client2vec: Towards systematic baselines for banking applications

Problem Statement

We need to build an embedding system for different types of clients in any system, which posses some properties like that neighboring points in the space of embeddings correspond to clients with similar behavior.

INTRODUCTION:

- client2vec is a library developed at BBVA D&A (a data science company) to speed up the construction of informative baselines for behavior centric banking applications.
- It focuses on behaviors which can be extracted from account transactions data by encoding that information into vector form (client embedding).

APPROACH:

- client2vec follows analogy with unsupervised word embeddings whereby account transactions can be seen as words, clients as documents (bags or sequence of words) and the behavior of a client as the summary of a document.
- So, it should exhibit the fundamental property that neighboring points in the space of embeddings correspond to clients with similar behaviors.
- Two possible paths of doing this:
  i). one is to extract vector representations of transactions and compose them into client embeddings.
  ii). the other is to embed clients straight away

- The first approach can be explored by applying famed word2vec algorithm to our data and then pooling the embeddings of individual transactions into client representations with a variety of methods.
- For the latter approach, which is the one currently employed by client2vec, we built client embeddings via a marginalized stacked denoising autoencoder (mSDA).
- For comparison and benchmarking purposes, we also tested the embedding comprising the raw transactional data of a client and the one produced by sociodemographic variables.

ACCOUNT TRANSACTIONS DATA:

	CAT1	CAT2	CAT3	···	CAT70
client 1	-10.15	-527.11	1250.67	···	-
client 2	-	-12.26	800.32	···	322.81
⋮					
client N	-45.65	-600.00	2400.56	···	-241.67

Figure 1: Raw transaction data: a year worth of expenses aggregated averaged by client and expense category.

- We chose to focus client2vec on an aggregation of current account data by client, year and transaction category which can easily be applied to most commercial cases of interest.
- Specifically, for a year worth of data, we aggregate each client into a vector of 70 numbers, one per category, as depicted in Figure 1.

Sociodemographic Variables:

- The sociodemographic variables on which we compared all our methods are: age, gender, income range, postcode, city and province.
- Such variables are typically considered by banks, retailers and other organizations for purposes like segmentations or campaigns.
- All of these variables are categorical, even the income, having been binned in several ranges.
- So, we one-hot encode them and then reduce the dimensionality of the vector thus obtained in order to measure the Euclidean distance between two sociodemographic representations.

Raw transactions:

- The simplest behavioral embedding of a client could be expressed as $x_{raw} = (x_1, \ldots, x_k, \ldots, x_{70})$, where x_k is the aggregated expense of that customer in the kth category.
- This corresponds to one row of the table in Figure 1, possibly normalized (typically such that $||x_{raw}||^2 = 1$) or binarized.
- If an individual account operation in a certain category is interpreted as a “word”, we can think of the raw transactions embedding as one-hot encodings (when binarized), or as “bag of words” (when continuous).

Embedding via word2vec:

- In order to apply word2vec to account transactions it was key to transform our transactions data into sequences of meaningful tokens that word2vec could process.
- So, to introduce a word-like entity we created a suitable quantization of our data.
- First, percentiles are computed for all transactions of our dataset in each category, i.e. for all the columns of the table in Figure 1.
- The percentiles are then used to define bins for the transactions of each category. Each bin is labeled with a word composed by the category identifier and the bin's boundaries
- For example, referring to Figure 1, CAT1 -50.20:-7.16 is a possible label of a bin containing the transactions in CAT1 of both clients 1 and N.

Embedding via word2vec:

- Such words then correspond to expenses in a certain category and range.
- Finally, a quantized copy of the raw dataset is built by replacing individual transactions with the label of the bin they fall inside, hence yielding a finite set of repeating words
- The dataset thus transformed assumes some of the structural features of natural text, such as a finite vocabulary and a sentence-like organization.

From transactions to clients:

- Applying word2vec on the quantized dataset will yield transaction embeddings, which we then need to compose into client embeddings.
- Strategies such as concatenation or meanpooling have been proposed to compose transaction embeddings obtained through word2vec.

Embeddings via mSDA

Autoencoders

An autoencoder is a type of artificial neural network used to learn efficient data encodings in an unsupervised manner.

Denoising

Denoising autoencoders attempt to randomly corrupting input that the autoencoder must then reconstruct, or denoise.

