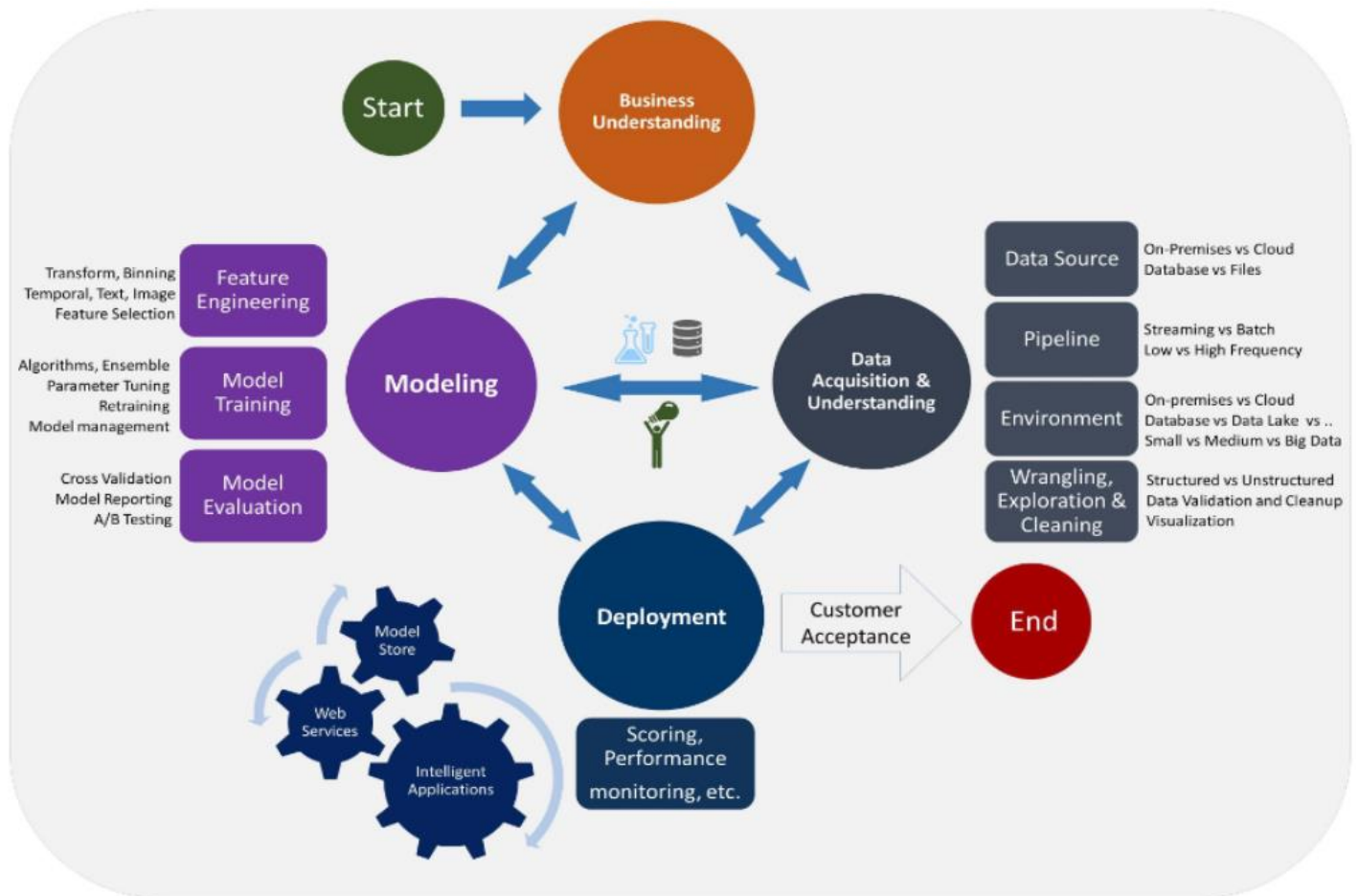


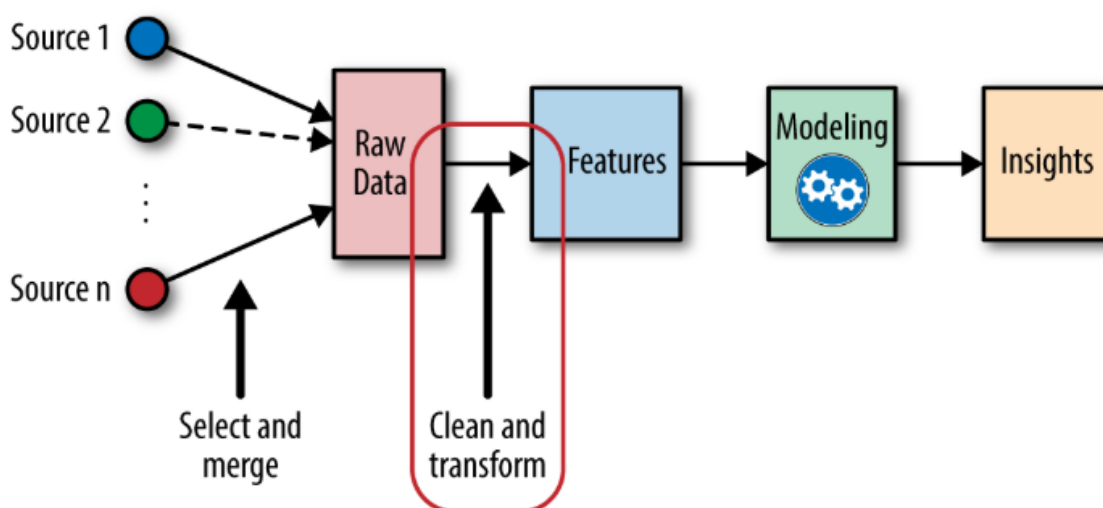
IMPORTANCE OF FEATURE ENGINEERING → PART-1

Following is typical process of Data Science Life Cycle



Reference: → <http://www.datascience-pm.com/microsoft-team-data-science-process/>

Above we can see typical Data science life cycle, in this FEATURE ENGINEERING is very importance part

