Distributed Tensor Decomposition for Large Scale Health Analytics

Huan He
Emory University
Atlanta, Georgia
huan.he@emory.edu

Jette Henderson
CognitiveScale
Austin, Texas
jette.henderson@gmail.com

Joyce C. Ho
Emory University
Atlanta, Georgia
joyce.c.ho.emory.edu

ABSTRACT

In the past few decades, there has been rapid growth in quantity and variety of healthcare data. These large sets of data are usually high dimensional (*e.g.* patients, their diagnoses, and medications to treat their diagnoses) and cannot be adequately represented as matrices. Thus, many existing algorithms can not analyze them. To accommodate these high dimensional data, tensor factorization, which can be viewed as a higher-order extension of methods like PCA, has attracted much attention and emerged as a promising solution. However, tensor factorization is a computationally expensive task, and existing methods developed to factor large tensors are not flexible enough for real-world situations.

To address this scaling problem more efficiently, we introduce `SGranite`, a distributed, scalable, and sparse tensor factorization method fit through stochastic gradient descent. `SGranite` offers three contributions: (1) *Scalability*: it employs a block partitioning and parallel processing design and thus scales to large tensors, (2) *Accuracy*: we show that our method can achieve results faster without sacrificing the quality of the tensor decomposition, and (3) *FlexibleConstraints*: we show our approach can encompass various kinds of constraints including l_2 norm, l_1 norm, and logistic regularization. We demonstrate `SGranite`'s capabilities in two real-world use cases. In the first, we use Google searches for flu-like symptoms to characterize and predict influenza patterns. In the second, we use `SGranite` to extract clinically interesting sets (i.e., phenotypes) of patients from electronic health records. Through these case studies, we show `SGranite` has the potential to be used to rapidly characterize, predict, and manage a large multimodal datasets, thereby promising a novel, data-driven solution that can benefit very large segments of the population.

CCS CONCEPTS

• **Information systems → Web mining**; **Data extraction and integration**; • **Computing methodologies → Factor analysis**; **Canonical correlation analysis**; **MapReduce algorithms**; • **Applied computing → Health informatics**.

KEYWORDS

Web Mining; User-Generated Content; Health Analytics; Tensor Decomposition; Distributed Algorithm; Apache Spark

ACM Reference Format:
Huan He, Jette Henderson, and Joyce C. Ho. 2019. Distributed Tensor Decomposition for Large Scale Health Analytics. In *Proceedings of the 2019 World Wide Web Conference (WWW '19), May 13–17, 2019, San Francisco, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3308558.3313548

1 INTRODUCTION

Increasingly large amounts of health-related data are released on the Internet and have great potential for enabling better disease surveillance and disease management. As a motivating example, search activities on diseases such as influenza can be used and correlated with actual influenza surveillance data. Estimation of influenza-like illness (ILI) rates is a well-studied task [25, 32], Google Flu Trends, while flawed, demonstrated a link between influenza related search queries and the Centers for Disease Control and Prevention's (CDC) ILI rates[19]. Similarly, programs such as the National Institute of Health's All for Us NIH [2], are looking to gather data and make it publicly available to researchers to enable precision medicine. Extracting influenza patterns or clinical characteristics from such high-dimensional data can pose challenges, even before considering whether the data has been appropriately labeled.

A vast majority of the algorithms for disease surveillance or disease prediction adopt a supervised learning approach, but the need for labels can limit the possible scope of the task. However, unsupervised learning methods such as tensor factorization have been successfully applied in many application domains including social network analysis [29, 30, 39] and health analytics [15, 16, 18, 22, 36]. Tensors can succinctly represent high-dimensional data, including various representations of time or different sources of data. For example, an existing work showed that factorizing a tensor that grouped ILI historical statistics by year, week, and region could tease out patterns that are commonly based on the weeks that influenza is highest, deliver insight into the degree to which regions are similar or different from one another in terms of influenza, and capture the changes in ILI intensity from one year to the next [13]. Moreover, a variety of constraints can be placed on the learned latent factors to extract meaningful patterns and reduce overfitting. Yet, efficient tensor decomposition of large datasets in the presence of such constraints can be challenging.

In this paper, we propose `SGranite`, a distributed tensor decomposition framework that can incorporate a variety of regularization terms to constrain the latent factors. In particular, we show that integrating three forms of regularization terms can achieve easier-to-interpret factors, provide robustness in the presence of noise, and map to existing domain knowledge. Moreover, `SGranite` is very fast and scalable. Using a Spark-based implementation, we demonstrate the ability to decrease computation time by distributing both the data as well as the parameters without sacrificing accuracy. To promote reproducibility, our code is open-sourced and available on GitHub[1].

The contributions of our work can be summarized as follows:

- **Flexibility:** Our framework supports a variety of meaningful constraints such as sparsity, diversity, and distinguishability.
- **Scalability:** Our scalability analysis of `SGranite` on a large tensor constructed from healthcare data achieves near linearity speed-up as we scale to the number of machines. Moreover, our framework achieves at least a 4× speed-up compared to an existing state-of-the-art distributed tensor factorization method.
- **Accuracy:** Our empirical results in two health-related case studies show that incorporating the variety of constraints improves interpretability and robustness compared to the standard decomposition models.

Table 1 summarizes the contributions in the context of existing works.

2 BACKGROUND AND NOTATION

This section briefly introduces tensors and a popular tensor decomposition model. We refer the reader to [24, 34] for comprehensive overviews of practical tensor decompositions.

2.1 Tensors

Tensors are generalizations of matrices and vectors to higher dimensions. An N-way tensor is denoted as $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ and each cell of the tensor represents the interactions between N types of data. Each dimension of the tensor is referred to as a *mode*. Tensors can be unfolded or flattened as a matrix, which is called *matricization*. $\mathbf{X}_{(n)}$ denotes the matricization of tensor $\mathcal{X}$ along mode-n. Table 2 lists the operations and symbols used in this paper.

Definition 2.1. A rank-one N-way tensor is the outer product of N vector

$$\mathcal{X} = a^{(1)} \circ a^{(2)} \circ \cdots \circ a^{(N)}.$$

Each element of a rank-one tensor is the product of the corresponding vector elements (i.e., $x_{i_1 i_2 i_3 \cdots i_n} = a^{(1)}_{i_1} \circ a^{(2)}_{i_2} \circ \cdots \circ a^{(N)}_{i_N}$).

2.2 Tensor Decompositions

The CANDECOMP / PARAFAC (CP) model [7, 14] is one of the most popular and well-studied tensor decomposition method. In CP decomposition, the tensor is factored into a sum of rank-one tensors:

$$\mathcal{X} \approx \mathcal{M} = \sum_{r=1}^{R} \mathbf{A}^{(1)}(:,r) \circ \mathbf{A}^{(2)}(:,r) \circ \cdots \circ \mathbf{A}^{(n)}(:,r),$$

where $\mathbf{A}^{(n)}(:,r)$ is the rth column of $\mathbf{A}^{(n)}$. CP-decomposition can be also expressed as $[[\lambda; \mathbf{A}^{(1)}; \mathbf{A}^{(2)}; \cdots; \mathbf{A}^{(n)}]]$, where λ is a vector of the weights λ_r. Several benefits of the CP decomposition includes its intuitive output structure, uniqueness property that makes the model reliable to interpret, and the ability to learn a model even with relatively small amount of observations [24]. Figure 1 provides

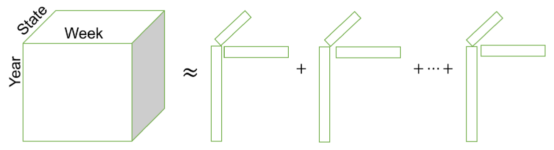


Figure 1: An example of CP decomposition for influenza search data. A tensor is constructed of time series data is decomposed into the weighted sum of rank-one tensors based on the minimization of an objective function. Each rank-one tensor, formed by taking the outer product of factor vectors, constitutes a latent factor.

an example of the CP decomposition for an influenza-based tensor, where each rank-one tensor represents a pattern over time for a group of states and a set of search queries.