Stacked

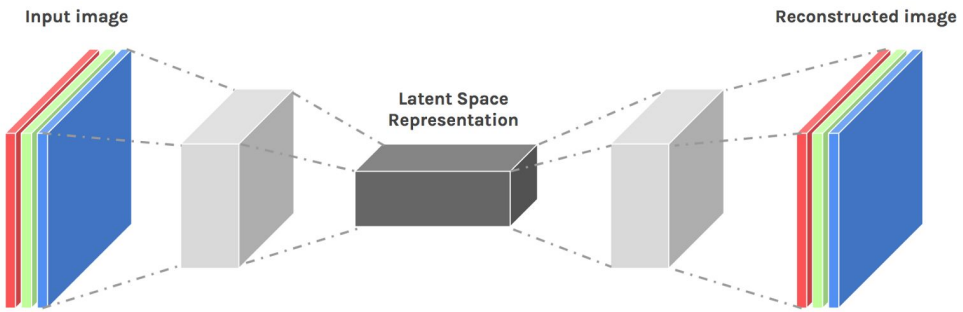
A stacked autoencoder is a neural network consist several layers where output of each hidden layer is connected to the input of the successive hidden layer.

Marginalised

It used to marginalize out the noise distribution and train the autoencoder as if it had observed infinitely many noisy variations of the training set.

Autoencoder

- It is a type of artificial neural network used to learn efficient data coding in an unsupervised manner.
- Aim is to learn a representation for a set of data.
- It is using two main parts :
  - an encoder that maps the input into the code
  - a decoder that maps the code to a reconstruction of the original input.



Component of Autoencoders :

01	Encoders	• It input x_i into a hidden representation h_i. • $h_i = g(Wx_i + b)$ • $h_i \varepsilon R^d, x_i \varepsilon R^n, W\varepsilon R^{d\ xn}, b\varepsilon R^{dx1}$
02	Latent Feature	• These are the features we want to learn • They should have a property that they should cover all the important features of the original data.
03	Decoders	• It input h_i into a hidden representation x'_i. • $x'_i = g(W^*h_i + c)$ • $h_i \varepsilon R^d, x'_i \varepsilon R^n, W^* \varepsilon R^{nxd}, c \varepsilon R^{nx1}$

Basic Structure

An autoencoder consists of two parts, the encoder and the decoder, which can be defined as transitions ϕ and Ψ such that:

$\phi : X \longrightarrow F$
$\Psi : F \longrightarrow X$

given one hidden layer, the encoder stage of an autoencoder takes the input $x \, \varepsilon \, R^n = X$ and maps it to $H \, \varepsilon \, R^d = F$.

$$H = \sigma (WX + b) \quad \ldots (1)$$

here, h is known as latent feature representation of original input.
After that, the decoder stage of the autoencoder maps *h* to the reconstruction x' of the same shape as x:

$$X' = \sigma' (W^*H + c) \quad \ldots (2)$$

W, b and W*,c are the weight matrix and bais for encoder and decoder respectively.

Choice of activation function :

Case 1: all inputs are binary
- use logistic function it restricts output between 0 to 1.

Case 2: all inputs are real values
- We need output to also be real values then we use linear function in decoder side and we can use sigmoid or any other activation in encoder side.

Choice of loss function :

Case 1: all inputs are binary
- Cross entropy loss

$$min\,(-\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{n} x_{ij}\log\left(\hat{x}_{ij}\right) + (1-x_{ij})\log\left(1-\hat{x}_{ij}\right))$$

Case 2: all inputs are real value
- Squared error loss

$$min\,(\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{n} (\hat{x}_{ij}-x_{ij})^{2})$$

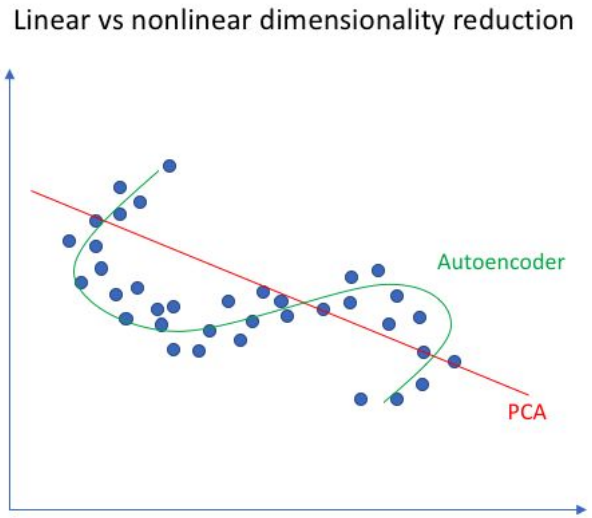
Consider a case:

- dim(H) < dim(X)
  - It is called under complete autoencoders.
  - If we are able to reconstruct the input then it is called loss-free encoding of x_i.

