SMF: Drift-Aware Matrix Factorization with Seasonal Patterns

Bryan Hooi[*] [†] Kijung Shin[*] Shenghua Liu[‡] Christos Faloutsos[*]

Abstract

Consider a stream of time-stamped events, such as taxi rides, where we record the start and end locations of each ride. How do we learn a matrix factorization model which takes into account seasonal patterns (such as: rides toward office areas occur more frequently in the morning), and use it to forecast taxi rides tomorrow? Also, how can we model drift (such as population growth), and detect sudden changes (or anomalies)? Existing matrix factorization algorithms do not take seasonal patterns into account. We propose SMF (Seasonal Matrix Factorization), a matrix factorization model for seasonal data, and a streaming algorithm for fitting it. SMF is (a) **accurate in forecasting:** outperforming baselines by 13% to 60% in RMSE; (b) **online:** requiring fixed memory even as more data is received over time, and scaling linearly; (c) **effective:** providing interpretable results. In addition, we propose SMF-A, an algorithm which detects anomalies in a computationally feasible way, without forecasting every observation in the matrix.

1 Introduction

Consider a stream of events, represented as tuples of the form (*entity*$_1$, *entity*$_2$, *time*). Given such data, a natural goal is to model patterns, and to forecast future data: for example, the number of taxi rides from Brooklyn to Manhattan tomorrow. Other similar applications includes disease forecasting, movie recommendation, retweet prediction, etc. In all these cases, seasonal patterns are present, and relevant to making accurate forecasts. Hence, our problem is:

Informal Problem 1. (Forecasting)

- ***Given:*** *a stream of past events, containing seasonal patterns;*
- ***Forecast:*** *the number of events between each pair of entities at any future time tick.*

Another goal is to detect when anomalies occur. These could involve a sudden increase in activity, but can also be any unusual shift in activity (e.g. a road accident redirecting traffic from one lane to another).

Informal Problem 2. (Anomaly Detection)

- ***Given:*** *a stream of past events, containing seasonal patterns;*
- ***Find:*** *a measure of how much each entity deviates from normal behavior at each time tick.*

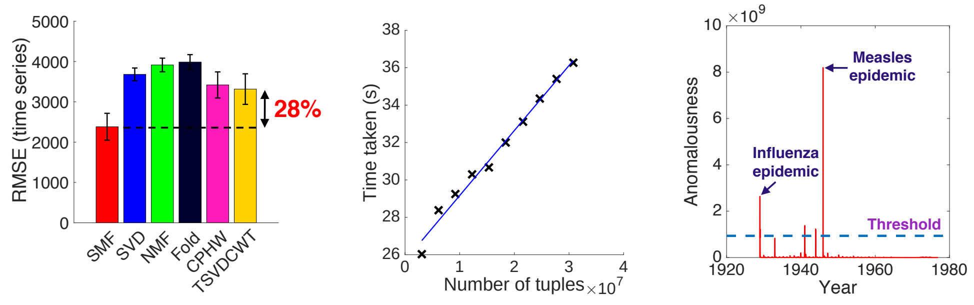
Standard matrix factorization approaches model this data by ignoring the time dimension, resulting in a matrix. However, these approaches ignore **seasonal** patterns. Taxi activity typically follows a daily bimodal pattern, peaking at morning and evening peak hours. In addition, standard matrix factorization cannot capture **drift**, or changes in the components over time: such as population growth, or people entering or leaving a community.

Scalability is also a major challenge, both in memory and running time, since matrix factorization often involves large numbers of entities. The entire dataset may not fit in memory, or may even have no finite size, in an **online** setting. Hence, we propose SMF (Seasonal Matrix Factorization), a drift-and-seasonality aware matrix factorization model which can be fit using an online algorithm. Our contributions are:

- **Model:** we propose a novel matrix factorization model incorporating seasonal patterns and drift, and an online algorithm for fitting this model.
- **Effectiveness:** in experiments, SMF has lower forecasting error than baselines by 13% to 60% (Figure 1a), and provides interpretable results in case studies on real data.
- **Scalability:** SMF is online, and scales linearly (Figure 1b). In experiments, it was 12 to 103 times faster than seasonal baselines.
- **Fast Anomaly Detection:** we propose SMF-A for detecting anomalies (Figure 1c) in a computationally feasible way, without forecasting every possible observation in the matrix.

Reproducibility: our code and datasets are publicly available at `www.andrew.cmu.edu/bhooi/smf`.

[*]School of Computer Science, Carnegie Mellon University
[†]Department of Statistics, Carnegie Mellon University
[‡]CAS Key Laboratory of Network Data Science & Technology, Institute of Computing Technology, Chinese Academy of Sciences



(a) Accurate (b) Scales linearly (c) Detects anomalies

Figure 1: **SMF is accurate, scales linearly, and detects anomalies.** (a) Forecast error of SMF compared to state-of-the-art baselines. (b) Running time scales linearly. (c) SMF-A detects two large epidemics, which have been previously reported in the medical literature, in a diseases dataset.

2 Background and Related Work

Static Matrix Factorization Matrix Factorization (MF) techniques including SVD [17], NMF [15], and pLSI [10] have been widely explored, particularly in collaborative filtering [13, 14, 21]. Other work incorporated bias terms [13], and alternating least squares [2].

Dynamic Matrix Factorization Time-weighting schemes weight past data by their recency [7, 8]. Other approaches include temporal regularization [23] and Kalman filters [21, 3]. timeSVD++ [14] modifies SVD with a temporal bias term. [6] uses dynamic MF with priors. [23] proposes a Bayesian approach. [20] proposes a dynamic tensor analysis algorithm. However, none of these consider seasonal patterns.

Seasonal Patterns in Matrix Factorization Fold [9] combines data at the same point in the season, then uses 3-way tensor decomposition (CPD). [22] similarly separates recurring patterns from outliers. CPHW [8] uses 3-way CPD, then extends the temporal factor matrix using the Holt-Winters algorithm. [5] uses a similar approach, also incorporating coupled tensor factorizations.