The CP decomposition, $\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \cdots, \mathbf{A}^{(n)}$, is performed by minimizing the loss between $\mathcal{X}$ and $\mathcal{M}$ as defined by an objective function. The form of the objective function is determined by assumptions about how the data in the tensor was generated. A standard method of fitting a CP decomposition is the least squares formulation, which assumes the random variation in the tensor data follows a Gaussian distribution. Unfortunately, this may not be appropriate for count data, which is common in many applications including those considered in this paper [17].

A more appropriate objective function for count data assumes the underlying distribution of the data is Poisson [8]. This assumption results in an objective function that minimizes the Kullback-Leibler (KL) divergence:

$$f(\mathcal{M}) = \sum_i m_i - x_i \log m_i.$$

There have been several recent works to accelerate the computational speed of the CP decomposition. FlexiFact partitions the tensor into smaller tensors that are then decomposed in parallel using Apache Hadoop. While their method scales well with the input size, the model is not well-suited for count data and performs excessive disk operations. More recently, DisTenC, a Spark-based distributed tensor completion algorithm was proposed that regularizes the trace norm of the tensor [11]. Similar to FlexiFact, it is designed for numeric data and does not support any regularization or constraints on the factors.

3 SGRANITE

We propose `SGranite`, a distributed and flexible constrained CP model, to impose a variety of constraints on the latent factors. Our algorithm uses distributed stochastic gradient descent (DSGD) approaches to scale the CP decomposition on count data to huge datasets. `SGranite` has the following benefits:

- Simultaneously supports multiple constraints on the factor matrices.
- Learns patterns even when data cannot be stored on a single server.
- Maintains computational efficiency across a large number of workers.

[1]https://github.com/hehuannb/SGranite-WWW

Table 1: A comparison of the supported features between `SGranite` versus state of arts methods

Feature	`SGranite`	CP-APR [8]	Granite [16] (fit using SGD)	FlexiFact[6]	DisTenC [11]
Scalable	Yes	No	Yes	Yes	Yes
Memory Efficient	Yes	No	Yes	No	Yes
Time efficient	Yes	No	No	Yes	Yes
Appropriate for count data	Yes	Yes	Yes	No	No
Works with constraints	Yes	No	Yes	Yes	No

Table 2: Table of symbols and their associated definitions

Symbol	Definition
$\mathcal{X}$, $\mathbf{X}$, $\mathbf{x}$, x	Tensor, Matrix, Column Vector, Scalar
$\mathbf{X}_{(n)}$	n-mode matricization of a tensor $\mathcal{X}$
$\mathbf{X}(\mathbf{r},:)$	rth row of $\mathbf{X}$
$\mathbf{X}(:,\mathbf{r})$	rth column of $\mathbf{X}$
$\mathbf{A}^{(n)}$	nth factor matrix
$\|a\|_2$, $\|\mathbf{A}\|_F$	l_2 norm, Frobenius norm
$*$	Hadamard (elementwise) product
$\circ$	outer product
$\otimes$	Kronecker product
$\odot$	Khatri-Rao product (column-wise $\otimes$)

A distributed framework for incorporating a variety of constraints in CP decomposition is appealing for several reasons including the ability to extract patterns from large datasets that cannot be readily stored on a centralized server, to encode prior knowledge, to improve interpretability, and to democratize high-dimensional learning by running on standard commodity servers.

In this section, we will first provide a general overview and then formulate the optimization problem.

3.1 General Optimization Problem

`SGranite`, builds on several existing nonnegative CP decomposition algorithms to model sparse count data using the Poisson distribution [8, 16]. Let $\mathcal{X}$ denote an observed tensor constructed from count data with size $I_1 \times I_2 \times \cdots \times I_N$ and M represent a same-sized tensor of Poisson parameters for $\mathcal{X}$. In addition to KL divergence, we introduce generalized constraints on the factor matrices, $\mathcal{R}(\mathbf{A}^{(n)})$ to the objective function. Thus, the optimization problem is defined as:

$$\min f(\mathcal{M}) = \sum_{\vec{i}}(m_{\vec{i}} - x_i \log m_{\vec{i}}) + \sum_k \underbrace{\beta_k \mathcal{R}_k\left(\mathbf{A}^{(n)}\right)}_{\text{regularization terms}}$$

$$\text{s.t. } \mathcal{M} = [\![\boldsymbol{\lambda}; \mathbf{A}^{(1)}, \cdots, \mathbf{A}^{(N)}]\!] \qquad (1)$$

$$\lambda_r \geq 0, ||\mathbf{a}_r^{(n)}||_1 = 1, \ \forall r$$

$$\mathbf{A}^{(n)} \in [0,1]^{I_n \times R}, \ \forall n$$

The Poisson parameters, $\mathbf{m}$, can be determined by minimizing the negative log-likelihood of the observed data $\mathbf{x}$. We also maintain the stochasticity (i.e., elements sum to 1) and non-negativity constraints (i.e., factor elements and weights, or λ, must be non-negative) that were introduced in the original CP-APR model [8].

3.2 Example of Useful Regularization Terms

Equation 1 supports a variety of regularization items, $\mathcal{R}(\mathbf{A}^{(n)})$. While we describe three forms of special regularizations that are useful for analyzing health data, `SGranite` was developed to handle any regularization that is either smooth and differentiable or has an easy-to-compute proximal operator [31].

3.2.1 *Diversity on $\mathcal{A}^{(n)}$.*

For analyzing flu patterns or clinical characteristics of patient subgroups, it is preferable for the rank-one factor components to be distinct from each other. This allows domain experts to more easily interpret the patterns. While several mechanisms for encouraging diversity have been proposed [16, 21, 36], we adopt the angular penalty term in [16] that encourages diversity between rank-one tensors by penalizing overlapping elements. There are two benefits to this regularization. It does not require prior knowledge to construct a similarity matrix that is used in [21]. Similarly, it does not require the discovered patterns to be orthogonal to one another [36], which may be too restrictive. Under angular regularization, any element that has large values in multiple columns in the factor matrix are penalized. Thus, the angular penalty for the n^{th} factor matrix, $\mathbf{A}^{(n)}$, is formulated as follows:

$$\mathcal{R}_k\left(\mathbf{A}^{(n)}\right) = \sum_{r=1}^{R}\sum_{p=1}^{r} \max\left(0, \frac{(a_p^n)^T a_r^n}{||a_p^n||_2 ||a_r^n||_2} - \theta_n\right)^2$$

3.2.2 *Sparsity and Smoothness on $\mathcal{A}^{(n)}$.*

Sparsity and smoothness constraints have been introduced in a wide range of applications to improve interpretability and increase robustness to noise. Our framework supports a general class of ℓ_p penalties including simplex constraint term ($||\mathbf{a}_r||_1 = 1, a_{ir} \in [0,1]$); ℓ_2 regularization on the weight and the first factor matrix, $\lambda\mathcal{A}^{(1)}$ to mitigate overfitting to large count data; and the ℓ_0-norm regularization which caps the number of non-zeros elements in the factor.

We first consider the simplex constraint term, which can yield sparse factors while providing a probabilistic interpretation. For the n^{th} factor matrix, $\mathbf{A}^{(n)}$, we restrict the elements to lie on the ℓ_1-ball of diameter s, where s is a user-specified parameter, such that:

$$\mathcal{R}_k\left(\mathbf{A}^{(n)}\right) = \sum_{r=1}^{R}(s - ||\mathbf{a}_r^{(n)}||_1)$$

When $s = 1$, this results in the projection of the factor onto the probabilistic (or canonical) simplex [9]. By decreasing s to be less than 1, the resulting factors will be sparser.

