In the first lecture, the discussion was around the basics of recommendation systems and their applications in different domains. Later, we defined the problem statement for recommendation systems and discussed two types of solutions - Averaging, where we assume that every user/item is the same, and Content-Based where we use additional information/features about users and/or items.

In the second lecture, we discussed more complex techniques to solve the problem of recommendation systems - Clustering, Collaborative Filtering, Singular Value Thresholding, and Optimization using Alternate Least Squares.

Now, in this final lecture, we are going to use the algorithms we have learned so far as building blocks to come up with algorithms that can be used to solve even more complex problems. In general, problems in recommendations systems have three dimensions:

1. **Multiple Measurements**: Data for observed preferences
2. **Content or Exogenous features**: Features of users/items
3. **Dynamics:** Time-varying aspect

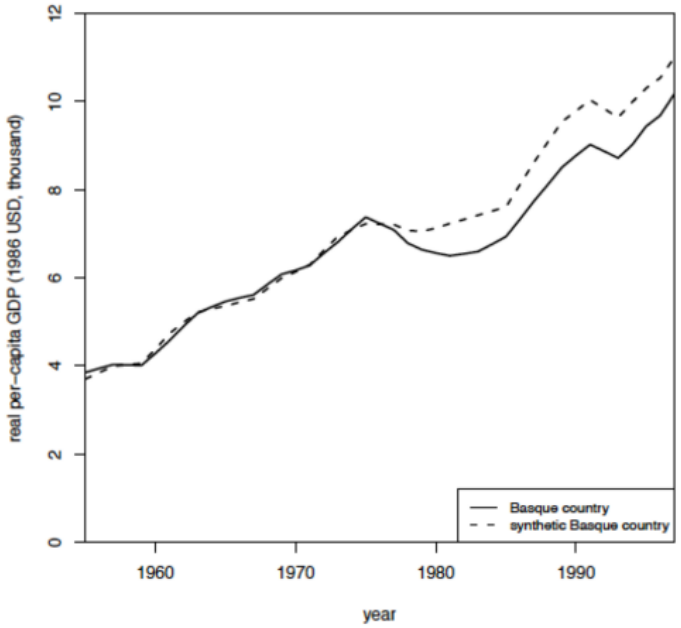
So far we have only touched upon the second dimension in our problems, but in this lecture, we will try to combine all three dimensions.

The main idea of this lecture is to demonstrate that **the problem of recommendation systems and their techniques can be used to solve complex machine learning problems** and are not limited to just providing recommendations. Before discussing algorithms, let's go through some examples to establish this idea:

- **Did Terrorism have an impact on the Economy of Basque Country?**

The Basque Country is a region in northern Spain. It was affected by terrorism from the mid to late 70s and it also experienced a decline in the per-capita GDP income around the same time. The question is whether this decline was due to that terrorism or due to some other factors?

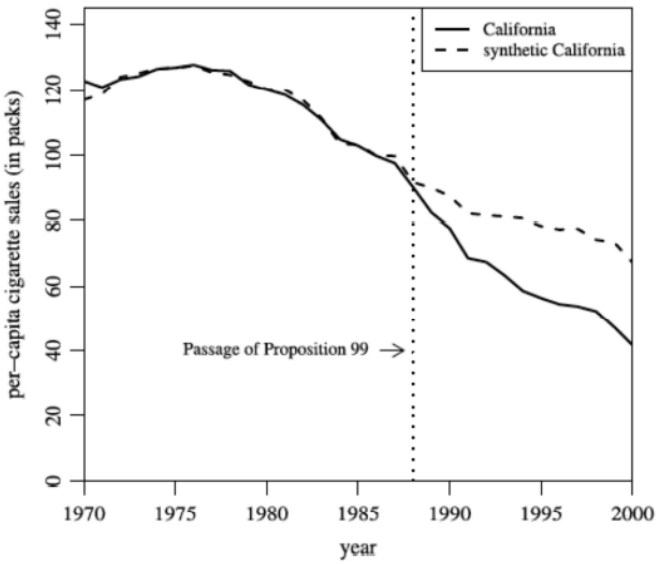
To answer this question, we need to come up with a counterfactual prediction, or a **synthetic prediction**, of the per-capita GDP of the Basque Country i.e. the GDP if there would have been no terrorism. In the below graph, the dark line shows the actual per-capita GDP of Basque Country and the dotted line shows the synthetic prediction.