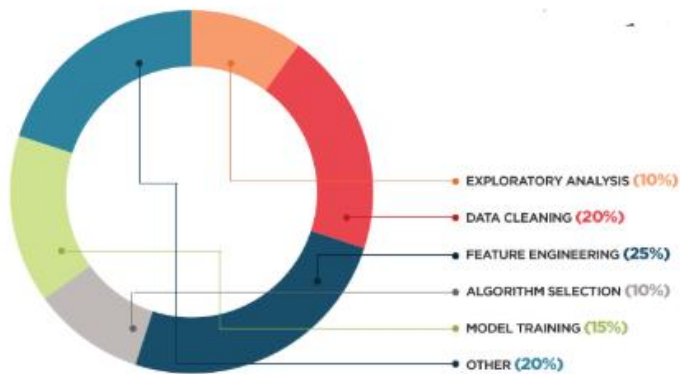


Machine learning fits mathematical models to data in order to derive insights or make predictions. These models take features as input. A feature is a numeric representation of an aspect of raw data. Features sit between data and models in the machine learning pipeline. Feature engineering is the act of extracting features from raw data, and transforming them into formats that is suitable for the machine learning model. It is a crucial step in the machine learning pipeline, because the right features can ease the difficulty of modelling, and therefore enable the pipeline to output results of higher quality.

A picture relevant to our discussion on feature engineering is the front-middle of this process. It might look something like the following:

1. (tasks before here...)
2. Select Data: Integrate data, de-normalize it into a dataset, collect it together.
3. Pre-process Data: Format it, clean it, sample it so you can work with it.
4. Transform Data: Feature Engineer happens here.
5. Model Data: Create models, evaluate them and tune them.
6. (tasks after here...)

Most of the time Data scientist spend on cleaning, data extraction and engineering features



What is Feature Engineering & Why it is important?

Feature engineering is a process of transforming the given data into a form which is easier to interpret. Here, we are interested in making it more transparent for a machine learning model, but some features can be generated so that the data visualization prepared for people without a data-related background can be more digestible. However, the concept of transparency for the machine learning models is a complicated thing as different models often require different approaches for the different kinds of data.

Reference: → [kdnuggets](#)

- Feature Engineering is a data preparation process
- Feature engineering is about creating new input features from your existing ones.
- It is the one of the most time consuming task in creating models
- Based on type of data different Feature Engineering techniques are used.
- Depending on kind of data domain expertise are involved to create good features.

{Coming up with features is difficult, time-consuming, requires expert knowledge. "Applied machine learning" is basically feature engineering.

— Andrew Ng, *Machine Learning and AI via Brain simulations*}

Why Feature Engineering is important?

- 1) We can highlight & extract key information, which helps algorithms to achieve good results.
- 2) once we understand the depth of feature engineering of any specific data, we can bring in other people's domain expertise.
- 3) Feature engineering is the process of transforming raw data into features that better represent the underlying problem to the predictive models, resulting in improved model accuracy on unseen data.
- 4) Improving the performance of machine learning models.

Iterative Process of Feature Engineering

The process might look as follows:

- 1) Brainstorm features: Really get into the problem, look at a lot of data, study feature engineering on other problems and see what you can steal.
- 2) Devise features: Depends on your problem, but you may use automatic feature extraction, manual feature construction and mixtures of the two.
- 3) Select features: Use different feature importance scorings and feature selection methods to prepare one or more "views" for your models to operate upon.
- 4) Evaluate models: Estimate model accuracy on unseen data using the chosen features.

There are different feature engineering techniques used depending on data, it always depends on type of data and variety of feature engineering is vary high.

FEATURE UNDERSTANDING

In this we begin to explore different kind of data & identify characteristics of different attributes then we will understand type of transformations that are allowed and that promise to improve our machine learning algorithm

Some examples →

- Identifying missing data values
- Exploratory data analysis
- Descriptive statistics
- Data visualization

FEATURE IMPROVEMENT

Here we will be using mathematical transformation to enhance given data but not remove or insert any new attributes

- Data imputation
- Dealing with missing values in a dataset
- Normalization of data
 - Standardization (z-score normalization)
 - Min-max scaling
 - L1 and L2 normalization

FEATURE SELECTION

Here we understand which columns/features are not helping my ML pipeline and should be removed so there are different techniques used to make the decision of which attributes to get rid of in our dataset

Some example →

- Statistical based feature selection
 - Correlation coefficients
 - Identifying and removing multicollinearity
 - Chi-squared tests
 - Anova tests
 - Interpretation of p-values
- Model based feature selection
 - Using machine learning to measure entropy & information gain
 - Using CountVectorizer built in max_features, min_df, max_df

FEATURE CONSTRUCTION

Here we try techniques in creating brand new features and placing them correctly within our dataset

These new features ideally holds new information & generate new patterns that ML pipelines will be able to exploit and use to increase performance

Some examples →

- Imputing categorical features
- Encoding categorical variables
- Extending numerical features
- Text specific feature construction
 - BagOfWords, Count Vectorizer, Tf-Idf vectorizer

FEATURE TRANSFORMATIONS

Here we started to look at the automatic creation of these features as it applies to mathematical dimensionality. If we regard our data as in an n-space (n being number of columns) we will see if we can create a new dataset in a k-space (where $k < n$) that fully or nearly represents the original data, but might give us speed boosts or performance enhancements in machine learning.

Some examples →

- Principal component Analysis
- Linear Discriminant Analysis

FEATURE LEARNING

Here we outline algorithms that are not in and of themselves a mathematical formulation but an architecture attempting to understand and model data in such a way that it will exploit patterns in data in order to create new data.

Some examples →

- Restricted Boltzmann machines
- Word2vec/Glove for word embedding

Now we will discuss about different data and feature engineering techniques used for different types of data which are easily available on Kaggle.

Following data are Kaggle dataset for more details about data you can check data on Kaggle as well.

1) NETFLIX MOVIE RECOMMENDATIONS
#####



ABOUT DATA: →

Netflix is all about connecting people to the movies they love.

If you check data available on Kaggle for Netflix, The first line of each file [combined_data_1.txt, combined_data_2.txt, combined_data_3.txt, combined_data_4.txt] contains the movie id followed by a colon. Each subsequent line in the file corresponds to a rating from a customer and its date in the following format:

CustomerID,Rating,Date

MovieIDs range from 1 to 17770 sequentially.

CustomerIDs range from 1 to 2649429, with gaps. There are 480189 users.

Ratings are on a five star (integral) scale from 1 to 5.

Dates have the format YYYY-MM-DD.

