

Recommendation Systems

Part III

Devavrat Shah

Massachusetts Institute of Technology

Recall — Part I

Introduction, simple methods

Module 1: background

Recommendation systems: why and what?

Example datasets: Yelp & MovieLens

Module 2: problem statement

Recommendation systems: a prediction problem

Model: estimating time varying tensor with side information

Module 3: simple solutions

Solution I: averaging

Solution II: content-based

Recall: problem statement

We will start with simple problem statement: complete the matrix

| | | j | | |
|-----|---|-----|--------------|---|
| | | 1 | * | |
| i | 1 | 0 | users | |
| | * | 0 | $L_{ij} = ?$ | |
| * | * | 1 | 1 | 0 |
| * | * | 1 | 0 | 1 |

items / providers

Recall: Solution 1 — Averaging

What if, instead we assume

All items or providers are identical

Then estimate: row average (+ correction for number of observations)

How to put these two simple estimators together?

$$2L_{ij} = L_{i\cdot} + \frac{1}{\sqrt{n_{i\cdot}}} + L_{\cdot j} + \frac{1}{\sqrt{n_{\cdot j}}}$$

where $L_{i\cdot}$ is average of observed entries in row i

$n_{i\cdot}$ is number of observed entries in row i

$L_{\cdot j}$ is average of observed entries in column j

$n_{\cdot j}$ is number of observed entries in column j

Recall: Solution 2 — Content-based Filtering

A little more involved assumption

Users and items have features

that are observed and can predict the likelihood

Let features of user i be x_i

MovieLens: demographics of users

Let features of item j be y_j

MovieLens: attributes of movies

Then, goal is to learn f where $L_{ij} = f(x_i, y_j)$

Outline — Part II

Solution evolves, Matrix estimation

Module 1: clustering

Finding user and item clusters

Averaging *within* clusters

Module 2: collaborative filtering aka personalized clustering

Finding users and items similar to a given user, item

Averaging *amongst* user-item specific *similar* users, items

Module 3: singular value thresholding, optimization

It's a Matrix! find Singular Value Decomposition

Solve for least-squares

Clustering

How to cluster users (or items)?

Step 1. Compute similarity between each pair of N users [How?]

This gives $N \times N$ similarity matrix (that is symmetric)

Step 2. Obtain representation of each user in low-dim space [How?]

This assigns co-ordinates to each user in d dim space

Step 3. Perform k-means clusterings [How?]

Iteratively find k clusters till they make sense

Iterative collaborative filtering

Collaborative filtering

Use average over *similar* users (or items)

a user is *similar* when shared observations are *close enough*

A common challenge: not enough users with shared observations

Solution: friend of your friend is (almost) your friend

Find users *similar* to user of interest

Next find users *similar* to these users

And continue *iterating* this procedure to find more *similar* users

till *enough similar* users are found

Use the experiences of such users to obtain the estimate

Optimization perspective

Singular value decomposition: optimization perspective

Let U, V be solution of

$$\begin{aligned} & \text{minimize}_{i,j} \sum_{k=1}^r (X_{ij} - \sum_{k=1}^r u_{ik} v_{jk})^2 \\ & \text{over } u_{ik} \in \mathbb{R}, 1 \leq i \leq N, 1 \leq k \leq r \\ & \text{over } v_{jk} \in \mathbb{R}, 1 \leq j \leq M, 1 \leq k \leq r. \end{aligned}$$

Then, U and V are (effectively) left/right singular vectors

And UV^T provides the best rank r approximate of X

Optimization perspective

This suggests the following algorithm

Find solution U, V of optimization problem

$$\begin{aligned} \text{minimize}_{\substack{(i,j):\text{obs}}} \quad & \sum_{k=1}^r (L_{ij} - \sum_{k=1}^r u_{ik}v_{jk})^2 \\ \text{over } u_{ik} & \in \mathbb{R}, \quad 1 \leq i \leq N, \quad 1 \leq k \leq r \\ \text{over } v_{jk} & \in \mathbb{R}, \quad 1 \leq j \leq M, \quad 1 \leq k \leq r. \end{aligned}$$

For any i, j , produce estimate

$$\hat{L}_{ij} = \sum_{k=1}^r u_{ik}v_{jk}$$

Optimization perspective

The algorithm uses *only* observed entries

and does not require filling missing values as in for SVD

The optimization problem can be solved

via iteratively solving for U and V

also known as *Alternating Least Squares*

Food for thought:

Will it converge?

Optimization perspective

Exercise: Singular Value Thresholding meets Alternative Least Squares

Initialize by filling missing values with 0

Singular Value Thresholding to obtain an estimate.

Use outcome to fill missing values and then perform Alt. Least. Sq.

Iterate.

What are the advantages?

Use MovieLens and/or Yelp data to answer.