Why not use 3-way CPD? 3-way CPD [12] treats the temporal dimension as a discrete, unordered variable. Hence, it cannot be directly used for forecasting, and also does not model component drift. To forecast, we could modify it, e.g. as in Fold or CPHW. Compared to Fold and CPHW, which have fixed components, SMF allows both the components and seasonal patterns to drift: this includes both drifting component strength (e.g. a community growing more active) and drifting component structure (e.g. a community changing in composition). We verify that SMF learns meaningful such drifts in taxi data in Section 5.3, and our experiments show that SMF outperforms Fold and CPHW in forecasting accuracy. Another difference is that both Fold and CPHW are offline algorithms, while SMF is online.

Table 1: Comparison between methods.

	SVD/NMF [17, 15]	Dynamic MF [6, 7]	3-way CPD [12]	TimeCrunch [18]	Fold [9]	CPHW, etc [8, 5]	SMF
Component-based	✓	✓	✓		✓	✓	✓
Seasonal patterns				✓	✓	✓	✓
Drifting component strength		✓				✓	✓
Drifting component structure							✓
Online algorithm	some	✓	some				✓

3 Model

Preliminaries The input data is a series of sparse matrices $\mathbf{A}(t), t = 1, 2, \cdots, r$. For example, in the taxi case, if there were 100 taxi rides from location i to location j at time t, then the (i, j)th entry of $\mathbf{A}(t)$ is 100. Table 2 shows the symbols used. For matrix indexing, $\mathbf{X}(:, 1:2)$ is the submatrix of X with all its rows and the first 2 columns.

Table 2: Symbols and definitions

Symbol	Definition
$\mathbf{A}(t)$	$m \times n$ sparse data matrix at time t
m, n	Number of rows and columns in $\mathbf{A}(t)$
r	Number of time steps
k	Number of components
$\mathbf{u}_i(t), \mathbf{v}_i(t)$	Factorization component i at time t
$w_i(t)$	(Scalar) seasonal multiplier i at time t
$\mathbf{U}(t)$	$m \times k$ matrix form of $\mathbf{u}_i(t)$: $\mathbf{U}(t) = [\mathbf{u}_1(t) \cdots \mathbf{u}_k(t)]$
$\mathbf{V}(t)$	$n \times k$ matrix form of $\mathbf{v}_i(t)$: $\mathbf{V}(t) = [\mathbf{v}_1(t) \cdots \mathbf{v}_k(t)]$
$\mathbf{W}(t)$	$k \times k$ diagonal matrix: $\mathbf{W}(t) = \text{diag}(w_1(t), \cdots, w_k(t))$
s	Period (e.g. 7 for daily data with weekly periodicity)
α	Gradient step size hyperparameter
$z(t)$	Number of nonzeros in $\mathbf{A}(t)$
z	Total number of nonzeroes: $z = \sum_{t=1}^{r} z(t)$
$\mathbf{X}(:, 1:2)$	Submatrix of $\mathbf{X}$ with all rows and first 2 columns

3.1 Proposed SMF Model

To capture the desired seasonality patterns, we introduce **seasonal weights**

$w_i(t)$: $w_i(t)$ is a (scalar) multiplier that applies to component i at time t. Thus:

$$\mathbf{A}(t) \approx \sum_{i=1}^{k} \mathbf{u}_i(t) w_i(t) \mathbf{v}_i(t)^T \tag{3.1}$$

$w_i(t)$ will allow us to capture seasonal patterns, because we will ensure that $w_i(t)$ itself is close to periodic over time. Following Figure 2, in matrix notation this is:

$$\mathbf{A}(t) \approx \mathbf{U}(t)\mathbf{W}(t)\mathbf{V}(t)^T \tag{3.2}$$

We model the data using smoothly varying components

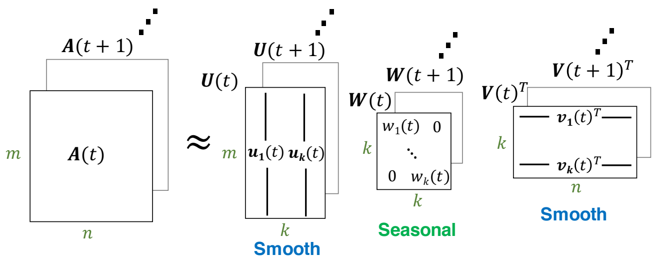


Figure 2: An illustration of our model. Section 4 explains how we model smoothness in **U** and **V**, and seasonality in w.

u and **v**, and seasonally varying 'multipliers' w, which govern the seasonal patterns in the data. This model captures multiple types of change: drifting component strength corresponds to variation in $w_i(t)$. Drifting community structure corresponds to variation in $\mathbf{u}_i(t)$.

4 Proposed SMF Algorithm

We now propose SMF, an online optimization algorithm, which has two steps: **Initialization**, where we use a short initial time period to train an initial model, then **Online Update**, where we repeatedly observe the next time point and update our model. Note that standard fitting methods cannot be used as they are generally offline: since we have defined $\mathbf{U}, \mathbf{W}, \mathbf{V}$ as functions of time, storing all of them simultaneously would require too much memory, and cause memory usage to grow over time.

4.1 Initialization Step We start by initializing $\mathbf{u}, \mathbf{v}$, and w. A reasonable initialization requires a few seasons of data: we use the first 3 seasons, following a common practice for initializing seasonal ETS models [11]. We thus 'stack' up $\mathbf{A}(1), \cdots, \mathbf{A}(3s)$ into a $m \times n \times 3s$ sparse tensor $\mathcal{T}_{\text{init}}$. Next, we 'fold' this into a $m \times n \times s$ sparse tensor $\mathcal{T}_{\text{fold}}$:

$$\mathcal{T}_{\text{fold}} = \frac{1}{3}\sum_{i=1}^{3} \mathcal{T}_{\text{init}}(:, :, (i-1)\cdot s + 1 : i \cdot s) \tag{4.3}$$

We then run nonnegative CP decomposition [24] on $\mathcal{T}_{\text{fold}}$. We use the resulting component as $\mathbf{U}(0)$ and $\mathbf{V}(0)$. For the third component, we use its value in component i at time t as the seasonal multipler $w_i(t)$, for $t = -s+1, \cdots, 0$. The negative indices for t are used so that starting at $t = 1$, we have valid values when we access the 'previous season' of $w_i(t)$. To allow the w_i to reflect component strength, we normalize $\mathbf{u}_i(0)$ by dividing by its norm $\|\mathbf{u}_i(0)\|$, compensating by multiplying $\|\mathbf{u}_i(0)\|$ into each of the $w_i(t)$ for $t = -s+1, \cdots, 0$ instead. We do the same for $\mathbf{v}_i(0)$.