These synthetic predictions can be generated by simply using the Matrix Estimations that were discussed in previous lectures. Before discussing how to come up with these synthetic predictions, let's go through one more example:

- **Did the California tobacco control program (Prop 99) work?**

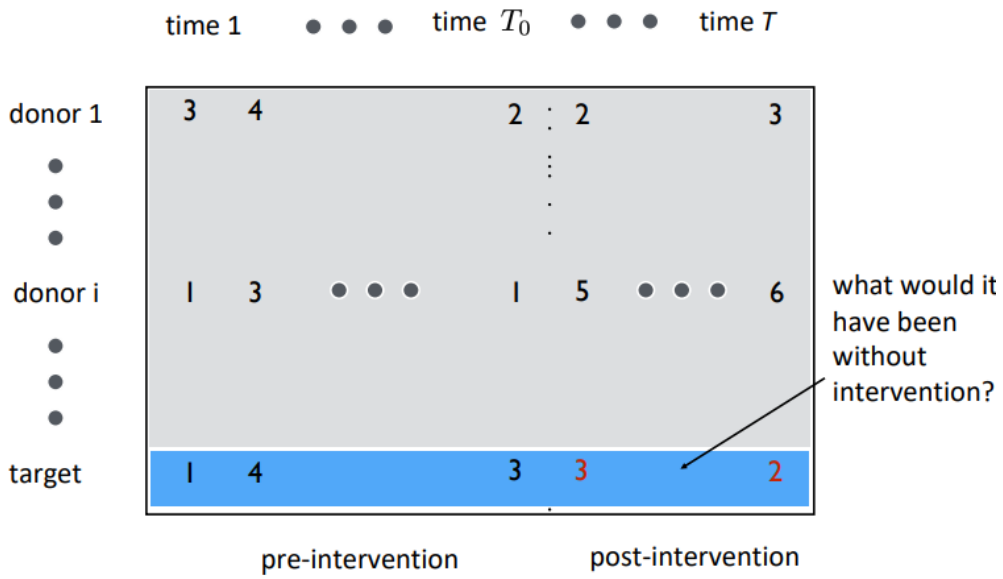
In 1988, California voters enacted Proposition 99, increasing the tax on cigarettes by 25 cents per pack on the sale of tobacco cigarettes within California. The per-capita cigarette sales declined after this proposition. The question is whether this decline in the sales is due to the proposition or some other factors?



Now, **how do we find these predictions?**

We will use Matrix Estimation to determine the synthetic predictions. The idea is similar to what we have learned in **Collaborative Filtering** - find users and items similar to a given user, item and then averaging (simple or weighted) amongst user-item specific similar users, items.

In the first example, we can observe the per-capita GDP of different regions in Spain or Europe which were not affected by terrorism, similarly for the second example, we can observe the per-capita cigarettes sales where Proposition 99 was not introduced. We will call this main factor (terrorism or Proposition 99) an **intervention.**