Matrix Estimation: Generic Model

Prediction problem: complete the matrix

Latent features

v_j

Latent features

u_i

$$L_{ij} = f(u_i, v_j)$$

$$\text{low-rank: } L_{ij} = u_i^T v_j$$

user i

item j

Problem reduces to learning “factorization” of the matrix

either through similarities or algebraic approaches

Appendix: Matrix Estimation with Neural Networks

Singular Value Thresholding

bi-linear function of latent features

“Generalized” Singular Value Thresholding

generic “activation” function of latent features

multiple layers

this provides neural network implementation

Exercise: compare performance with other methods

Outline — Part III

Getting things together, some more and connections

Module 1: Matrix estimation meets content-based

Model

Estimation algorithm

Module 2: Matrix estimation across time

Model

Estimation algorithm

Module 3: Everything together

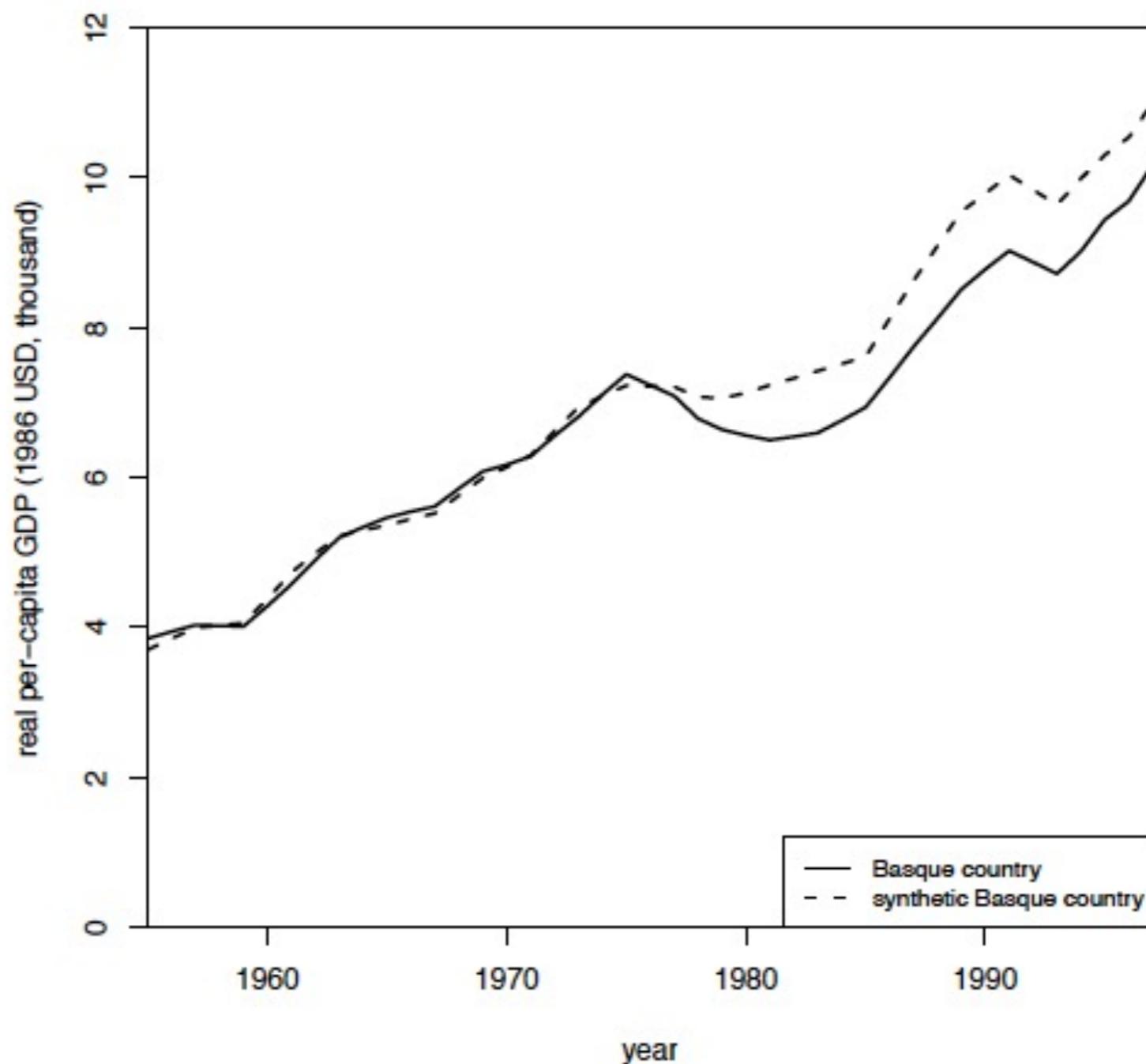
Model

Estimation algorithm

Some more applications

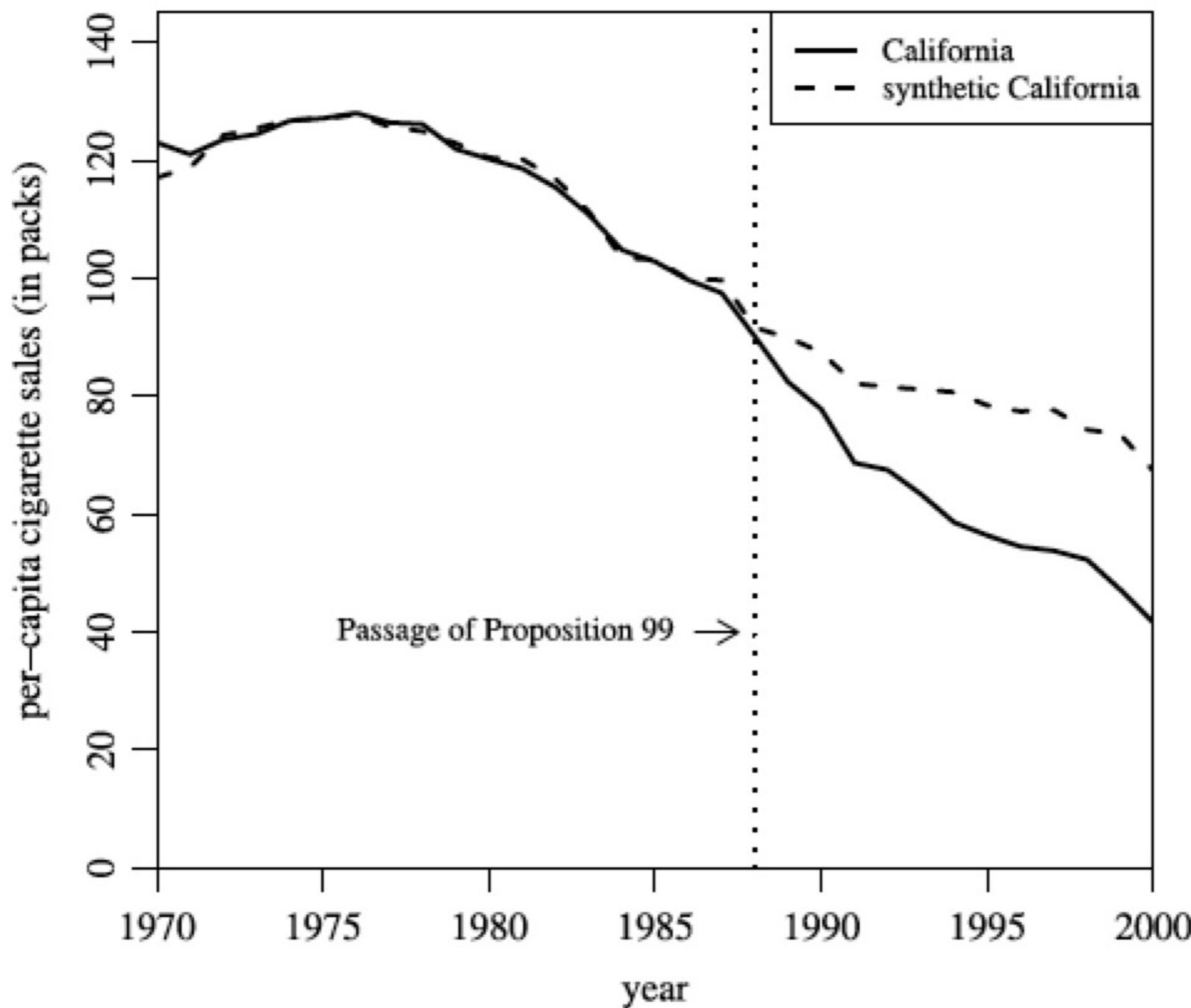
A Question

Did Terrorism have impact on Economy (of Basque Country)?

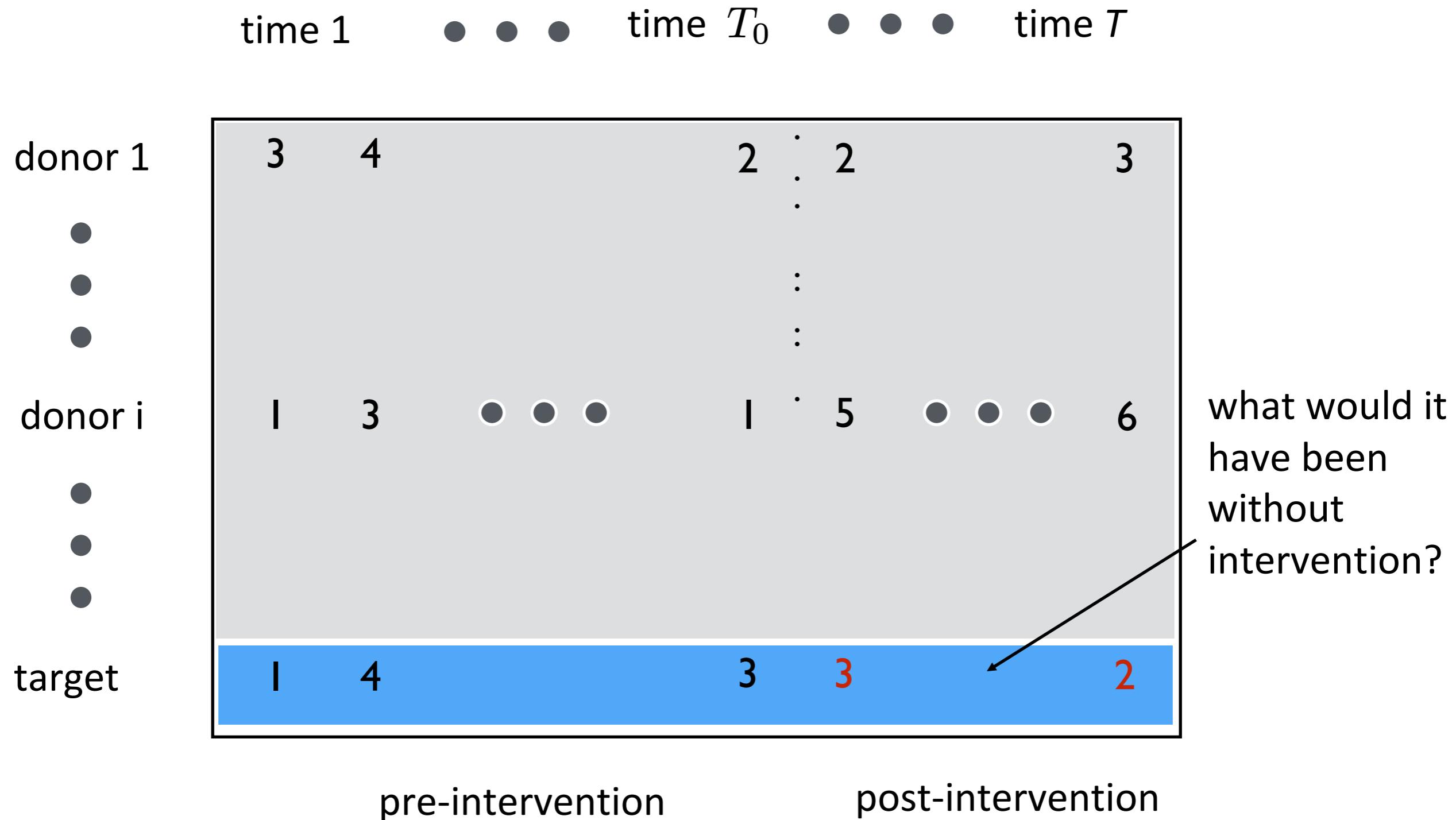


A Question

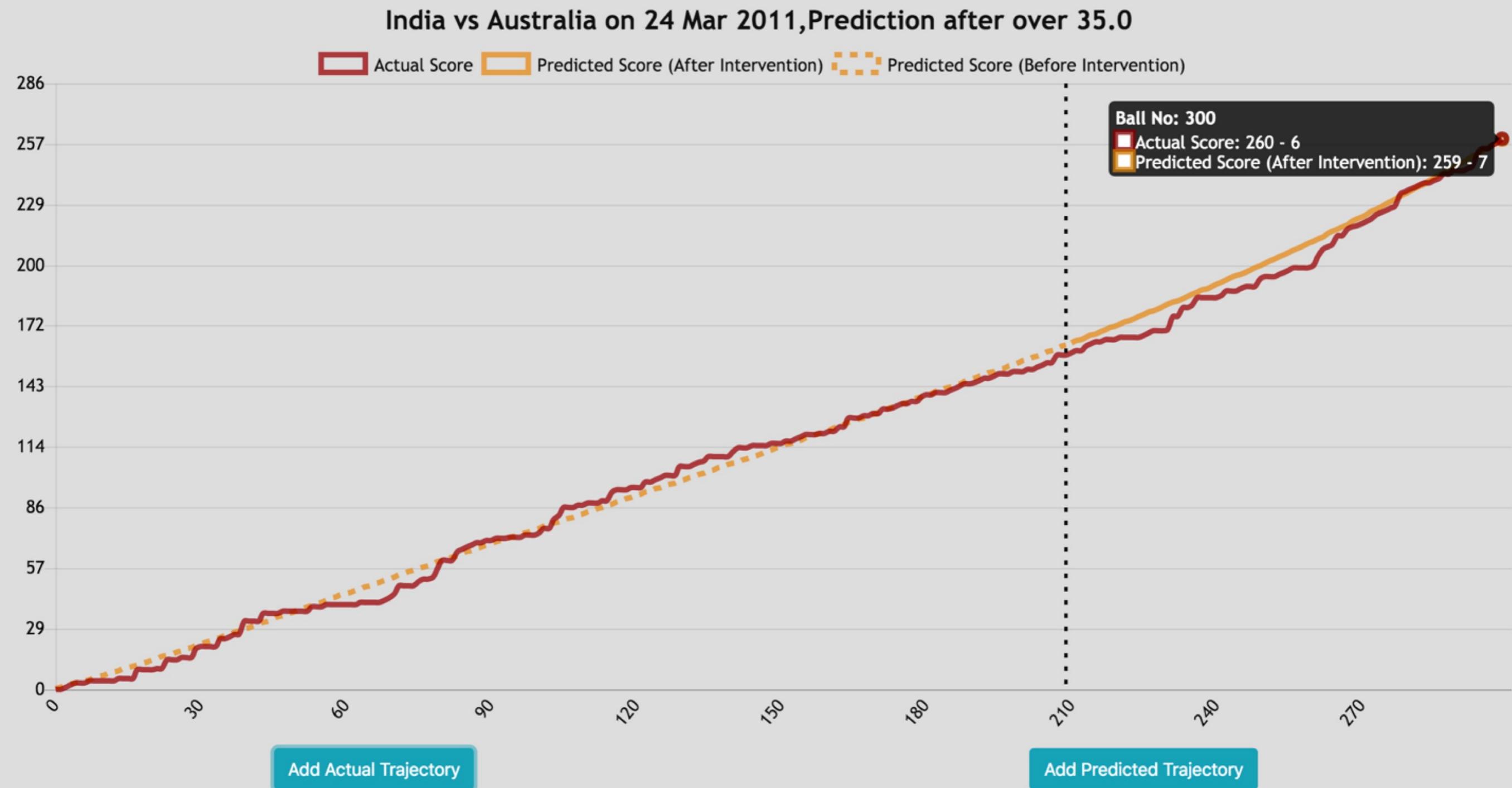
Did California tobacco control program (Prop 99) work?