- dim(H) >= dim(X)
  - It is called over complete autoencoders.
  - here, autoencoder could learn a trivial encoding by simply copying data into hidden layers.
  - Used where you need to disentangled the features.
  - ex. BMI function using height and weight.

Autoencoders VS PCA

- PCA is essentially a linear transformation but auto-encoders are capable of modelling complex nonlinear functions.
- PCA features are totally linearly uncorrelated with each other since features are projections onto the orthogonal basis. But autoencoded features might have correlations.
- PCA is faster and computationally cheaper than autoencoders.
- A single layered autoencoder with a linear activation function is very similar to PCA.
- Autoencoder is prone to overfitting due to high number of parameters.



Autoencoders are equivalent to PCA :

Conditions are:

- Normalised input

$$\hat{x}_{ij} = \frac{1}{\sqrt{m}} \left(x_{ij} - \frac{1}{m} \sum_{k=1}^{m} x_{ij} \right)$$

- Data now has 0 mean along each dimension j.
- Now, $X = 1/(m)^{½} * X'$
- Then, $(X)^T X = 1/m * (X')^T X'$ is the covariance matrix. ….(1)

- **If we use linear decoder and a squared error loss function then optimal solution is obtained when we use a linear encoder.**

Given: Linear decoder, Squared error loss function

Proof:

$$E = min\left(\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{n}(\hat{x}_{ij} - x_{ij})^2\right) \approx \min(\,||\, X\text{-}HW^* \,||_F\,)^2$$

We know that: $X' = U\sum V^T \approx HW^*$ … *using SVD*

Let, $H = U_{<=K} \sum_{k*k}$ and $W^* = V^T_{<=K}$ … using best k-approximation

Now we have to show that this is a linear encoding and find an expression for the encoder weights W.

$$H = U_{<=K} \sum_{k*k}$$

$$= (XX^T)(XX^T)^{-1} U_{<=K} \sum_{k*k}$$

$$= (XV\sum^T U^T)(U\sum V^T V\sum^T U^T)^{-1} U_{<=K} \sum_{k*k} \quad \ldots X=U\sum V^T$$

$$= XV\sum^T U^T U(\sum\sum^T)^{-1} U^T U_{<=K} \sum_{k*k} \quad \ldots (ABC)^{-1}=C^{-1}B^{-1}A^{-1}$$

$$= XV\sum^T (\sum^T)^{-1} \sum^{-1} U^T U_{<=K} \sum_{k*k}$$

$$= XV\sum^T I_{<= K} \sum_{k*k}$$

$$= XVI_{<=K}$$

$$= XV_{<=K}$$

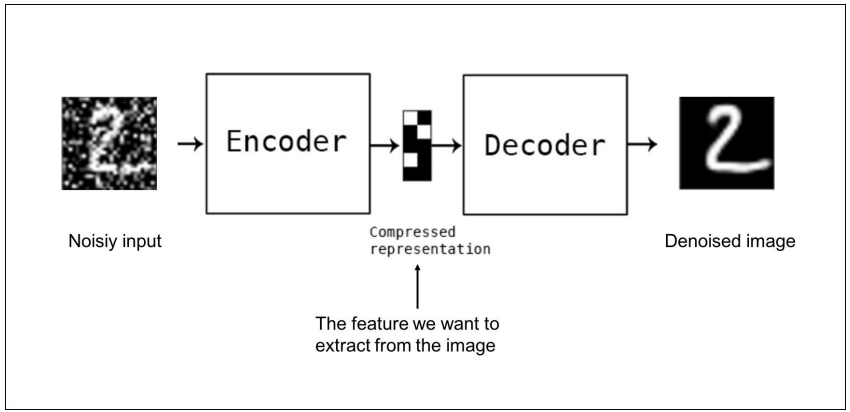
- We have encoded : $W=V_{<=K}$
- From SVD we know that V is the matrix of eigenvector of (XX^T). …(2)
- From PCA we know that P is the matrix of the eigenvectors of the covariance matrix.
- Using (1) and (2) we can say that W is the matrix of the eigenvectors of the covariance matrix.
- Thus the encoder matrix for linear encoder (W) and the projection matrix (P) for PCA could indeed be the same.