The ℓ_2-norm regularization was introduced in [16] to encourage terms in the factor matrix vectors to be similar-sized. Together with the simplex projection, the interaction of these two regularizations achieved further sparsity by driving specific elements to 0 more quickly in a similar manner to the elastic net regularization [41].

$$\mathcal{R}_k\left(\mathbf{A}^{(n)}\right) = \sum_{r=1}^{R} \|a_r^{(n)}\|_2$$

The ℓ_0-norm regularization, introduced in [4], is an alternative to the simplex projection that limits the number of non-zero elements. While its usage in Equation 1 results in a non-convex optimization problem, the hard thresholding properties can yield easy to interpret factors (top-k elements). To perform hard-thresholding on the n^{th} factor matrix, $\mathbf{A}^{(n)}$, the regularization term is:

$$\mathcal{R}_k\left(\mathbf{A}^{(n)}\right) = \sum_{r=1}^{R} \|a_r^{(n)}\|_0$$

3.2.3 Discriminative Factors. In some scenarios, the discovered patterns should be discriminative of a certain outcome of interest. For example, we may want to use the clinical characteristics to predict things like mortality or whether or not the patient is likely to be readmitted in 30 days. [21] introduced a logistic regression regularization that encouraged the derivation of latent factors that can distinguish in-hospital mortality outcomes. `SGranite` also adopts the regularization term to derive discriminative latent factors when such information exists. Without loss of generality, we assume that the first mode has labeled records. Then the discriminative regularization is of the form:

$$\mathcal{R}_k\left(\mathbf{A}^{(1)}\right) = \log P(\mathcal{A}^{(1)}, y|\theta)$$

The probability of a sample $\mathbf{a}(i,:)$ (i^{th} row in $\mathbf{A}^{(1)}$) having the outcome of interest, $P(\mathcal{A}^{(1)}, y|\theta)$, is obtained by training a logistic regression model on the factor matrix $\mathbf{A}^{(1)}$.

3.2.4 Sparse, Diverse, and Discriminative Patterns. To demonstrate the flexibility of `SGranite`, we introduce all three forms of regularization into our final optimization problem. Thus, the final objective function is:

$$\begin{aligned} f(\mathcal{M}) = &\sum_{\vec{i}} (m_{\vec{i}} - x_i \log m_{\vec{i}}) + \\ &\beta_1 \sum_{n=1}^{N}\sum_{r=1}^{R}\sum_{p=1}^{r} \max\left(0, \frac{(a_p^n)^T a_r^n}{\|a_p^n\|_2 \|a_r^n\|_2} - \theta_n\right)^2 + \\ &\beta_2 \sum_{n=1}^{N}\sum_{r=1}^{R} (s - \|\mathbf{a}_r^{(n)}\|_2) + \beta_3 \log P(\mathcal{A}^{(1)}, y|\theta) \end{aligned} \tag{2}$$

3.3 SGD Updates

This section provides details of how to solve our optimization problem efficiently (Equation 2). `SGranite` uses an alternating minimization approach, cycling through each mode while fixing all the other modes. For each mode, the resulting subproblem is solved using stochastic gradient descent (SGD). To derive the SGD updates, we first re-write the objective function as a scalar-valued function of the parameter vector y using the same approach as [3]. The parameter vector y represents the vectorization of the factor matrices, with the weights $\boldsymbol{\lambda}$ absorbed into the first factor matrix.

$$y = \begin{bmatrix} vec(\boldsymbol{\lambda}\mathbf{A}^{(1)}) \\ vec(\mathbf{A}^{(2)}) \\ \vdots \\ vec(\mathbf{A}^{(n)}) \end{bmatrix}$$

As a result, the gradients of the objective function can be formed by vectorizing the partial derivatives with respect to each component of this parameter vector:

$$\nabla f(y) = \left[vec(\frac{\partial f}{\partial \mathbf{A}^{(1)}}) \cdots vec(\frac{\partial f}{\partial \mathbf{A}^{(n)}}) \right]$$

For notational convenience, we also represent the matricized form of the tensor decomposition as:

$$[\![\boldsymbol{\lambda}; \mathbf{A}^{(1)}, \cdots, \mathbf{A}^{(N)}]\!]_{(n)} = \boldsymbol{\lambda}\mathbf{A}^{(n)}(\mathbf{A}^{(-n)})^T$$

where

$$\mathbf{A}^{(-n)} = \mathbf{A}^{(N)} \odot \cdots \odot \mathbf{A}^{(n+1)} \odot \mathbf{A}^{(n-1)} \odot \cdots \odot \mathbf{A}^{(1)}.$$

Thus, the partial derivatives of Equation 2 with respect to the factor matrix, $\mathbf{A}^{(n)}$ are the following:

$$\begin{aligned} \frac{\partial f}{\partial A_r^{(n)}} = &\left[1 - X_{(n)} \oslash Z_{(n)}\right] a_r^{(-n)} + \\ &\beta_1 \sum_{p \neq r} \max\left(0, g(a_r^{(n)}, a_p^{(n)})\right) \frac{\partial g(a_r^{(n)}, a_p^{(n)})}{\partial \mathbf{a}_r^{(n)}} + \\ &\beta_2 \mathbf{a}_r^{(n)} + \beta_3 y \frac{1}{1 + \exp(\mathbf{y}\mathbf{A}_r^{(1)})} \theta \end{aligned} \tag{3}$$

We refer the reader to [16, 21] for the detailed derivation of the gradients.

For large datasets, the calculation of the derivatives simultaneously for all modes is computationally expensive. Thus, `SGranite` uses an SGD approach to avoid storing the entire tensor in memory. For faster convergence, we adopt a variant of SGD named Adaptive Moment Estimation (Adam) to adaptively update the learning rate [23]. Our preliminary experiments on a single machine showed that SGD with Adam converged faster and more accurately than using a fixed learning rate.

Algorithm 1 SGD updating process

```
1: for $l = 1 : L$ do
2:     Randomly select $n$ samples
3:     Calculate the gradients for samples using Equation 3
4:     Compute the decaying averages of past and past squared
       gradients
5:     Take a step using averaged gradients
6: end for
```

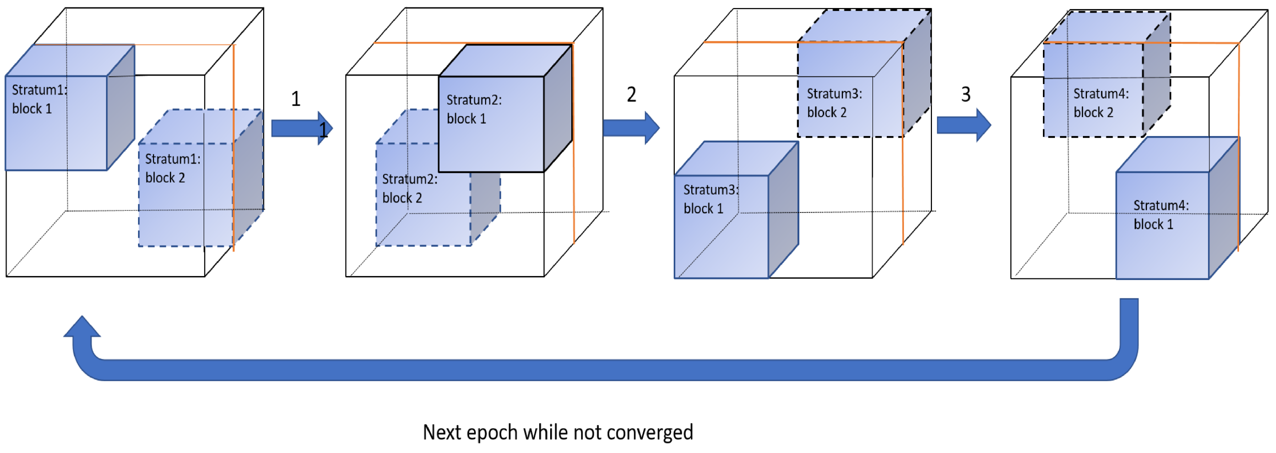


Figure 2: A graphical example of our `SGranite`: Suppose there are 2 workers, we will have 8 blocks and 4 strata after partition. We run this process iteratively until convergence. In each epoch, start from strata one, each worker runs SGD for its own assigned block in parallel. Check the convergence until all strata are iterated. We repeat above algorithm again if the stopping criteria is not satisfied. All intermediate results are saved as Resilient Distributed Datasets (RDD) collections and cached in memory.