4.2 Online Updates As we receive each $\mathbf{A}(t)$ for $t = 1, 2, \cdots$, we need to update $\mathbf{U}, \mathbf{V}$ and $\mathbf{W}$ in an online way to preserve good model fit. Assume that we have fit $\mathbf{U}, \mathbf{V}$ and $\mathbf{W}$ up to time $t-1$. At time t, we start by setting $\mathbf{U}(t)$ and $\mathbf{V}(t)$ equal to $\mathbf{U}(t-1)$ and $\mathbf{V}(t-1)$, and $\mathbf{W}(t)$ equal to $\mathbf{W}(t-s)$. We then adjust $\mathbf{U}$ and $\mathbf{V}$ by taking a small gradient step in the direction given by minimizing error with respect to $\mathbf{A}(t)$. The gradient step keeps error with respect to $\mathbf{A}(t)$ low (i.e. $\mathbf{A}(t) \approx \mathbf{U}(t)\mathbf{W}(t)\mathbf{V}(t)^T$). Taking a small step ensures that $\mathbf{U}$ and $\mathbf{V}$ are smooth ($\mathbf{U}(t) \approx \mathbf{U}(t-1)$), while $\mathbf{W}$ is near-seasonal ($\mathbf{W}(t) \approx \mathbf{W}(t-s)$). The fitted parameters 'track' the true values over time as we perform gradient updates. Meanwhile, each update is highly efficient as it only involves gradients with respect to $\mathbf{A}(t)$.

Let $\hat{\mathbf{A}}(t) = \mathbf{U}(t-1)\mathbf{W}(t-s)\mathbf{V}(t-1)^T$. For adjusting $\mathbf{u}_i(t)$ and $\mathbf{v}_i(t)$, the gradient update to $\mathbf{u}_i(t)$ and $\mathbf{v}_i(t)$ can be computed by differentiating fitting error. $\alpha > 0$ determines the learning rate.

$$\mathbf{u}_i(t) \leftarrow \mathbf{u}_i(t-1) + \alpha(\mathbf{A}(t) - \hat{\mathbf{A}}(t))\mathbf{v}_i(t-1)w_i(t-s)$$
$$\mathbf{v}_i(t) \leftarrow \mathbf{v}_i(t-1) + \alpha(\mathbf{A}(t) - \hat{\mathbf{A}}(t))^T\mathbf{u}_i(t-1)w_i(t-s)$$

Next, we ensure that the nonnegativity constraint is met by projecting $\mathbf{u}$ and $\mathbf{v}$: $\mathbf{u}_i(t) \leftarrow \max(0, \mathbf{u}_i(t))$ and $\mathbf{v}_i(t) \leftarrow \max(0, \mathbf{v}_i(t))$, where max is applied elementwise. Finally, we re-normalize $\mathbf{u}_i(t)$ and $\mathbf{v}_i(t)$ by dividing by their norms $\|\mathbf{u}_i(t)\|$ and $\|\mathbf{v}_i(t)\|$ respectively while multiplying these norms into $w_i(t)$:

$$w_i(t) \leftarrow w_i(t-s) \cdot \|\mathbf{u}_i(t)\| \cdot \|\mathbf{v}_i(t)\| \tag{4.4}$$

$$\mathbf{u}_i(t) \leftarrow \mathbf{u}_i(t)/\|\mathbf{u}_i(t)\|; \quad \mathbf{v}_i(t) \leftarrow \mathbf{v}_i(t)/\|\mathbf{v}_i(t)\| \tag{4.5}$$

This allows our seasonality pattern w to adapt over time, while also ensuring that the normalization constraint is met for **u** and **v**.

Note that only the last time step of **U** and **V**, and the last s time steps of **W** are needed at any time. This prevents memory usage from growing over time.

4.3 Speeding up Updates $\hat{\mathbf{A}}(t)$ is a dense $m \times n$ matrix, so explicitly forming it is inefficient both in

Algorithm 1: Online Updates

```
Input : Sparse matrices $\mathbf{A}(1), \cdots, \mathbf{A}(r)$,
        initialization for $\mathbf{U}, \mathbf{V}, \mathbf{W}$
Output: $\mathbf{U}, \mathbf{V}, \mathbf{W}$
1  for $t = 1$ to $r$ do
2      ▷Perform gradient updates
3      $\hat{\mathbf{A}}(t) = \mathbf{U}(t-1)\mathbf{W}(t-s)\mathbf{V}(t-1)^T$
4      $\mathbf{U}(t) \leftarrow \mathbf{U}(t-1) + \alpha(\mathbf{A}(t) - \hat{\mathbf{A}}(t))\mathbf{V}(t-1)\mathbf{W}(t-s)$
       $\mathbf{V}(t) \leftarrow \mathbf{V}(t-1) + \alpha(\mathbf{A}(t) - \hat{\mathbf{A}}(t))^T\mathbf{U}(t-1)\mathbf{W}(t-s)$
       for $i = 1$ to $k$ do
5          ▷Project onto nonnegative constraint
6          $\mathbf{u}_i(t) \leftarrow \max(0, \mathbf{u}_i(t))$, $\mathbf{v}_i(t) \leftarrow \max(0, \mathbf{v}_i(t))$
7          ▷Renormalize and multiply into $w$
8          $w_i(t) \leftarrow w_i(t-s) \cdot \|\mathbf{u}_i(t)\| \cdot \|\mathbf{v}_i(t)\|$
9          $\mathbf{u}_i(t) \leftarrow \mathbf{u}_i(t)/\|\mathbf{u}_i(t)\|$
10         $\mathbf{v}_i(t) \leftarrow \mathbf{v}_i(t)/\|\mathbf{v}_i(t)\|$
```

running time and memory. Instead, we can rewrite the gradient updates in Lines 4 and 4 of Algorithm 1 into a more efficiently computable form. Let $z(t)$ be the number of nonzeroes in $\mathbf{A}(t)$.