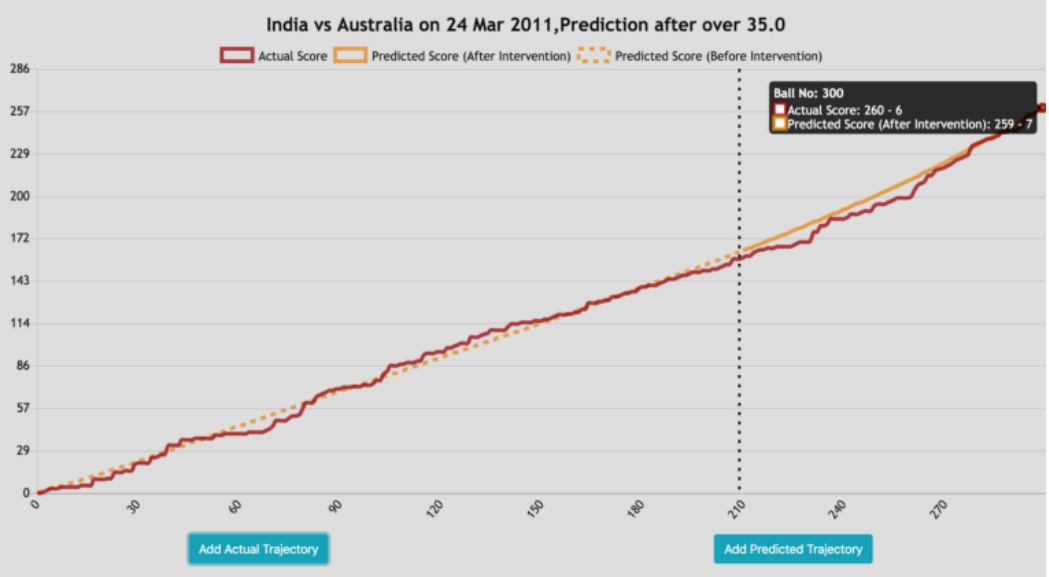


In the above image, time T_0 is the time when the intervention is introduced, the **target** is the main user for which we need to find predictions (California or Basque Country), and **donors** are similar users.

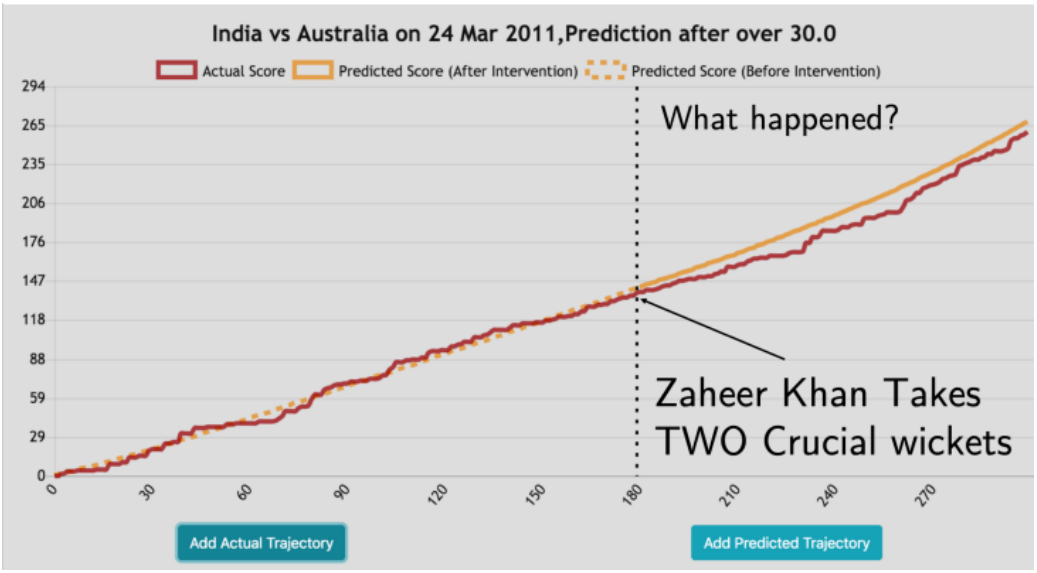
We will observe the pre-intervention data, find the similarity of each donor with the target, and use it to create synthetic post-intervention data. This method is called **synthetic control**.

There are many more applications where the technique of matrix estimation can be used to solve such complex time series problems. We can even use this technique to **predict scores in an ongoing game**. In such applications, the choice of T_0 matters greatly. Let's look at an example of **Forecasting Cricket Trajectory** in an India vs Australia game from 2011.

