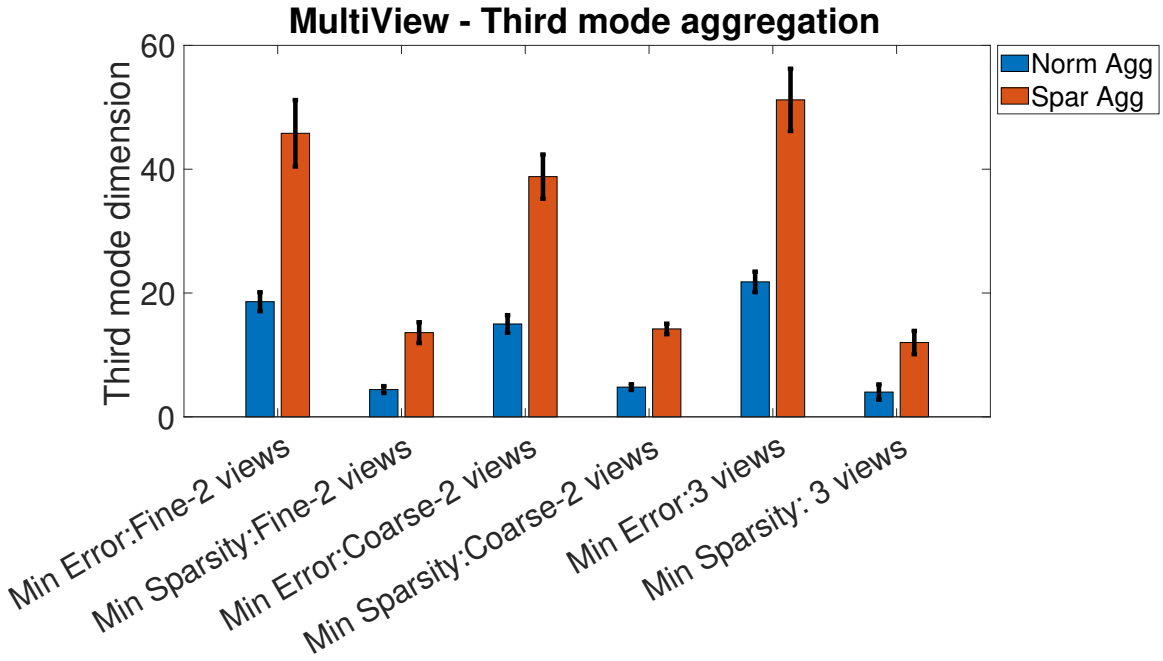


Figure 6.10: Harvester: MultiView CORCONDIA

6.14, we report time taken by harvester for both of this inputs. When third mode is smaller both aggregation roughly take the same time, but as the dimension grows we see that HARVESTER with coarser aggregation runs faster than the with finer aggregation.

We also evaluated scalability of HARVESTER against ICEBREAKER++. Figure 6.15 shows the time taken by each method. Since ICEBREAKER++ depends on the performance of the utility functions, we ran ICEBREAKER++ with no missing value prediction utility function as it's the most time consuming of all the utility function used in that work. We also ran ICEBREAKER++ with one iteration of missing value prediction utility function as opposed to 10 times in the paper [72]. We compared it with HARVESTER with both

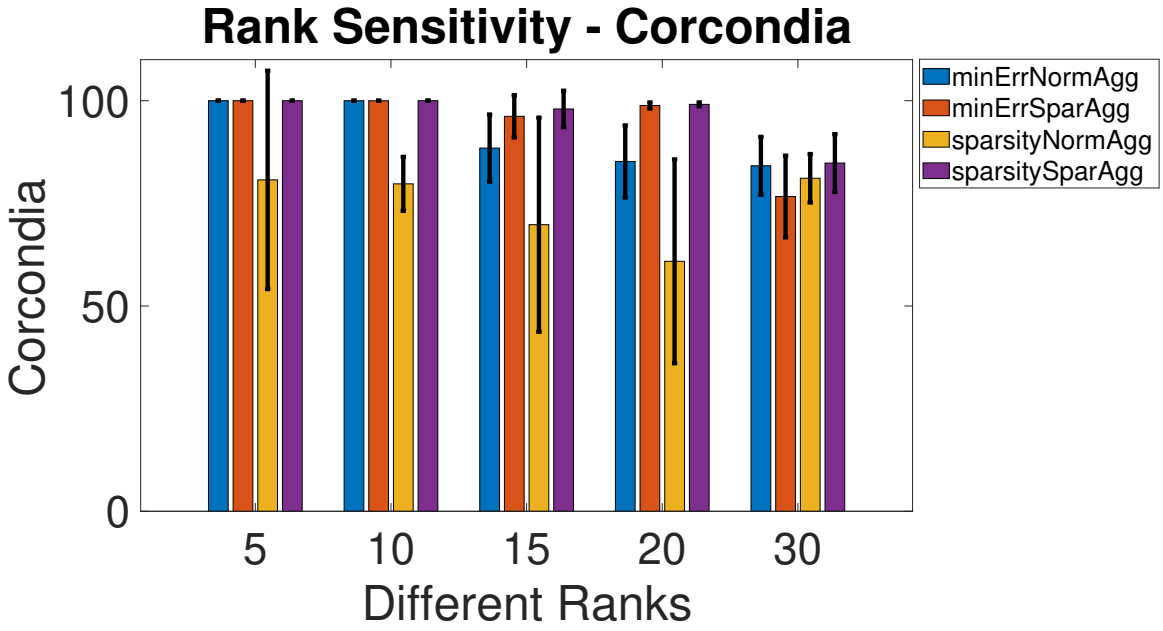


Figure 6.11: Harvester Rank Sensitivity CORCONDIA on dataset: $100 \times 100 \times 10$ rank 10

coarser and fine grained views. As shown in figure 6.15, HARVESTER is way faster than ICEBREAKER++.

Scalability as the Harvester rank grows

In this section we evaluate the behaviour of HARVESTER as rank grows which is used to compute the coupled tensor factorization to set up the problem with $\mathbf{C}_i$'s. For an input dataset to harvester of size $100 \times 100 \times 500$ we ran it 10 times for each rank we report the mean and standard deviation of time taken by harvester in figure 6.16. We observe that HARVESTER scales linearly with respect to rank.