3.4 Parallel Algorithm using Spark

Although SGD scales well to sparse data, we would like to distribute the computation to achieve results faster. FlexiFact proposed distributing the computation by dividing the tensor such that no two blocks share any elements (along with any dimension) [6]. Thus, the SGD algorithm can be run in parallel on each block without sacrificing accuracy. We refer the reader to [6, 12] for detailed proof of convergence. `SGranite` uses the same approach to distribute the non-zero elements of the count tensor using this tensor partition. However, we note several main differences between our framework and Flexifact: (1) support for sparse, count data by using an appropriate objective function (KL divergence), (2) flexibility to incorporate a variety of constraints beyond sparsity and non-negativity, and (3) distributed computation using Apache Spark.

Unlike `SGranite`, FlexiFact uses the Hadoop Map-Reduce platform to distribute the data collection across multiple nodes. Unfortunately, a Hadoop workflow spends an exorbitant amount of time on disk operations, as it needs to read and write intermediary results on the disk. On the other hand, Apache Spark [40] has been proposed as an alternative that eliminates unnecessary disk operations for iterative algorithms. By performing the data analytic operations in-memory and in near real-time, Spark can achieve lower computation times. Thus, `SGranite` is developed using Spark to distribute the computation.

3.4.1 *Tensor Partition.*

First, we define a stratum as a set of independent blocks, and we denote the number of blocks in each stratum by d. Suppose we have d available workers, in order to iterate all regions of $\mathcal{X}$, we need d^3 blocks and thus d^2 strata. For a stratum s we have d blocks $Z_i^{(s)}$ for $i = 0, 1, \cdots, n-1$. A detailed partition function for a size of $I \times J \times K$ tensor $\mathcal{X}$ is provided below:

$$\begin{aligned}
b_i &= (i\lceil I/d\rceil, (i+1)\lceil I/d\rceil)\\
b_j &= (j\lceil J/d\rceil, (j+1)\lceil J/d\rceil)\\
b_k &= (k\lceil K/d\rceil, (k+1)\lceil K/d\rceil)\\
j_{s,i} &= (j+s)\\
k_{s,i} &= (j+s) \mod d\\
Z_i^{(s)} &= \mathcal{X}_{(b_i, b_{j_{s,i}}, b_{k_{s,i}})}
\end{aligned} \tag{4}$$

Figure 2 provides an example of how to divide a count tensor for 2 available workers.

3.4.2 *Block Parallelization.*

Prior to introducing how `SGranite` iteratively solves the optimization problem in parallel, we introduce some definitions. A full epoch is defined as when the algorithm has seen all the d^3 blocks in the tensor. Since we need d^2 strata to cover all the blocks, we need to perform d^2 inner iterations. Therefore, we refer to each stratum training as a single inner iteration. Thus, in `SGranite`, the computation of each stratum is performed sequentially in each epoch. But for each stratum, we run SGD on the d^2 blocks in parallel. After each inner iteration, we update the factor matrices and use them as the initialization for the next stratum. Figure 3 provides an example of training using a single stratum. Upon the completion of an epoch (all strata have been run), the factor matrices are combined from all the workers, and then re-normalized for identifiability. The normalization can be performed for a user-specified mode, otherwise it defaults to the first mode. Convergence is checked between epochs by measuring the changes in the KL divergence to see if it is below a given tolerance. The details for the parallel-version of `SGranite` is described in Algorithm 2.

3.4.3 *Spark Implementation Details.*

The non-zero elements of the count tensor are stored in a list using the coordinate format and

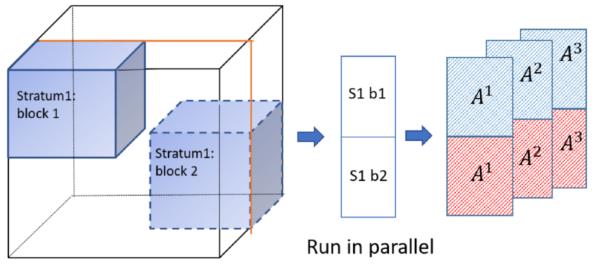


Figure 3: A graphical example of one stratum training: Given one stratum of training data and factor matrices $A^{(1)}, A^{(2)}, A^{(3)}$, we run SGD on each block in parallel. Then factor matrices $A^{(1)}, A^{(2)}, A^{(3)}$ are updated and used as the initialization for the next stratum training.

Algorithm 2 `SGranite`

```
1:  Randomly initialize $[\![\boldsymbol{\lambda}; \mathbf{A}^{(1)}; \cdots; \mathbf{A}^{(N)}]\!]$
2:  Partition the tensor and construct $d^2$ stratas using 4
3:  for $m = 1 : M$ do
4:      for $l = 1 : d^2$ do
5:          Assign each block in $l$th strata to a worker
6:          Each worker runs Algorithm 1 in parallel
7:          Update the factors $[\![\boldsymbol{\lambda}; \mathbf{A}^{(1)}; \cdots; \mathbf{A}^{(N)}]\!]$
8:      end for
9:      Gather results from each worker
10:     Normalize factor matrices according to the specified mode
11:     if converged then
12:         break
13:     end if
14: end for
15: Return $[\![\boldsymbol{\lambda}; \mathbf{A}^{(1)}; \cdots; \mathbf{A}^{(N)}]\!]$
```

loaded as Resilient Distributed Datasets (RDDs), and then it is shared throughout our cluster as a broadcast variable. A broadcast variable in Spark is immutable, meaning that it cannot be changed later on. This may seem inconvenient but it truly suits our case since we only need to read values from the tensor to calculate gradients in each iteration.

We do not broadcast factor matrices since we need to update them in each iteration. Due to our partition function, each worker has a chance to update factors matrices with different boundaries. The best way is to partition factor matrices using Block ID. In this way, we can reduce the memory and communication cost. Specifically, we applied `map` and `aggregateByKey` functions to partition the factor matrices into blocks. The function `map` transforms each entry of the sparse tensor into an element in the RDD whose key is a block ID. Then `aggregateByKey` groups each block together and persists in memory. In each inner iteration, we use `groupWith` to build a stratum partitioned using `partitionBy` and then use `mapPartitions` to assign tasks to each node.

We found storing factor matrices RDDs and partitioned result in a significant acceleration, but not doing this will cause virtual memory issues in our experiments. Our experiments suggest such a design will enable us to obtain better speed-up and scalability.

4 EXPERIMENTS AND RESULTS

In this section, we first provide descriptions for two real-world health datasets. We then give an overview of baseline methods and provide qualitative and quantitative results.