In a cricket game, there are 300 balls bowled by each team. The below images show the prediction of Australia's score when 210 balls have been bowled i.e. 70% of Australia's innings is finished. The prediction is pretty accurate, the lines for predicted and actual scores are almost overlapping.



But what if we change the T_0 from 70% to 60% i.e. 180 balls?



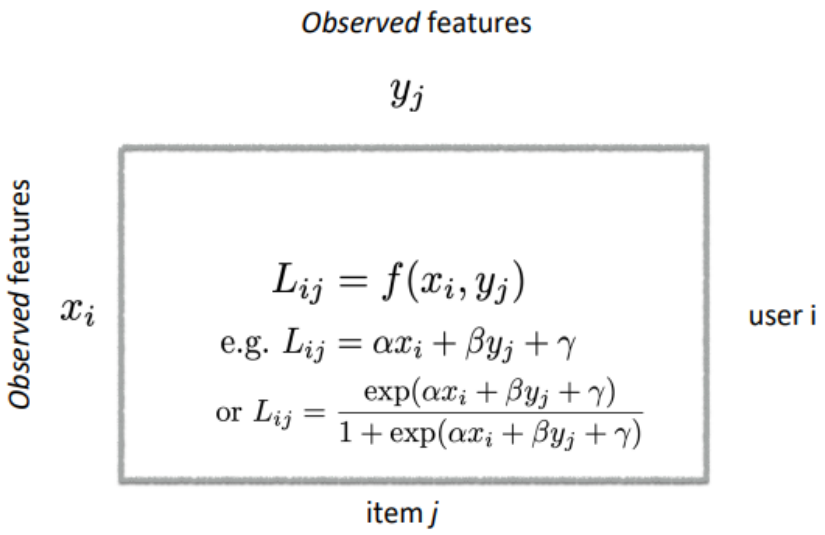
The above images show that the predictions change and some gap between the actual and predicted score can be observed. This is because a crucial moment happened at this point in the game which resulted in Australia's lower score than predicted. This implies that an important breakthrough happened around this time that affected the outcome of the game. Hence, this method can be used to find **highlights or breakthroughs in a game or history**.

Now that we have an understanding that recommendation system techniques can be used to solve other complex problems, let's go through the **first module of this lecture**.

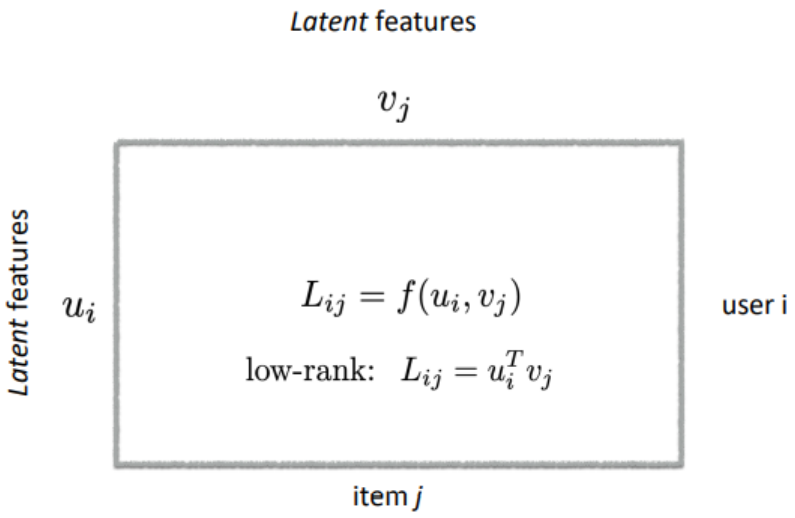
Matrix Estimations & Content-Based Recommendations

In previous lectures, we discussed the prediction problem of recommendation systems - complete the matrix where we need to find the likelihood of user i matching with item j i.e. L_{ij} .

In the content-based model, we can do this using **observed features** of user i, denoted by x_i , and item j, denoted by y_j. The below image shows that the problem of estimating L_{ij} is reduced to learning a model f which is a function of x_i and y_j. The learned model can be a simple regression model or classification model depending on the target values.

