PyTorch Geometric Graph Embedding

Using SAGEConv in PyTorch Geometric module for embedding graphs

Anuradha Wickramarachchi · Follow
Published in Towards Data Science
5 min read · Sep 3, 2021

Graph representation learning/embedding is commonly the term used for the process where we transform a Graph data structure to a more structured vector form. This enables the downstream analysis by providing more manageable fixed-length vectors. Ideally, these vectors should incorporate both graph structure (topological) information apart from the node features. We use graph neural networks (CNN) to perform this transformation. To have a basic high-level idea about GNNs, you can take a peek at the following article.

What Can You Do With GNNs

Manipulation, Utility and Advantages of Graph Neural Networks

towardsdatascience.com

In this article, I will talk about the GraphSAGE architecture which is a variant of message passing neural networks (MPNN). MPNN is a fancy term for how GNNs are efficiently implemented.

Generalized GNN representation

Any MPNN can be formally represented using the two functions **aggregate** and **combine**.

$$a_v^{(k)} = AGGREGATE^{(k)}\left(\left\{h_u^{(k-1)} : u \in \mathcal{N}(v)\right\}\right)$$

Equation by Author Reference(https://arxiv.org/pdf/1810.00826.pdf)

The **aggregate** function governs how neighbour information is gathered or aggregated for a given node.

$$h_v^{(k)} = COMBINE^{(k)}\left(h_v^{(k-1)}, a_v^{(k)}\right)$$

Equation by Author Reference(https://arxiv.org/pdf/1810.00826.pdf)

The **combine** function governs how the information of a node itself is combined with the information from the neighbourhood.

GraphSAGE

GraphSAGE stands for Graph-SAmple-and-aggreGatE. Let's first define the aggregate and combine functions for GraphSAGE.

Combine — Use element-wise mean of features of neighbours

Aggregate — Concatenate aggregated features with current node features

Graphical Explanation

GraphSAGE layers can be visually represented as follows. For a given node v, we aggregate all neighbours using mean aggregation. The result is concatenated with the node v's features and fed through a multi-layer perception (MLP) followed by a non-linearity like RELU.



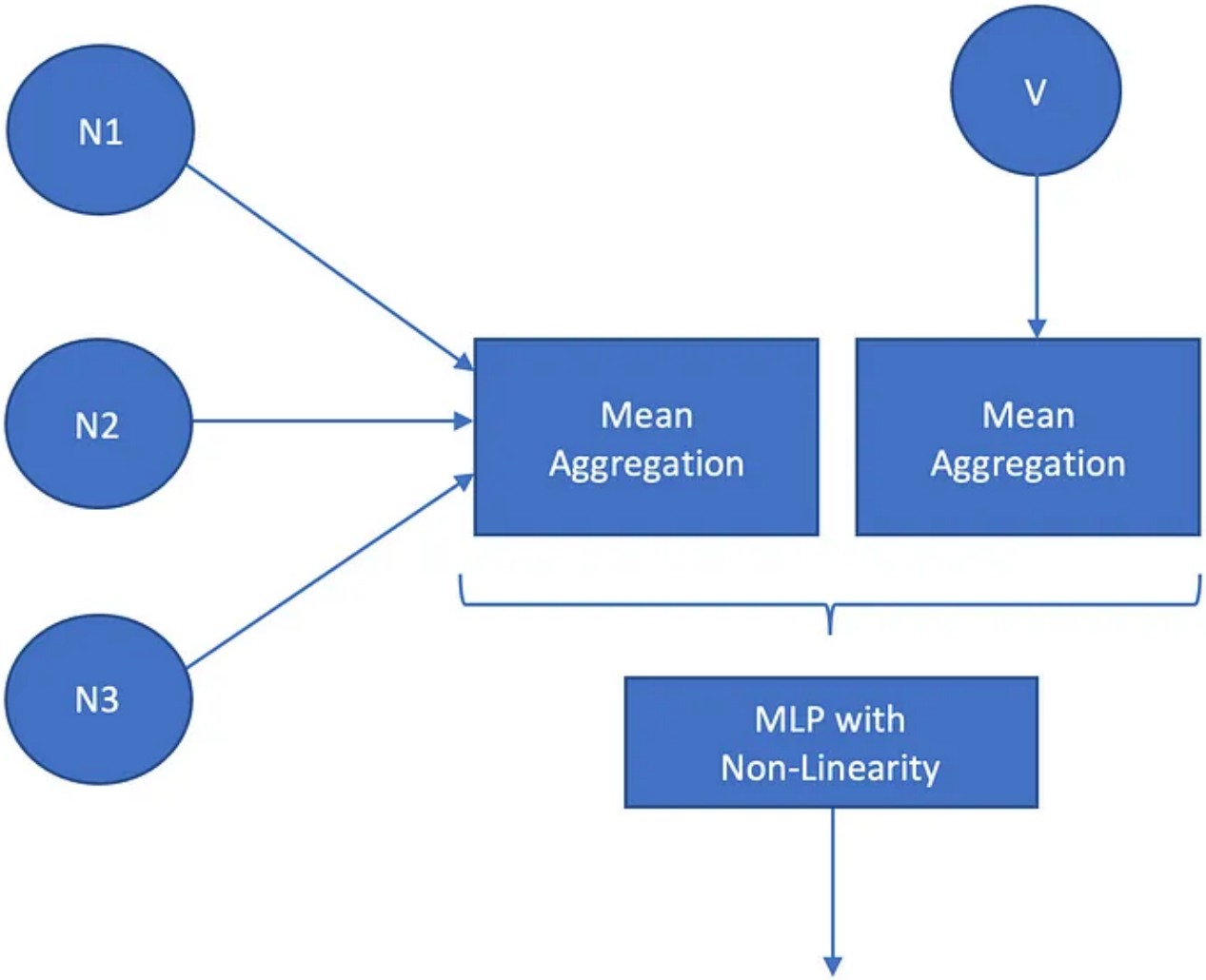



Image by Author

One can easily use a framework such as PyTorch geometric to use GraphSAGE. Before we go there let's build up a use case to proceed. One major importance of embedding a graph is visualization. Therefore, let's build a GNN with GraphSAGE to visualize Cora dataset. Note that here I am using the provided example in PyTorch Geometric repository with few tricks.