How? Synthetic controls aka Matrix Estimation (!!)



Another Example: Forecasting Cricket Trajectory

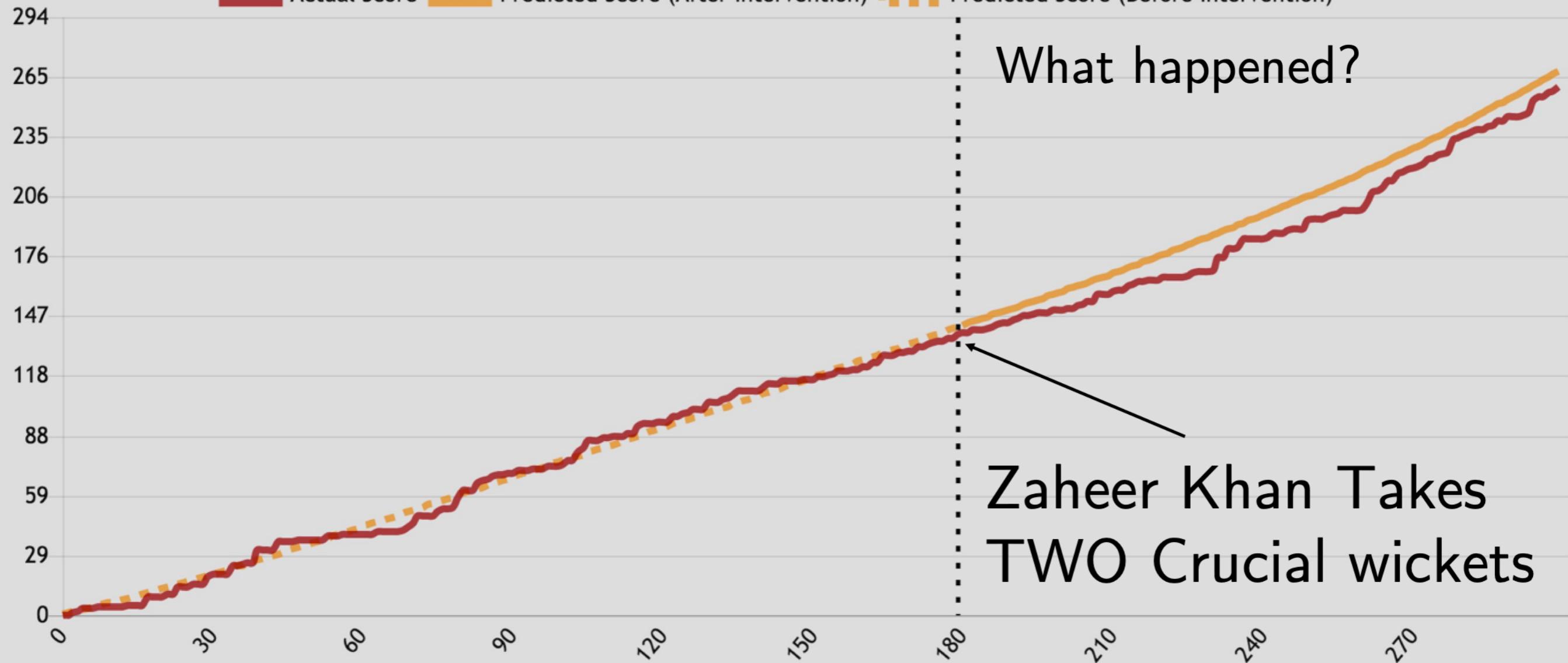


- to further the applications of sophisticated Data/Sports Analytics
- to contribute to academic literature on Sports Analytics
- to propose novel and practical solutions to (real) problems in the sport

Another Example: Forecasting Cricket Trajectory

India vs Australia on 24 Mar 2011, Prediction after over 30.0

Actual Score Predicted Score (After Intervention) Predicted Score (Before Intervention)



Add Actual Trajectory

Add Predicted Trajectory

Module 1: matrix est & content-based

Recommendation: Problem statement

Prediction problem: complete the matrix

| | | j | | |
|-----|---|-----|--------------|---|
| | | 1 | * | |
| i | 1 | 0 | users | |
| | * | 0 | $L_{ij} = ?$ | |
| * | * | * | 1 | 1 |
| * | | | 0 | 0 |

items / providers

Content-based: Model

Prediction problem: complete the matrix

Observed features

y_j

Observed features

x_i

$$L_{ij} = f(x_i, y_j)$$

e.g. $L_{ij} = \alpha x_i + \beta y_j + \gamma$

or $L_{ij} = \frac{\exp(\alpha x_i + \beta y_j + \gamma)}{1 + \exp(\alpha x_i + \beta y_j + \gamma)}$

user i

item j

Problem reduces to learning model f

That is, traditional supervised learning (regression or classification)

Matrix Estimation: Model

Prediction problem: complete the matrix

Latent features

v_j

Latent features

u_i

$$L_{ij} = f(u_i, v_j)$$

$$\text{low-rank: } L_{ij} = u_i^T v_j$$

user i

item j

Problem reduces to learning “factorization” of the matrix

either through similarities or algebraic approaches

Matrix Estimation meets Content-based: Model

Prediction problem: complete the matrix

item features

latent: v_j , obs'ed: y_j

user features

latent: u_i
obs'ed: x_i

$$L_{ij} = f_{\text{obs}}(x_i, y_j) + f_{\text{latent}}(u_i, v_j)$$

$$L_{ij} = \alpha x_i + \beta y_j + u_i^T v_j + \gamma$$

user i

item j

Problem reduces to learning model f

That is, traditional supervised learning (regression or classification)

Matrix Estimation meets Content-based: Algorithm

Prediction problem: complete the matrix

Step 1. Content-based supervised learning

Learn the regressor (or classifier) f_{obs}

$$L_{ij}^{\text{obs}} = f_{\text{obs}}(x_i, y_j)$$

Step 2. Matrix estimation

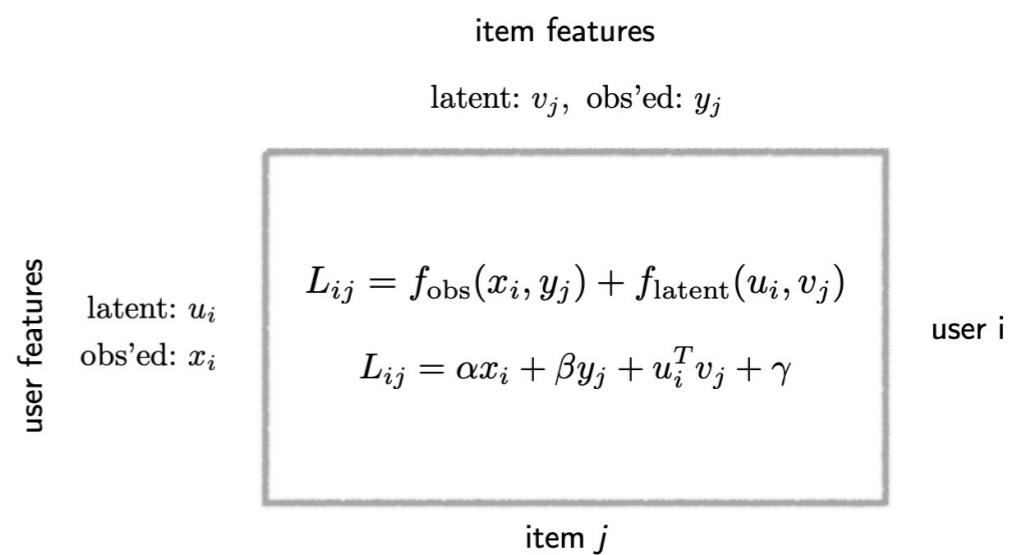
Compute “difference” matrix

$$L_{ij}^{\text{diff}} = L_{ij} - L_{ij}^{\text{obs}}, \text{ over obs. entries}$$