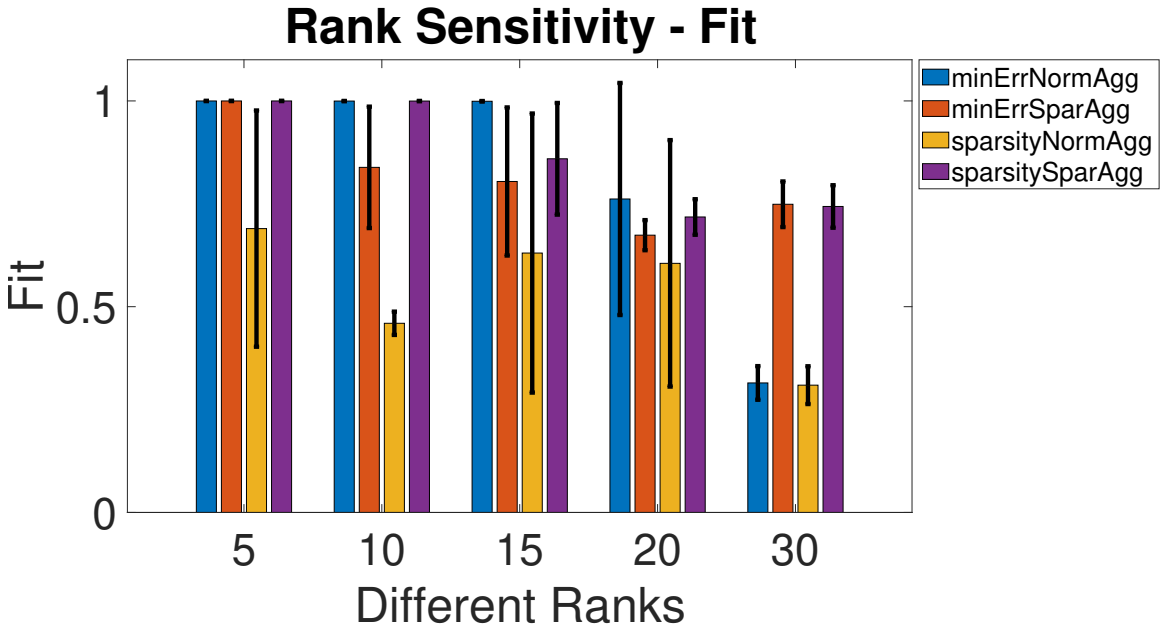


Figure 6.12: Harvester Rank Sensitivity Fit on dataset: $100 \times 100 \times 10$ rank 10

6.4.6 Real-world case study: Foursquare Dataset

We evaluated the efficiency of HARVESTER on real world dataset. For this purpose we used the foursquare dataset[2] [89] for check-ins in New York City collected over 10 months from 12 April, 2012 to 16 February, 2014. We constructed a 3-mode tensor with user, venue categories and time as the three modes where each entry in tensor is number of check-in for a user for a particular venue category for a given day. We aggregated temporal mode on the daily basis to create an input tensor and we used top 126 venue categories (venue categorises which had more check-ins than the median check-in value of each categories). We, thus, end up with a tensor of size $1083 \times 126 \times 250$ (user, venue categories, days). We create two views from the input tensor of size $1083 \times 126 \times 125$ and $1083 \times 126 \times 50$ respectively. These two

[2] https://www.kaggle.com/datasets/chetanism/foursquare-nyc-and-tokyo-checkin-dataset

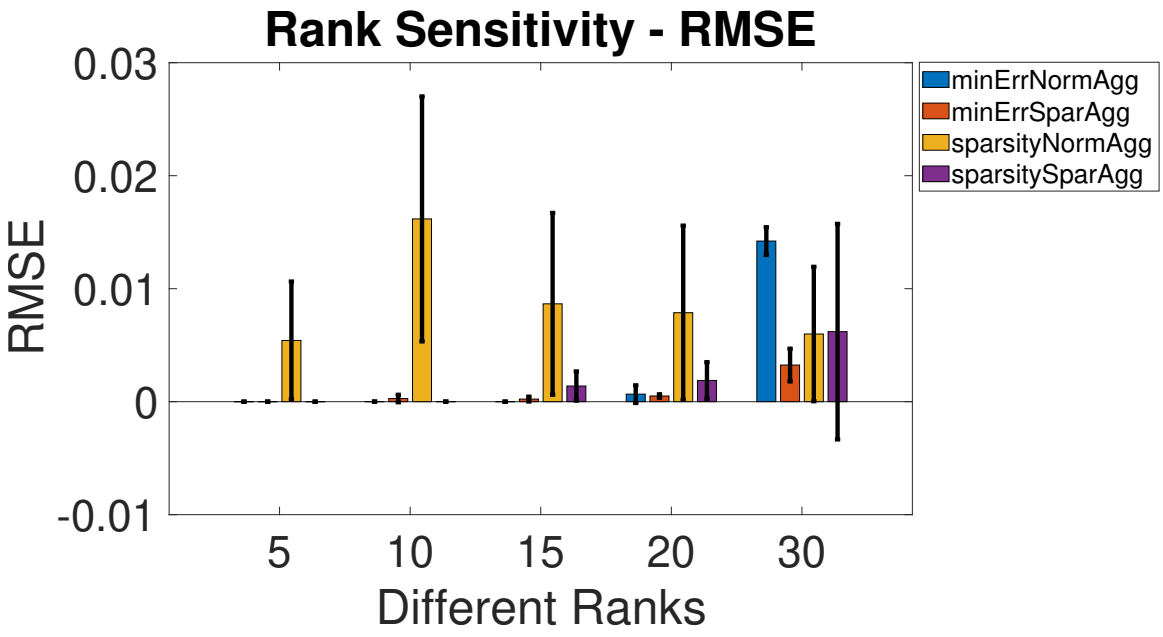


Figure 6.13: Harvester Rank Sensitivity RMSE on dataset: $100 \times 100 \times 10$ rank 10

views were used by HARVESTER to find tensor of optimal granularity. Figure 6.17 shows the plot for relative fit vs CORCONDIA for HARVESTER and baselines. In figure 6.18 plots the relative fit vs $1/RMSE$ for HARVESTER and baselines. We see that only tensors generated using HARVESTER are on the Pareto frontier.

The two points on the Pareto frontier are $\underline{\mathbf{Y}}_{norm}$ of size $1083 \times 126 \times 15$ and $\underline{\mathbf{Y}}_{spar}$ of size $1083 \times 126 \times 48$ created from the total minimum error point as mentioned in the starting of section 6.4. Both of tensor were decomposed using rank 10, which was determined by using AUTOTEN [62]. We decided to drill down onto those tensor generated with optimal granularity using HARVESTER. In consideration of space we show some of the latent factors for one of the tensors namely $\underline{\mathbf{Y}}_{norm}$, figure 6.19, 6.20, 6.21, and 6.22 shows the word cloud for the top-50 venue categories and the temporal activity of the that factor. The date shown