Example: →


```
1:
1488844,3,2005-09-06
822109,5,2005-05-13
885013,4,2005-10-19
30878,4,2005-12-26
823519,3,2004-05-03
893988,3,2005-11-17
124105,4,2004-08-05|
1248029,3,2004-04-22
1842128,4,2004-05-09
```

We convert data to dataframe and data looks like following

1	df.head()				
	movie	user	rating	date	
56431994	10341	510180	4	1999-11-11	
9056171	1798	510180	5	1999-11-11	
58698779	10774	510180	3	1999-11-11	
48101611	8651	510180	2	1999-11-11	
81893208	14660	510180	2	1999-11-11	

After this we create Sparse Matrix from DataFrame

MOVIE_ID	USER_ID	RATING
1	1	3
2	1	4
3	1	2
3	2	1
4	2	4
8	2	2
1	3	3
7	3	1
10	3	5



	1	2	3	4	5	6	7	8	9	10	(movie)
1	3	4	2	-	-	-	-	-	-	-	
2	-	-	1	4	-	-	-	-	-	-	
3	3	-	-	-	-	-	1	-	-	5	
(user)											

Data we have is in following format

USER-ITEM Matrix							
	I_1	I_2	...	I_j	...	I_{m-1}	I_m
U_1				
U_2				

U_i			...	A_{ij}	...		

U_{n-1}				
U_n				

USER-ITEM Matrix

FEATURE ENGINEERING FOR NETFLIX DATA

Before going into details, we need to understand Content Based and Collaborative filtering

Collaborative Filtering

$U_1 \rightarrow m_1, m_2, m_3$

$U_2 \rightarrow m_1, m_3, m_4$

$U_3 \rightarrow m_1$

U_1 & U_2 both like movie m_3 so there is high chance that U_3 also like m_3

Core Idea: \rightarrow Users who agreed in the past to also agree in the future

Content Based RS

Here we are using Movie j features like Genre of movie, Actor, Director, Year of release etc.

Similarly, for User i we use features like U_i likes action movie or romantic movie, Age, Location of user etc.

There are different types of collaborating filtering

Similarity Matrices

- \rightarrow User-User similarity Matrix
- \rightarrow Movie-Movie similarity Matrix

USER-USER similarity Matrix

Reference → <https://machinelearningcoban.com/2017/05/24/collaborativefiltering/>

Reference → Appliedai course

We have data in User-Item Matrix format then we convert it into User-User matrix format

In User Movie Matrix each row represents a user which contains ratings given by a user to all the movies.

Using Cosine similarity to convert the data into User-User similarity matrix, this matrix is not sparse.

Here for each user let's say U_i it will try to calculate dot product similarity with each other users & find top similarities with U_i .

$$\text{cosine_similarity}(\mathbf{u}_1, \mathbf{u}_2) = \cos(\mathbf{u}_1, \mathbf{u}_2) = \frac{\mathbf{u}_1^T \mathbf{u}_2}{\|\mathbf{u}_1\|_2 \cdot \|\mathbf{u}_2\|_2}$$

It took a lot of time to calculate

	I_1	I_2	...	I_j	...	I_{m-1}	I_m
U_1				
U_2				
...
U_i			...	A_{ij}	...		
...
U_{n-1}				
U_n				

USER-ITEM Matrix

	U_1	U_2	...	U_j	...	U_{n-1}	U_n
U_1	1	Sim_{12}	...	Sim_{1j}	...		
U_2		1		
...
U_i		Sim_{i2}	...	Sim_{ij}	...		
...
U_{n-1}			1	
U_n				1

USER-USER Similarity

→ Once we have this similarity matrix and we need to recommend new movies for User 10 (U_{10})

- We will check for U_{10} and find that U_1, U_2, U_7 are the most similar users to U_{10}
- Now let's pick Movies that are like by $U_1, U_2, \& U_7$ that are not yet watched by U_{10}
- Recommend them to U_{10}

Problem with User-User Similarity: → User preferences changes with time, If they are changing to often then problem but if it changes not much often then we can build matrix w.r.t time

	u_0	u_1	u_2	u_3	u_4	u_5	u_6
i_0	5	5	2	0	1	?	?
i_1	4	?	?	0	?	2	?
i_2	?	4	1	?	?	1	1
i_3	2	2	3	4	4	?	4
i_4	2	0	4	?	?	?	5
\bar{u}_j	3.25	2.75	2.5	1.33	2.5	1.5	3.33

a) Original utility matrix \bar{Y} and mean user ratings.

	u_0	u_1	u_2	u_3	u_4	u_5	u_6
i_0	1.75	2.25	-0.5	-1.33	-1.5	0	0
i_1	0.75	0	0	-1.33	0	0.5	0
i_2	0	1.25	-1.5	0	0	-0.5	-2.33
i_3	-1.25	-0.75	0.5	2.67	1.5	0	0.67
i_4	-1.25	-2.75	1.5	0	0	0	1.67

b) Normalized utility matrix \bar{Y} .

	u_0	u_1	u_2	u_3	u_4	u_5	u_6
u_0	1	0.83	-0.58	-0.79	-0.82	0.2	-0.38
u_1	0.83	1	-0.87	-0.40	-0.55	-0.23	-0.71
u_2	-0.58	-0.87	1	0.27	0.32	0.47	0.96
u_3	-0.79	-0.40	0.27	1	0.87	-0.29	0.18
u_4	-0.82	-0.55	0.32	0.87	1	0	0.16
u_5	0.2	-0.23	0.47	-0.29	0	1	0.56
u_6	-0.38	-0.71	0.96	0.18	0.16	0.56	1

c) User similarity matrix S .

	u_0	u_1	u_2	u_3	u_4	u_5	u_6
i_0	1.75	2.25	-0.5	-1.33	-1.5	0.18	-0.63
i_1	0.75	0.48	-0.17	-1.33	-1.33	0.5	0.05
i_2	0.91	1.25	-1.5	-1.84	-1.78	-0.5	-2.33
i_3	-1.25	-0.75	0.5	2.67	1.5	0.59	0.67
i_4	-1.25	-2.75	1.5	1.57	1.56	1.59	1.67

d) \hat{Y}

Predict normalized rating of u_1 on i_1 with $k = 2$

Users who rated i_1 : $\{u_0, u_3, u_5\}$

Corresponding similarities: $\{0.83, -0.40, -0.23\}$

⇒ most similar users: $\mathcal{N}(u_1, i_1) = \{u_0, u_5\}$

with normalized ratings $\{0.75, 0.5\}$

$$\Rightarrow \hat{y}_{i_1, u_1} = \frac{0.83 * 0.75 + (-0.23) * 0.5}{0.83 + |-0.23|} \approx 0.48$$

e) Example

	u_0	u_1	u_2	u_3	u_4	u_5	u_6
i_0	5	5	2	0	1	1.68	2.70
i_1	4	3.23	2.33	0	1.67	2	3.38
i_2	4.15	4	1	-0.5	0.71	1	1
i_3	2	2	3	4	4	2.10	4
i_4	2	0	4	2.9	4.06	3.10	5

f) Full \hat{Y}

MOVIE-MOVIE similarity Matrix

As described in User-User similarity similarly we are calculating Movie-Movie similarity matrix

For any movie M_i we want to calculate similarity with all other movies, top similar movies data is what we need.