Use matrix estimation on L^{diff} to produce L^{ME}

Step 3. Combine estimates:

$$\hat{L}_{ij} = L_{ij}^{\text{obs}} + L_{ij}^{\text{ME}}$$



Module 2: Matrix Estimation Across Time

Matrix Estimation over Time: Model

Prediction problem: complete the matrix over time, indexed by t

Latent features

$$v_j(t)$$

Latent features

$$u_i(t)$$

$$L_{ij}(t) = f(u_i(t), v_j(t))$$

$$\text{low-rank: } L_{ij}(t) = u_i^T(t)v_j(t)$$

$u_i(\cdot), v_j(\cdot)$ time-series

user i

item j

Problem reduces estimating time varying matrix where

Latent factors are time varying, observations are partial and noisy

Matrix Estimation over Time: Model

Prediction problem: complete the matrix over time, indexed by t

Multiple time series:

$$L_{ij}(t), \quad i \in [N], \quad j \in [M]$$

They obey latent structure:

$$L_{ij}(t) = u_i(t)^T v_j(t), \quad u_i(t), \quad v_j(t) \in \mathbb{R}^d$$

where each component of u_i , v_j is a structured time-series

Observations:

Partial, noisy observations of the $L_{ij}(t), \quad i \in [N], \quad j \in [M]$

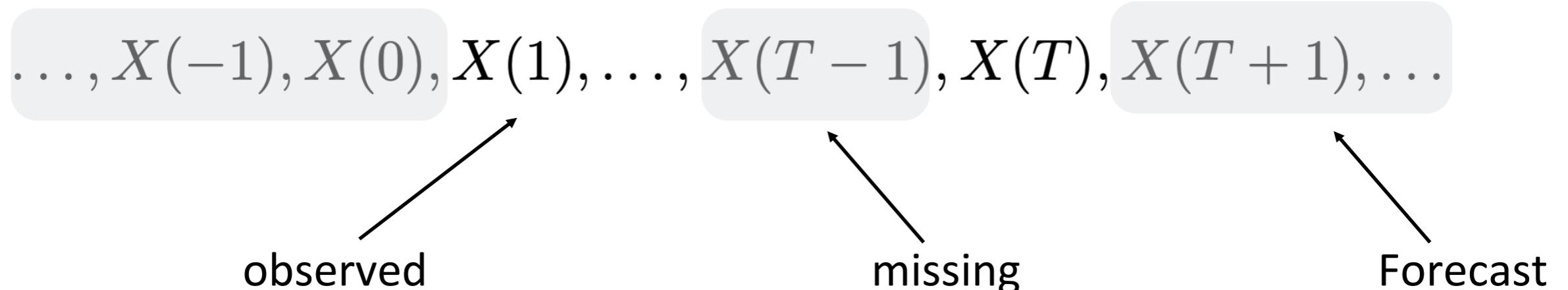
NOT $u_i(t), \quad v_j(t) \in \mathbb{R}^d$

A digression: time series imputation, forecasting

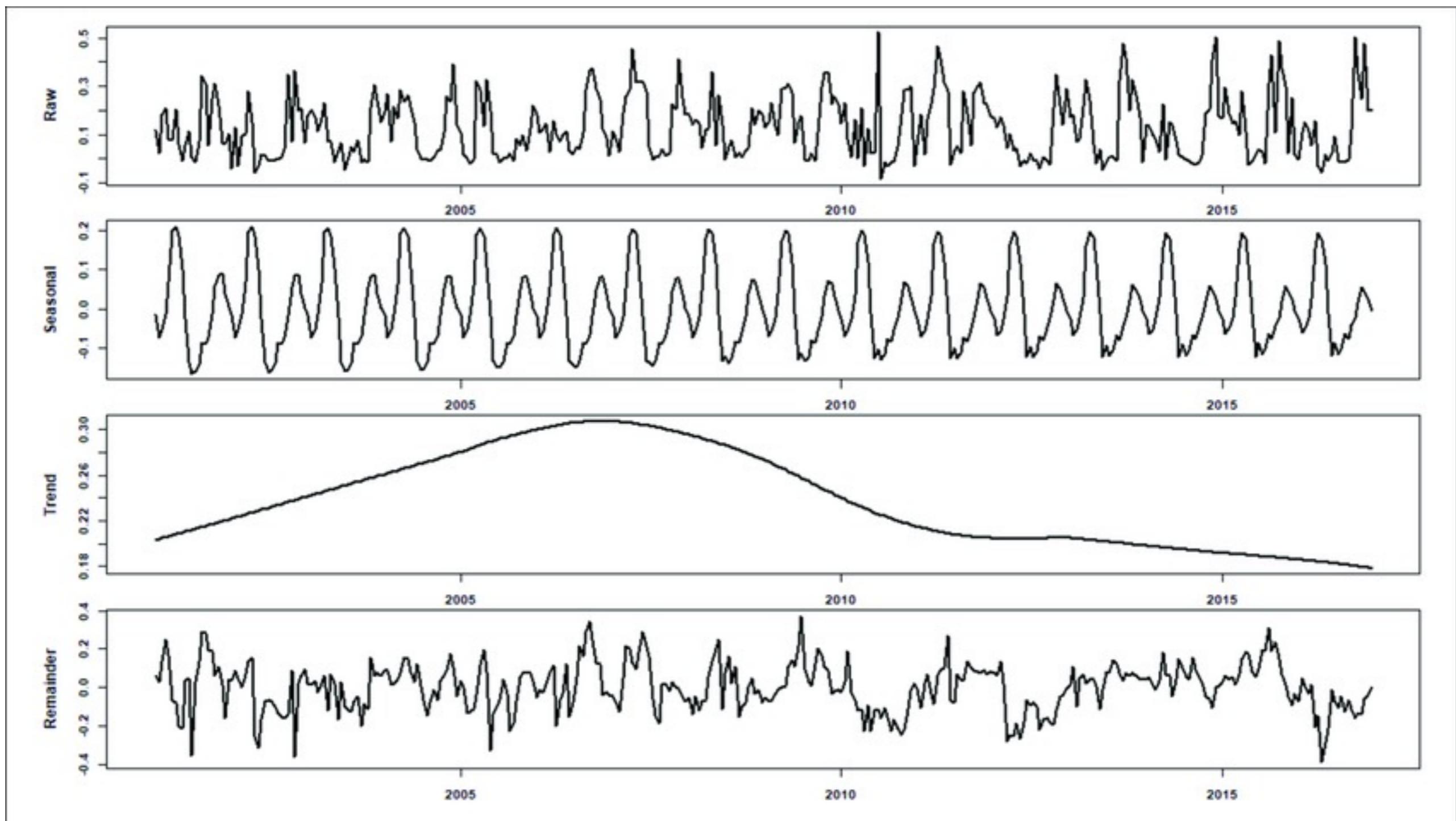
Ground truth of interest:

$$\dots, X(-1), X(0), X(1), \dots, X(T-1), X(T), X(T+1), \dots$$

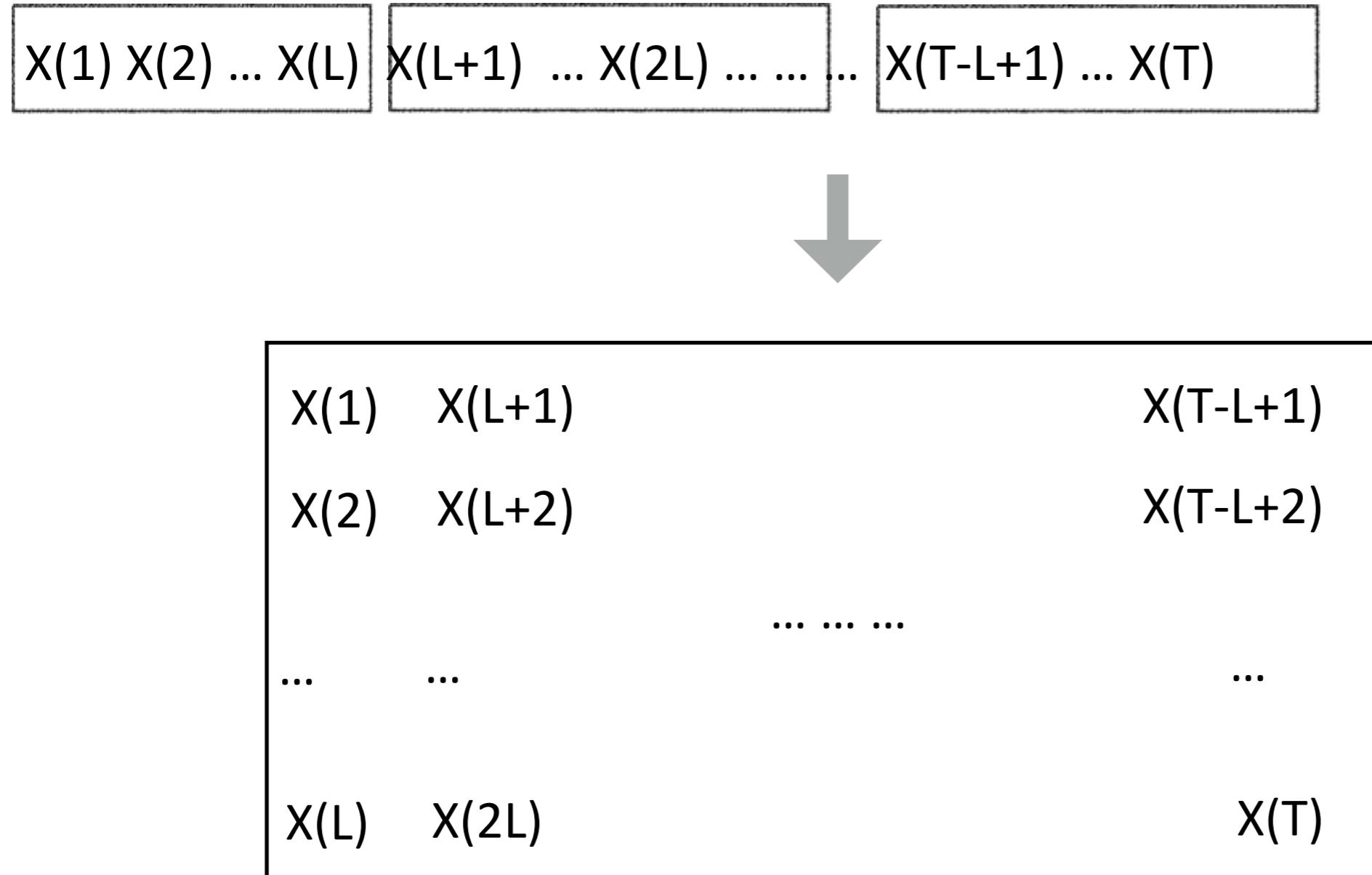
Observations:



Characteristics of time series data



Page Matrix



$$P = [P_{ij} = X(i + (j - 1)L)] \in \mathbb{R}^{L \times \frac{T}{L}}$$

Imputation of time series data

Transform to Matrix, **Do Matrix Estimation**, Undo Transformation

| | | |
|------------------------|----------------------------------|-----------------------|
| $X(1) X(2) \dots X(L)$ | $X(L+1) \dots X(2L) \dots \dots$ | $X(T-L+1) \dots X(T)$ |
|------------------------|----------------------------------|-----------------------|

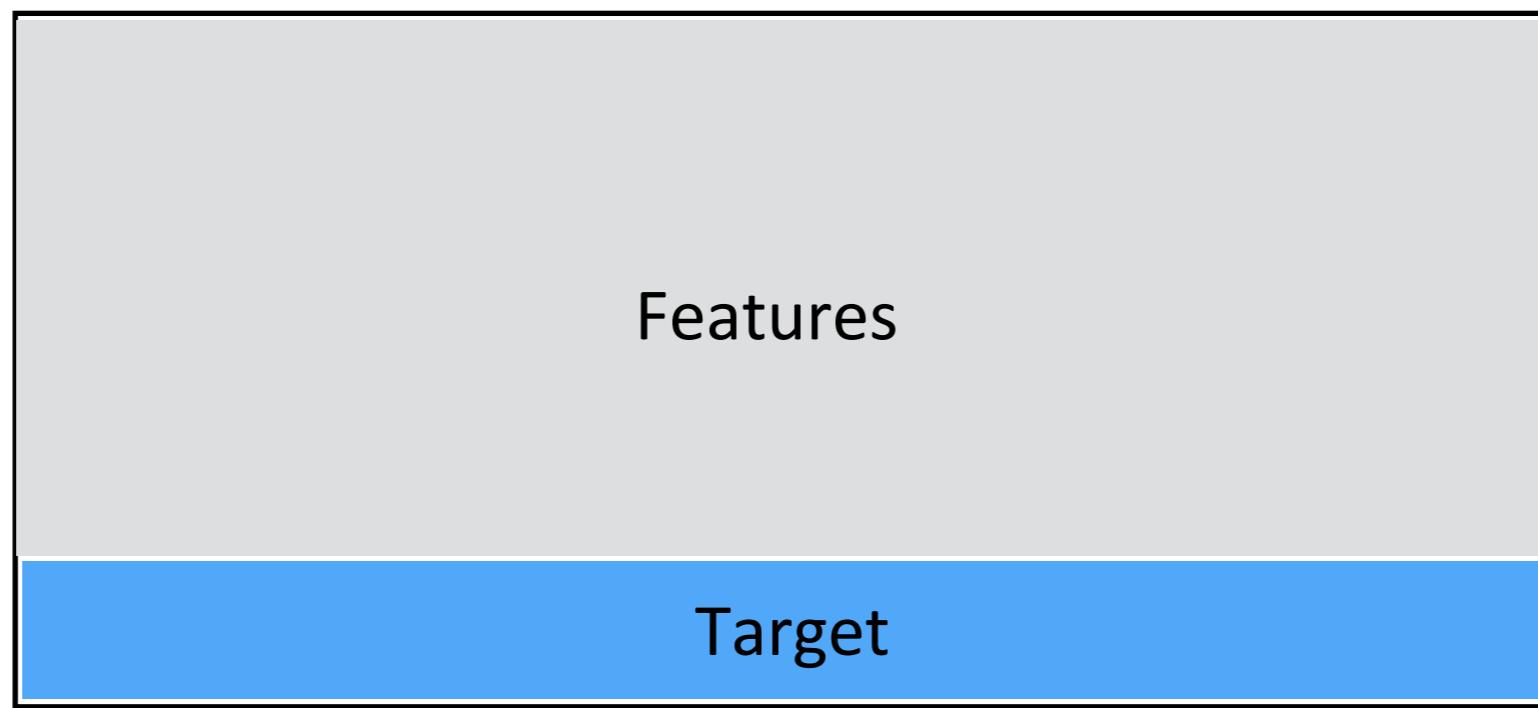


| | | |
|--------|----------|------------|
| $X(1)$ | $X(L+1)$ | $X(T-L+1)$ |
| $X(2)$ | $X(L+2)$ | $X(T-L+2)$ |
| ... | ... | ... |
| $X(L)$ | $X(2L)$ | $X(T)$ |

That's it!