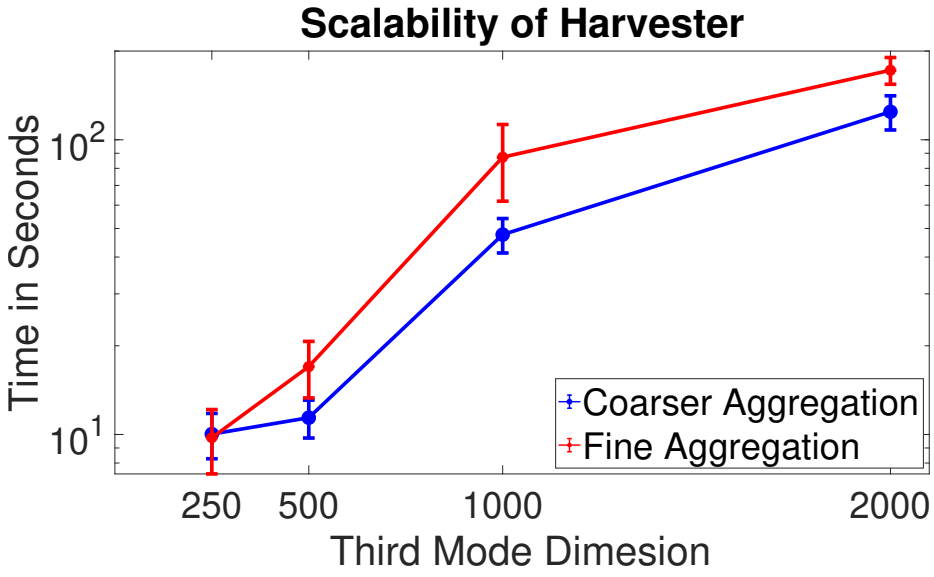


Figure 6.14: Harvester Scalability analysis

on the x-axis of the temporal activity corresponds to the dates which were aggregated up until that point from the previous date. In figure 6.19 corresponds to latent factor which as highest venue category as "Bar", which make sense since bar is the one of the most checked-in venues in the dataset. Figure 6.20 corresponds to the latent factor of restaurant, foods and drink places. Figure 6.21 corresponds to the latent factor of park and outdoors, and it has a temporal spike during spring season. Lastly the figure 6.22 corresponds to the latent factor of coffee shops. Other latent factors corresponds for this tensor are Home (2nd largest checked-in venue), Office (3rd largest checked-in venue), Gym & fitness center (5th largest checked-in venue), Subway & Train Station(4th and 8th largest), Hotel, Resident Building & Airport etc.

As a side note, without any external information about different events that coincide with the activity observed, we can only make speculations about the spikes and

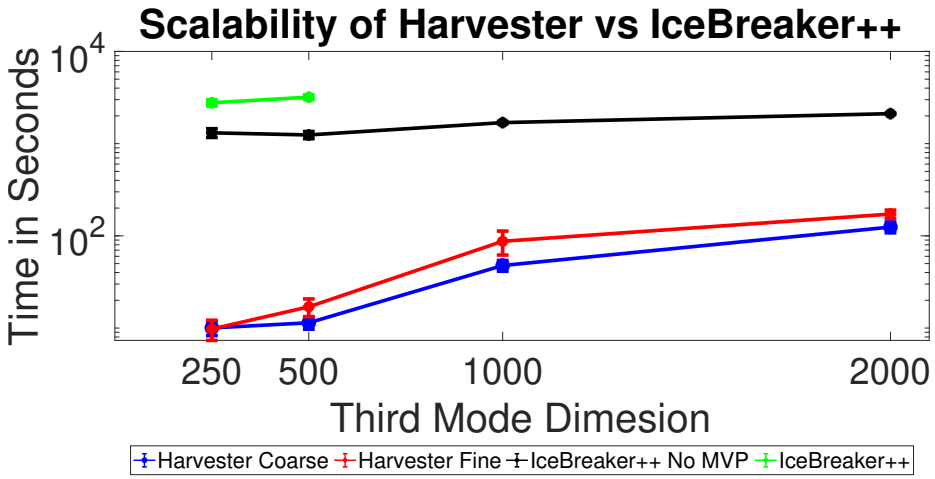


Figure 6.15: HARVESTER vs ICEBREAKER++ scalability analysis

fluctuations we observe in the temporal activity of different components. However, the component corresponding to Figure 6.21 is rather intuitive, since most outdoors activities in New York City flourish over the time where the spike is observed. Such an intuitive explanation lends more credence to the quality of the components extracted. A platform like Foursquare, or different vendors, however, can use such extracted components to understand different spatio-temporal trends over the City, and potentially integrate those components with other information (such as deals, specials, etc) which may not be available to us but would certainly be available to vendors or Foursquare and would provide more tangible information for time points where we discover different spikes of activity.