Hence Proved

Denoising autoencoders:

A denoising autoencoders corrupts the input data using a probabilistic process before feeding it to the network.
We use a probability function $P(x'_{ij} | x_{ij})$ as :

- $P(x'_{ij} = 0 | x_{ij}) = q$
- $P(x'_{ij} = x_{ij} | x_{ij}) = 1-q$



How does this help?

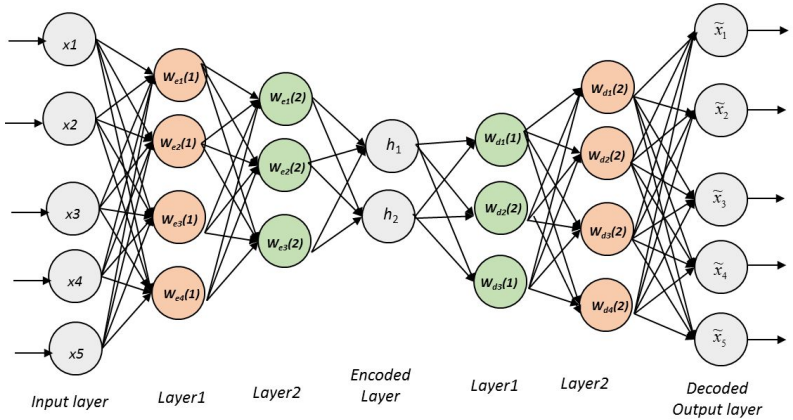
This will help us because the objective is still to reconstruct the original x_i.

Stacked autoencoders:

A stacked autoencoder is a neural network consist several layers where output of each hidden layer is connected to the input of the successive hidden layer.

Stacked autoencoder mainly consists of three steps :

- Train autoencoder using input data and acquire the learned data.
- The learned data from the previous layer is used as an input for the next layer and this continues until the training is completed.
- Once all the hidden layers are trained use the backpropagation algorithm to minimize the cost function and weights are updated with the training set to achieve fine tuning.



Marginalized Denoising Autoencoder:

To lower the variance, we perform multiple passes over the training set, each time with different corruption.
Solve for the W that minimizes the overall squared loss :

$$\mathcal{L}_{sq}(\mathbf{W}) = \frac{1}{2mn}\sum_{j=1}^{m}\sum_{i=1}^{n}\|\mathbf{x}_i - \mathbf{W}\tilde{\mathbf{x}}_{i,j}\|^2 \quad \approx \quad \mathcal{L}_{sq}(\mathbf{W}) = \frac{1}{2nm}\mathrm{tr}\left[\left(\overline{\mathbf{X}} - \mathbf{W}\widetilde{\mathbf{X}}\right)^{\top}\left(\overline{\mathbf{X}} - \mathbf{W}\widetilde{\mathbf{X}}\right)\right]$$

here, m is the number of passes over the training set or number of times we are corrupted our data.
The larger m, the more corruption we average over.

Aim of this architecture is to learn the statistical dependencies among the variables.
This type of embedding has the following interpretation:

- Each entry holds the most likely average transaction value in the corresponding category, given all the other transaction of that client.

Case Study

Three different base test cases were performed to check the offering of the embedding.