Here advantage is rating of an given movie do not changes significantly after the initial period for example titanic movie rating

	u_0	u_1	u_2	u_3	u_4	u_5	u_6	
i_0	5	5	2	0	1	?	?	→ 2.6
i_1	4	?	?	0	?	2	?	→ 2
i_2	?	4	1	?	?	1	1	→ 1.75
i_3	2	2	3	4	4	?	4	→ 3.17
i_4	2	0	4	?	?	?	5	→ 2.75

a) Original utility matrix \mathbf{Y} and mean item ratings.

	u_0	u_1	u_2	u_3	u_4	u_5	u_6
i_0	2.4	2.4	-0.6	-2.6	-1.6	0	0
i_1	2	0	0	-2	0	0	0
i_2	0	2.25	-0.75	0	0	-0.75	-0.75
i_3	-1.17	-1.17	-0.17	0.83	0.83	0	0.83
i_4	-0.75	-2.75	1.25	0	0	0	2.25

b) Normalized utility matrix $\tilde{\mathbf{Y}}$.

	i_0	i_1	i_2	i_3	i_4
i_0	1	0.77	0.49	-0.89	-0.52
i_1	0.77	1	0	-0.64	-0.14
i_2	0.49	0	1	-0.55	-0.88
i_3	-0.89	-0.64	-0.55	1	0.68
i_4	-0.52	-0.14	-0.88	0.68	1

c) Item similarity matrix \mathbf{S} .

	u_0	u_1	u_2	u_3	u_4	u_5	u_6
i_0	2.4	2.4	-0.6	-2.6	-1.6	-0.29	-1.52
i_1	2	2.4	-0.6	-2	-1.25	0	-2.25
i_2	2.4	2.25	-0.75	-2.6	-1.20	-0.75	-0.75
i_3	-1.17	-1.17	-0.17	0.83	0.83	0.34	0.83
i_4	-0.75	-2.75	1.25	1.03	1.16	0.65	2.25

d) Normalized utility matrix $\tilde{\mathbf{Y}}$.

After preparing it we try to find top 10 similar movies for "Vampire Journals" and we got following results

Top 10 similar movies

```
1 movie_titles.loc[sim_indices[:10]]
```

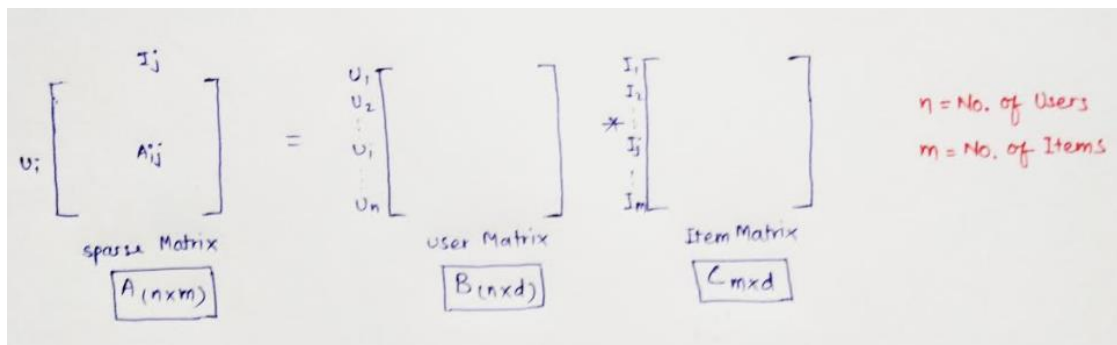
movie_id	year_of_release	title
323	1999.0	Modern Vampires
4044	1998.0	Subspecies 4: Bloodstorm
1688	1993.0	To Sleep With a Vampire
13962	2001.0	Dracula: The Dark Prince
12053	1993.0	Dracula Rising
16279	2002.0	Vampires: Los Muertos
4667	1996.0	Vampirella
1900	1997.0	Club Vampire
13873	2001.0	The Breed
15867	2003.0	Dracula II: Ascension

MATRIX FACTORIZATION

First, we will discuss about matrix factorization and then we will connect matrix factorization to recommender system problem.

If we are able to decompose matrix A ---- to ---- > matrix B, matrix C, matrix D

PCA, SVD is example of matrix factorization



$$A_{ij} = B_i^T * C_j \quad \text{or} \quad A_{ij} = B_i * C_j^T$$

$(d \times 1) \quad (1 \times d) \qquad \qquad (1 \times d) \quad (d \times 1)$

Our goal is to find $B + C$
we have A_{ij} not empty in Sparse Rating matrix
So $\sum_{i,j} (A_{ij} - B_i^T * C_j)^2$ is minimized

\Downarrow

so our optimization problem will be

$$\underset{B, C}{\operatorname{argmin}} \sum_{i,j} (A_{ij} - B_i^T C_j)^2$$

across all i, j
such that
 A_{ij} is not empty

Here $\{A_{ij} - B_i^T * C_j\}$ is nothing but an error which we have to minimize.

Therefore, this is an optimization problem. This is nothing but a **squared-loss** which can be minimized by SGD algorithm. This looks like a regression problem. In ideal case $(A_{ij} - B_i^T * C_j)$ will become 0.

After solving this problem using SGD, we will get matrices ' B ' & ' C '. ' B ' will be of size ' $N \times d$ ' and ' C ' will be of size ' $M \times d$ '. Once we get matrices ' B ' & ' C ' then this problem will become a **matrix completion** problem. Now for any empty cell A_{ij} in matrix ' A ', we can calculate $\{B_i^T * C_j\}$. Let say $A_{10,5}$ is empty. We can now fill $A_{10,5}$ as " $B_{10}^T * C_5$ ". By this we can fill all of the empty cells in matrix ' A '. These ratings which we got for empty cells are nothing but the predicted ratings.