4.1 Datasets

We use the following two publicly available datasets:

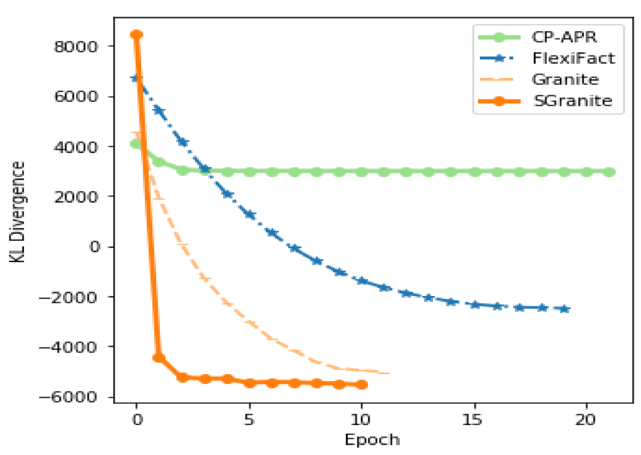
- Influenza: Using Google Flu Trends historical data[2] from 2003 to 2015, we generated a tensor to uncover temporal influenza patterns that are unique and similar across multiple states. For each region in the United States, we collected the number of search queries related to influenza on a weekly basis over 11 years. The resulting tensor is 12 regions × 52 weeks × 11 years. Although the data quality has been shown to be low Olson, Donald R et al. [28], this dataset is used to demonstrate the feasibility of `SGranite` on search data.
- MIMIC-III [20]: MIMIC-III is large database containing de-identified health data associated with approximately sixty thousand admissions of critical care unit patients from the Beth Israel Deaconess Medical Center collected between 2001 and 2012. For each patient, we extract medications and the International Classification of Diseases (ICD-9) diagnosis codes. ICD-9 codes are aggregated using Clinical Classification Software (CCS) categories[3], a standard preprocessing step in healthcare analysis. Similarly, medications are grouped using the Anatomical Therapeutic Chemical (ATC) Classification via the RxNorm RESTful Web API, a web service developed by the National Library of Medicine[4]. The aggregation step results in a 38159 patient × 234 diagnosis × 511 medication tensor.

4.2 Baseline Approaches

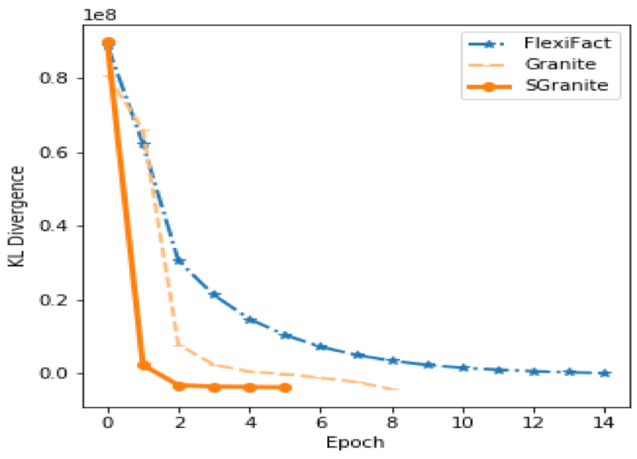
We will compare `SGranite` to both centralized and distributed CP decomposition methods.

- CP-APR [8]: The first algorithm proposed for modeling sparse count data using a Poisson distribution. There is no support for constraints, and the updates are performed using multiplicative updates. The algorithm has been ported to Python by the authors of [16].
- Granite [16]: A centralized extension of CP-APR that incorporates the angular penalty, ℓ_2, and the simplex projection as regularization terms. The authors shared a python implementation of Granite fit using SGD.
- FlexiFact [6]: A distributed algorithm based on the DSGD approach that factorizes a coupled tensor and matrix using a similar partition method. However, it uses least squares as an objective and only supports non-negativity and ℓ_1 constraints. For a fair comparison, we implemented the algorithm in Spark according to the paper.

[2]https://www.google.org/flutrends
[3]The mapping from ICD-9 to CCS can be found at https://www.hcup-us.ahrq.gov/toolssoftware/ccs/ccs.jsp.
[4]Details for the RxNorm RESTful Web API can be found at https://mor.nlm.nih.gov/download/rxnav/RxNormAPIREST.html.



(a) Influenza



(b) MIMIC

Figure 4: Comparison of two distributed and two non-distributed CP models using KL divergence. `SGranite` converges in less epochs than the other methods. The negative KL divergence arises from the fact that the observed values are not probability measurements.

4.3 Implementation and Hardware Details

`SGranite` is implemented in Python and the source code is publicly available[5]. The experiments were conducted on AWS. The cluster has one master and three worker nodes. Each node has four virtual cores and 16 GB of RAM. Results in our paper were reported using 4 workers.

4.3.1 Hyperparameter tuning. The logistic regularization penalty β_3, simplex projection β_2 and the angular penalty β_1 were set as 0.06, 0, and 0.02, respectfully. θ_n for angular penalty for each mode, the learning rate, and batch sizes were selected as 0.9, 0.0001, and 200 respectively. All hyperparameters were chosen based on a grid search over values.

4.4 Results

4.4.1 Scalability and Accuracy. First, we assess the quality of the approximation (measured by KL divergence) for `SGranite` and the other baseline methods. Figure 4 shows the KL divergence as the function of the number of epochs on both datasets. For the centralized algorithms (CP-APR and Granite), each epoch corresponds to a full iteration. The plots demonstrate that `SGranite` converges at least 4× faster than FlexiFact and also faster than the centralized algorithms. Moreover, the quality of the approximation is better than any of the existing methods. This suggests that SGD-based methods may help escape undesirable local minima (compared to CP-APR). The figure also highlights the importance of appropriately modeling the data distribution as opposed to using the least-squares loss (FlexiFact) may not yield the best approximation.

Next, we evaluate the scalability of our algorithm with respect to the number of workers. We calculate the speed-up as the ratio between the total execution time and the sequential execution time. Figure 5 demonstrates the speed-up of `SGranite` with respect to the number of workers. As can be seen in the figure, the speed up for the MIMIC tensor is very close to the ideal speed-up, as it is relatively large. However, there is a limited improvement on the Influenza tensor, a small dataset. This is due to the communication cost that is incurred in coordinating the different nodes. We note that because `SGranite` caches the updated factor matrices in memory to minimize disk accesses between consecutive iterations, it is able to scale to a large dataset and a large number of workers. Since this speed-up would not be possible on a system like Hadoop, we do not provide a comparison with FlexiFact.