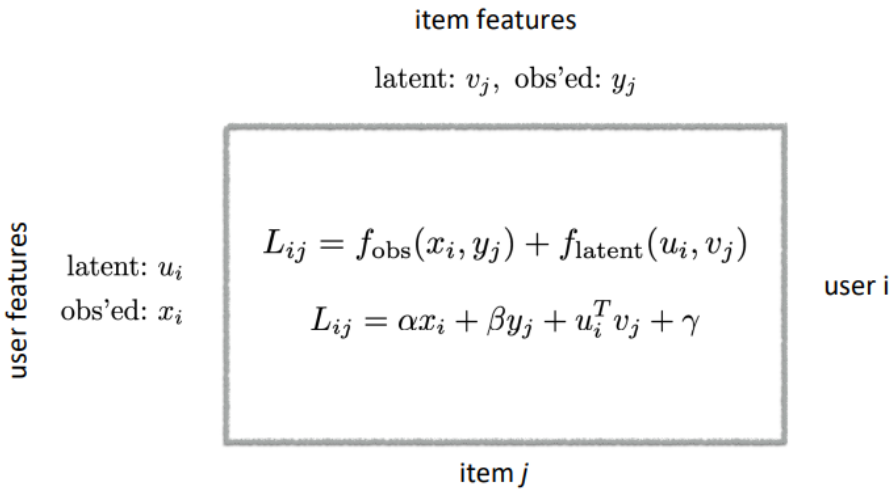


In matrix estimations, we assume that there are **latent features** of user i, denoted by u_i and item j, denoted by v_j, and L_{ij} is a function of those latent features.



While using one algorithm - content-based or matrix estimation, we lose information about the other set of features - latent or observed. **What if we can combine these two methods?**

We can take into account the function of latent features, u_i and v_j, as well as observed features, x_i and y_j, and combine them to get a more accurate estimation of L_{ij}.



The algorithm can be given in three steps as follows:

- Step 1: Content-based supervised learning
  - Learn the regressor (or classifier) using the observed features f_{obs} such that

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

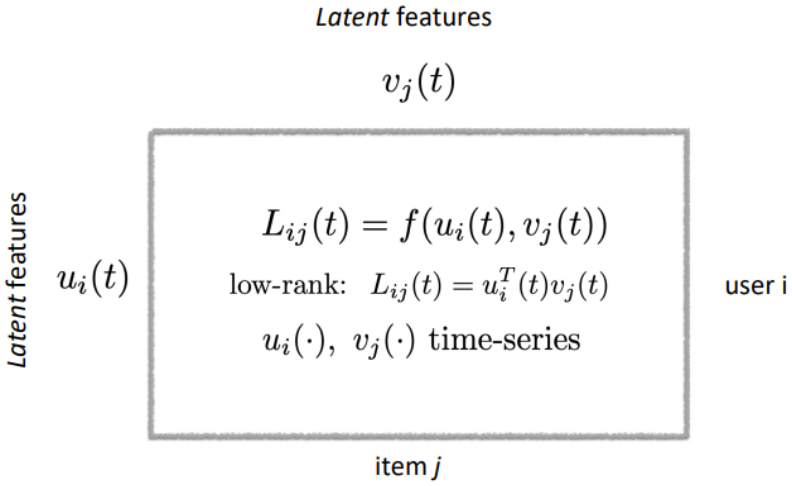
- Step 2: Matrix estimation
  - Compute "difference" matrix $L_{ij}^{diff} = L_{ij} - L_{ij}^{obs}$, over **observed entries**
  - Use matrix estimation on L_{ij}^{diff} to produce L^{ME}
- Step 3: Combine the final estimates $\widehat{L_{ij}} = L_{ij}^{obs} + L_{ij}^{ME}$

Now, let's move to the **second module of this lecture** that takes into account the time-varying factor while performing matrix estimation.