1. Clustering of Clients
2. The Typical Traveler
3. Missing Category Prediction

Case 1: Clustering of Clients

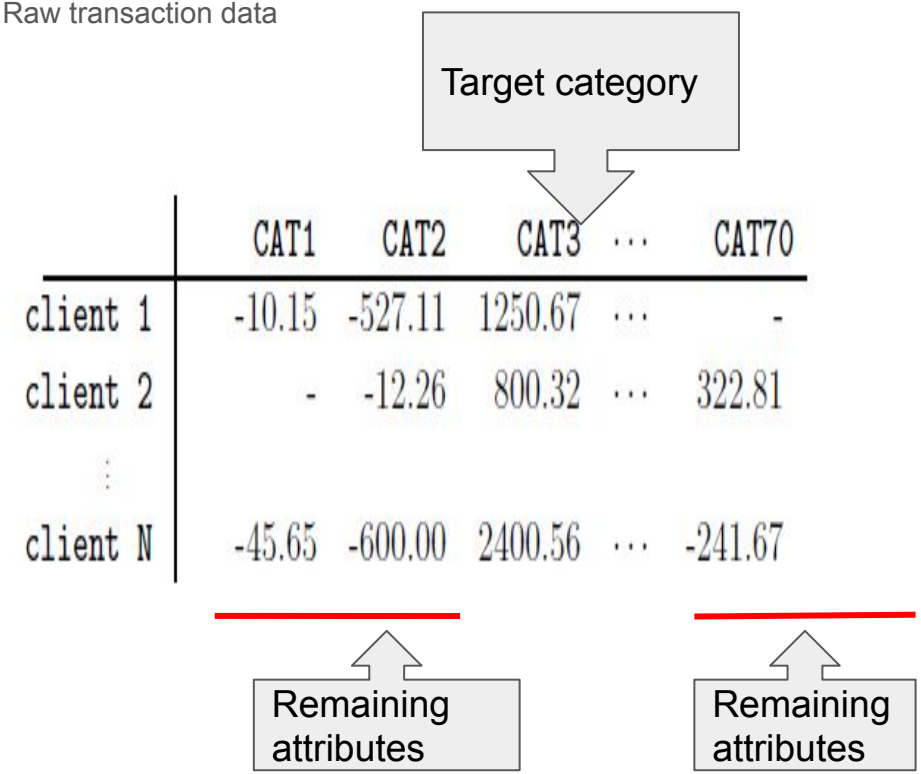
- One of the most immediate use cases for a client embedding is generating a segmentation of the client base.
- Example - “ Compare Yourself ” in banking application.
- customers can compare their own expenses in certain categories against those of their socio demographic segment, which is defined by 4 variables (age, income, gender and postcode).

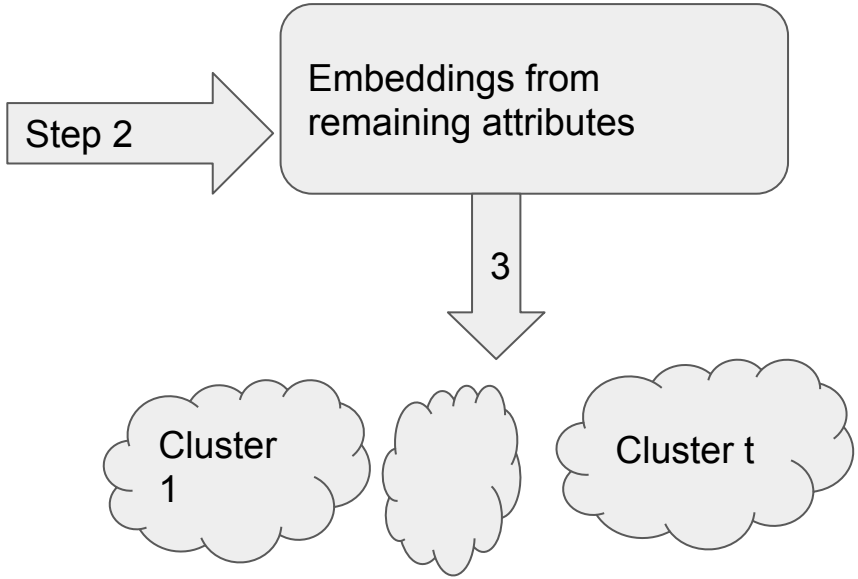
Approach - plug the embeddings into a clustering algorithm and compare the **homogeneity**

of the values of a left-out target category in the resulting clusters.

workflow

Raw transaction data





Setup and evaluation

A dataset with a year worth of data for 120k clients.

For each target **category t** and **embedding E**, calling X_t the expense of client X in category t , so the vector representation can be written as

$$X_{-t} = (x_{1,} , x_2 , \ldots, x_{t-1} , x_{t+1}, \ldots, x_k)$$

This vector can be used to compute an embedding $E(X_{-t})$,

Steps:-

- Each time one target category for evaluation purpose is left out.
- The embeddings thus obtained are clustered via K-means in 300 clusters.
- Evaluation of embeddings using average dispersion across all category.

cont.