GraphSAGE Specifics

The key idea of GraphSAGE is sampling strategy. This enables the architecture to scale to very large scale applications. The sampling implies that, at each layer, only up to **K** number of neighbours are used. As usual, we must use an order invariant aggregator such as Mean, Max, Min, etc.

Loss Function

In graph embedding, we operate in an unsupervised manner. Therefore, we use the graph topological structure to define the loss.

$$J_{\mathcal{G}}(\mathbf{z}_u) = -\log\left(\sigma(\mathbf{z}_u^\top \mathbf{z}_v)\right) - Q \cdot \mathbb{E}_{v_n \sim P_n(v)} \log\left(\sigma(-\mathbf{z}_u^\top \mathbf{z}_{v_n})\right)$$

From GraphSAGE paper: https://arxiv.org/pdf/1706.02216.pdf

Here **Zu** demonstrates the final layer output for node **u**. **Zvn** indicates the negative sampled node. In simple terms, the second term of the equation indicates that the negated dot product of negative (node **u** and any random node **v**) should be maximized. In other words, the cosine distance of random nodes should be further. First-term says otherwise for node **v**, which is a node that we need to be embedded closer to **u**. This **v** is called a positive node, which is typically obtained using a random walk starting from **u**. **Evn~Pn(v)** indicates that the negative nodes are taken from a negative sampling approach. In actual implementation, we take direct neighbours as positive samples and random nodes as negative samples.

Building a Graph Embedding Network

We can start by importing the following python modules.

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch_cluster import random_walk
from sklearn.linear_model import LogisticRegression

import torch_geometric.transforms as T
from torch_geometric.nn import SAGEConv
from torch_geometric.datasets import Planetoid
from torch_geometric.data import NeighborSampler as
                                    RawNeighborSampler

import umap
import matplotlib.pyplot as plt
import seaborn as sns
```

Initializing the Cora dataset;

```
dataset = 'Cora'
path = './data'
dataset = Planetoid(path, dataset, transform=T.NormalizeFeatures())
data = dataset[0]
```

Note that the dataset object is a list of subgraphs. In the case of Cora, we have one so we pick the graph in index 0.

Sampler components (here we extend NeighborSampler class's sample method to create batches with positive and negative samples);

```
# For each batch and the adjacency matrix
pos_batch = random_walk(row, col, batch,
                        walk_length=1,
                        coalesced=False)[:, 1]
# row are source nodes, col are target nodes from Adjacency matrix
# index 1 is taken as positive nodes

# Random targets from whole adjacency matrix
neg_batch = torch.randint(0, self.adj_t.size(1), (batch.numel(), ),
                                  dtype=torch.long)
```

GNN can be declared in PyTorch as follows;

```
class SAGE(nn.Module):
    def __init__(self, in_channels, hidden_channels, num_layers):
        super(SAGE, self).__init__()
        self.num_layers = num_layers
        self.convs = nn.ModuleList()

        for i in range(num_layers):
            in_channels = in_channels if i == 0 else hidden_channels
            self.convs.append(SAGEConv(in_channels,
                                       hidden_channels))

    def forward(self, x, adjs):
        for i, (edge_index, _, size) in enumerate(adjs):
            x_target = x[:size[1]]
            x = self.convs[i]((x, x_target), edge_index)
            if i != self.num_layers - 1:
                x = x.relu()
                x = F.dropout(x, p=0.5, training=self.training)
        return x