Forecasting of time series data

Transform to Matrix, Do Matrix Estimation, Regression, Prediction

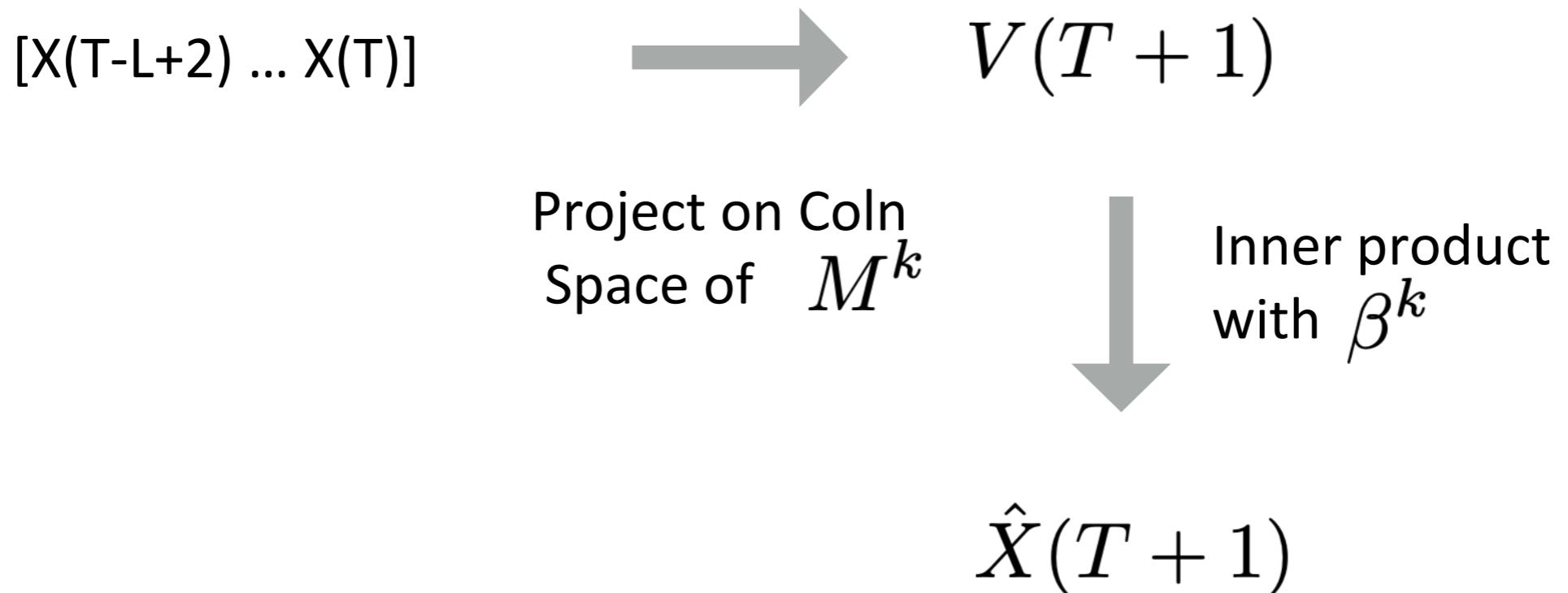


Linear
Regression

$$\beta^k$$

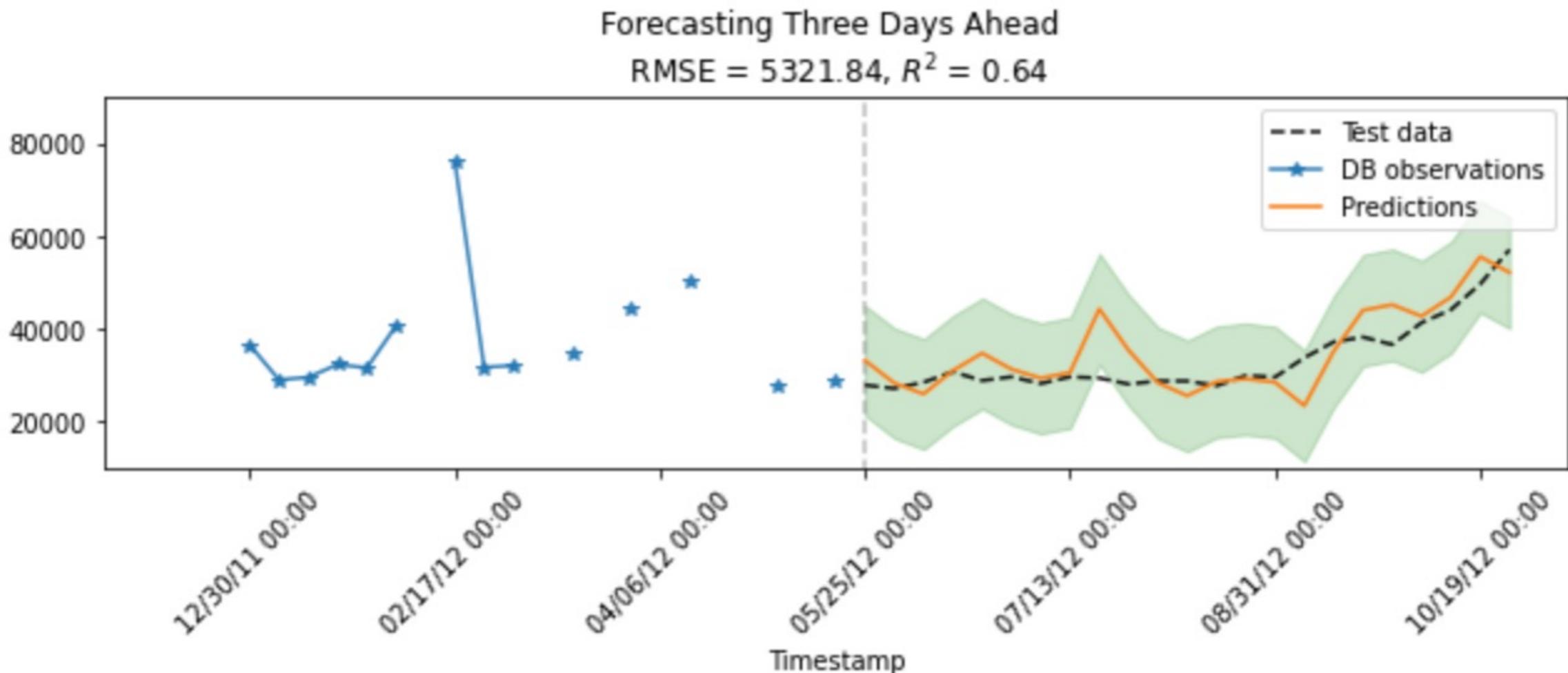
Forecasting

Transform to Matrix, Do Matrix Estimation, Regression, Prediction



where $k = (T + 1 \bmod L) + 1$

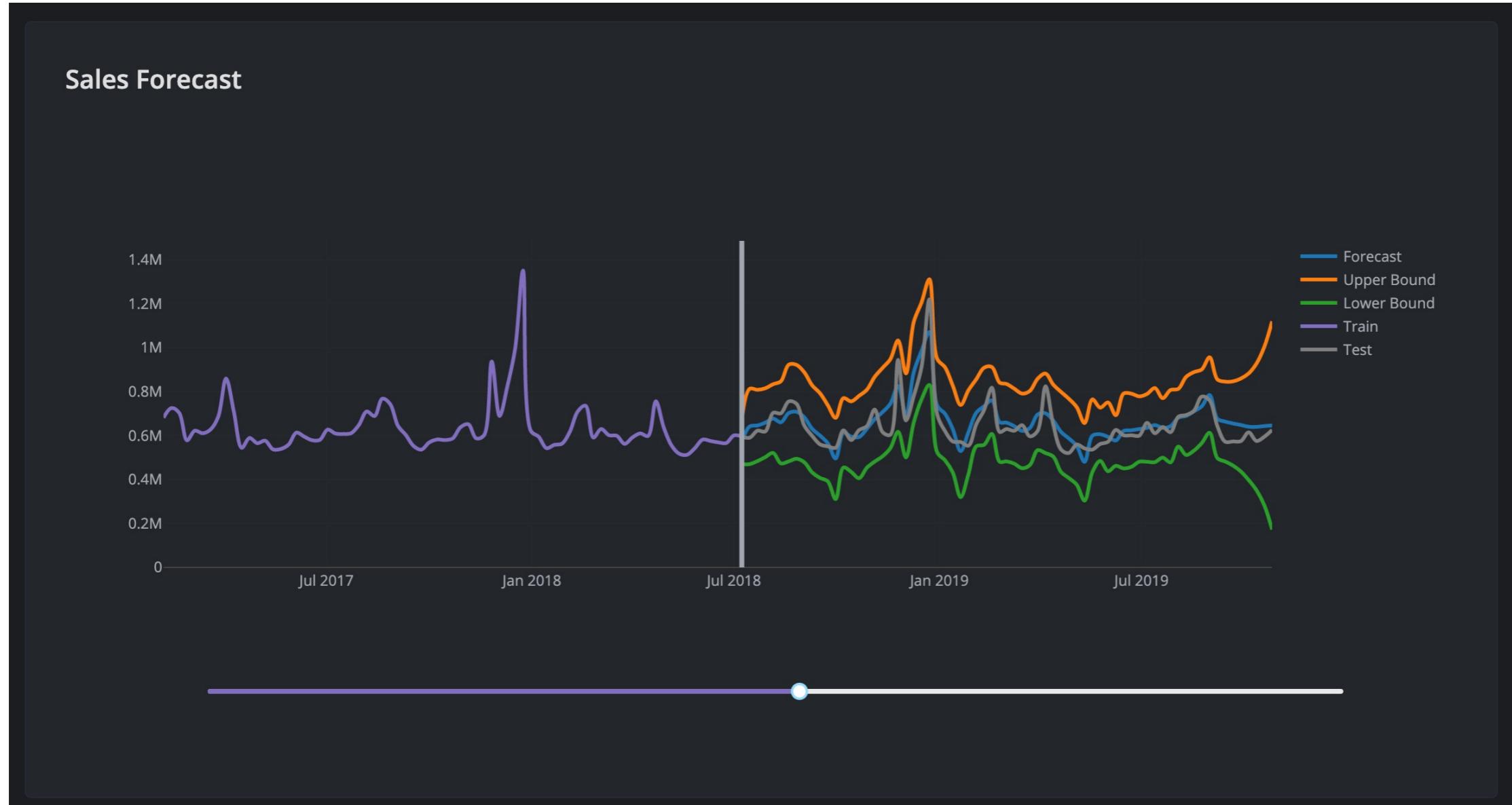
Does it really work?



Data URL:

<https://www.kaggle.com/c/walmart-recruiting-store-sales-forecasting>

Does it really work?

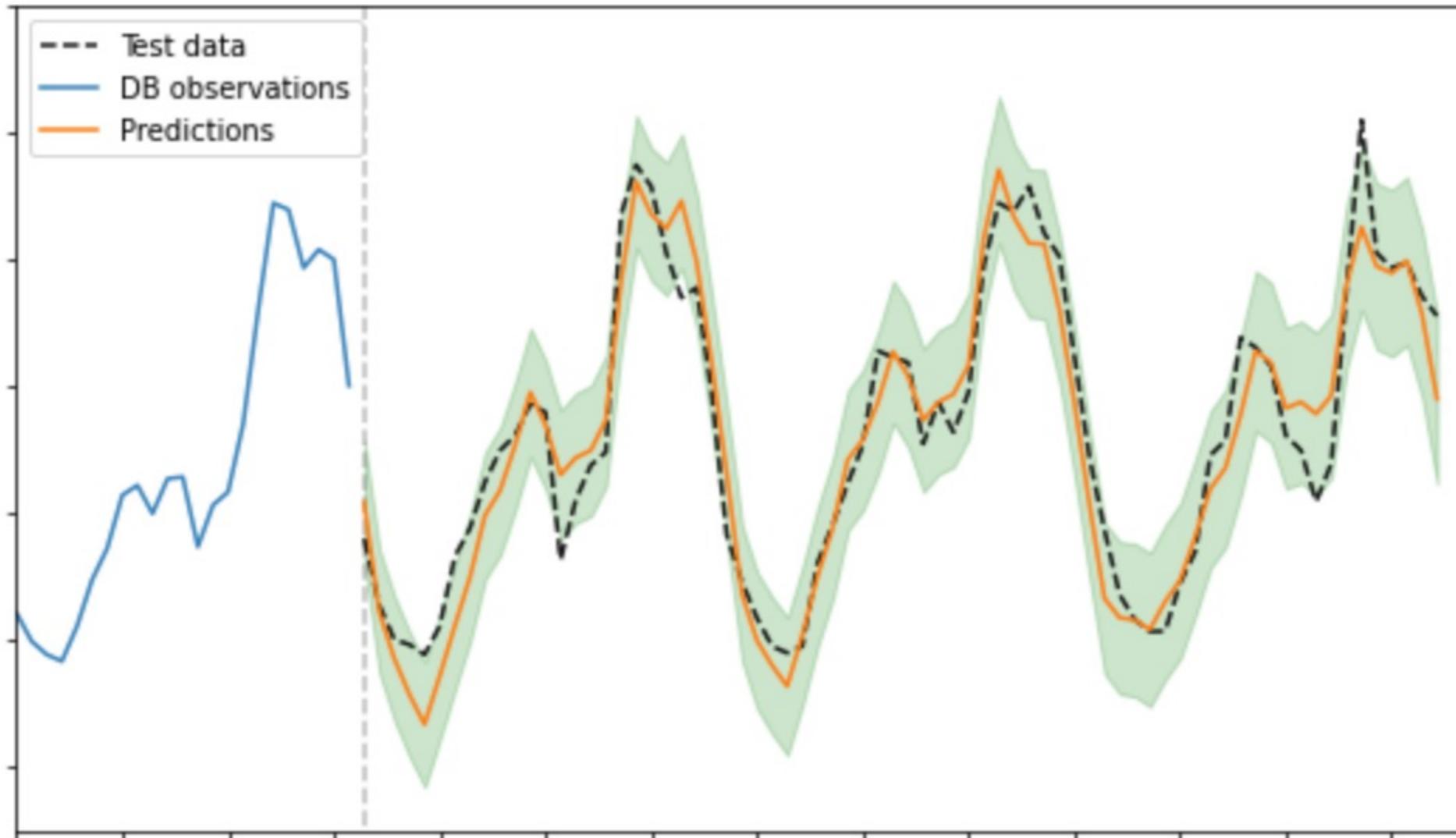


Data URL:

<https://www.kaggle.com/c/walmart-recruiting-store-sales-forecasting>

Does it really work?