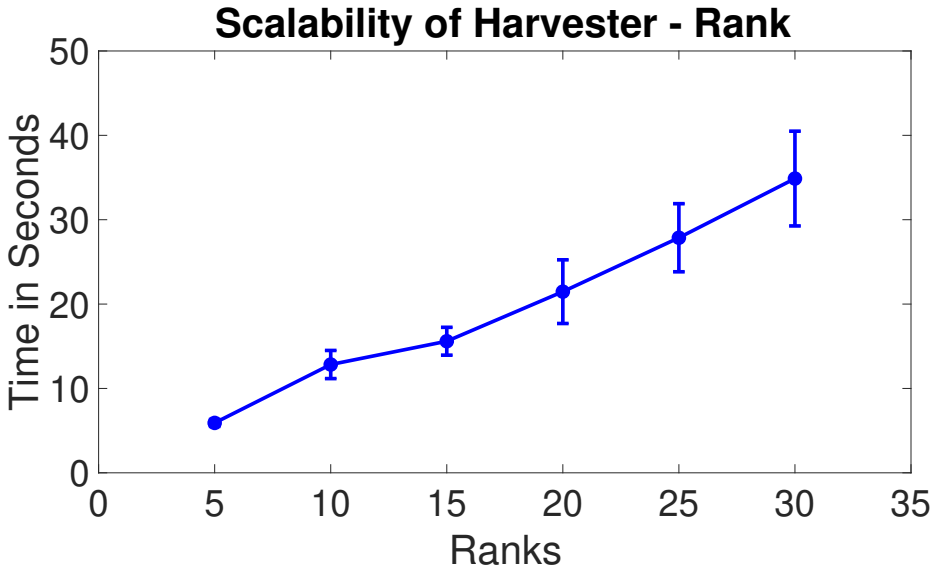


Figure 6.16: HARVESTER rank scalability analysis

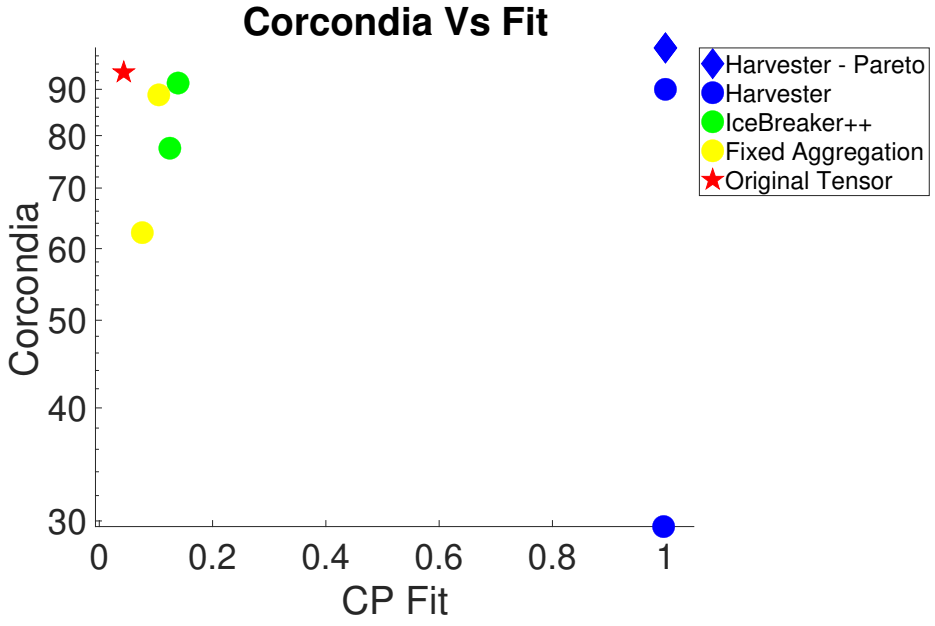


Figure 6.17: Foursquare: Fit vs. CORCONDIA

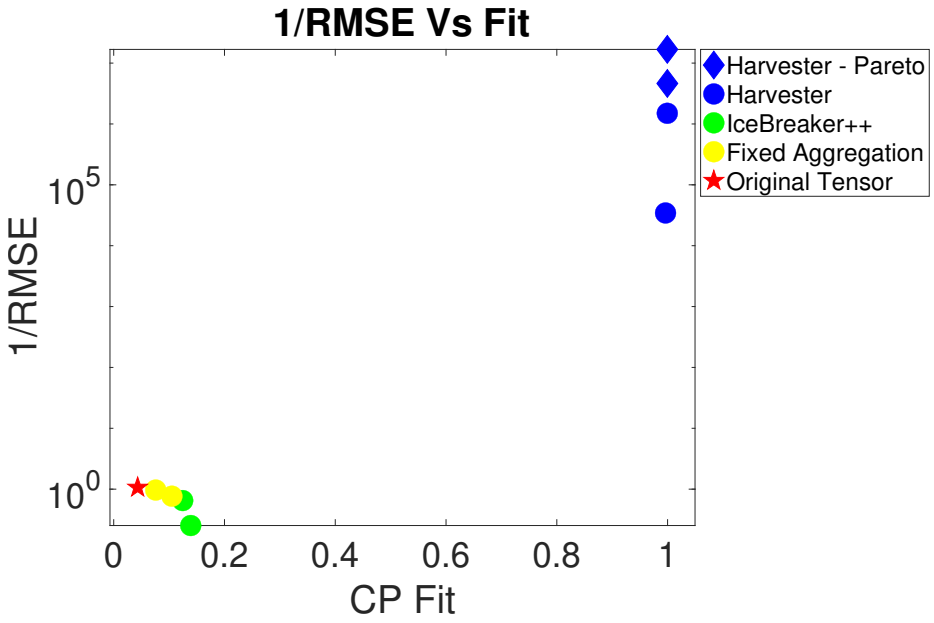


Figure 6.18: Foursquare: Fit vs. 1/RMSE

6.5 Related Work

There has been good amount of work when it comes to temporal edge aggregation in graph literature [82, 83, 84]. There has been also been some work in tensor literature to deal with temporal nature of the data [32, 50]. However these works deals with either finding aggregating the temporal edges in the graph case or finding communities in temporal graph and dealing with streaming tensor decomposition in tensor case. Our work does take inspiration from [4] and [5], which solves the inverse problem of our problem. Given aggregated data in one or two modes, they recover tensor in the original dis-aggregated granularity. However closest to our work is [72, 70], which provides a greedy solution to the adaptive aggregation in tensors problem. However to best of our knowledge we are the first work to introduce a principled way to finding an optimal aggregation in the temporal mode

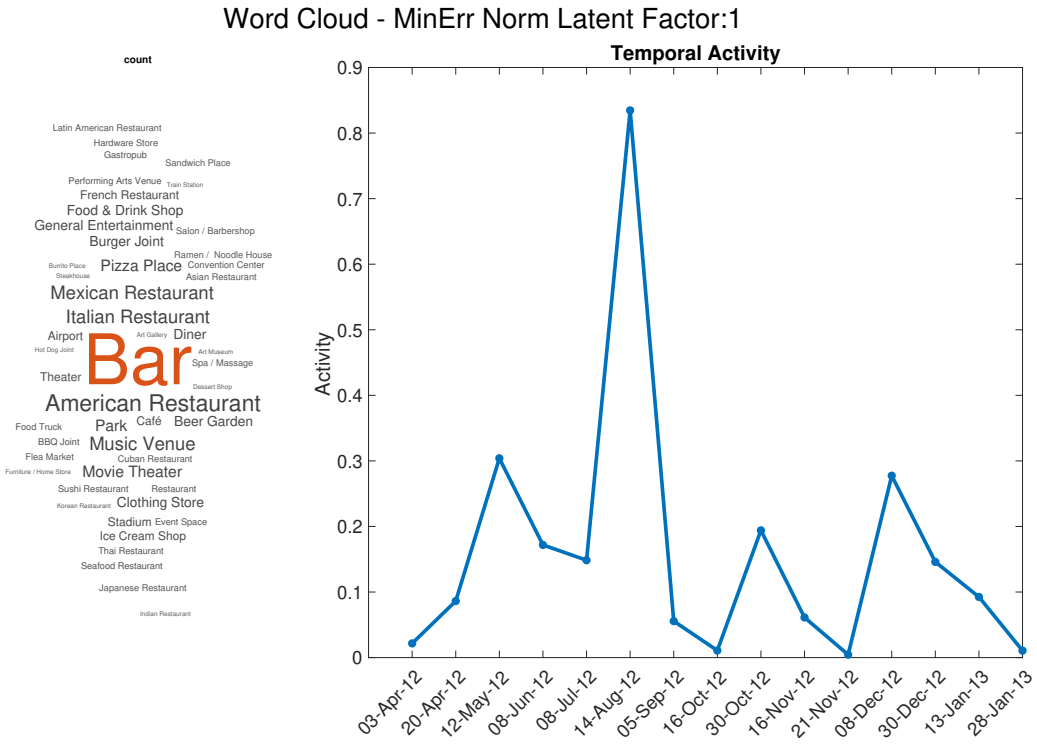


Figure 6.19: Foursquare: Word Cloud of Top 50 venue categories and Temporal Activity. Latent factor seems to correspond to venue category: Bar

using aggregated views of the input data and formalizing it in terms of an optimization problem.

6.6 Contributions

Our contributions in this chapter are as follows:

- **Novel Problem Formulation**: We provide a new and novel way to formulate the problem in terms of a factorization problem which can be solved using optimization techniques.

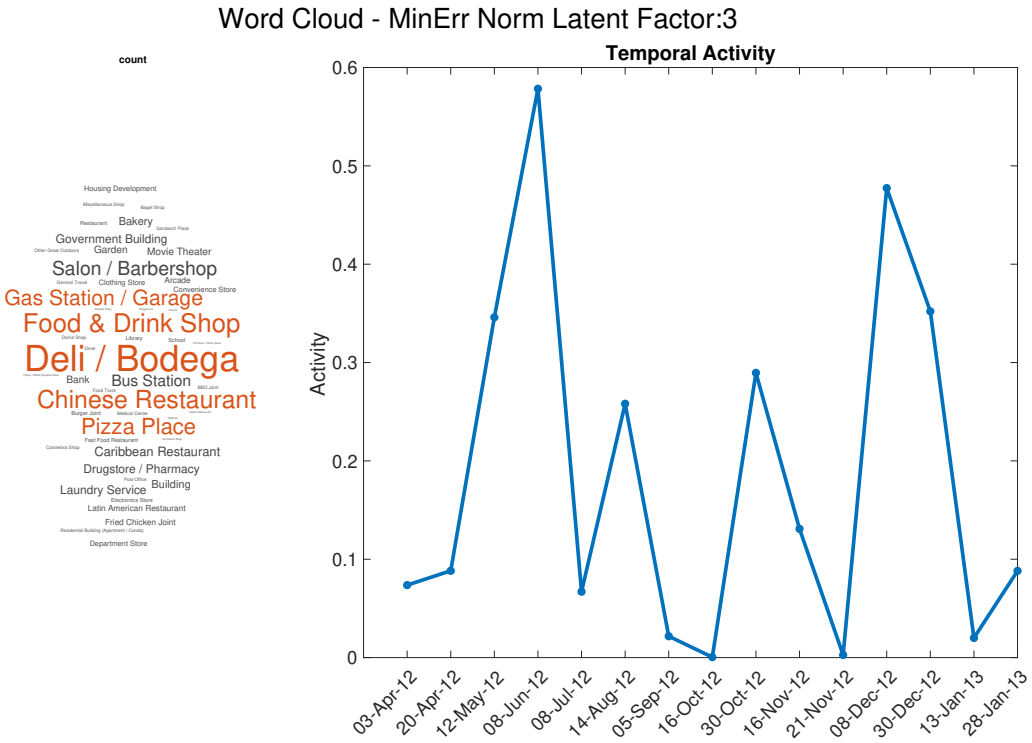


Figure 6.20: Foursquare: Word Cloud of Top 50 venues and Temporal Activity - Latent factor seems to correspond to venue category: Restaurant