    def full_forward(self, x, edge_index):
        for i, conv in enumerate(self.convs):
            x = conv(x, edge_index)
            if i != self.num_layers - 1:
                x = x.relu()
                x = F.dropout(x, p=0.5, training=self.training)
        return x
```

Note that we use SAGEConv layers from PyTorch Geometric framework. In the forward pass, the NeighborSampler provides us with data to be passed over in each layer as data indices. This is a rather complex module so I suggest readers read the Minibatch Algorithm from paper(page 12) and the NeighborSampler module docs from PyTorch Geometric.

Visualization

Without using graph structure, the following is the UMAP plot.

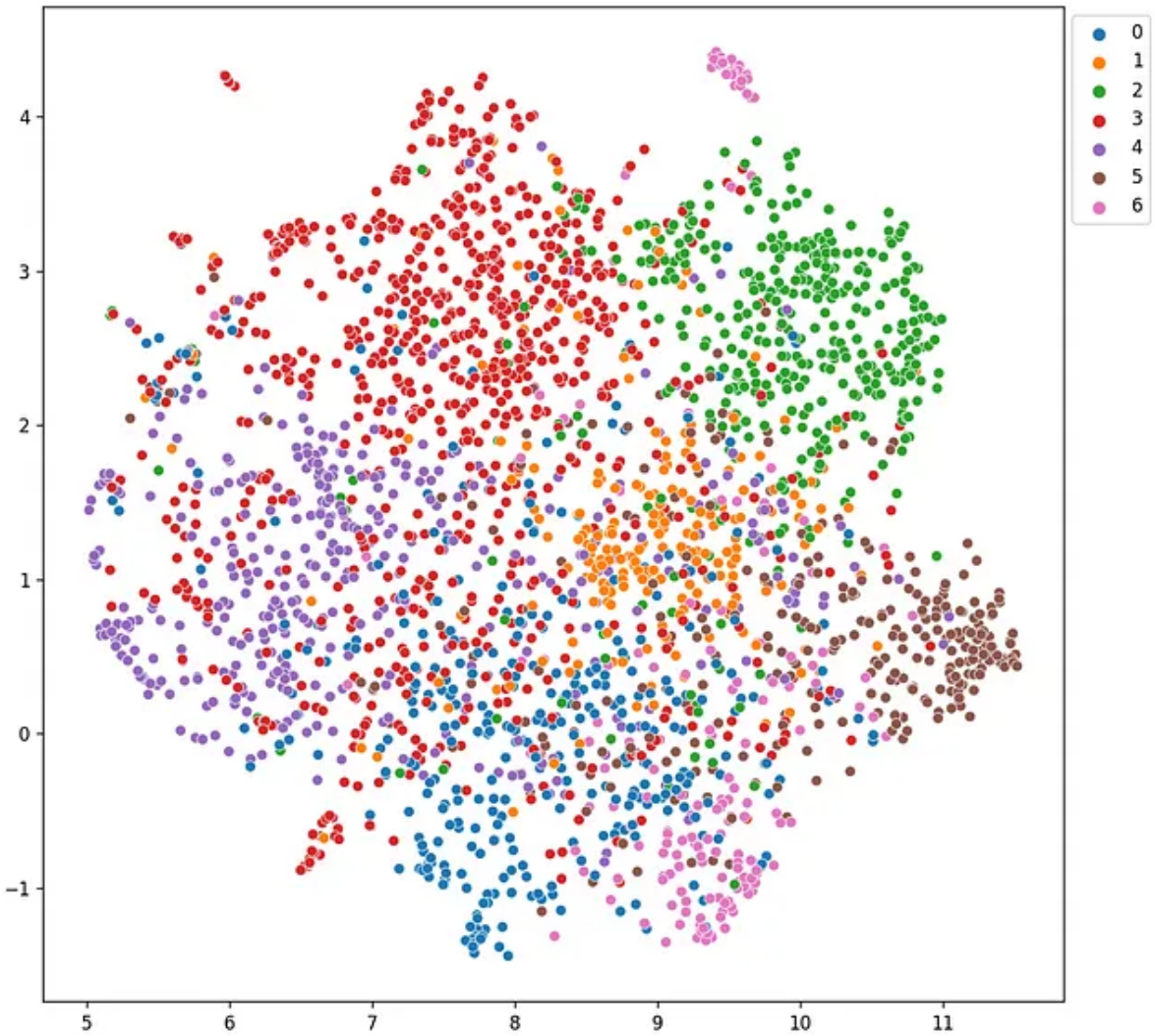


Image by Author

When we embed using GraphSAGE we can have a better embedding as follows;

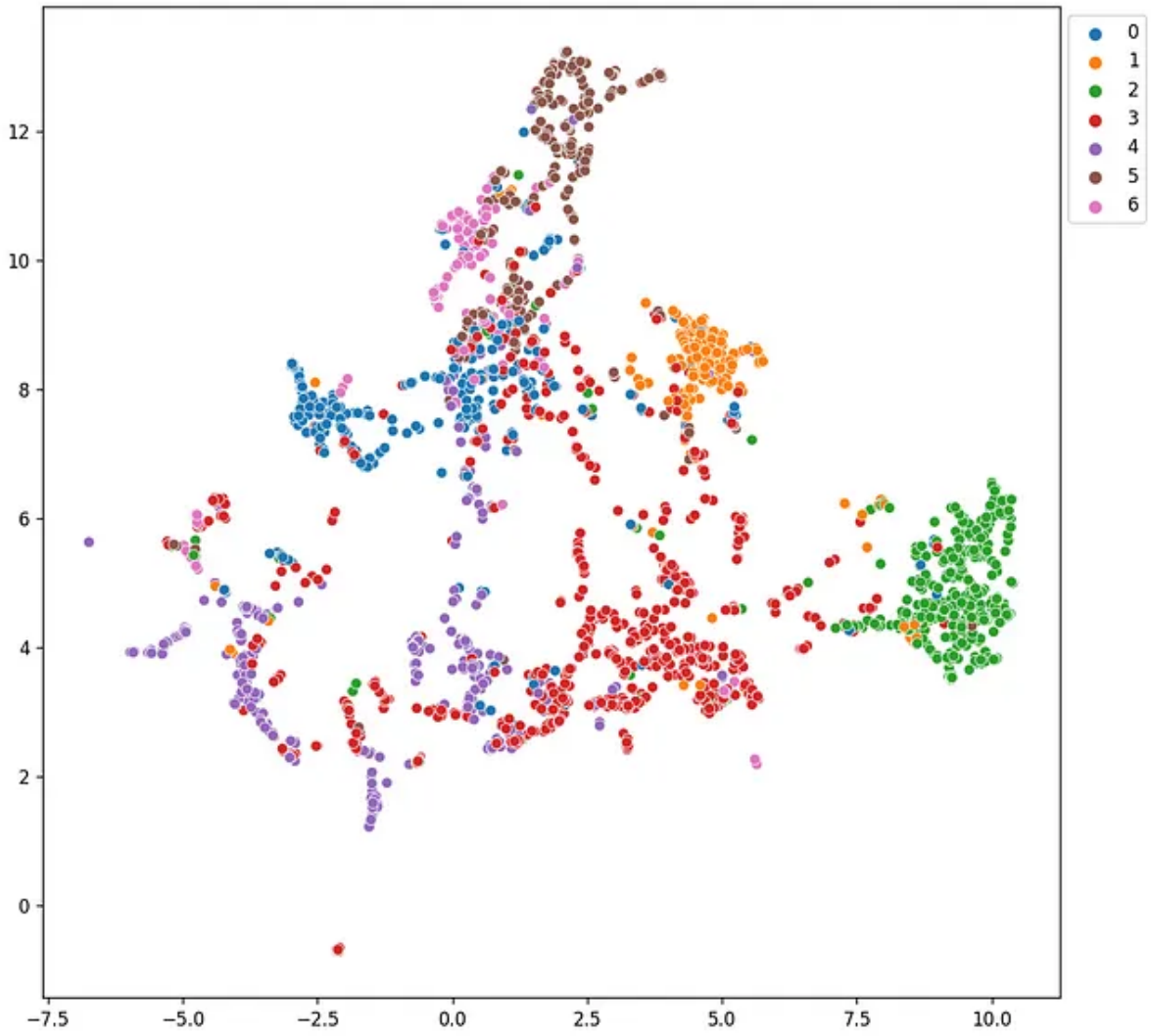


Image by Author

We can see that the embeddings are much better and well separated compared to naive UMAP embedding. However, this is not perfect and needs more work. But I hope this is a good enough demonstration to convey the idea. 😊

Hope you enjoyed this article!

The complete code and Jupyter Notebook is available here.

Machine Learning | Data Science | Pytorch | Graph Neural Networks | Graphsage

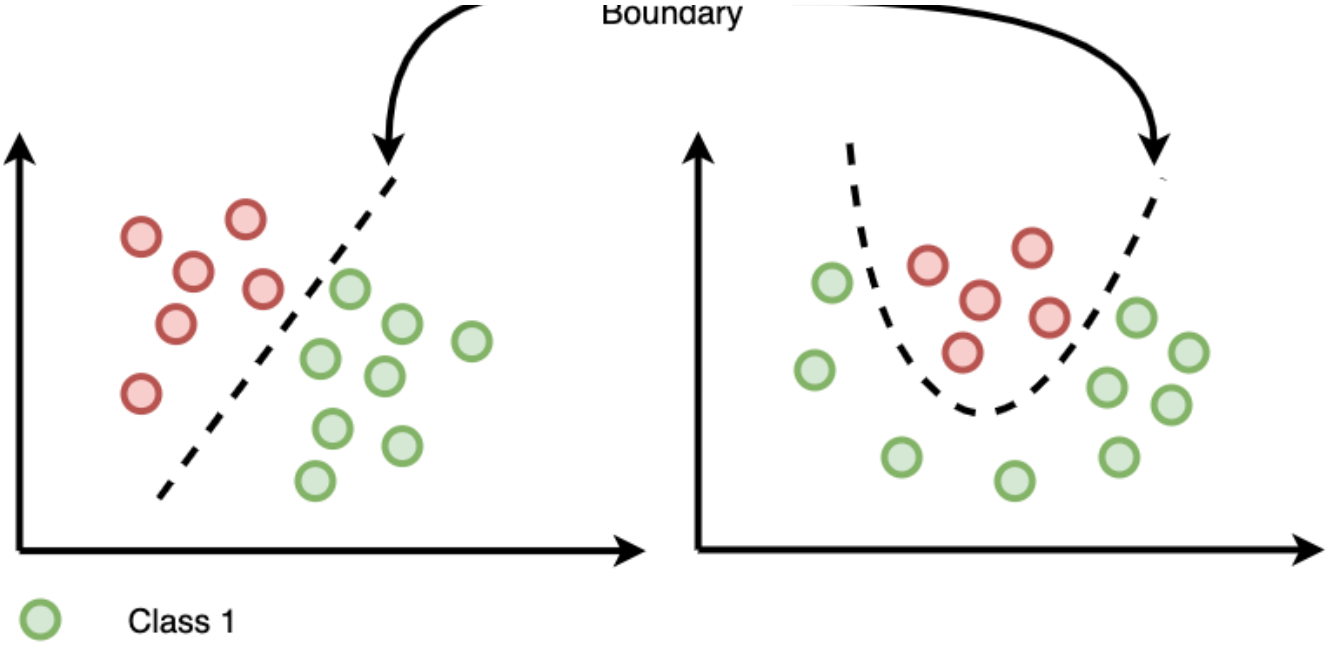
Follow

Written by Anuradha Wickramarachchi

2K Followers · Writer for Towards Data Science

https://anuradhawick.com/