[5]https://github.com/hehuannb/SGranite-WWW

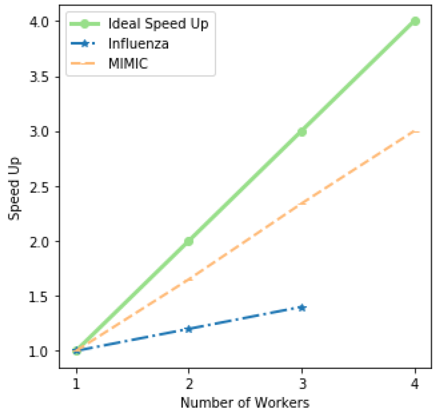
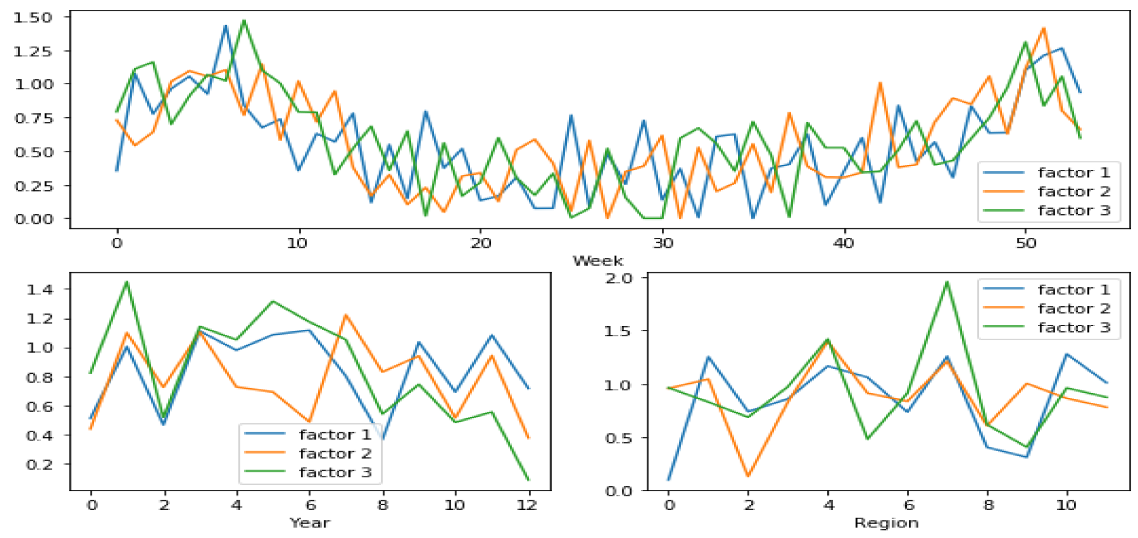
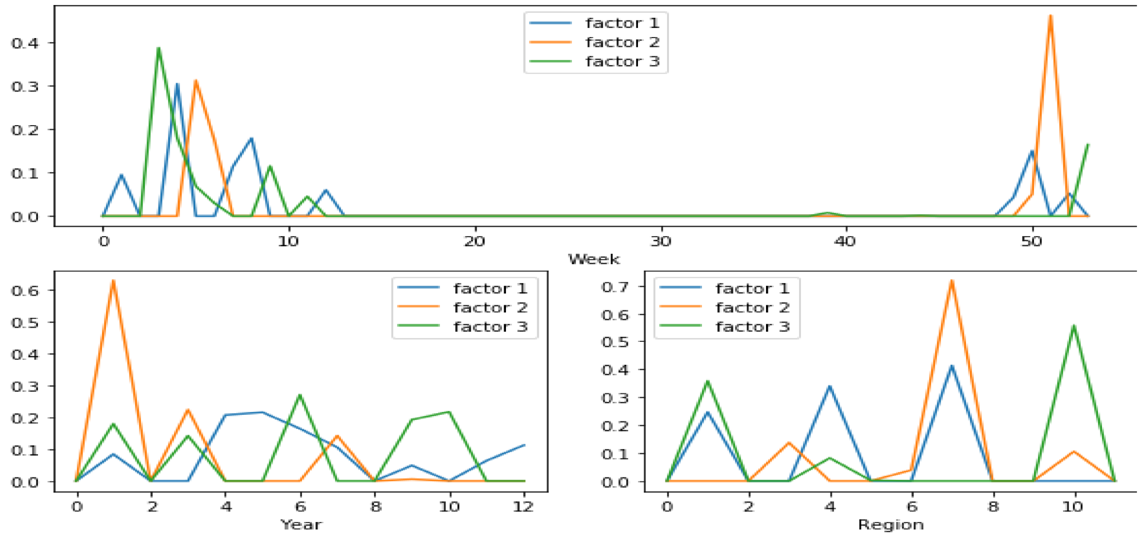


Figure 5: The speed-up curve for both datasets. It shows analysis of large datasets will gain an obvious speed up by using `SGranite`.

4.4.2 Qualitative and Quantitative Assessment of the Constraints. To examine the impact of the constraints, we first compare the results from `SGranite` on the influenza dataset both with and without the angular and simplex regularization terms. Figure 6a shows the



(a) No angular penalty and simplex projection ($\beta_1 = \beta_2 = 0$)



(b) Angular penalty and simplex projection constraints

Figure 6: A comparison of the learned latent factors with and without constraints using $R = 3$. Year from 2003 to 2015

latent factors learned without the regularization terms, and Figure 6b shows the latent factors learned with the regularization terms. From these plots, we observe that the learned factors without regularization are highly correlated and can be difficult to distinguish from one another. In Figure 6a, it is hard to discern any noticeable pattern across the weeks and the different regions. In comparison, Figure 6b demonstrates the potential of incorporating both diversity and sparsity. We can observe that factor 2 predominantly captures the peak influenza season that occurs both towards the end of December and in mid-February in region 7, whereas factor 3 is slightly delayed and captures the influenza trend in regions 1 and 10. Furthermore, all three factors capture the peak in influenza season that occurs in late December and early February through March.

Next, we quantitatively assessed the impact of the logistic regression and angular penalty on the MIMIC-III dataset. To evaluate the discriminative power and distinctiveness of the learned factors, we used the in-hospital mortality cohort similar to that proposed in [21]. We used 37,000 patients, including all 5,014 patients who died during admission. We split our dataset into 80% training and 20% testing. We measured the discrimination on the test set using the area under the receiver operating characteristic curve (AUC). Distinctiveness is measured using the average overlap or the degree of overlapping between latent factors. It is defined as the average of cosine similarities between all latent factor pairs:

$$\text{Avg Overlap} = \frac{\sum_{r_1}^{R}\sum_{r_2>r_1}^{R}\cos(a_{r_1}^{(2)}, a_{r_2}^{(2)}) + \cos(a_{r_1}^{(3)}, a_{r_2}^{(3)})}{R(R-1)}$$

Table 3 summarizes the AUC, total computation time (or running time), and the average overlap. We observe that SGranite can not only accelerate the tensor decomposition but also provides better prediction than other baseline methods. Moreover, the average overlap is smaller than Granite even without the angular constraints. This suggests that the partition function may also have some beneficial impact in terms of reducing overlapping factors. Moreover, incorporating the angular constraints further helps the discriminative ability of the model. This suggests that adding diversity constraints to yield less correlated latent features may also help the resulting predictive model. Therefore, SGranite supports a variety of flexible constraints and yields improved predictive performance.

Model	**AUC**	**Time**	**Avg Overlap**
CP-APR [8]	0.63	> 1 hour	0.3
FlexiFact [6]	0.65	35 mins	0.37
Granite [16]	0.67	> 1 hour	0.3
SGranite ($\beta_1 = 0$)	0.68	**20 mins**	0.1
SGranite ($\beta_1 > 0$)	**0.71**	25 mins	**0.07**

Table 3: Table of AUC, running time, and average overlapping using different methods. The highest AUC value means extracted phenotypes have stronger discrimination. The lowest running time indicates our distributed method can significantly accelerate the computation time. Compared to CP-APR and FlexiFact, adding angular penalty improved the distinction significantly.

4.4.3 Case Study 1: Flu Patterns. We provide a further qualitative assessment of our learned latent patterns from the influenza dataset. First, we comment on the ability to capture the overall flu season trends. Although flu season can vary across region to region, the flu season is typically between October through May (week 43 to week 22) [1]. We observe this phenomenon even with and without angular penalty constraints as illustrated in Figure 6. The variance in region and slight shifts in the week are further evident when angular penalty and simplex projection constraints are present (Figure 6b). We can see that some of the regions are present only in 1 of the factors. Moreover, slight shifts along the week are observed (top chart), depending on which latent factor with the higher elements occurring between weeks 48 and 13. This provides further confirmation that each region will have slightly different times when influenza will be more prominent.

We also assessed the learned flu patterns with FlexiFact, the other distributed CP algorithm that supports non-negativity and sparsity. Figure 7 presents the learned latent factors using FlexiFact. We observe that the peak level regions that are discovered using SGranite are more consistent with the CDC influenza positive test results, shown in Figure 8. The peaks that are discovered by FlexiFact are inconsistent with the observed CDC reports. FlexiFact factors suggest two different peaks, one between weeks 8-10 and one 18-20, whereas the CDC reports note a peak around 7-9 and by week 20, it has mostly died down. Moreover, we observe that the FlexiFact latent factors are more difficult to interpret as the region and year factors are fairly correlated. We also compared with the learned factors from a previous study [13] and found our learned patterns were more consistent with the observed results. This suggests that the incorporation of constraints not only improves interpretability but also provides robustness to noise.

4.4.4 Case Study 2: Phenotypes. We conducted a second case study to examine SGranite's ability to extract discriminative and distinct clinical characteristics from the MIMIC III dataset. The identification of clinical phenotypes from EHR data can help advance our understanding of disease risk and drug response as well as support the practice of precision medicine on a national scale [33, 38].

For clinicians, diversity is important to discover rare phenotypes in a patient population. Moreover, diverse phenotypes are likely easier to implement, as a clinician may find it difficult to rank-order or apply phenotypes that have substantial overlap. In addition, discriminative phenotypes are better predictors of mortality (shown in Table 3) and thus can be used to assist the decision-making process.