Let ℐ be the set of transaction categories.

Let $C_t(E)$ be the set of clusters produced for categories $t \in ℐ$ by the embedding E.

1. So, $S_c = \{x : E(X_{-t}) \in c\}$: the set of customers whoso embeddings is in $c \in C_t(E)$
2. $S_c^{out} = \{x_l : x \in S_c\}$: the corresponding values in the left-out category

The average dispersion across all categories:

$$\Delta(E) = 1/ |ℐ| \sum_{t \in ℐ} \text{Median} [\{ \text{std} (S_c^{out})\} c \in C_t(E)]$$

The rationale is that if customers are really similar within a cluster, this similarity should generalize to the unseen category and produce a low dispersion.

Case 2: The typical traveler

Consider the case of clients that have travel-related expenses,and conjecture that a typical traveler should also have expenses in the hotels category.

Aim :- to extract different definitions of typical travelers according to different embeddings.

Approach:-

1. Different embeddings were obtained by assigning a person to a random socio demographic segment.
2. In each embedding space we look at the 10 most central clients in each of the 10 densest clusters.
3. The clients considered in different embedding spaces may differ , so the proximity of a client to the cluster and cluster density were considered for evaluation.

cont.

- Each matrix corresponds to a different embedding.
- The Roman-numbered blocks of rows correspond to clusters,
- rows to clients and columns to transaction categories.
- Green dots correspond to incoming transactions (e.g. salary payments),while grey dots to outgoing transactions.
- The blue column corresponds to expenses in the Hotels category.

Case 3: Missing category prediction

This study measures the effectiveness of a client embedding is being able to reconstruct whether a person had an expense in a target category.

Can be considered as approximation of product propensity prediction.

A dataset of 120K clients was used, splitting it in 100K customers as the training set, 10K as a validation set and 10K as the test set.

$$\Theta_t(x) = |sgn(x_t)| \quad \text{.....eq 1.}$$

indicates whether client x has a transaction in the target category t.

evaluation

To estimate previous equation Nearest neighbor regression in the embedding space is performed.

$$\square_k^E(x) = \{k \text{ nearest neighbors to } x\} \text{.....eq 2.}$$

the neighbors are defined by a dot-product similarity in the space of the embedding E

So the preference score for the event that client x had an expense in category t , can be obtained as-

$$\theta_t\hat{}(x) = 1/k \sum_{z \in \square} \theta_t(z)$$

For a target category t,where the sum is taken over the set X of all clients ordered by decreasing values of θˆ and P(θˆt(x)) is the precision computed on those clients with score higher than θˆt(x)

$$\mathrm{AP}_{\theta_t}(\mathcal{X}) = \frac{\sum_{x \in \mathcal{X}} P(\hat{\theta}_t(\mathbf{x})) \cdot \theta_t(\mathbf{x})}{\sum_{x \in \mathcal{X}} \theta_t(x)} ,$$

Inference drawn from the case studies

From case 1 a baseline requiring one call to the client2vec library to obtain client embeddings, plus K-Means, yields more homogeneous customer segments than those of hand-crafted, domain-specific attributes.

From case 2 it is evident that looking at different dense clusters,will prevent us from focusing exclusively on a single behavioral pattern or socio demographic segment.

The structural information is lost with the sociodemographic embedding, where the spending behavior appears fragmented and uncorrelated to the cluster structure. Which implies sociodemographic variables capture close to no information about people's spending habits.

Using case 3 scenario , we can form a product propensity prediction.

Conclusion

- Client2vec is an approach that could catalyze the data-driven decision making.
- Solution that is simple to use, fast to deploy and integrate processes and that required minimal preprocessing.
- Composing transactional embeddings extracted with word2vec into customer embeddings doesn’t always offer an acceptable performance, while mSDAs help us capture a good deal of behavioral information.
- The approach showed that information can be extracted even from simple, coarse transactional data.

References

- client2vec: Towards Systematic Baselines for Banking Applications-*Leonardo Baldassini, Jose Antonio Rodríguez Serrano*
- Nptel : Autoencoder *Mitesh M. Khapra*
- Autoencoder

Thank You