More from Anuradha Wickramarachchi and Towards Data Science



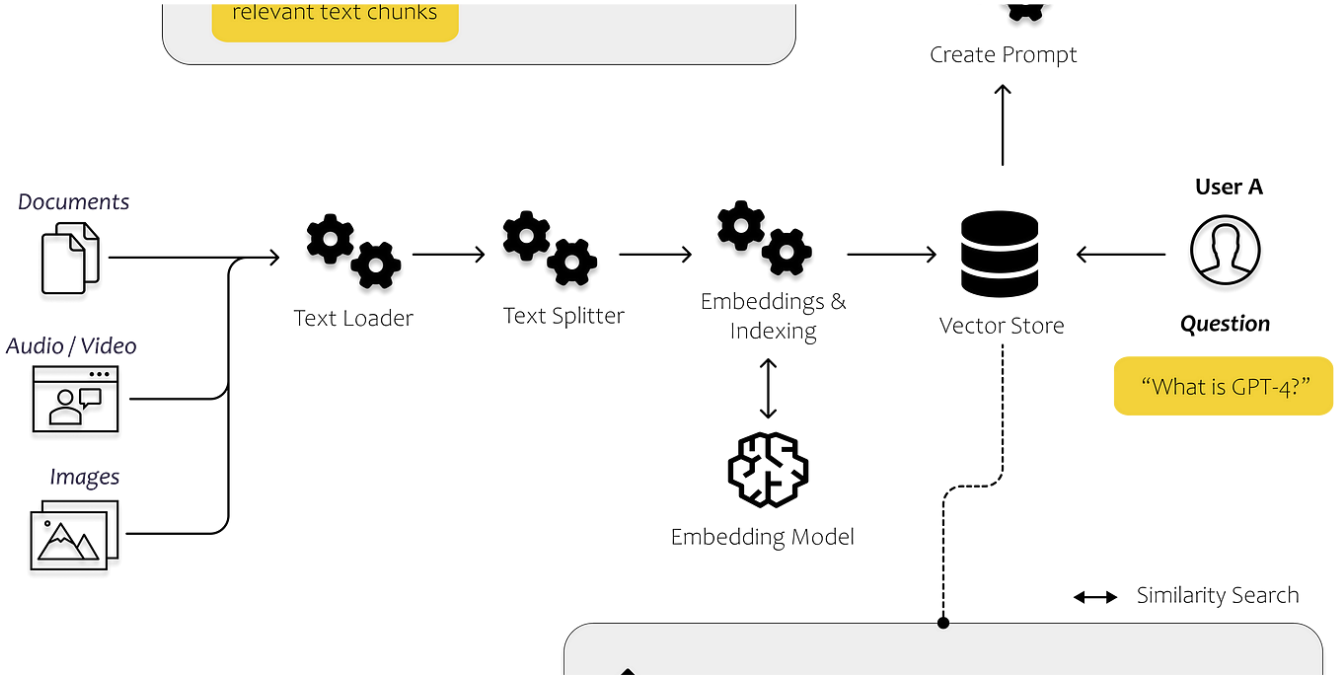
Anuradha Wickramarachchi in Towards Data Science

Logistic Regression and Decision Boundary

Understanding logistic regression and its utility in classification

6 min read · May 19, 2020

169 2



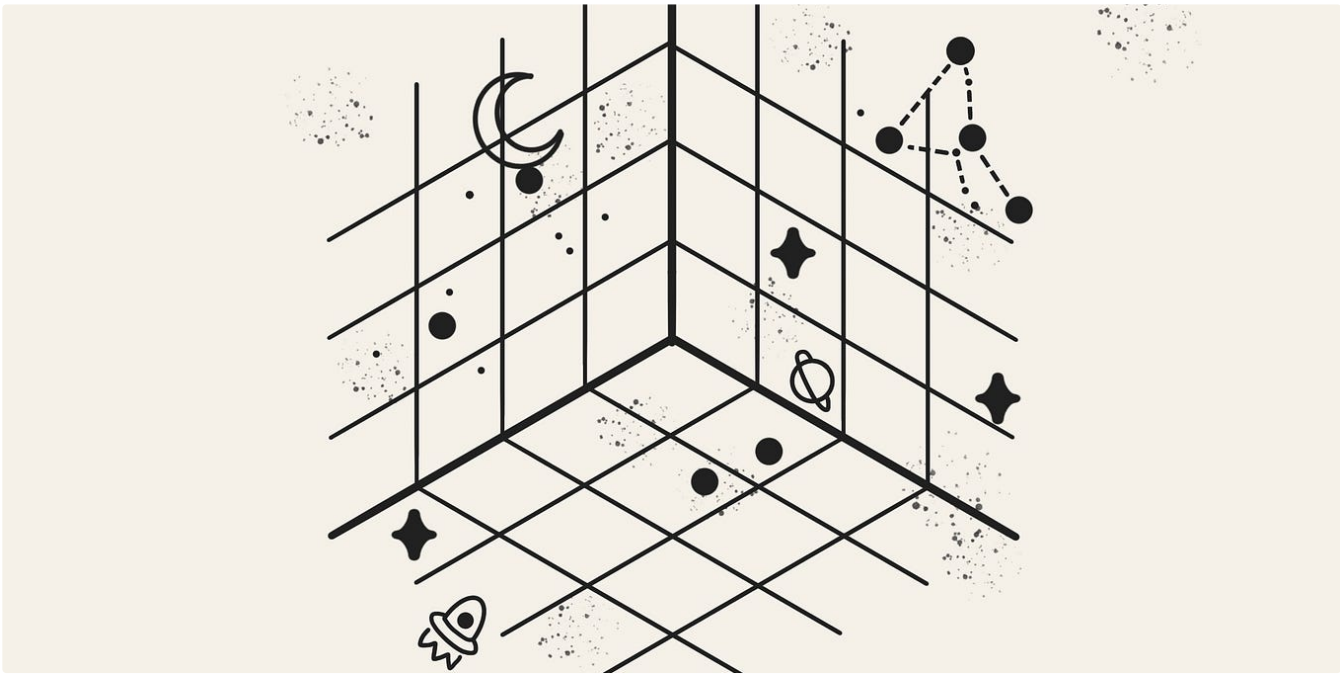
Dominik Polzer in Towards Data Science

All You Need to Know to Build Your First LLM App

A step-by-step tutorial to document loaders, embeddings, vector stores and prompt templates

· 26 min read · Jun 22

3.3K 28



Leonie Monigatti in Towards Data Science

Explaining Vector Databases in 3 Levels of Difficulty

From noob to expert: Demystifying vector databases across different backgrounds