- **Efficient & Scalable Algorithm**: We propose an efficient and scalable algorithm called HARVESTER, which solves the formulated problem in alternating fashion using non-negative multiplicative updates.

- **Experimental Evaluation**: We demonstrate the effectiveness of HARVESTER by performing extensive experiments on synthetic, semi-synthetic and real datasets. In evaluation, HARVESTER was preferred over the baseline for majority of the time.

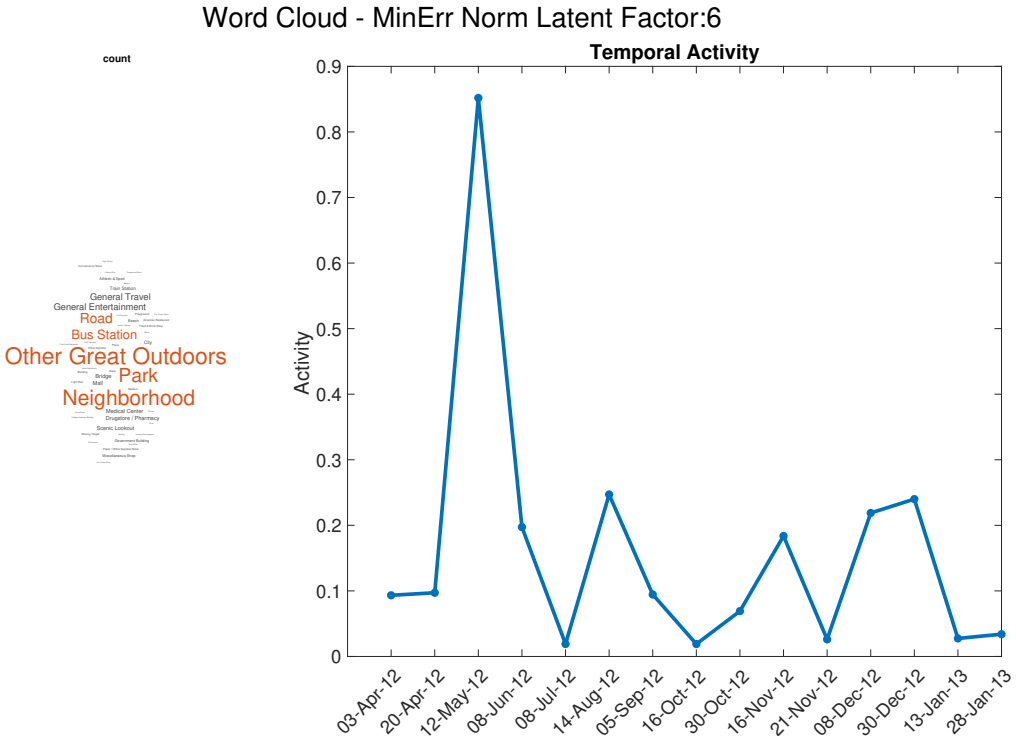


Figure 6.21: Foursquare: Word Cloud of Top 50 venues and Temporal Activity - Latent factor seems to correspond to venue category: Park and Outdoor

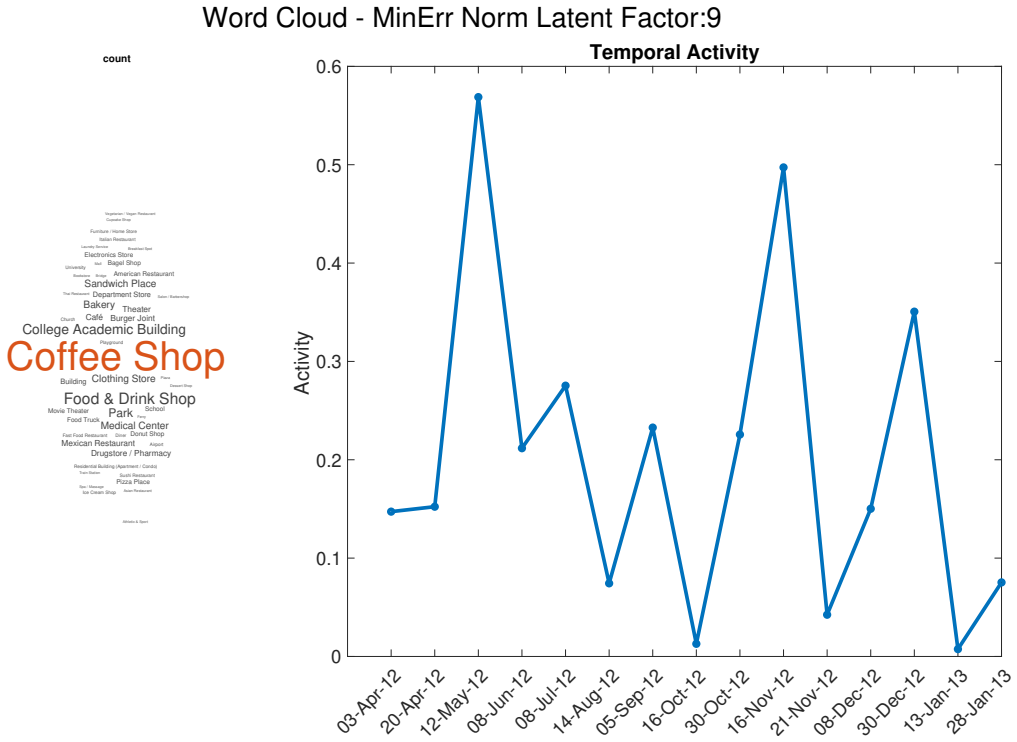


Figure 6.22: Foursquare: Word Cloud of Top 50 venues and Temporal Activity - Latent factor seems to correspond to venue category: Coffee Shop

Chapter 7

Conclusion

In this thesis, we revisited some of the conventional assumptions in tensor decomposition domain and developed novel solutions for the problems arose from relaxing those assumptions.

7.1 Summary

Hyperspectral Image Classification (Chapter 3): We developed a novel hyperspectral pixel classification model that employs tensor factorization to generate a new feature space. Specifically, our method Orion exploits the multi-linear structure of the tensor to find a richer space. We showcase the effectiveness of our method by conducting experiments on publicly available hyperspectral image datasets where we compared Orion against baseline methods like Kernel SVMs and Multi-layer Perceptrons. Orion is significantly more accurate in a majority of the cases, even with limited training samples.