LEMMA 4.1. $(\mathbf{A}(t) - \hat{\mathbf{A}}(t))\mathbf{V}(t-1)$ *can be computed in* $O(kz(t) + k^2(m+n))$ *time.*

Proof.

$$\begin{aligned}&(\mathbf{A}(t) - \hat{\mathbf{A}}(t))\mathbf{V}(t-1)\\ &= (\mathbf{A}(t) - \mathbf{U}(t-1)\mathbf{W}(t-s)\mathbf{V}(t-1)^T)\mathbf{V}(t-1)\\ &= \mathbf{A}(t)\mathbf{V}(t-1) - \mathbf{U}(t-1)\mathbf{W}(t-s)(\mathbf{V}(t-1)^T\mathbf{V}(t-1)).\end{aligned}$$

Performing the $\mathbf{V}(t-1)^T\mathbf{V}(t-1)$ multiplication produces a $k \times k$ matrix, so the subsequent multiplications are fast. Specifically, performing $\mathbf{V}(t-1)^T\mathbf{V}(t-1)$ takes $O(nk^2)$ time, while multiplying it by $\mathbf{U}(t-1)$ takes $O(mk^2)$ time. $\mathbf{A}(t)\mathbf{V}(t-1)$ requires $O(kz(t)+nk)$ time, which add up to give $O(kz(t) + k^2(m+n))$.

Letting $z = \sum_{t=1}^{r} z(t)$ be the number of nonzeroes:

LEMMA 4.2. *Algorithm 1 is* $O(kz + rk^2(m+n))$*.*

Proof. By Lemma 4.1, lines 4 and 4 take $O(kz(t) + k^2(m+n))$ time. Lines 5 to 10 are $O(m+n)$, so the inner loop (line 4) is $O(k(m+n))$. For the outer loop, summing $O(kz(t) + k^2(m+n))$ over $t = 1, \cdots, r$ gives $O(kz + rk^2(m+n))$.

4.4 Forecasting Given any $t > r$, we forecast $\mathbf{A}(t)$ using the most recent $\mathbf{U}$ and $\mathbf{V}$ (i.e. $\mathbf{U}(r)$ and $\mathbf{V}(r)$) and the $\mathbf{W}$ in the most recent season at the time corresponding to t (e.g. forecasting next Monday using last Monday), i.e. t_{seas} where $t_{\text{seas}} = r - (r - t \bmod s)$, where mod is the modulo operation:

$$\hat{\mathbf{A}}(t) = \mathbf{U}(r)\mathbf{W}(t_{\text{seas}})\mathbf{V}(r)^T \tag{4.6}$$

4.5 Anomaly Detection: SMF-A Algorithm

Having fit the above model, how do we identify anomalies, e.g. an epidemic, or a road diversion? How anomalous is entity i at time t? We measure anomalousness of an entity at time t is by its **residuals**: i.e. the difference between its observed data at time t, and our model's fitted values. If a large anomaly occurred at time t, this difference will be large.

Define the fitted values as $\tilde{\mathbf{A}}(t) = \mathbf{U}(t)\mathbf{W}(t)\mathbf{V}(t)^T$. Then for any entity i in the first mode (i.e. row i), its anomalousness is the sum of squared differences between the data and the fitted values:

DEFINITION 1. (ROW ANOMALOUSNESS)

$$Anom_i(t) = \sum_{j=1}^{n}(\mathbf{A}_{ij}(t) - \tilde{\mathbf{A}}_{ij}(t))^2 \tag{4.7}$$

Anomalousness along the second mode is analogous.

Eq. (4.7) is infeasible to compute directly as $\tilde{\mathbf{A}}(t)$ is a dense $m \times n$ matrix. We now show that Eq. (4.7) can be computed more efficiently: to do this, we first rewrite $\text{Anom}_i(t)$ into a form such that the slowest part of its computation can be precomputed and then re-used when computing $\text{Anom}_i(t)$ for each i. Across m entities, this provides large savings (up to a factor of m). In the following, we suppress the 't' notation since all terms are taken at time t.

LEMMA 4.3. *An equivalent, faster to compute form is:*

$$\begin{aligned}Anom_i &= \mathbf{U}(i,:)\mathbf{W}\mathbf{V}^T\mathbf{V}\mathbf{W}\mathbf{U}(i,:)^T\\ &\quad + \sum_{i,j:\mathbf{A}_{ij}>0}\left((\mathbf{A}_{ij} - \tilde{\mathbf{A}}_{ij})^2 - \tilde{\mathbf{A}}_{ij}^2\right).\end{aligned}$$

Proof. Note that $(\mathbf{A}_{ij} - \tilde{\mathbf{A}}_{ij})^2 = \tilde{\mathbf{A}}_{ij}^2$ when $\mathbf{A}_{ij} = 0$. Hence:

$$\begin{aligned}\text{Anom}_i &= \sum_{j=1}^{n}(\mathbf{A}_{ij} - \tilde{\mathbf{A}}_{ij})^2\\ &= \sum_{j:\mathbf{A}_{ij}=0}\tilde{\mathbf{A}}_{ij}^2 + \sum_{j:\mathbf{A}_{ij}>0}(\mathbf{A}_{ij} - \tilde{\mathbf{A}}_{ij})^2\\ &= \sum_{j=1}^{n}\tilde{\mathbf{A}}_{ij}^2 + \sum_{j:\mathbf{A}_{ij}>0}\left((\mathbf{A}_{ij} - \tilde{\mathbf{A}}_{ij})^2 - \tilde{\mathbf{A}}_{ij}^2\right)\\ &= \|\mathbf{U}(i,:)\mathbf{W}\mathbf{V}^T\|_2^2 + \sum_{j:\mathbf{A}_{ij}>0}\left((\mathbf{A}_{ij} - \tilde{\mathbf{A}}_{ij})^2 - \tilde{\mathbf{A}}_{ij}^2\right).\\ &= \mathbf{U}(i,:)\mathbf{W}\mathbf{V}^T\mathbf{V}\mathbf{W}\mathbf{U}(i,:)^T\\ &\quad + \sum_{j:\mathbf{A}_{ij}>0}\left((\mathbf{A}_{ij} - \tilde{\mathbf{A}}_{ij})^2 - \tilde{\mathbf{A}}_{ij}^2\right).\end{aligned}$$

The key point is that $\mathbf{W}\mathbf{V}^T\mathbf{V}\mathbf{W}$ can be computed once, then re-used for all i, greatly reducing runtime:

Lemma 4.4. *Computing $Anom_i$ for all i is $O(kz(t) + k^2(m+n))$.*

Proof. Computing $\mathbf{W}\mathbf{V}^T\mathbf{V}\mathbf{W}$ takes $O(mk^2)$ time. Then, computing $\mathbf{U}(i,:)\mathbf{W}\mathbf{V}^T\mathbf{V}\mathbf{W}\mathbf{U}(i,:)^T$ takes $O(k^2)$ for each i, thus $O(nk^2)$ overall. Computing the next term $\sum_{j:\mathbf{A}_{ij}>0}\left((\mathbf{A}_{ij} - \tilde{\mathbf{A}}_{ij})^2 - \tilde{\mathbf{A}}_{ij}^2\right)$ for every row i takes $O(z(t)k)$ time, since computing $\tilde{\mathbf{A}}_{ij}$ for each nonzero $\mathbf{A}_{ij}$ takes $O(k)$. Thus the total runtime is $O(kz(t) + k^2(m+n))$.

Identifying Anomalous Events Given the $\text{Anom}_i(t)$ scores, how do we identify when an anomaly occurred? One way would be to sum $\text{Anom}_i(t)$ over entities, and plot the resulting time series. However, some entities have much higher natural variation and thus larger typical values of $\text{Anom}_i(t)$ than others, and summing in this way would drown out other true anomalies. Hence, we instead use a 'majority vote' approach that aims for both accuracy and interpretability: intuitively, time points with much higher $\text{Anom}_i(t)$ scores than the next highest time point are particularly suspicious.

Definition 2. (Weighted Vote) *Each entity i votes for time point $t_i^{(1)} = \arg\max_t Anom_i(t)$, and the* **weight** *of this vote is $Anom_i(t_i^{(1)}) - Anom_i(t_i^{(2)})$, where $t_i^{(2)}$ is the time of the next highest $Anom_i(t)$.*

We repeat this for each entity i. Then, the final anomalousness of each time point is the sum of the weighted votes given to it. This allows for interpretability: 1) the set of time points with at least 1 vote acts as a restricted set that practitioners can focus their attention on. 2) Each time point in this set has an 'explanation' in the form of the entities that voted for it. Hence, a practitioner can examine whether this entity and time point are truly anomalous.

5 Experiments

We design experiments to answer the following:

- **Q1. Accuracy:** how accurately does SMF forecast?
- **Q2. Scalability:** how does it scale?
- **Q3. Real-World Effectiveness:** does it provide meaningful components and anomalies in real data?

Our code and datasets are publicly available at `www.andrew.cmu.edu/bhooi/smf`. We implement our algorithms in Matlab; experiments were done on a 2.4 GHz Intel Core i5 Macbook Pro, 16 GB RAM, running OS X 10.11.2. Dataset details are in Table 3. `NY-Taxi` data points are hourly, with weekly periodicity ($s = 168$). `Disease` data points are weekly, with yearly periodicity ($s = 52$).

	Rows	Columns	Time points	Nonzero entries
`NY-Taxi`	2167	2167	2184	28.5M
`Disease`	39	50	2601	0.5M
`Synthetic`	5000	5000	5000	31M

Table 3: Datasets used in our experiments.

Baselines Our baselines are static approaches 1) SVD and 2) NMF; the seasonal approaches 3) Fold [9] and 4) CPHW [8], and 5) TSVDCWT (Truncated SVD with Collapsed Weighted Tensors) [8], a dynamic (but non-seasonal) approach. [14, 6] are also dynamic approaches, but are designed for ratings (e.g. 1 to 5 stars) and do not work in our case. For fair comparison, we use $k = 15$ components for all algorithms. Since our algorithm uses nonnegative components, we also use nonnegative CPD for the Folding and CPHW baselines.

5.1 Q1: Forecasting Accuracy

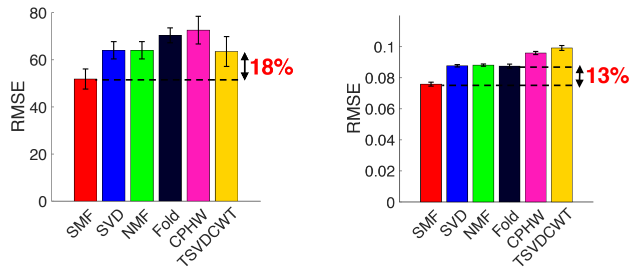
We evaluate SMF compared to baselines on `NY-Taxi` and `Disease`. Each algorithm takes the first r_{train} time steps and forecasts the next r_{test}. This is repeated for multiple values of r_{train}, and the results are averaged: for `NY-Taxi`, we use $r_{train} = 1600, 1800, 2000$ and $r_{test} = 100$. For `Disease`, we use $r_{train} = 1000, 1500, 2000$ and $r_{test} = 500$.

Metrics We use 1) **RMSE**; 2) Since RMSE values in large and sparse matrices are hard to interpret, **Time-series RMSE** aims to more directly answer questions like: 'how many taxi rides will happen each day from Brooklyn to Manhattan,' forecasting a subset of the matrix rows and columns, as a time-series. Moreover, we want this forecast to be accurate for any such subsets. Hence, in time-series RMSE, we select a random subset of rows and columns (each with probability 1/2). Each algorithm compute a time series of its forecasted number of events within this subset of rows and columns, and compares this to the true time series using RMSE. We average this result over 10 such random subsets.

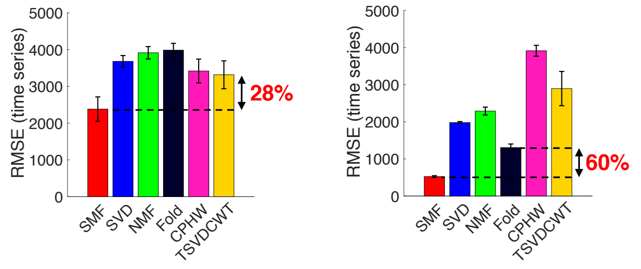
As seen in Figure 3, SMF outperforms baselines in accuracy. The baselines cannot capture changes in the components, or seasonal pattern, over time. Fairly large changes happen over time (e.g. see Section 5.3), so this causes high error.