· 8 min read · Jul 4

1.7K 18

Anuradha Wickramarachchi in Towards Data Science

Extract Health Data From Your Samsung

Exporting and interpreting health data from Samsung Health

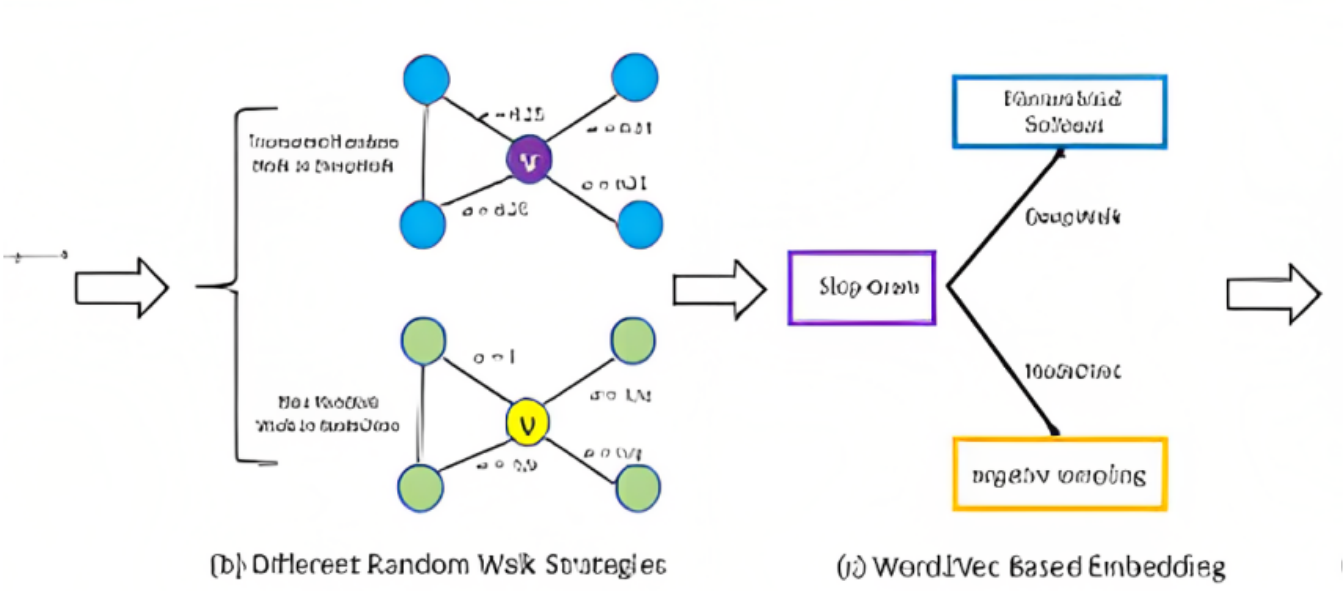
5 min read · Aug 22, 2020

121

See all from Anuradha Wickramarachchi

See all from Towards Data Science

Recommended from Medium



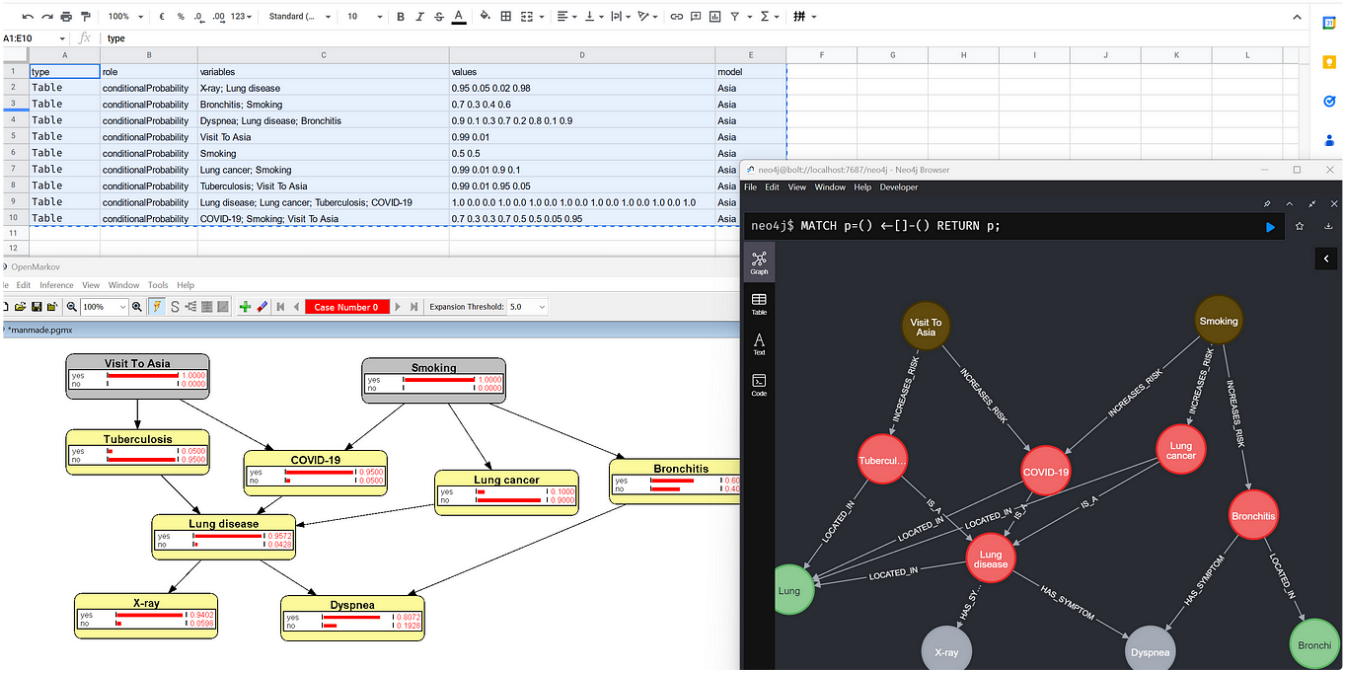
Tejpal Kumawat

Deep Walk and Node2Vec: Graph Embeddings

Investigating Node2Vec and DeepWalk to extract embeddings from graphs

6 min read · Mar 14

10



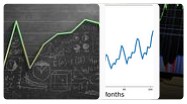
Sixing Huang in Geek Culture

How to Build a Bayesian Knowledge Graph