Forecasting electricity consumption

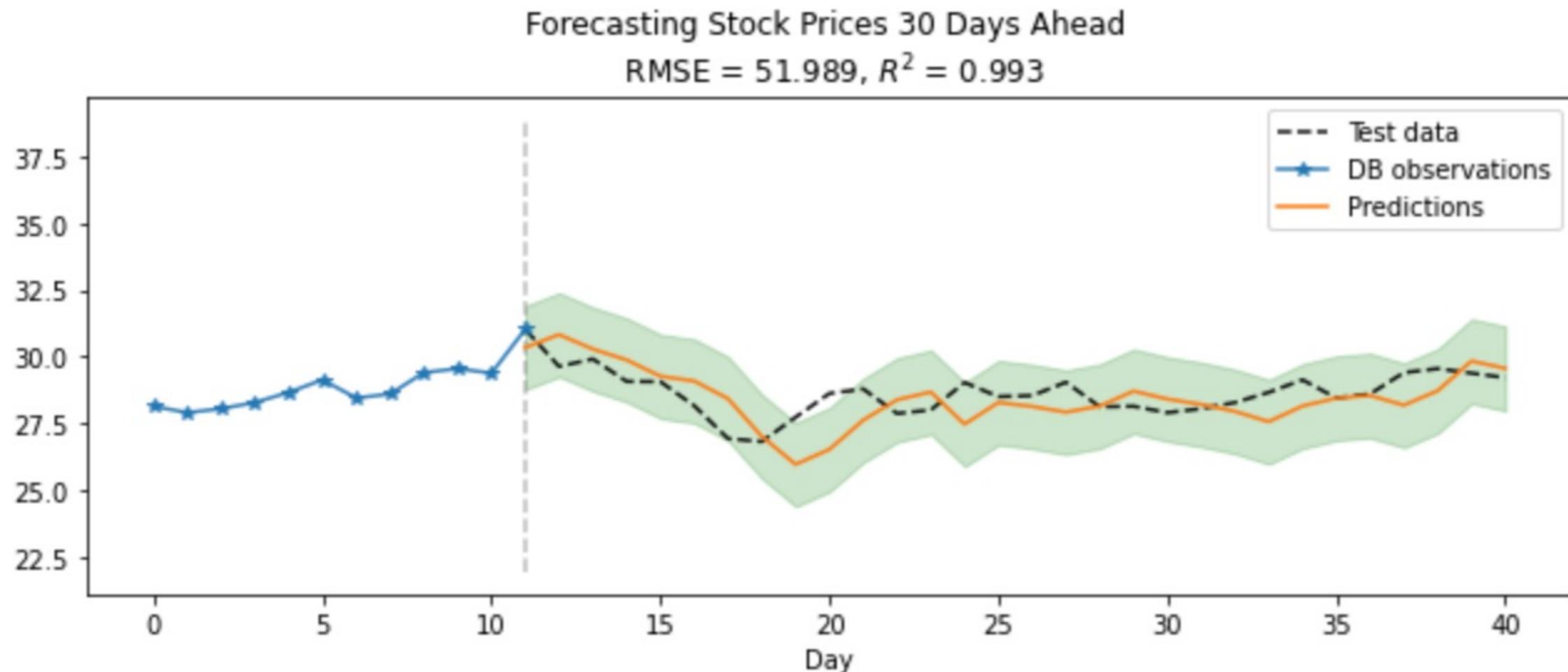


For more, see:

tspdb.mit.edu/#demo

Does it really work?

Financial data



For more, see:

tspdb.mit.edu/#demo

Does it really work?

| | Mean Imputation (NRMSE) | | | | Mean Forecasting (NRMSE) | | | |
|---------|----------------------------|--------------|--------------|--------------|-----------------------------|--------------|--------------|--------------|
| | Electricity | Traffic | Synthetic | Financial | Electricity | Traffic | Synthetic | Financial |
| mSSA | 0.391 | 0.494 | 0.253 | 0.283 | 0.483 | 0.525 | 0.196 | 0.358 |
| SSA | 0.519 | 0.608 | 0.626 | 0.466 | 0.552 | 0.704 | 0.522 | 0.592 |
| LSTM | NA | NA | NA | NA | 0.551 | 0.473 | 0.444 | 1.203 |
| DeepAR | NA | NA | NA | NA | 0.484 | 0.474 | 0.331 | 0.395 |
| TRMF | 0.694 | 0.512 | 0.325 | 0.513 | 0.534 | 0.570 | 0.267 | 0.464 |
| Prophet | NA | NA | NA | NA | 0.582 | 0.617 | 1.005 | 1.296 |

Out-performs:

DeepAR from Amazon

Prophet from Facebook

LSTM, the *shiny* neural network approach

Easy to use implementation:

tspdb.mit.edu

NeurIPS 2020 demo:

<https://www.powtoon.com/s/fkVh3axA4Jy/1/m>

Back to Matrices Changing with Time

Step 1. Convert time series for each component into a (Page) matrix

$$\boxed{L_{ij}(1), \dots, L_{ij}(P)} \quad \boxed{L_{ij}(P+1), \dots, L_{ij}(2P)} \quad \dots \dots \dots$$



$$\begin{matrix} L_{ij}(1) & L_{ij}(P+1) \\ \vdots & \vdots & \dots \dots \dots \\ L_{ij}(P) & L_{ij}(2P) \end{matrix}$$

Matrix Estimation over Time: Algorithm

Step 2. Concatenate matrices of all components along columns
observations over total of T time units

| | |
|-------------|---------------|
| $L_{11}(1)$ | $L_{11}(P+1)$ |
| : | : |
| $L_{11}(P)$ | $L_{11}(2P)$ |

0 0 0

| | |
|-------------|---------------|
| $L_{ij}(1)$ | $L_{ij}(P+1)$ |
| : | : |
| $L_{ij}(P)$ | $L_{ij}(2P)$ |

This combined matrix, say Z, has

P rows

$T/P \times N \times M$ columns

Matrix Estimation over Time: Algorithm

Step 3. Perform matrix estimation over Z

$L_{11}(1) \ L_{11}(P+1)$

$\vdots \ \vdots$

$L_{11}(P) \ L_{11}(2P)$

0 0 0

$L_{ij}(1) \ L_{ij}(P+1)$

$\vdots \ \vdots$

$L_{ij}(P) \ L_{ij}(2P)$

Let estimated matrix from Z is \hat{Z}

By reversing the map we can obtain

$$\hat{L}_{ij}(t), t \in [T], i \in [N], j \in [M]$$

Module 3: Everything together

Recommendation: Problem statement

Prediction problem: complete the matrix

| | | j | | |
|-----|---|-----|--------------|---|
| | | 1 | * | |
| i | 1 | 0 | users | |
| | * | 0 | $L_{ij} = ?$ | |
| * | * | * | 1 | 1 |
| * | | | 0 | 0 |

items / providers

Recommendation: Model 1

Content-based

Observed features

y_j

Observed features

x_i

$$L_{ij} = f(x_i, y_j)$$

e.g. $L_{ij} = \alpha x_i + \beta y_j + \gamma$

or $L_{ij} = \frac{\exp(\alpha x_i + \beta y_j + \gamma)}{1 + \exp(\alpha x_i + \beta y_j + \gamma)}$

user i

item j

Problem reduces to learning model f

That is, traditional supervised learning (regression or classification)