Concept Drift in Streaming Tensor Decomposition(Chapter 4): We introduce the notion of "concept drift" in streaming tensors and developed an algorithm called SeekAndDestroy, which detects and alleviates concept drift without making any assumptions about the rank of the tensor. SeekAndDestroy outperforms other state-of-the-art methods when the rank is unknown and is effective in detecting concept drift.

Greedy Algorithm for Adaptive Granularity (Chapter 5): To the best of our knowledge, this work is first to define and formalize the *Trapped Under Ice* problem in constructing a tensor from raw sparse data. We demonstrate that an optimal solution is intractable and subsequently proposed IceBreaker++, a practical solution that is able to identify good tensor structure from raw data, and construct tensors from the same dataset that pertains to multiple resolutions. Our experiments demonstrate the merit of IceBreaker++ in discovering exploitable structure, as well as providing tools to data analysts in automatically extracting multi-resolution patterns from raw multi-aspect data.

Factorization-based Granularity Estimation (Chapter 6): In this work we developed a factorization-based tensor temporal granularity estimation algorithm called Harvester, which provides a new framework to deal with the *Trapped Under Ice* problem. Using multiple aggregated views of the input data we were able to find a tensor of optimal aggregation which provide better utility latent factors for tensor decomposition. We demonstrated using extensive experimentation that Harvester performs better than the baselines and finds tensor which has "better" latent structure.

7.2 Future Work

There are several directions in which these works can be extended or build upon.

For instance we only investigated ORION (Chapter 3) in hyperspectral image data, either ORION or some modified version of ORION can be used in other application like EEG data, sensor data where a richer feature space is needed for classification.

In Chapter 4, we introduced an algorithm called SEEKANDDESTROY which keep tracks of evolving latent factor only in one mode. However, there could be datasets which can evolve in multiple mode (like spatio-temporal data). An interesting variation of this problem would be to keep track of evolving latent factors based on two or more mode of dataset evolving with time.

Another way to solve the problem presented in Chapter 4 to keep track of new, overlap and missing latent factor can be to keep track of the subspace of the rank-1 latent factor as the data grows larger.

Finally, Chapters 5 and 6 deal with finding optimal aggregation of the tensor only in single mode (temporal mode). However a dataset can have poor granularity in multiple modes. A naive way would be to use HARVESTER (or ICEBREAKER++) on each mode separately until optimal aggregation is found in each mode. However more interesting problem would be to use the information from multiple modes (or all the modes) to find an optimal aggregation in all of the modes by exploiting multi-linear structure of the data.

Bibliography

[1] Chicago data portal. https://data.cityofchicago.org/Public-Safety/Crimes-2001-to-present/ijzp-q8t2.

[2] Evrim Acar, Canan Aykut-Bingol, Haluk Bingol, Rasmus Bro, and Bülent Yener. Multiway analysis of epilepsy tensors. *Bioinformatics*, 23(13):i10–i18, 2007.

[3] Evrim Acar, Daniel M Dunlavy, Tamara G Kolda, and Morten Mørup. Scalable tensor factorizations for incomplete data. *Chemometrics and Intelligent Laboratory Systems*, 106(1):41–56, 2011.

[4] Faisal M Almutairi, Charilaos I Kanatsoulis, and Nicholas D Sidiropoulos. Tendi: Tensor disaggregation from multiple coarse views. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 867–880. Springer, 2020.

[5] Faisal M Almutairi, Charilaos I Kanatsoulis, and Nicholas D Sidiropoulos. Prema: Principled tensor data recovery from multiple aggregated views. *IEEE Journal of Selected Topics in Signal Processing*, 15(3):535–549, 2021.

[6] José Manuel Amigo, Marta J Popielarz, Raquel M Callejón, Maria L Morales, Ana M Troncoso, Mikael A Petersen, and Torben B Toldam-Andersen. Comprehensive analysis of chromatographic data by using parafac2 and principal components analysis. *Journal of Chromatography a*, 1217(26):4422–4429, 2010.

[7] Charlotte Møller Andersen and Rasmus Bro. Practical aspects of parafac modeling of fluorescence excitation-emission data. *Journal of Chemometrics: A Journal of the Chemometrics Society*, 17(4):200–215, 2003.

[8] Miguel Araujo, Spiros Papadimitriou, Stephan Günnemann, Christos Faloutsos, Prithwish Basu, Ananthram Swami, Evangelos E Papalexakis, and Danai Koutra. Com2: fast automatic discovery of temporal ('comet') communities. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 271–283. Springer, 2014.

[9] Brett W Bader, Michael W Berry, and Murray Browne. Discussion tracking in enron email using parafac. In *Survey of text mining II*, pages 147–163. Springer, 2008.

[10] Brett W Bader, Richard A Harshman, and Tamara G Kolda. Analysis of latent relationships in semantic graphs using dedicom. Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States); Sandia ..., 2006.

[11] Brett W Bader, Richard A Harshman, and Tamara G Kolda. Temporal analysis of semantic graphs using asalsan. In *Seventh IEEE international conference on data mining (ICDM 2007)*, pages 33–42. IEEE, 2007.

[12] Brett W. Bader, Tamara G. Kolda, et al. Matlab tensor toolbox version 2.6. Available online, February 2015.

[13] Brett W. Bader, Tamara G. Kolda, et al. Matlab tensor toolbox version 3.1. Available online, June 2019.

[14] Marion F. Baumgardner, Larry L. Biehl, and David A. Landgrebe. 220 band aviris hyperspectral image data set: June 12, 1992 indian pine test site 3, Sep 2015.

[15] Stephen Becker. L-bfgs-b-c. https://github.com/stephenbeckr/L-BFGS-B-C, 2015.

[16] Albert Bifet, Joao Gama, Mykola Pechenizkiy, and Indre Zliobaite. Handling concept drift: Importance, challenges and solutions. *PAKDD-2011 Tutorial, Shenzhen, China*, 2011.

[17] José M Bioucas-Dias, Antonio Plaza, Gustavo Camps-Valls, Paul Scheunders, Nasser Nasrabadi, and Jocelyn Chanussot. Hyperspectral remote sensing data analysis and future challenges. *IEEE Geoscience and remote sensing magazine*, 1(2):6–36, 2013.

[18] José M Bioucas-Dias, Antonio Plaza, Nicolas Dobigeon, Mario Parente, Qian Du, Paul Gader, and Jocelyn Chanussot. Hyperspectral unmixing overview: Geometrical, statistical, and sparse regression-based approaches. *IEEE journal of selected topics in applied earth observations and remote sensing*, 5(2):354–379, 2012.

[19] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008.

[20] Rasmus Bro. Parafac. tutorial and applications. *Chemometrics and intelligent laboratory systems*, 38(2):149–171, 1997.

[21] Rasmus Bro and Henk AL Kiers. A new efficient method for determining the number of components in parafac models. *Journal of Chemometrics: A Journal of the Chemometrics Society*, 17(5):274–286, 2003.

[22] Gustavo Camps-Valls and Lorenzo Bruzzone. Kernel-based methods for hyperspectral image classification. *IEEE Transactions on Geoscience and Remote Sensing*, 43(6):1351–1362, 2005.