To explain it I have attached screenshot to explain it.

① solve optimization problem SGD

$$\arg \min_{B, c} \sum_{i,j} (A_{ij} - B_i^T c_j)^2$$

(2) $B = [\leftarrow B_i \rightarrow]$; $C = [\leftarrow C_j \rightarrow]$

③ Matrix completion

$A = \begin{bmatrix} & & 5 & & \\ & \square & | & \square & \\ 10 & - & \square & \square & \square \end{bmatrix}$
empty
 $A_{10,5} = \text{empty}$

using above calculated B + C we can fill our empty cells in A

for ex $A_{10,5} = \beta_{10}^T * C_5$

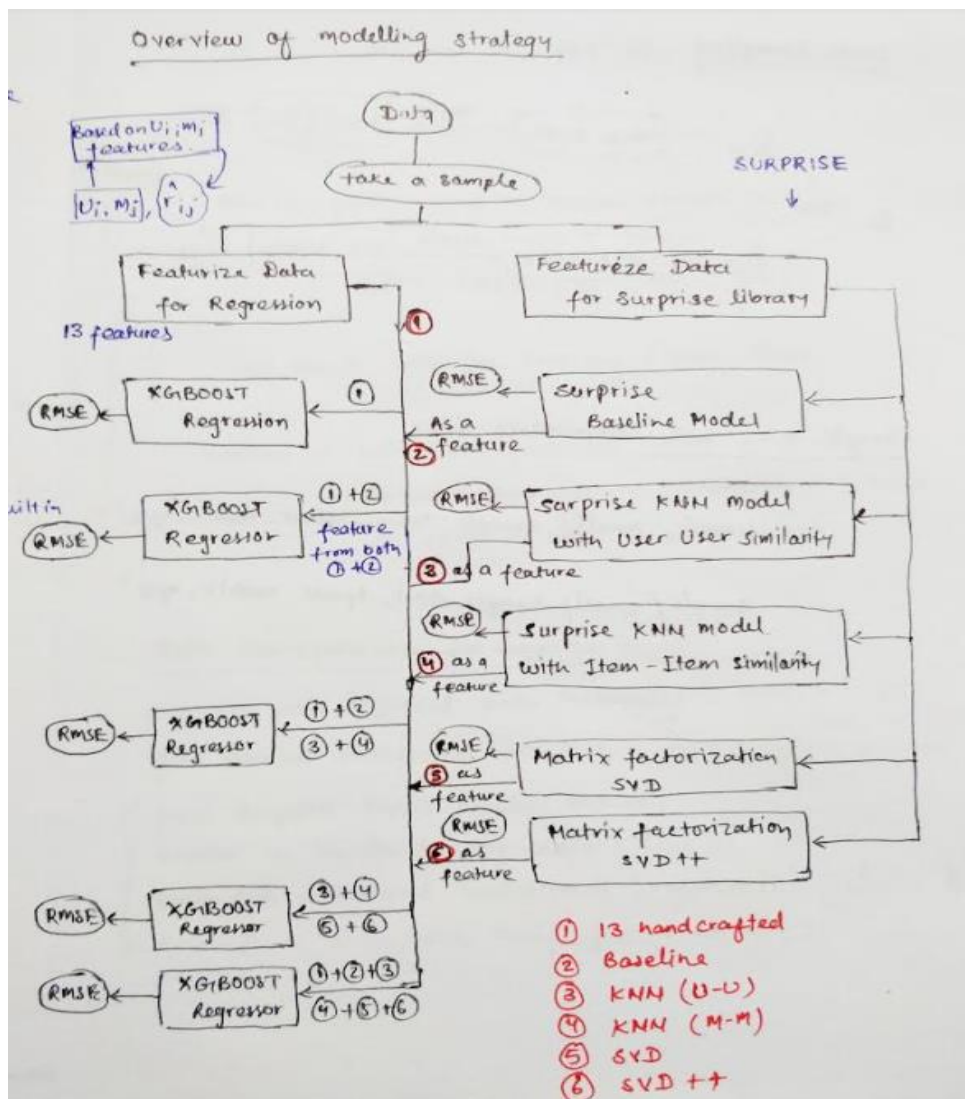
initially we have A

$A = \begin{matrix} & I_j \\ v_i & \end{matrix} \left[\begin{array}{c} \\ \\ \\ \end{array} \right]_{n \times m}$ → solve optimization problem we get B,C → $\hat{A} = \left[\begin{array}{c} \\ \\ \\ \end{array} \right]$

Total of empty cells sparse matrix

\hat{A} is completed form of A

So above we have explained different feature engineering techniques used for NETFLIX MOVIE RECOMMENDATION problem. Once our features are ready we can apply models on top of it we have applied models in below format



2) FACEBOOK FRIEND RECOMMENDATION

PROBLEM STATEMENT: → Given a directed social graph, we have to predict missing links to recommend friends/connections/followers

DATA OVERVIEW: →

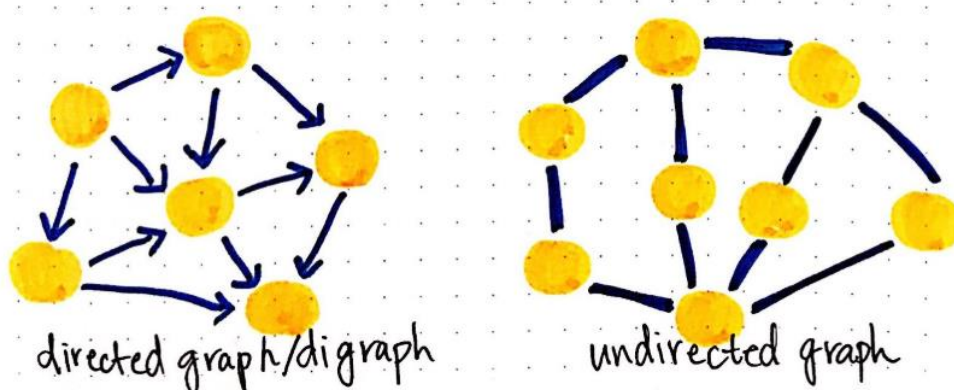
Reference : → <https://medium.com/basics/a-gentle-introduction-to-graph-theory-77969829ead8>