Recommendation: Model 2

Matrix estimation

Latent features

v_j

Latent features

u_i

$$L_{ij} = f(u_i, v_j)$$

$$\text{low-rank: } L_{ij} = u_i^T v_j$$

user i

item j

Problem reduces to learning “factorization” of the matrix

either through similarities or algebraic approaches

Recommendation: Model 3

Matrix estimation over time

Latent features

$v_j(t)$

Latent features

$u_i(t)$

$$L_{ij}(t) = f(u_i(t), v_j(t))$$

$$\text{low-rank: } L_{ij}(t) = u_i^T(t)v_j(t)$$

$u_i(\cdot), v_j(\cdot)$ time-series

user i

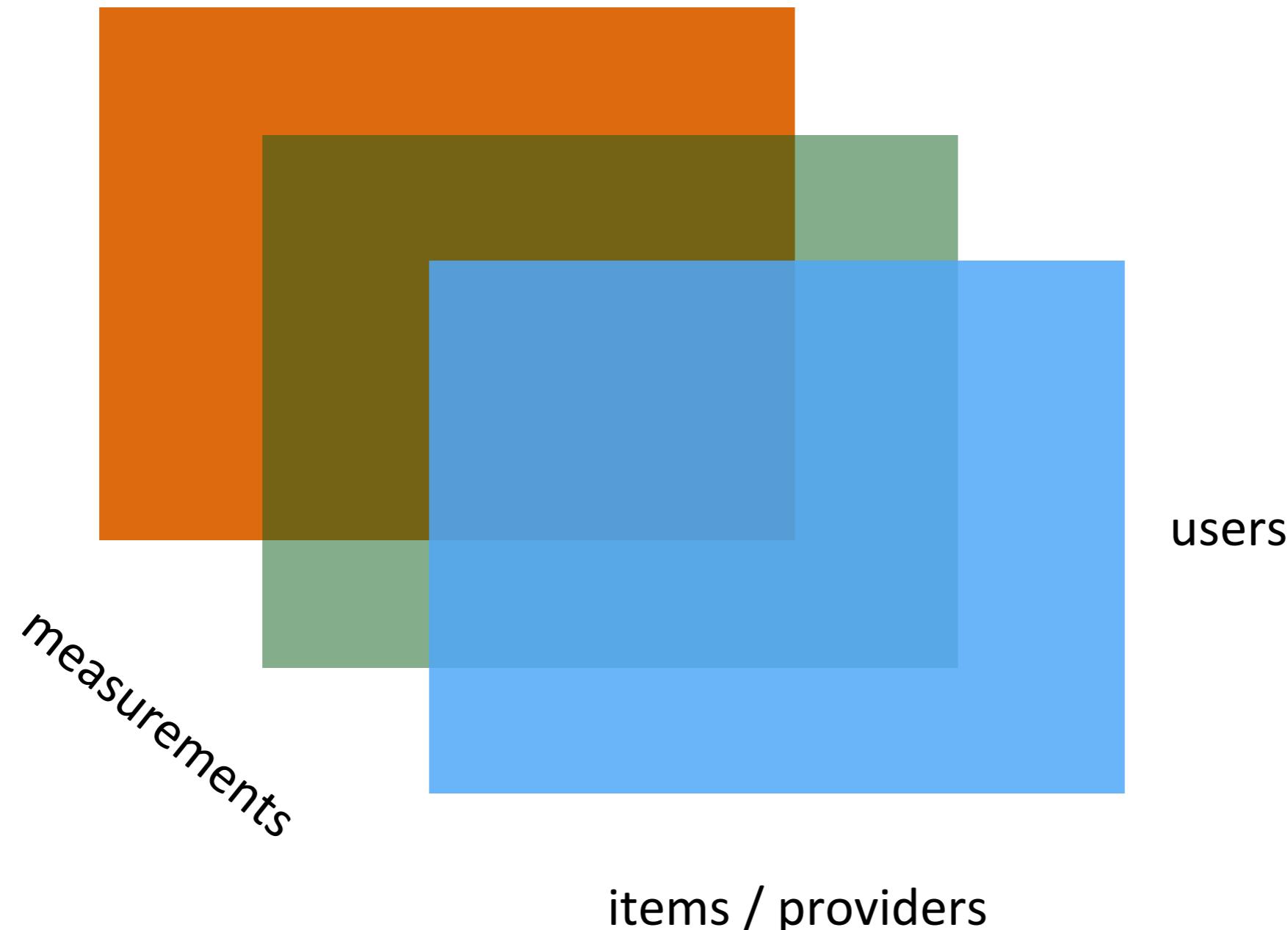
item j

Problem reduces estimating time varying matrix where

Latent factors are time varying, observations are partial and noisy

Recommendation: Model 4

Multiple measurements



Recommendation: Model Everything

Put everything together

Users: N, Items: M

Time horizon: T

Measurements: K

Quantity of interest: user i, item j, measurement k at time t

$$L_{ijk}(t)$$

Problem statement:

Estimate the above using noisy, sparse observations

Recommendation: Model Everything

The model

$$L_{ijk}(t) = f_{\text{obs}}^k(x_i, y_j) + f_{\text{latent}}^k(u_i(t), v_j(t))$$

$u_i(\cdot)$, $v_j(\cdot)$ time-series

A useful special instance

$$f_{\text{obs}}^k(x_i, y_j) = \alpha^k x_i + \beta^k y_j + \gamma^k$$

$$f_{\text{latent}}^k(u_i(t), v_j(t)) = \sum_{\ell=1}^d u_{i\ell}(t) v_{j\ell}(t) w_{k\ell}$$

Recommendation: Algorithm

Step 1. Content based learning

For each measurement k, learn via supervised learning

f_{obs}^k , that is $(\alpha^k, \beta^k, \gamma^k)$

Step 2. Obtain difference (not learnt through content)

$$L_{ijk}^{\text{diff}}(t) = L_{ijk}(t) - L_{ijk}^{\text{obs}}$$

where $L_{ijk}^{\text{obs}} = f_{\text{obs}}^k(x_i, y_j)$

Recommendation: Algorithm

Step 3. Build stacked Page matrice across entries, slices of tensor

That is, for measurement k, create Page matrix with
P rows, T/P x N x M columns

$$L_{ijk}(1) \ L_{ijk}(P+1)$$

:

:

...

$$L_{ijk}(P) \ L_{ijk}(2P)$$

Recommendation: Algorithm

Step 3. Build stacked Page matrice across entries, slices of tensor

That is, for measurement k, create Page matrix with

P rows, T/P x N x M columns

Call it Z^k

$$\begin{matrix} & L_{ijk}(1) & L_{ijk}(P+1) \\ \text{o o o} & \vdots & \vdots & \dots & \dots & \dots \\ & L_{ijk}(P) & L_{ijk}(2P) \\ & \text{o o o} & & & & \end{matrix}$$

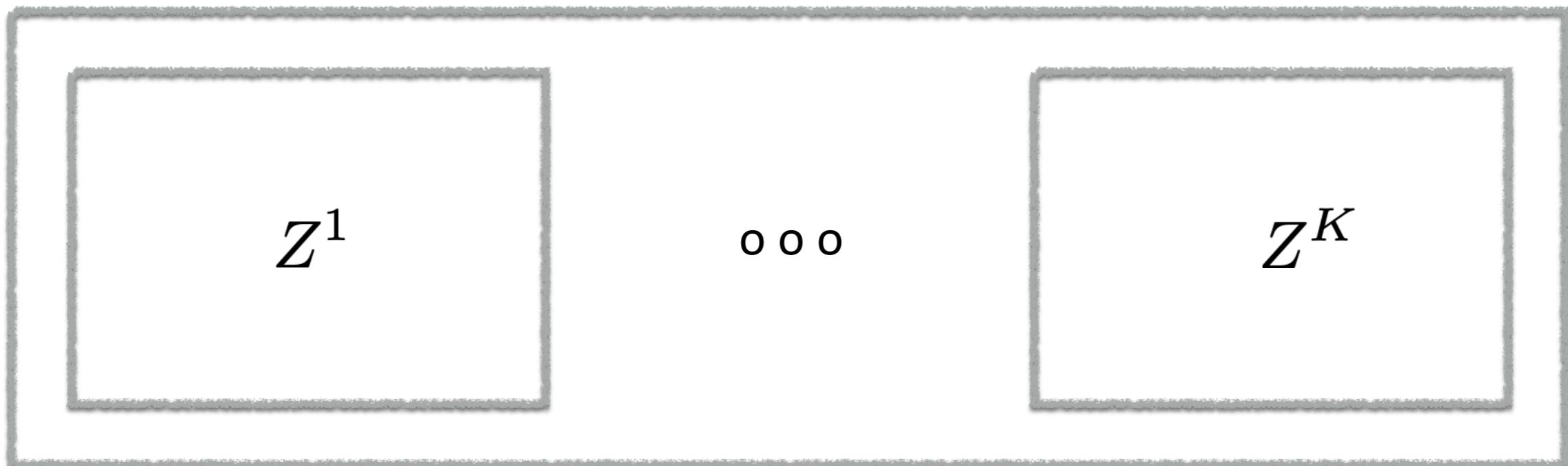
Recommendation: Algorithm

Step 3. Build stacked Page matrice across entries, slices of tensor

That is, for measurement k, create Page matrix with

P rows, T/P x N x M columns

Call it Z^k



Step 4. Perform matrix estimation on it to obtain

$$\widehat{L}_{ijk}^{\text{diff}}(t)$$

Recommendation: Algorithm

Step 5. Final estimate

$$\hat{L}_{ijk}(t) = \hat{L}_{ijk}^{\text{diff}}(t) + L_{ijk}^{\text{obs}}$$

Some remarks:

We *flattened* tensor in matrix to estimate to $\hat{L}_{ijk}^{\text{diff}}(t)$

In extremely sparse data regime, directly estimating tensor can help

But, it comes at the cost of increased computation

In Summary — Part I

Introduction, simple methods

Module 1: background

Recommendation systems: why and what?

Example datasets: Yelp & MovieLens

Module 2: problem statement

Recommendation systems: a prediction problem

Model: estimating time varying tensor with side information

Module 3: simple solutions

Solution I: averaging

Solution II: content-based

In Summary — Part II

Solution evolves, Matrix estimation

Module 1: clustering

Finding user and item clusters

Averaging *within* clusters

Module 2: collaborative filtering aka personalized clustering

Finding users and items similar to a given user, item

Averaging *amongst* user-item specific *similar* users, items

Module 3: singular value thresholding, optimization

It's a Matrix! find Singular Value Decomposition

Solve for least-squares

In Summary — Part III

Getting things together, some more and connections

Module 1: Matrix estimation meets content-based

Model

Estimation algorithm

Module 2: Matrix estimation across time

Model

Estimation algorithm

Module 3: Everything together

Model

Estimation algorithm

That's all, Folks!