[23] Gustavo Camps-Valls, Luis Gomez-Chova, Jordi Muñoz-Marí, Joan Vila-Francés, and Javier Calpe-Maravilla. Composite kernels for hyperspectral image classification. *IEEE geoscience and remote sensing letters*, 3(1):93–97, 2006.

[24] J Douglas Carroll and Jih-Jie Chang. Analysis of individual differences in multidimensional scaling via an n-way generalization of "eckart-young" decomposition. *Psychometrika*, 35(3):283–319, 1970.

[25] Yi Chen, Nasser M Nasrabadi, and Trac D Tran. Hyperspectral image classification via kernel sparse representation. *IEEE Transactions on Geoscience and Remote sensing*, 51(1):217–231, 2012.

[26] Andrzej Cichocki, Danilo Mandic, Lieven De Lathauwer, Guoxu Zhou, Qibin Zhao, Cesar Caiafa, and Huy Anh Phan. Tensor decompositions for signal processing applications: From two-way to multiway component analysis. *IEEE signal processing magazine*, 32(2):145–163, 2015.

[27] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[28] Grupo de Inteligencia Computacional. Hyperspectral remote sensing scenes. http://www.ehu.eus/ccwintco/index.php/Hyperspectral_Remote_Sensing_Scenes.

[29] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM computing surveys (CSUR)*, 46(4):1–37, 2014.

[30] Alexander FH Goetz, Gregg Vane, Jerry E Solomon, and Barrett N Rock. Imaging spectrometry for earth remote sensing. *science*, 228(4704):1147–1153, 1985.

[31] Ehsan Goodarzi, Mina Ziaei, and Edward Zia Hosseinipour. *Introduction to optimization analysis in hydrosystem engineering*. Springer, 2014.

[32] Alexander Gorovits, Ekta Gujral, Evangelos E Papalexakis, and Petko Bogdanov. Larc: Learning activity-regularized overlapping communities across time. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1465–1474, 2018.

[33] Gisel Bastidas Guacho, Sara Abdali, Neil Shah, and Evangelos E Papalexakis. Semi-supervised content-based detection of misinformation via tensor embeddings. In *2018 IEEE/ACM international conference on advances in social networks analysis and mining (ASONAM)*, pages 322–325. IEEE, 2018.

[34] Ekta Gujral, Ravdeep Pasricha, and Evangelos E Papalexakis. Sambaten: Sampling-based batch incremental tensor decomposition. In *Proceedings of the 2018 SIAM International Conference on Data Mining*, pages 387–395. SIAM, 2018.

[35] Richard A Harshman et al. Foundations of the PARAFAC procedure: models and conditions for an explanatory multimodal factor analysis. *UCLA Working Papers in Phonetics*, 16:1–84, 1970.

[36] Johan Håstad. Tensor rank is np-complete. *Journal of Algorithms*, 11(4):644–654, 1990.

[37] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

[38] Christopher J Hillar and Lek-Heng Lim. Most tensor problems are np-hard. *Journal of the ACM (JACM)*, 60(6):1–39, 2013.

[39] Joyce C Ho, Joydeep Ghosh, and Jimeng Sun. Marble: high-throughput phenotyping from electronic health records via sparse nonnegative tensor factorization. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 115–124, 2014.

[40] Gordon Hughes. On the mean accuracy of statistical pattern recognizers. *IEEE transactions on information theory*, 14(1):55–63, 1968.

[41] Ali Jahan, Kevin L Edwards, and Marjan Bahraminasab. *Multi-criteria decision analysis for supporting the selection of engineering materials in product design*. Butterworth-Heinemann, 2016.

[42] Mohamad Jouni, Mauro Dalla Mura, and Pierre Comon. Hyperspectral image classification using tensor cp decomposition. In *IGARSS 2019-2019 IEEE International Geoscience and Remote Sensing Symposium*, pages 1164–1167. IEEE, 2019.

[43] Nirmal Keshava and John F Mustard. Spectral unmixing. *IEEE signal processing magazine*, 19(1):44–57, 2002.

[44] Tamara G Kolda and Brett W Bader. Tensor decompositions and applications. *SIAM review*, 51(3):455–500, 2009.

[45] Tamara G Kolda, Brett W Bader, and Joseph P Kenny. Higher-order web link analysis using multilinear algebra. In *Fifth IEEE International Conference on Data Mining (ICDM'05)*, pages 8–pp. IEEE, 2005.

[46] Yehuda Koren. Collaborative filtering with temporal dynamics. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 447–456, 2009.

[47] Jean Kossaifi, Yannis Panagakis, Anima Anandkumar, and Maja Pantic. Tensorly: Tensor learning in python. *Journal of Machine Learning Research (JMLR)*, 20(26), 2019.

[48] Denis Krompaß, Maximilian Nickel, Xueyan Jiang, and Volker Tresp. Non-negative tensor factorization with rescal. In *Tensor Methods for Machine Learning, ECML workshop*, pages 1–10, 2013.

[49] Taehyung Kwon, Inkyu Park, Dongjin Lee, and Kijung Shin. Slicenstitch: Continuous CP decomposition of sparse tensor streams. In *37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021*, pages 816–827. IEEE, 2021.

[50] Taehyung Kwon, Inkyu Park, Dongjin Lee, and Kijung Shin. Slicenstitch: Continuous cp decomposition of sparse tensor streams. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 816–827. IEEE, 2021.

[51] Daniel Lee and H Sebastian Seung. Algorithms for non-negative matrix factorization. *Advances in neural information processing systems*, 13, 2000.

[52] Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1):503–528, 1989.

[53] Ji Liu, Przemyslaw Musialski, Peter Wonka, and Jieping Ye. Tensor completion for estimating missing values in visual data. *IEEE transactions on pattern analysis and machine intelligence*, 35(1):208–220, 2012.

[54] Alexander V Lotov and Kaisa Miettinen. Visualizing the pareto frontier. In *Multiobjective optimization*, pages 213–243. Springer, 2008.

[55] Konstantinos Makantasis, Anastasios D Doulamis, Nikolaos D Doulamis, and Antonis Nikitakis. Tensor-based classification models for hyperspectral data analysis. *IEEE Transactions on Geoscience and Remote Sensing*, 56(12):6884–6898, 2018.

[56] Konstantinos Makantasis, Konstantinos Karantzalos, Anastasios Doulamis, and Nikolaos Doulamis. Deep supervised learning for hyperspectral data classification through convolutional neural networks. In *2015 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, pages 4959–4962. IEEE, 2015.

[57] Morten Mørup and Lars Kai Hansen. Automatic relevance determination for multiway models. *Journal of Chemometrics: A Journal of the Chemometrics Society*, 23(7-8):352–363, 2009.

[58] Morten Mørup, Lars Kai Hansen, Christoph S Herrmann, Josef Parnas, and Sidse M Arnfred. Parallel factor analysis as an exploratory tool for wavelet transformed event-related eeg. *NeuroImage*, 29(3):938–947, 2006.

[59] Lichao Mou, Pedram Ghamisi, and Xiao Xiang Zhu. Deep recurrent neural networks for hyperspectral image classification. *IEEE Transactions on Geoscience and Remote Sensing*, 55(7):3639–3655, 2017.

[60] Dimitri Nion, Kleanthis N Mokios, Nicholas D Sidiropoulos, and Alexandros Potamianos. Batch and adaptive parafac-based blind separation of convolutive speech mixtures. *IEEE Transactions on Audio, Speech, and Language Processing*, 18(6):1193–1207, 2009.