Reference → Appliedai course

Let me explain keywords which we will use in this dataset

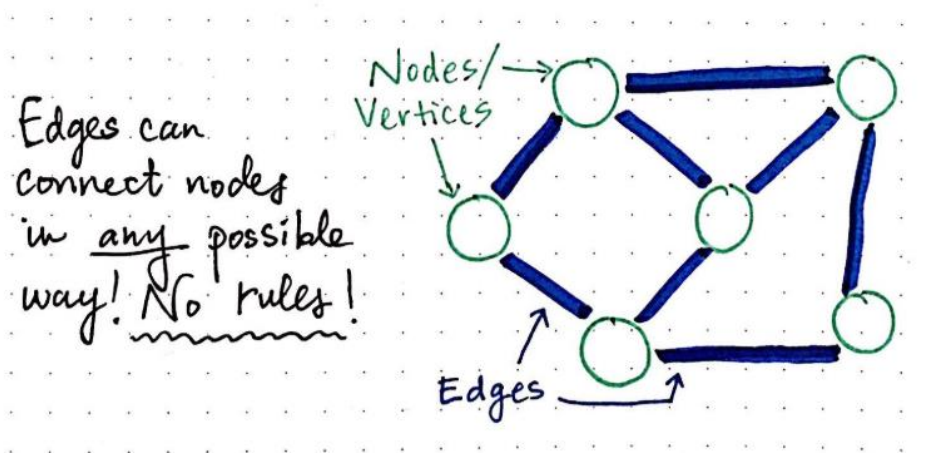
Graphs: node/vertex, edge/link, directed-edge path

Let's discuss two types of graphs: directed graphs, and undirected graphs.



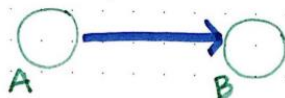
Directed graphs as compared to undirected graphs

Now in a graph what is Node/Vertices and Edge/Link can be easily seen in below diagram

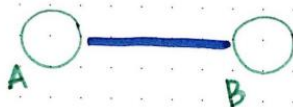


Edges can be of two types Directed and Undirected

Different types of edges in graphs



directed edge: there is only a path from A, the origin, to B, the destination



undirected edge: the path between A and B is bidirectional, meaning origin + destination are not fixed.

Directed edges compared to undirected edges

So, Graph is set of Vertices and set of Edges

In Facebook friend recommendation we have directed graph and Users are Nodes/Vertices for example $u_1, u_2, u_3 \dots$

Edges/Link explains if any User following another User for example if user u_1 following User2 then there will be directed edge from u_1 to u_2 .

Path is set of edges (valid edges)

Data for Facebook Kaggle challenge contains two columns

- ➔ Source Edge pairs
- ➔ Destination Edge pairs

Train.csv ➔ Source_node, Destination_node

Given is pair of source and destination or pair of nodes or pair of vertices

Total Number of vertices/Nodes ➔ 1862220

Total Number of directed Edges ➔ 9437520

We are given with one snapshot at time t which is dynamic at time changes graph grows.

Suppose we have U_i and U_j pair in train.csv which means there is edge between them but there is no entry between U_i and U_k which means we need to predict new connection based on available edges.

The diagram shows a handwritten table with three columns: 'Source Node', 'Destination Node', and ' Y_i '. The first part of the table has rows with source nodes u_1, u_1, \dots and destination nodes u_5, u_6, \dots , with corresponding Y_i values of 1. A bracket on the right of this section points to the text 'for all $Y_i = 1$ ' and '➔ train.csv', which also lists '1.86 M Nodes' and '9.43 M Edges'. The second part of the table has rows with empty source and destination cells and Y_i values of 0. A bracket on the right of this section points to the text 'Need to generate randomly because if generated for all than data become very HIGH'.

Source Node	Destination Node	Y_i
u_1	u_5	1
u_1	u_6	1
\vdots	\vdots	\vdots
		1
		0
		0
		\vdots
		0

1 ➔ Having Edge
0 ➔ Not having an edge

for all $Y_i = 1$

➔ train.csv
1.86 M Nodes
9.43 M Edges

Need to generate randomly because if generated for all than data become very HIGH

- ➔ We generated class 0 data to balance with class 1
- ➔ For class 0 another point we take care is we will consider only nodes which have shortest path distance greater than 2

Now we are familiar with data so we will discuss about **FEATURE ENGINEERING on GRAPH based DATA**

1. JACCARD & COSINE SIMILARITIES
2. PAGE RANK
3. SHORTEST PATH
4. CONNECTED COMPONENT
5. KARTZ CENTRALITY
6. HITS SCORE
7. SVD
8. WEIGHT FEATURES

JACCARD & COSINE SIMILARITIES

Jaccard Distance → J

For User u1 → X → {u5, u3, u4}

For User u2 → Y → {u3, u4, u6}

$$J = \frac{|X \cap Y|}{|X \cup Y|} = (\text{size of intersection b/w 2 sets X \& Y}) / (\text{size of union b/w 2 sets X \& Y})$$

We can compute Jaccard distance between followers set and we can also compute Jaccard distance b/w followee set

J follower & J followee

If Jaccard distance value is HIGH or If we have higher overlap b/w followers sets or followee set then chance of having edge will be high