Store data in Google Sheets, visualize the knowledge graph in Neo4j, and do Bayesian reasoning with OpenMarkov

· 10 min read · Feb 5

Lists



Predictive Modeling w/ Python

18 stories · 169 saves

Practical Guides to Machine Learning

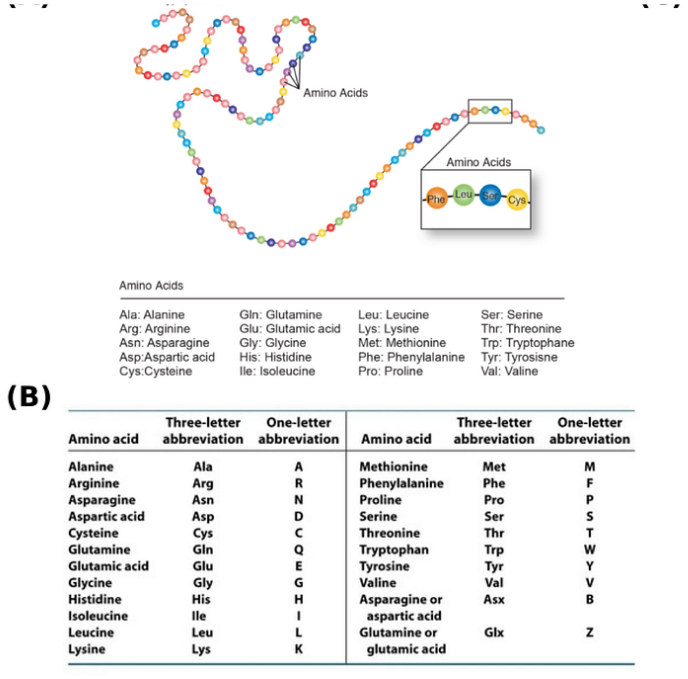
10 stories · 187 saves

Natural Language Processing

438 stories · 80 saves

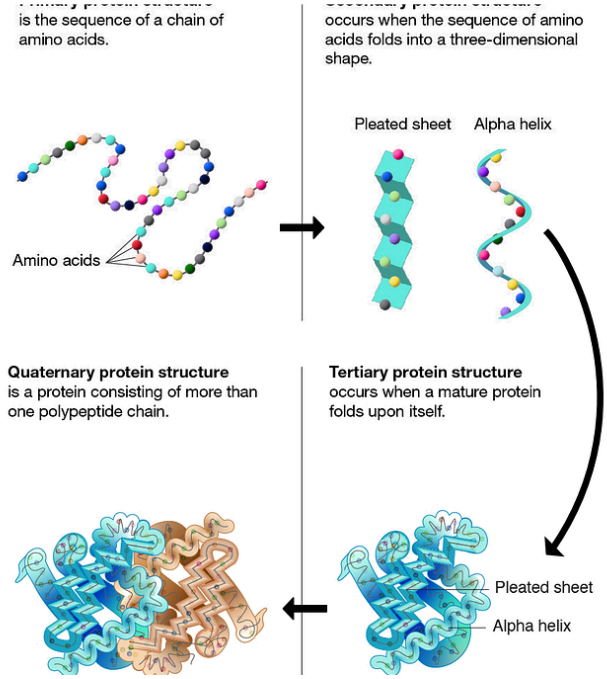
New_Reading_List

174 stories · 37 saves



(B)

Amino acid	Three-letter abbreviation	One-letter abbreviation	Amino acid	Three-letter abbreviation	One-letter abbreviation
Alanine	Ala	A	Methionine	Met	M
Arginine	Arg	R	Phenylalanine	Phe	F
Asparagine	Asn	N	Proline	Pro	P
Aspartic acid	Asp	D	Serine	Ser	S
Cysteine	Cys	C	Threonine	Thr	T
Glutamine	Gln	Q	Tryptophan	Trp	W