[61] Dimitri Nion and Nicholas D Sidiropoulos. Adaptive algorithms to track the parafac decomposition of a third-order tensor. *IEEE Transactions on Signal Processing*, 57(6):2299–2310, 2009.

[62] Evangelos E Papalexakis. Automatic unsupervised tensor mining with quality assessment. In *Proceedings of the 2016 SIAM International Conference on Data Mining*, pages 711–719. SIAM, 2016.

[63] Evangelos E Papalexakis, Leman Akoglu, and Dino Ience. Do more views of a graph help? community detection and clustering in multi-graphs. In *Proceedings of the 16th International Conference on Information Fusion*, pages 899–905. IEEE, 2013.

[64] Evangelos E Papalexakis, Christos Faloutsos, and Nicholas D Sidiropoulos. Parcube: Sparse parallelizable tensor decompositions. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 521–536. Springer, 2012.

[65] Evangelos E Papalexakis, Christos Faloutsos, and Nicholas D Sidiropoulos. Tensors for data mining and data fusion: Models, applications, and scalable algorithms. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(2):1–44, 2016.

[66] Evangelos E Papalexakis, Konstantinos Pelechrinis, and Christos Faloutsos. Location based social network analysis using tensors and signal processing tools. In *2015 IEEE 6th International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*, pages 93–96. IEEE, 2015.

[67] Evangelos E Papalexakis, Nicholas D Sidiropoulos, and Rasmus Bro. From k-means to higher-way co-clustering: Multilinear decomposition with sparse latent factors. *IEEE transactions on signal processing*, 61(2):493–506, 2012.

[68] SungJin Park, Suan Lee, and Jinho Kim. Estimating revenues of seoul commercial alley services using tensor decomposition & generating recommendation system. In *2020 IEEE International Conference on Big Data and Smart Computing (BigComp)*, pages 287–294. IEEE, 2020.

[69] Ravdeep Pasricha, Ekta Gujral, and Evangelos Papalexakis. Adaptive granularity in time evolving graphs as tensors. In *Proceedings of the 16th International Workshop on Mining and Learning with Graphs (MLG)*, 2020.

[70] Ravdeep Pasricha, Ekta Gujral, and Evangelos E Papalexakis. Adaptive granularity in time evolving graphs as tensors. *Ratio*, 20(40):60.

[71] Ravdeep Pasricha, Ekta Gujral, and Evangelos E Papalexakis. Identifying and alleviating concept drift in streaming tensor decomposition. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 327–343. Springer, 2018.

[72] Ravdeep Pasricha, Ekta Gujral, and Evangelos E Papalexakis. Adaptive granularity in tensors: A quest for interpretable structure. *arXiv preprint arXiv:1912.09009*, 2019.

[73] Ravdeep S Pasricha, Pravallika Devineni, Evangelos E Papalexakis, and Ramakrishnan Kannan. Tensorized feature spaces for feature explosion. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 6298–6304. IEEE, 2021.

[74] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[75] Ioakeim Perros, Robert Chen, Richard Vuduc, and Jimeng Sun. Sparse hierarchical tucker factorization and its application to healthcare. In *2015 IEEE International Conference on Data Mining*, pages 943–948. IEEE, 2015.

[76] Kaare Brandt Petersen, Michael Syskind Pedersen, et al. The matrix cookbook. *Technical University of Denmark*, 7(15):510, 2008.

[77] Carey E. Priebe, John M. Conroy, David J. Marchette, and Youngser Park Park. Enron data set, 2006.

[78] Nicholas D Sidiropoulos and Rasmus Bro. On the uniqueness of multilinear decomposition of n-way arrays. *Journal of Chemometrics: A Journal of the Chemometrics Society*, 14(3):229–239, 2000.

[79] Nicholas D Sidiropoulos, Lieven De Lathauwer, Xiao Fu, Kejun Huang, Evangelos E Papalexakis, and Christos Faloutsos. Tensor decomposition for signal processing and machine learning. *IEEE Transactions on Signal Processing*, 65(13):3551–3582, 2017.

[80] Nicholas D Sidiropoulos, Georgios B Giannakis, and Rasmus Bro. Blind parafac receivers for ds-cdma systems. *IEEE Transactions on Signal Processing*, 48(3):810–823, 2000.

[81] Shaden Smith, Jee W. Choi, Jiajia Li, Richard Vuduc, Jongsoo Park, Xing Liu, and George Karypis. FROSTT: The formidable repository of open sparse tensors and tools, 2017.

[82] Sucheta Soundarajan, Acar Tamersoy, Elias B Khalil, Tina Eliassi-Rad, Duen Horng Chau, Brian Gallagher, and Kevin Roundy. Generating graph snapshots from streaming edge data. In *Proceedings of the 25th International Conference Companion on World Wide Web*, pages 109–110, 2016.

[83] Rajmonda Sulo, Tanya Berger-Wolf, and Robert Grossman. Meaningful selection of temporal resolution for dynamic networks. In *Proceedings of the Eighth Workshop on Mining and Learning with Graphs*, pages 127–136, 2010.

[84] Jimeng Sun, Christos Faloutsos, Spiros Papadimitriou, and Philip S Yu. Graphscope: parameter-free mining of large time-evolving graphs. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 687–696, 2007.

[85] Ledyard R Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, 1966.

[86] Geoffrey I Webb, Roy Hyde, Hong Cao, Hai Long Nguyen, and Francois Petitjean. Characterizing concept drift. *Data Mining and Knowledge Discovery*, 30(4):964–994, 2016.

[87] Geoffrey I Webb, Loong Kuan Lee, François Petitjean, and Bart Goethals. Understanding concept drift. *arXiv preprint arXiv:1704.00362*, 2017.

[88] Alex H Williams, Tony Hyun Kim, Forea Wang, Saurabh Vyas, Stephen I Ryu, Krishna V Shenoy, Mark Schnitzer, Tamara G Kolda, and Surya Ganguli. Unsupervised discovery of demixed, low-dimensional neural dynamics across multiple timescales through tensor component analysis. *Neuron*, 98(6):1099–1115, 2018.

[89] Dingqi Yang, Daqing Zhang, Vincent W Zheng, and Zhiyong Yu. Modeling user activity preference by leveraging user spatial temporal characteristics in lbsns. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 45(1):129–142, 2014.

[90] Kai Yang, Xiang Li, Haifeng Liu, Jing Mei, Guotong Xie, Junfeng Zhao, Bing Xie, and Fei Wang. Tagited: Predictive task guided tensor decomposition for representation learning from electronic health records. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[91] Kejing Yin, William K Cheung, Yang Liu, Benjamin CM Fung, and Jonathan Poon. Joint learning of phenotypes and diagnosis-medication correspondence via hidden interaction tensor factorization. In *IJCAI*, pages 3627–3633, 2018.

[92] Yu Zheng, Licia Capra, Ouri Wolfson, and Hai Yang. Urban computing: concepts, methodologies, and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 5(3):1–55, 2014.

[93] Shuo Zhou, Nguyen Xuan Vinh, James Bailey, Yunzhe Jia, and Ian Davidson. Accelerating online cp decompositions for higher order tensors. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1375–1384, 2016.