5.2 Q2: Scalability

Computation time We use a 5000×5000 matrix for 5000 timesteps, with 200M tuples generated from a realistic power-law slice sum distribution in the first two modes (with exponent fitted to the `NY-Taxi` dataset), and a uniform distribution in the temporal mode. After combining overlaps, there are 31M nonzeroes. For Fig-



(a) RMSE (`Disease`) (b) RMSE (`NY-Taxi`)



(c) Time-series RMSE (`Disease`) (d) Time-series RMSE (`NY-Taxi`)

Figure 3: (a) **SMF outperforms baselines in accuracy:** SMF has 13% to 60% lower error than the best performing baseline. Error bars indicate one standard deviation.

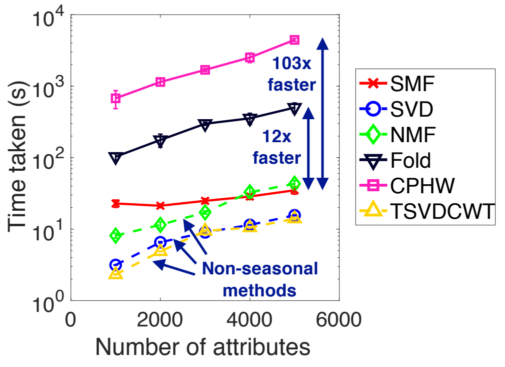
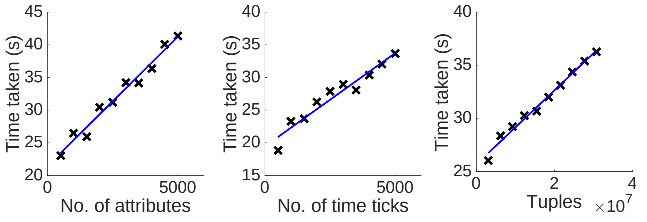


Figure 4: **SMF is fast:** it outperforms seasonal baselines, Fold and CPHW. Error bars (small) indicate one standard deviation.

ure 4, we subsample the first mode in $(1000, \cdots, 5000)$, and plot time taken against size. Each trial is averaged over 4 repetitions. Among the seasonal baselines (CPHW and Fold), CPHW is slowest as it performs CPD on the entire tensor. Fold performs CPD on a shrunken (folded) tensor. SMF is much faster than these, requiring time comparable to NMF. Note that SVD and NMF ignore temporal information completely, operating on a static matrix, and are expected to be fast.

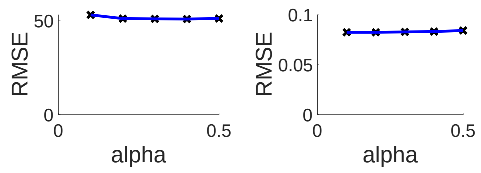
In an online setting where we require incremental results, the offline methods would need to be re-run for each time step, while SMF would not. Figure 4 does not take this into account, and runs each algorithm on the whole dataset. In such an online setting, the speedup of SMF over CPHW and Fold would be much greater. Figure 5 shows that SMF scales linearly in attributes (a), timesteps (b), and entries (c).



(a) Attributes (b) Time ticks (c) Nonzero entries

Figure 5: (a) **SMF scales linearly**.

Parameter Selection We select α using cross-validation between $\{0.1, \cdots, 0.5\}$, producing 0.3 and 0.1 on `Disease` and `NY-Taxi`. The period s often can be deduced from domain knowledge, but if needed, it can also be estimated by cross validation between a few reasonable candidates (daily, weekly, yearly, etc.). Figure 6 shows that SMF is insensitive to the hyperparameter α, and performs well in all cases.

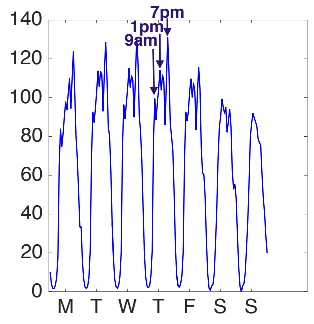


(a) `Disease` (b) `NY-Taxi`

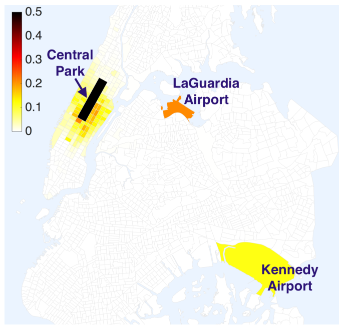
Figure 6: SMF performs well across parameter values.

5.3 Q3: Real-World Effectiveness

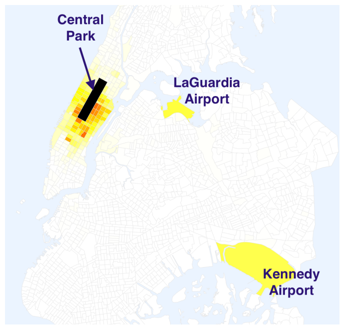
We now show that SMF provides useful and interpretable results on the `NY-Taxi` dataset. A similar (but Manhattan-only) dataset was studied by [9]. Figure 7 shows the results for 3 components, one per row. The first is concentrated around Central Park and the nearby museums, and likely to represent tourism. Its peaks coincide with mealtimes (9am, 1pm, 7pm). The second component peaks at 8-9am on weekdays, and is concentrated on the major railway stations and airports, and likely represents commuting trips, particularly in the morning 'rush-hour.' The third component peaks around Friday 10pm and Saturday midnight. This component is concentrated around southwest Manhattan, an area with a large number of bars and restaurants, suggesting entertainment related trips. In summary, our model finds meaningful seasonal components that give more insight than static approaches.