Cosine Distance →

In natural language processing we have same distance for vectors

Let's discuss its modified version on sets

$$\text{Cosine Distance} = \frac{|X \cap Y|}{\text{SQRT}(|X| * |Y|)}$$

Here, X & Y are sets, |X| & |Y| are size of X and size of Y

For example →

For User u1 → X → {u5, u3, u4}

For User u2 → Y → {u3, u4, u6}

Here Cosine Distance will be $\frac{2}{\sqrt{3*3}}$

We can compute it for followers and followee both

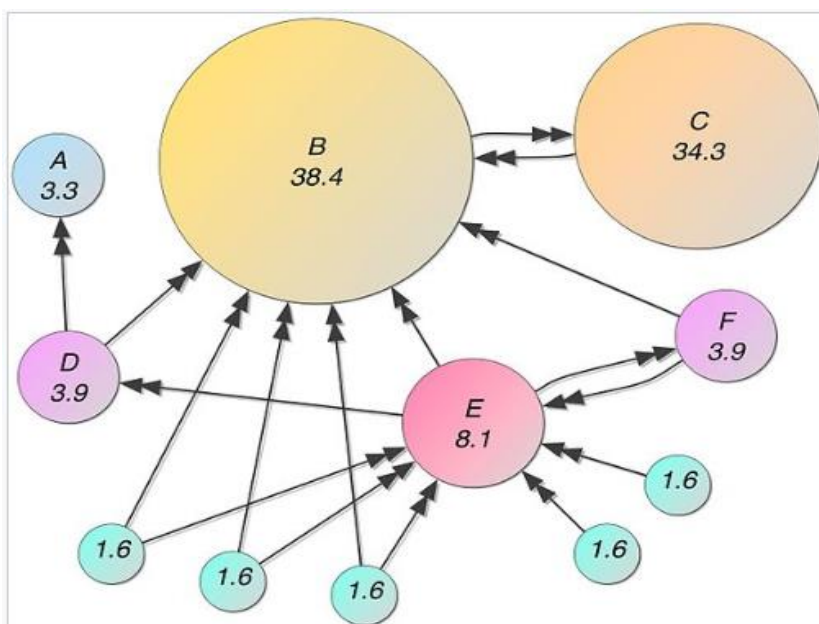
If cosine Distance is High → Chance of edge is HIGH

PAGE RANK

Reference → <https://en.wikipedia.org/wiki/PageRank>

It is for Directed Graph, PageRank (PR) is an algorithm used by Google Search to rank web pages in their search engine results. PageRank was named after Larry Page, one of the founders of Google. PageRank is a way of measuring the importance of website pages.

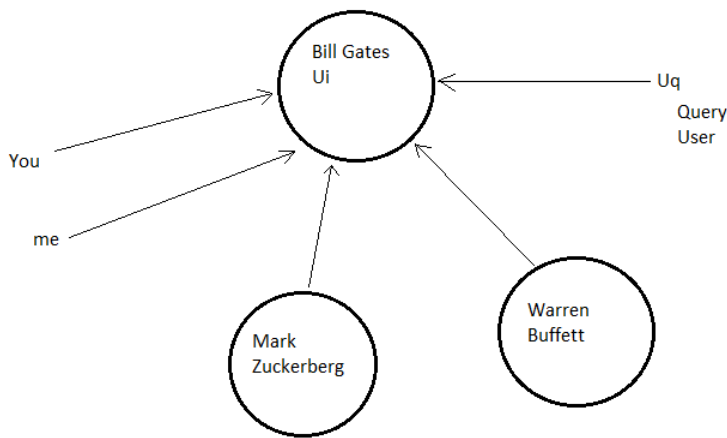
PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely to receive more links from other websites



Page C has a higher PageRank than Page E, even though there are fewer links to C; the one link to C comes from an important page and hence is of high value.

- 1) Lot of pages linking to page B → so trusted site
- 2) If there are lot of link from other popular pages → HIGH Rank

Page rank works by counting the number and quality of links to a page to determine a rough estimate of how important the websites is, important assumption is that more important websites are likely to receive more links from other websites



High Score of Bill Gates → Query user have higher chance to follow Bill gates
You and me have low score → so lower chance

For doing it we will use → Network X

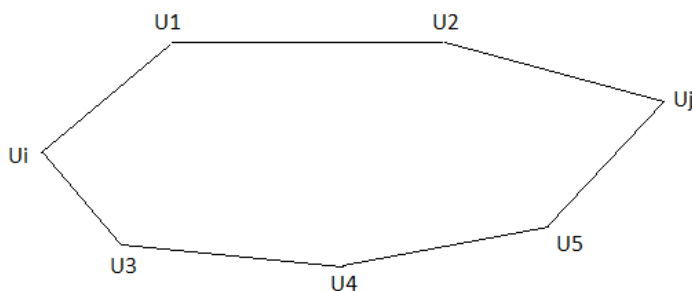
For each U_i → calculate page rank value – probability value

We will get 2 features

- Page Rank source
- Page Rank Destination

SHORTEST PATH

To understand shortest path let's consider below figure



If we want to calculate shortest path between U_i & U_j

U_i to U_j → 2 paths

- One is of length 3 via U_1 & U_2
- Another is of length 4 via U_3 , U_4 & U_5

→ Getting shortest path between 2 nodes

If nodes have an edge i.e. trivially connected than we need to remove that edge and calculate shortest path

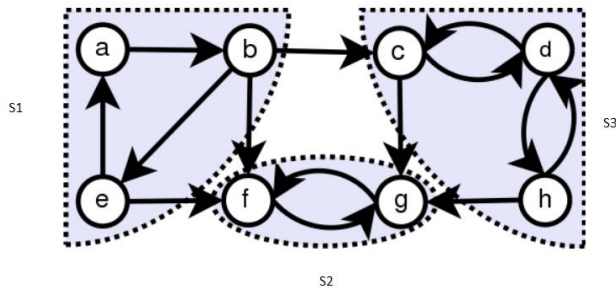
→ If no path between U_i & U_j consider -1 as shortest path

Note: If length of shortest path is low → chance of following each other increases

CONNECTED - COMPONENTS

STRONGLY CONNECTED COMPONENT

A directed graph is called strongly connected if there is a path in each direction between each pair of vertices of the graph. That is, a path exists from the first vertex in the pair to the second, and another path exists from the second vertex to the first. In a directed graph G that may not itself be strongly connected, a pair of vertices u and v are said to be strongly connected to each other if there is a path in each direction between them.



In each of subset we can reach to any of the node following given direction
Direction plays important role in strongly connected component
If we combine S1 and S2 we cannot reach a, b, e from f or g

WEAKLY CONNECTED COMPONENT

In mentioned figure if we remove direction then each of the components or node connected via some path

Why weakly connected components are useful?

- 1) As we explained above S1 and S2 cannot be strongly connected component but if we remove directions S1 and S2 become weakly connected component
- 2) Weakly connected component represents community like college, work
- 3) Having single connection between S1 & S2 bring both S1 and S2 in single set in weakly connected component

ADAR INDEX

This is specially designed for predicting link in a social network

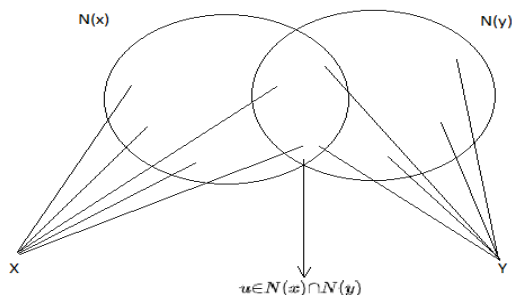
$$A(x, y) = \sum_{u \in N(x) \cap N(y)} \frac{1}{\log |N(u)|}$$

$N(u)$ is the set of nodes adjacent to User $\{u\}$. Such definition is based on the concept that common elements with very large neighbourhoods are lesser significant when predicting a connection between two nodes compared with elements shared between a small number of nodes

For example, $N(x) = \{U1, U2, U3\}$ & $N(y) = \{U4, U5, U6\}$

Any vertex or user connected to Node x ($N(x)$)

Any vertex or user connected to Node y ($N(y)$)



$$u \in N(x) \cap N(y)$$

Here U can also have a neighbourhood let's say U have a large neighbourhood then U might be a celebrity.