Matrix Estimation Across Time

So far we have only considered the completion matrix for a single instance in time but, in reality, time is an important factor as people's preferences or tastes change over time. For example, a successful movie from the 90s might not get very good reviews today.

So, the problem reduces to estimating time-varying matrices where latent features are time-varying, and observations are partial & noisy.



There are multiple time series, one for each entry i.e. each $L_{ij}(t)$ is a time series, indexed by t. So, if there are N users and M items, then there are $N \times M$ time series. We assume that each time series is explained by a series of latent features.

$$L_{ij}(t) = u_i(t)^T v_j(t)$$

Where each component of u_i and v_j is a structured time-series.

Note: For now, we are assuming that the observed features x_i and y_j are unknown or constant over time.

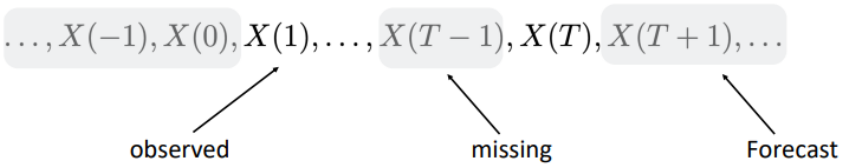
Since we might not observe entries for each user/item at every instance of time, the observations are partial and noisy. Also, we don't know $u_i(t)$ and $v_j(t)$ because they are latent features. So, how do we estimate the whole matrix L_{ij}? We will solve this massive time series by converting it into a recommendation system problem.

First, let's see how to **estimate a single entry** of the matrix L_{ij}, say $L_{1,1}$.

Suppose the following time series represents the **true values** of $L_{1,1}$:

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

Where T denotes the current time. Out of all these values, there might be some observations that are missing, some are observed and some we need to forecast.



As per the traditional time series approach, we might need to model all the components of the time series - seasonality, trend, and residuals. Here, we will ignore all of these, and try to **convert this time series into a matrix estimation problem**. How do we do that? We will do this in **three steps**:

Step 1: Transform the time series into a matrix

We will map the time series to a matrix called the **page matrix.** We will choose some $L > 1$, say 20, and divide the first T values of the time series into $\frac{T}{L}$ equal segments. We can treat each segment as a single column of the page matrix, denoted by P.

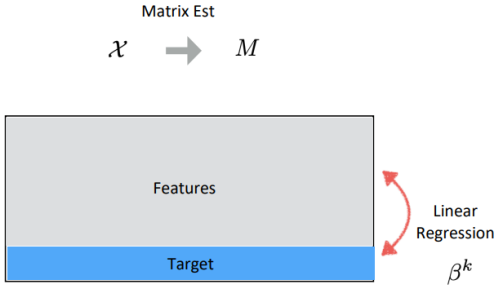
X(1)	X(L+1)		X(T-L+1)
X(2)	X(L+2)		X(T-L+2)
		... ...	
...	...		...
X(L)	X(2L)		X(T)

It is a matrix of dimension $L \times \frac{T}{L}$ whose each entry is given by:

$$P_{ij} = X(i + (j - 1)L)$$

Step 2: Do matrix estimation on the generated matrix

Once we do matrix estimation, we can treat each row as a feature and **forecast the time series** using simple regression models.



Step 3: Convert the matrix back to a univariate time series. That's it.

Now the question is **does this really work?**

Yes, it does. In practice, it works really well. The below table shows that this algorithm has **out-performed many state-of-the-art time series algorithms** in multiple domains.