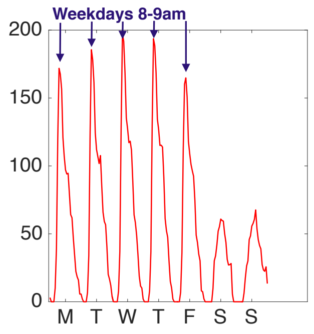
(a) 'Tourism': weekly pattern



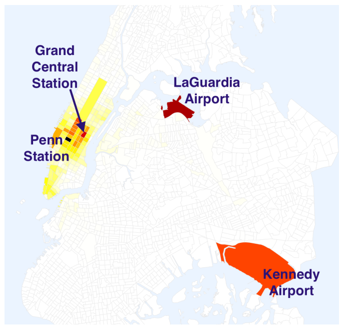
(b) 'Tourism': pickup location



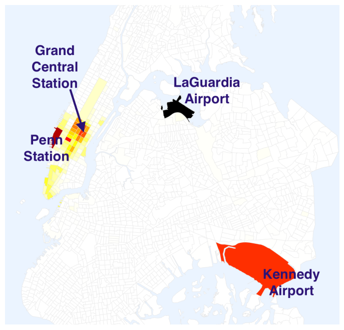
(c) 'Tourism': dropoff location



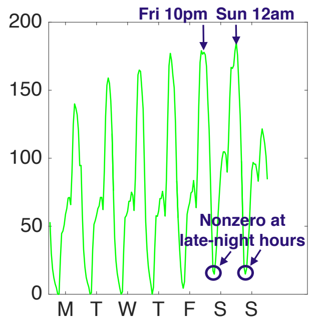
(d) 'Commute': weekly pattern



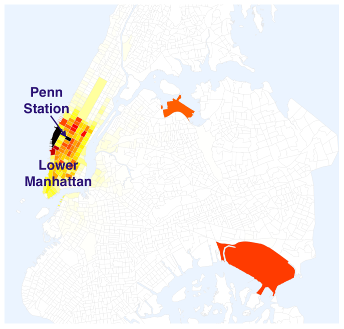
(e) 'Commute': pickup location



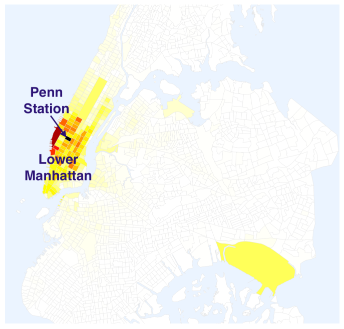
(f) 'Commute': dropoff location



(g) 'Entertainment': weekly pattern



(h) 'Entertainment': pickup location



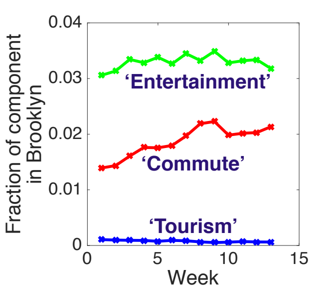
(i) 'Entertainment': dropoff location

Figure 7: (a) **SMF provides interpretable results with seasonal information:** the three components in the `Taxi` dataset correspond roughly to tourism-related trips (near Central Park and museums), morning rush-hour trips (airports and train stations), and entertainment (restaurants and bars).

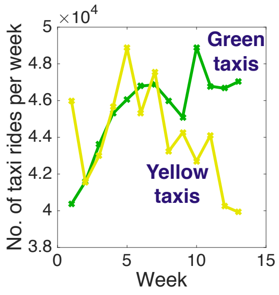
Drifting components An advantage of SMF is that it allows components to drift. We use this to find meaningful patterns in the `NY-Taxi` dataset. In New York City, prior to 2013, most pick-ups by traditional 'yellow' taxis occurred in Manhattan or at airports, resulting in low access to taxis for people living in outer areas ('boroughs') [1]. In 2013, the 'green' taxi program was introduced: these pick up passengers only in outer boroughs, except at airports. Did the green taxi program improve access to taxis in the outer boroughs, and what type of trips were affected?

Figure 8a plots the fraction of each component that lies within Brooklyn (an outer borough). Only the red ('commute') components shows clear movement towards Brooklyn, increasing by 53%. This suggests that more and more commute-related taxi trips are departing from Brooklyn, while no such change occurs for the other two components. This supports the claim of a shift towards outer boroughs, and further reveals what type of taxi trips were most affected: commute-related trips.

Did this change occur because of the green taxi program? We extract the top 20 locations in Brooklyn with the largest values of the 'commute' component. In these locations, Figure 8b plots the fraction of green and



(a) Shift towards Brooklyn



(b) Increase in green taxis

Figure 8: **SMF allows components to change over time:** (a) the 'commute' component (red) shifts toward Brooklyn, increasing by 53% in the fraction of the component in Brooklyn. (b) This can be explained by an increase in green taxis in the 'commute' component.

yellow taxis over time: green taxi rides rose significantly, while the yellow taxi rides decreased slightly. This suggests that green taxis were increasingly adopted, and since green taxis only pick up passengers in outer boroughs, this supports the claim that the rise of green taxis contributed to the shift towards Brooklyn. Thus, allowing the components to change over time reveals useful new information about the dataset.

5.4 Anomaly Detection We now evaluate the anomaly detection accuracy of SMF-A. Starting with the `NY-Taxi` dataset, we inject 100 anomalies of two types: 'add' anomalies represent an increase in activity (e.g. a major festival), while 'scramble' anomalies represent changes in behavior (e.g. a traffic accident redirecting traffic). For the 50 'add' anomalies, we select a random 50 rows and columns, and add 500 taxi trips to this block, distributed according the same power-law distribution as Section 5.2. For the 50 'scramble' anomalies, we select the 200 highest degree rows and columns, and randomly permute the rows and columns, changing the position of entries in this submatrix. We compare SMF-A to DenseAlert [19], a recent anomaly detection algorithm based on dense submatrix detection. The thresholds plotted in Figures 9 and 1c are 3 standard deviations from the mean (in log-space).

Results: Table 4 shows the precision at 100 of both methods on 'add' and 'scramble' anomalies, and Figure 9 shows the output of SMF-A. 100 is the true number of anomalies, so the number of false positives and negatives are equal. SMF-A is much more accurate than DenseAlert, and faster. This is because SMF-A takes into account differences from expected behavior, while DenseAlert only considers density. Moreover, DenseAlert cannot catch 'scramble' anomalies as it detects high activity, while SMF-A can detect any type of deviation from expected behavior.

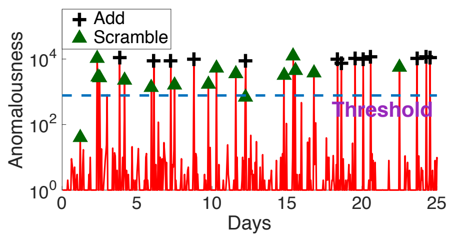


Figure 9: **SMF-A detects multiple types of anomalies:** it catches anomalies which add taxi trips ('add') or permute a subset of locations ('scramble'). We plot only the first 25 days, for visibility.