If X & Y following a celebrity that doesn't mean that X & Y should follow each other

In case of celebrity $\rightarrow N(U)$ increases $\rightarrow A(X, Y)$ decreases

User with small neighbourhood $\rightarrow N(U)$ decreases $\rightarrow A(X, Y)$ increases

KATZ CENTRALITY

Reference → https://en.wikipedia.org/wiki/Katz_centrality

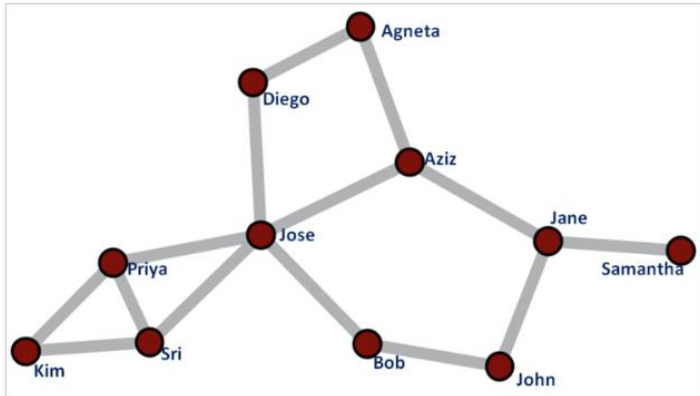
Katz centrality computes the relative influence of a node within a network by measuring the number of the immediate neighbours (first degree nodes) and also all other nodes in the network that connect to the node under consideration through these immediate neighbours.

Connections made with distant neighbours are, however, penalized by an attenuation factor α (alpha)

Similar to page rank and to the eigen vector centrality

To measure influence of one person or node just take influence of all the neighbour just like page rank

Now influence of neighbour can be measure by neighbour's neighbour & it will go on.



For example, in the figure on the right, assume that John's centrality is being measured and that α (alpha) = 0.5. The weight assigned to each link that connects John with his immediate neighbors Jane and Bob will be $\{(0.5)^1 = 0.5\}$. Since Jose connects to John indirectly through Bob, the weight assigned to this connection (composed of two links) will be $\{(0.5)^2 = 0.25\}$. Similarly, the weight assigned to the connection between Agneta and John through Aziz and Jane will be $\{(0.5)^3 = 0.125\}$ and the weight assigned to the connection between Agneta and John through Diego, Jose and Bob will be $\{(0.5)^4 = 0.0625\}$.

So, we will calculate katz centrality of each node.

HITS SCORE

Hyperlink Induced Topic Search

HITS also known as Hubs and authorities, It is link analysis algorithm that rates web pages

In early days of internet yahoo.com (hubs) → directory for other web pages

So there are two types of pages

- ➔ Hubs → site which have lots of outlinks
- ➔ Authorities → page from cnn.com/politics, bbc.com, mit.edu → have lots of inlinks

It gives every webpage W_i → 2 score hubs and authority score

For each user

- ➔ Celebrity => authority high => High inlink
- ➔ Normal User => Outlink => hub

Steps follows for Hubs and authority score

- 1) Initialize → To begin ranking $auth(p) = 1$ & $hub(p) = 1$
- 2) Update Authority rule →

$$auth(p) = \sum_{q \in P_{to}} hub(q) \text{ where } P_{to} \text{ is all pages which link to page } p.$$

Yahoo has high hub score so all pages cnn.com connected to we sum up hub score of them

- 3) Update Hub rule

$$hub(p) = \sum_{q \in P_{from}} auth(q) \text{ where } P_{from} \text{ is all pages which page } p \text{ links to.}$$

Example-> sum of auth of all pages which yahoo link out

- 4) Normalize

If we keep on updating Hub & Authority rule, it will lead to infinity

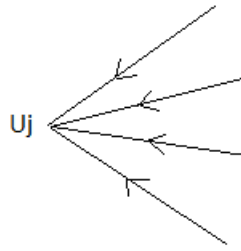
So, we need to do Normalization

WEIGHT FEATURES

Core idea →



1 million
 $U_i \Rightarrow$ Celebrity



30
 $U_j \Rightarrow$ Normal User

Possibility that random person know another random person for U_i followers → VERY LESS

Possibility that random person knows another random person for U_j followers → HIGH

So, if we somehow find out so that somebody is celebrity or not using just by number of vertex linking into it.

$$W(U_i) = \frac{1}{\sqrt{1+1 \text{ million}}} < W(U_j) = \frac{1}{\sqrt{1+30}}$$

Now if we see $W(U_i)$ is less compare to $W(U_j)$ so we have less weight for celebrity

Above explained are inweight feature and outweight feature

So, on the basis of weight features we added following features

- Weight of incoming edges
- Weight of outgoing edges
- Weight of incoming edges + weight of outgoing edges
- Weight of incoming edges + 2 * weight of outgoing edges
- 2 * Weight of incoming edges + weight of outgoing edges
- Weight of incoming edges * weight of outgoing edges

References links: →

<https://www.oreilly.com/library/view/feature-engineering-for/9781491953235/ch01.html>

<https://medium.com/basecs/a-gentle-introduction-to-graph-theory-77969829ead8>

<https://en.wikipedia.org/wiki/>

<http://www.datascience-pm.com/microsoft-team-data-science-process/>

Reference Book →

Feature Engineering made easy

Feature Engineering for machine learning

References Courses: →

Applied AI

About Me : → I am excited about opportunities for applying my machine learning and deep learning knowledge to real-world problems. **Currently, I am looking for opportunities either full-time or freelance projects, in the field of Machine Learning and**

Deep Learning. Feel free to contact me on [Linkdin](#) or you can reach me through [email](#) as well. I would love to discuss.

Do check out my other blogs [here](#)! & also on my [Github](#) profile