Table 4 presents the learned phenotypes that are important where importance is determined based on the magnitude of the phenotypes (or λ_r). Thus, these are the three sets of patient characteristics at which diagnosis and medication are dominant. First, we observe that the learned phenotypes have limited number of overlapping elements. In table 4, the most significant phenotype (λ_1) captures acute complications with heart diseases which can be riskier. In particular, acute respiratory distress syndrome has a mortality rate of 30-50% and is associated with long hospital stays [27]. The third phenotype (λ_3) captures more chronic diseases such as heart valve disorder, leukemias, and osteoarthritis. In addition, we observe that most medication codes in Table 4 are associated with diagnosis codes above. For example, potassiuman chloride and practolol are commonly used to lower blood pressure in hypertensives [10, 37]. An ACE inhibitor is used primarily for the treatment of hypertension and congestive heart failure [26]. And magnesium carbonate has shown to be effective for chronic kidney diseases and intracranial injury [5, 35].

5 CONCLUSION

In this paper, we presented a distributed, diverse, non-negative tensor decomposition framework that supports a variety of constraints including an angular penalty to encourage diversity and a simplex projection to encourage sparsity while scaling to large tensors. By imposing such regularization terms, SGranite successfully extracts meaningful latent factors in two real-world use cases. Moreover, by using Spark, SGranite successfully reduces processing time by dramatically reducing the workload and high communication cost. In addition, SGranite improves binary prediction tasks by incorporating logistic supervision into the fitting process. In the future, we plan to develop a distributed algorithm that can handle linear regression problem and also an extension that can use outside data sources as the guidance information.

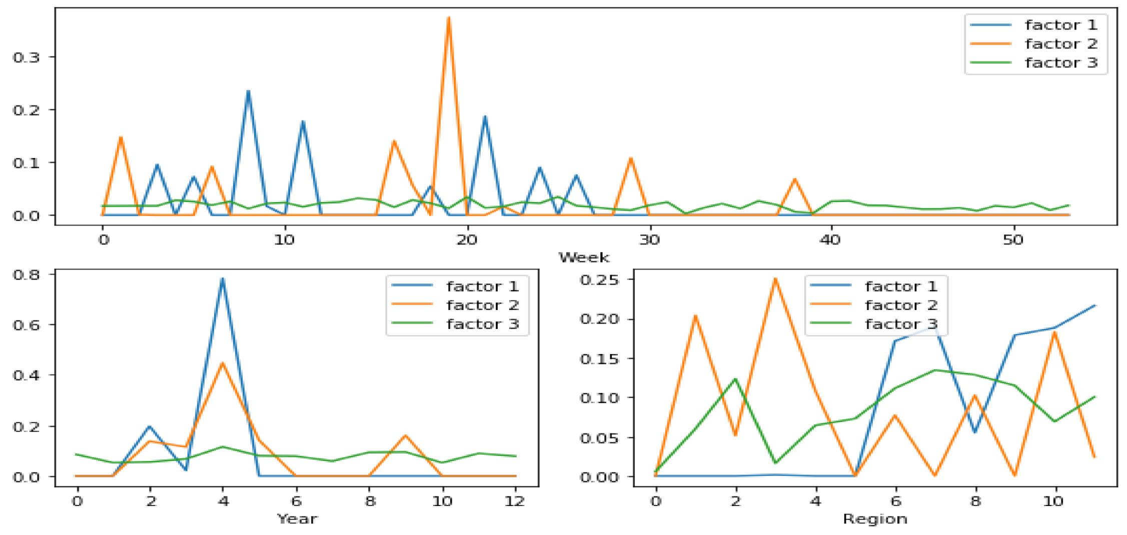


Figure 7: Latent factors obtained using Flexifact. Year from 2003 to 2015

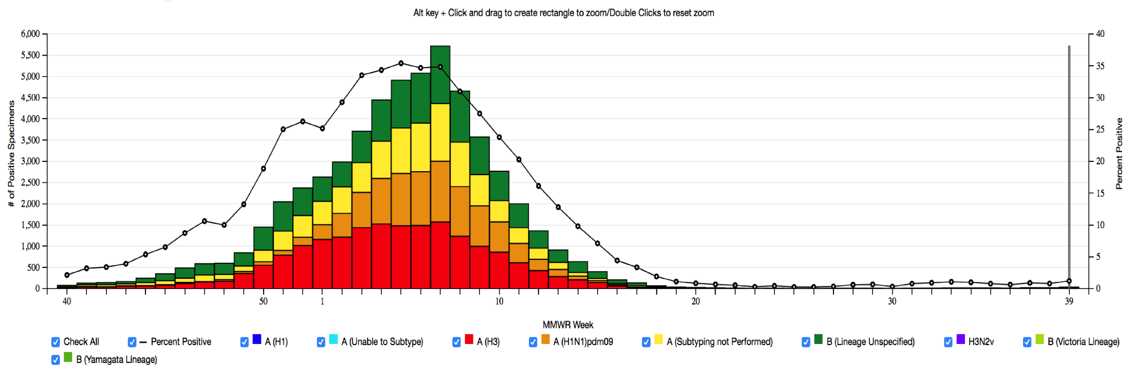


Figure 8: This figure is downloaded from CDC, it shows the actual influenza positive tests reported to CDC in 2010-2011, week ending Oct 01, 2001.

Phenotype 1: $\lambda_1 = 87$	**Phenotype 2:** $\lambda_1 = 79$	**Phenotype 3:** $\lambda_3 = 77$
Respiratory distress syndrome	Coronary atherosclerosis & other heart disease	Heart valve disorders
Acute cerebrovascular disease	Intracranial injury	Intracranial injury
Coronary atherosclerosis & other heart disease	Congestive heart failure; nonhypertensive	Leukemias
Hypertension w/ complications & secondary hypertension	Acute and unspecified renal	Osteoarthritis
Potassium chloride	Oxyphenisatine	Practolol
Sodium chloride	Adrenergics, inhalants	Magnesium carbonate
Omeprazole	ACE inhibitors	Anilides
Potassium chloride	Nitroprusside	Sodium chloride

Table 4: Table of top 3 phenotypes with high λ. Upper four rows are diagnosis codes and four rows below are medication codes correspondingly

6 ACKNOWLEDGEMENT

This work was supported by the National Science Foundation award IIS-#1838200 and the National Institute of Health award 1K01LM012924-01.

REFERENCES

[1] [n. d.]. Flu Season. https://en.wikipedia.org/wiki/Flu_season

[2] [n. d.]. National Institutes of Health. https://allofus.nih.gov/

[3] Evrim Acar, Daniel M Dunlavy, and Tamara G Kolda. 2011. A scalable optimization approach for fitting canonical tensor decompositions. *Journal of Chemometrics* 25, 2 (Feb. 2011), 67–86.

[4] Ardavan Afshar, Ioakeim Perros, Evangelos E Papalexakis, Elizabeth Searles, Joyce C Ho, and Jimeng Sun. 2018. COPA: Constrained PARAFAC2 for sparse & large datasets. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management.* 793–802.

[5] M F Arango and J H Mejia-Mantilla. 2006. Magnesium for acute traumatic brain injury.

[6] Alex Beutel, Partha Pratim Talukdar, Abhimanu Kumar, Christos Faloutsos, Evangelos E Papalexakis, and Eric P Xing. 2014. FlexiFaCT - Scalable Flexible Factorization of Coupled Tensors on Hadoop. *SDM* (2014).

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

[8] Eric C Chi and Tamara G Kolda. 2012. On tensors, sparsity, and nonnegative factorizations. *SIAM J. Matrix Anal. Appl.* 33, 4 (2012), 1272–1299.

[9] John C Duchi, Shai Shalev-Shwartz, Yoram Singer, and Tushar Chandra. 2008. Efficient projections onto the l1-ball for learning in high dimensions. *ICML* (2008).