	Prec. (add)	Prec. (scramble)	Runtime (sec.)
SMF-A	**1.00**	**0.86**	**304.94**
DenseAlert	0.06	0.06	1247.13

Table 4: **SMF-A outperforms baselines in anomaly detection:** precision of SMF-A in catching injected anomalies which add taxi trips ('add') or randomly reorder a subset of locations ('scramble').

Figure 1c shows our results of SMF-A on the `Disease` dataset. Two epidemics stand out, both of which have been reported in the medical literature: a large influenza outbreak in 1928 in Northeast US [4], and a measles epidemic in New York in 1946 [16].

6 Conclusion

We propose SMF, a drift and seasonality aware, online matrix factorization algorithm, and SMF-A, a fast anomaly detection algorithm. In contrast to existing methods (Fold and CPHW), our model uses smoothly varying components $\mathbf{u}$ and $\mathbf{v}$, and seasonally varying multiplers w to model seasonality. Our contributions are as follows:

- **Model:** we propose a novel matrix factorization model incorporating seasonal patterns and drift, and an online algorithm for fitting this model.
- **Effectiveness:** in experiments, SMF has lower forecasting error than baselines by 13% to 60% (Figure 1a), and provides interpretable results in case studies on real data.
- **Scalability:** SMF is online, and scales linearly (Figure 1b). In experiments, it was 12 to 103 times faster than seasonal baselines.
- **Fast Anomaly Detection:** we propose SMF-A for detecting anomalies (Figure 1c) in a computationally feasible way, without forecasting every possible observation in the matrix.

7 Acknowledgment

This material is based upon work supported by the National Science Foundation under Grant No. IIS-1247489, and by the Army Research Laboratory under Cooperative Agreement Number W911NF-09-2-0053, and by NSF of China, under Grant No. 61772498, and the Beijing NSF under Grant No. 4172059. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation, or other funding parties. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

References

[1] Background on the boro taxi program. `http://www.nyc.gov/html/tlc/html/passenger/shl_passenger_background.shtml`, 2013. Accessed: 2017-01-25.

[2] R. M. Bell and Y. Koren. Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In *ICDM*. IEEE, 2007.

[3] F. C. T. Chua, R. J. Oentaryo, and E.-P. Lim. Modeling temporal adoptions using dynamic matrix factorization. In *Data Mining (ICDM), 2013 IEEE 13th International Conference on*, pages 91–100. IEEE, 2013.

[4] S. D. Collins. The influenza epidemic of 1928-1929 with comparative data for 1918-1919. *American Journal of Public Health and the Nations Health*, 20(2):119–129, 1930.

[5] M. R. de Araujo, P. M. P. Ribeiro, and C. Faloutsos. Tensorcast: Forecasting with context using coupled tensors (best paper award). In *Data Mining (ICDM), 2017 IEEE International Conference on*, pages 71–80. IEEE, 2017.

[6] R. Devooght, N. Kourtellis, and A. Mantrach. Dynamic matrix factorization with priors on unknown values. In *KDD*. ACM, 2015.

[7] Y. Ding and X. Li. Time weight collaborative filtering. In *CIKM*. ACM, 2005.

[8] D. M. Dunlavy, T. G. Kolda, and E. Acar. Temporal link prediction using matrix and tensor factorizations. *TKDD*, 5(2):10, 2011.

[9] L. Espín Noboa, F. Lemmerich, P. Singer, and M. Strohmaier. Discovering and characterizing mobility patterns in urban spaces: A study of manhattan taxi data. In *Proceedings of the 25th International Conference Companion on World Wide Web*, pages 537–542. International World Wide Web Conferences Steering Committee, 2016.

[10] T. Hofmann. Latent semantic models for collaborative filtering. *ACM Transactions on Information Systems (TOIS)*, 22(1):89–115, 2004.

[11] R. J. Hyndman, Y. Khandakar, et al. *Automatic time series for forecasting: the forecast package for R*. Number 6/07. Monash University, Department of Econometrics and Business Statistics, 2007.

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

[13] Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 426–434. ACM, 2008.

[14] Y. Koren. Collaborative filtering with temporal dynamics. *Communications of the ACM*, 53(4):89–97, 2010.

[15] D. D. Lee and H. S. Seung. Algorithms for non-negative matrix factorization. In *NIPS*, 2001.

[16] J. E. Perkins, A. M. Bahlke, and H. F. Silverman. Effect of ultra-violet irradiation of classrooms on spread of measles in large rural central schools preliminary report. *American Journal of Public Health and the Nations Health*, 37(5):529–537, 1947.

[17] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Application of dimensionality reduction in recommender system-a case study. Technical report, Minnesota Univ Minneapolis Dept of Computer Science, 2000.

[18] N. Shah, D. Koutra, T. Zou, B. Gallagher, and C. Faloutsos. Timecrunch: Interpretable dynamic graph summarization. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1055–1064. ACM, 2015.

[19] K. Shin, B. Hooi, J. Kim, and C. Faloutsos. Densealert: Incremental dense-subtensor detection in tensor streams. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1057–1066. ACM, 2017.

[20] J. Sun, D. Tao, and C. Faloutsos. Beyond streams and graphs: dynamic tensor analysis. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 374–383. ACM, 2006.

[21] J. Z. Sun, D. Parthasarathy, and K. R. Varshney. Collaborative kalman filtering for dynamic matrix factorization. *IEEE Trans. Signal Processing*, 62(14):3499–3509, 2014.

[22] T. Takahashi, B. Hooi, and C. Faloutsos. Autocyclone: Automatic mining of cyclic online activities with robust tensor factorization. In *WWW*, 2017.

[23] L. Xiong, X. Chen, T.-K. Huang, J. Schneider, and J. G. Carbonell. Temporal collaborative filtering with bayesian probabilistic tensor factorization. In *SDM*. SIAM, 2010.

[24] Y. Xu and W. Yin. A block coordinate descent method for regularized multiconvex optimization with applications to nonnegative tensor factorization and completion. *SIAM Journal on imaging sciences*, 6(3):1758–1789, 2013.