	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

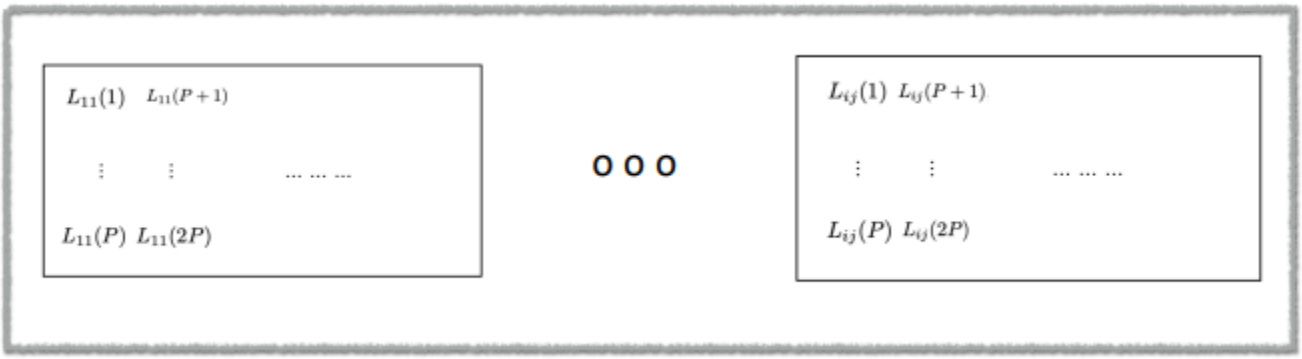
Remark: 'mSSA' stands for Multivariate Singular Spectrum Analysis.

Now that we understand the algorithm for a single entry $L_{1,1}$, let's look at how to extend this to estimate the whole matrix L_{ij}. We will again follow the below three steps:

Step 1: Convert time series for each entry into a (Page) matrix

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

Step 2: Concatenate matrices of all entries, along columns, into a big matrix, say Z.



The concatenated matrix has L rows and $\frac{T}{L} \times N \times M$ columns, where N and M are the numbers of users and items, respectively.

Step 3: Perform matrix estimation over Z

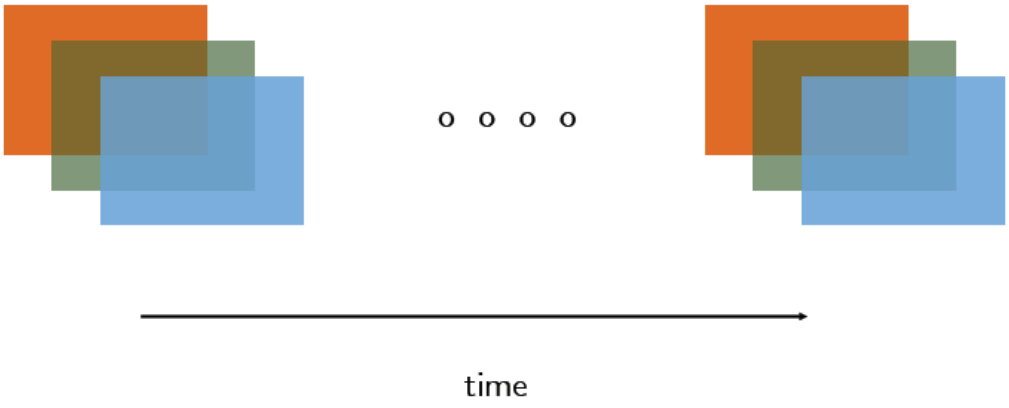
Once step 3 is completed, we can obtain the predictions by reversing the mapping.

So far we have discussed two dimensions - content and time but separately. Now, let's move to the **third module** of this lecture and combine everything together and also include the third dimension - multiple measurements.

Everything Together

In real-time, we have more pieces of information, we may have different interactions between users and items. These different interactions are called **measurements**. For example, in retail, we may have measurements like purchased an item or not, browsed an item or not, item added to cart or not, reviews for an item, etc., and these measurements are related to each other. This multilinear relationship among many slices of data can be represented by **tensors**.

The measurements that are available are changing over time. A **time-varying tensor** is a very high-dimensional complex object that changes over time with **very partial and noisy information**.



We need to model everything together now i.e. we need to estimate the time-varying tensor where each entry is represented as:

$$L_{i\,j\,k}(t)$$

Where i for the user, j is for the item, k is for the measurement, and t is for the time.