[10] Tommaso Filippini, Federica Violi, Roberto D'Amico, and Marco Vinceti. 2017. The effect of potassium supplementation on blood pressure in hypertensive subjects: A systematic review and meta-analysis. *International Journal of Cardiology* 230 (March 2017), 127–135.

[11] Hancheng Ge, Kai Zhang, Majid Alfifi, Xia Hu, and James Caverlee. 2018. DisTenC: A aistributed algorithm for scalable tensor completion on Spark. In *Proceedings of the 2018 IEEE 34th International Conference on Data Engineering.* 137–148.

[12] Gemulla, Rainer, Nijkamp, Erik, Haas, Peter J, and Sismanis, Yannis. 2011. *Large-scale matrix factorization with distributed stochastic gradient descent.* ACM, New York, New York, USA.

[13] Michael Joseph Haass, Mark Hilary Van Benthem, and Edward M Ochoa. 2014. *Tensor analysis methods for activity characterization in spatiotemporal data.* Technical Report.

[14] Harshman, R A. 1970. Foundations of the PARAFAC procedure: Models and conditions for an" explanatory" multimodal factor analysis. (1970).

[15] Jette Henderson, Ryan Bridges, Joyce C Ho, Byron C Wallace, and Joydeep Ghosh. 2017. PheKnow-Cloud: A Tool for Evaluating High-Throughput Phenotype Candidates using Online Medical Literature. *AMIA Joint Summits on Translational Science proceedings. AMIA Joint Summits on Translational Science* 2017 (2017), 149–157.

[16] Jette Henderson, Joyce C Ho, Abel N Kho, Joshua C Denny, Bradley A Malin, Jimeng Sun, and Joydeep Ghosh. 2017. Granite - Diversified, Sparse Tensor Factorization for Electronic Health Record-Based Phenotyping. *ICHI* (2017).

[17] Jette Henderson, Bradley A Malin, Joshua C Denny, Able N Kho, Jimeng Sun, Joydeep Ghosh, and Joyce C Ho. 2019. CP Tensor Decomposition with Cannot-Link Intermode Constraints. In *SDM.*

[18] Joyce C Ho, Joydeep Ghosh, and Jimeng Sun. 2014. Marble - high-throughput phenotyping from electronic health records via sparse nonnegative tensor factorization. *KDD* (2014).

[19] Ginsberg J, Mohebbi MH, Patel RS, Brammer L, Smolinski MS, and Brilliant L. 2008. Detecting influenza epidemics using search engine query data. *Nature* 457, 7232 (2008).

[20] Alistair EW Johnson, Tom J Pollard, Lu Shen, H Lehman Li-wei, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. 2016. MIMIC-III, a freely accessible critical care database. *Scientific data* 3 (2016), 160035.

[21] Yejin Kim, Robert El-Kareh, Jimeng Sun, Hwanjo Yu, and Xiaoqian Jiang. 2017. Discriminative and Distinct Phenotyping by Constrained Tensor Factorization. *Scientific Reports* 7, 1 (April 2017), 1114.

[22] Yejin Kim, Jimeng Sun, Hwanjo Yu, and Xiaoqian Jiang. 2017. Federated Tensor Factorization for Computational Phenotyping. *KDD* (2017), 887–895.

[23] Diederik P Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *arXiv.org* (Dec. 2014), arXiv:1412.6980. arXiv:cs.LG/1412.6980

[24] Kolda, Tamara G and Bader, Brett W. 2009. Tensor Decompositions and Applications. *SIAM Rev.* 51, 3 (2009), 455–500.

[25] V Lampos, A C Miller, S Crossan, C Stefansen Scientific reports, and 2015. [n. d.]. Advances in nowcasting influenza-like illness rates using search query logs. *nature.com* ([n. d.]).

[26] ECK Li, B S Heran, JM Wright Cochrane Database of, and 2014. [n. d.]. Angiotensin converting enzyme (ACE) inhibitors versus angiotensin receptor blockers for primary hypertension. *cochranelibrary.com* ([n. d.]).

[27] FCCP Nathaniel Marchetti, DO. 2018. Acute Respiratory Distress Syndrome (ARDS) | CHEST Foundation. Retrieved January 2018 from https://foundation.chestnet.org/patient-education-resources/acute-respiratory-distress-syndrome-ards/

[28] Olson, Donald R, Konty, Kevin J, Paladini, Marc, Viboud, Cécile, and Simonsen, Lone. 2013. Reassessing Google Flu Trends Data for Detection of Seasonal and Pandemic Influenza - A Comparative Epidemiological Study at Three Geographic Scales. *PLoS Computational Biology* 9 (2013), e1003256–.

[29] Evangelos E Papalexakis. 2016. Automatic Unsupervised Tensor Mining with Quality Assessment. In *Proceedings of the 2016 SIAM International Conference on Data Mining.* Society for Industrial and Applied Mathematics, Philadelphia, PA, 711–719.

[30] Evangelos E Papalexakis, Christos Faloutsos, and Nicholas D Sidiropoulos. 2015. ParCube - Sparse Parallelizable CANDECOMP-PARAFAC Tensor Decomposition. *TKDD* 10, 1 (2015), 1–25.

[31] Neal Parikh, Stephen Boyd, et al. 2014. Proximal algorithms. *Foundations and Trends® in Optimization* 1, 3 (2014), 127–239.

[32] Philip M Polgreen, Yiling Chen, David M Pennock, and Forrest D Nelson. 2008. Using Internet Searches for Influenza Surveillance. *Clinical Infectious Diseases* 47, 11 (2008), 1443–1448.

[33] R L Richesson, W E Hammond, M Nahm Journal of the, and 2013. [n. d.]. Electronic health records based phenotyping in next-generation clinical trials: a perspective from the NIH Health Care Systems Collaboratory. *academic.oup.com* ([n. d.]).

[34] Sidiropoulos, Nicholas D, De Lathauwer, Lieven, Fu, Xiao, Huang, Kejun, Papalexakis, Evangelos E, and Faloutsos, Christos. 2016. Tensor Decomposition for Signal Processing and Machine Learning. *CoRR* stat.ML, 13 (2016), 3551–3582.

[35] David M Spiegel, Beverly Farmer, Gerard Smits, and Michel Chonchol. 2007. Magnesium Carbonate Is an Effective Phosphate Binder for Chronic Hemodialysis Patients: A Pilot Study. *Journal of Renal Nutrition* 17, 6 (2007), 416–422.

[36] Yichen Wang, Robert Chen, Joydeep Ghosh, Joshua C Denny, Abel N Kho, You Chen, Bradley A Malin, and Jimeng Sun. 2015. Rubik - Knowledge Guided Tensor Factorization and Completion for Health Data Analytics. *KDD* (2015), 1265–1274.

[37] Gavin WK Wong, Heidi N Boyda, and James M Wright. 2016. Blood pressure lowering efficacy of beta-1 selective beta blockers for primary hypertension. *Cochrane Database of Systematic Reviews* (2016).

[38] Pranjul Yadav, Michael Steinbach, Vipin Kumar, and György J Simon. 2018. Mining Electronic Health Records (EHRs) - A Survey. *ACM Comput. Surv.* (2018).

[39] Shihao Yang, Mauricio Santillana, John S Brownstein, Josh Gray, Stewart Richardson, and S C Kou. 2017. Using electronic health records and Internet search information for accurate influenza forecasting. *BMC Infectious Diseases* 17, 1 (May 2017), 89.

[40] M Zaharia, R S Xin, P Wendell, T Das Communications of the, and 2016. [n. d.]. Apache spark: a unified engine for big data processing. *dl.acm.org* ([n. d.]).

[41] Hui Zou and Trevor Hastie. 2005. Regularization and variable selection via the elastic net. *J. R. Stat. Soc. Series B Stat. Methodol.* 67, 2 (2005), 301–320.