We will follow the same idea that we discussed in module 1 of this lecture - model using the observed features as well as latent features and then combine them. The general equation of the model is given as:

$$L_{i\,j\,k}(t) = f^{k}_{obs}(x_i, y_j) + f^{k}_{latent}(u_i(t), v_j(t)), \text{ where } u_i, v_j \text{ are time series}$$

The above equation implies that we are **modeling each slice of measurement separately** using the observed features that are not changing over time, and the latent features. But each of these **slices cannot be independent**, they must have some relation and the model would take it into account. For example, the below equations shows that f^{k}_{obs} is different for each k but f^{k}_{latent} has the same features, only multiplied by different weights for each k.

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

$$f^{k}_{latent}(u_i(t), v_j(t) = \sum_{l=1}^{d} u_{il}(t) v_{jl}(t) w_{kl}$$

So, how do we find $\widehat{L}$ i.e. predictions for the matrix L? The algorithm consists of 5 major steps. Let's go through them one by one.

Step 1: Content-based learning

For each measurement k, learn via supervised learning using the observed features

$$f^{k}_{obs}, \text{that is } (\alpha^{k}, \beta^{k}, \gamma^{k})$$

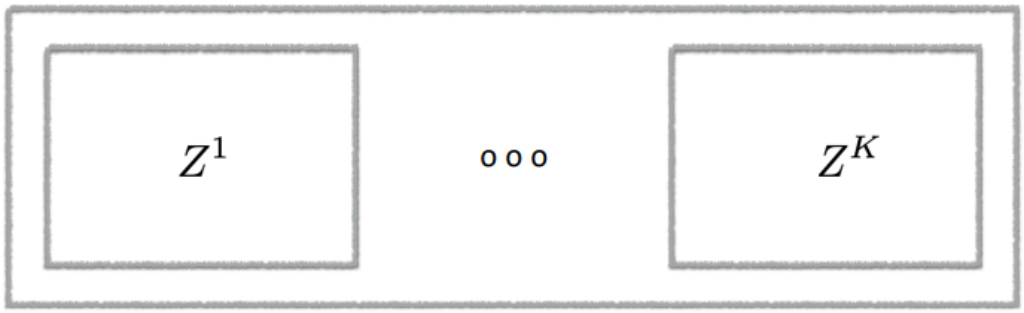
Since the observed features are independent of time, this would be a good model if things are not changing over time but most likely, that won't be the case which takes us to the next step.

Step 2: Obtain difference over observed entries

$$L^{diff}_{i\,j\,k}(t) = L_{i\,j\,k}(t) - L^{obs}_{i\,j\,k}, \text{ where } L^{obs}_{i\,j\,k} = f^{k}_{obs}(x_i, y_j)$$

Step 3: Build stacked page matrix across entries, slices of tensor

As discussed in the second module of this lecture, we can create a page matrix, Z^k, for each measurement k, from the differences obtained in step 2. Then we can stack all the matrices together to create a huge single matrix. This is called **flattening a tensor** in a matrix.



Step 4: Perform matrix estimation on the stacked matrix to obtain predictions on $L^{diff}_{i\,j\,k}(t)$, which is given as:

$$\hat{L}^{diff}_{i\,j\,k}(t)$$

Step 5: Compute the final estimate

$$\hat{L}_{i\,j\,k}(t) = \hat{L}^{diff}_{i\,j\,k}(t) + L^{obs}_{i\,j\,k}$$

In general, the final estimate works well as it takes into account all three dimensions of the problem.

Remark: We flattened the tensor in a matrix to estimate $\hat{L}^{diff}_{i\,j\,k}(t)$ but in an extremely sparse data regime, directly estimating the tensor can help. However, this will increase the computational cost significantly.

Links for Additional Reading

- Original paper on time series predict DB
- Links to get started/explore tspDB -
  - http://tspdb.mit.edu/
  - https://www.powtoon.com/s/fkVh3axA4Jy/1/m
- CricketML
- The original paper on Model Agnostic Time Series Analysis via Matrix Estimation