Glutamic acid	Glu	E	Tyrosine	Tyr	Y
Glycine	Gly	G	Valine	Val	V
Histidine	His	H	Asparagine or aspartic acid	Asx	B
Isoleucine	Ile	I			
Leucine	Leu	L	Glutamine or glutamic acid	Glx	Z
Lysine	Lys	K			

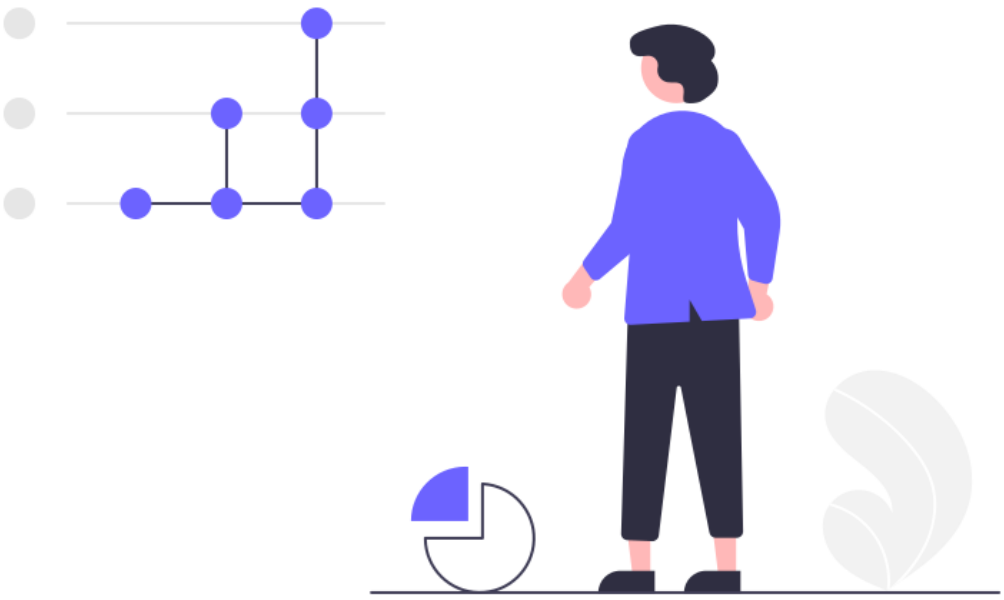


A Aviv Korman in Stanford CS224W GraphML Tutorials

Prediction of Protein Movement Upon Ligand Binding Using Equivariant Graph Neural Networks

Harnessing Graphein to transform protein structures into PyTorch-Geometric-ready residue-level spatial graphs and deploying equivariant...

12 min read · May 14



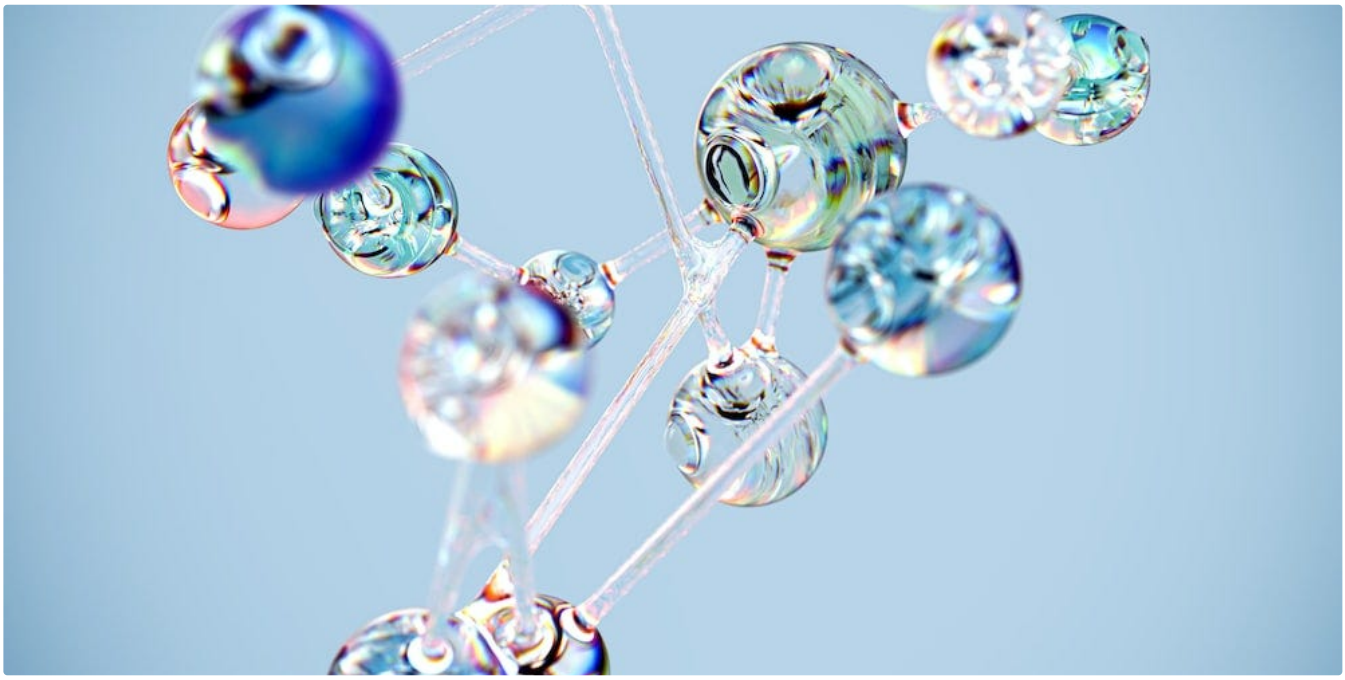
E Ravenspike in Dev Genius

Analyzing graph networks: Utilizing advanced methods

Story overview

· 16 min read · Jun 5

29



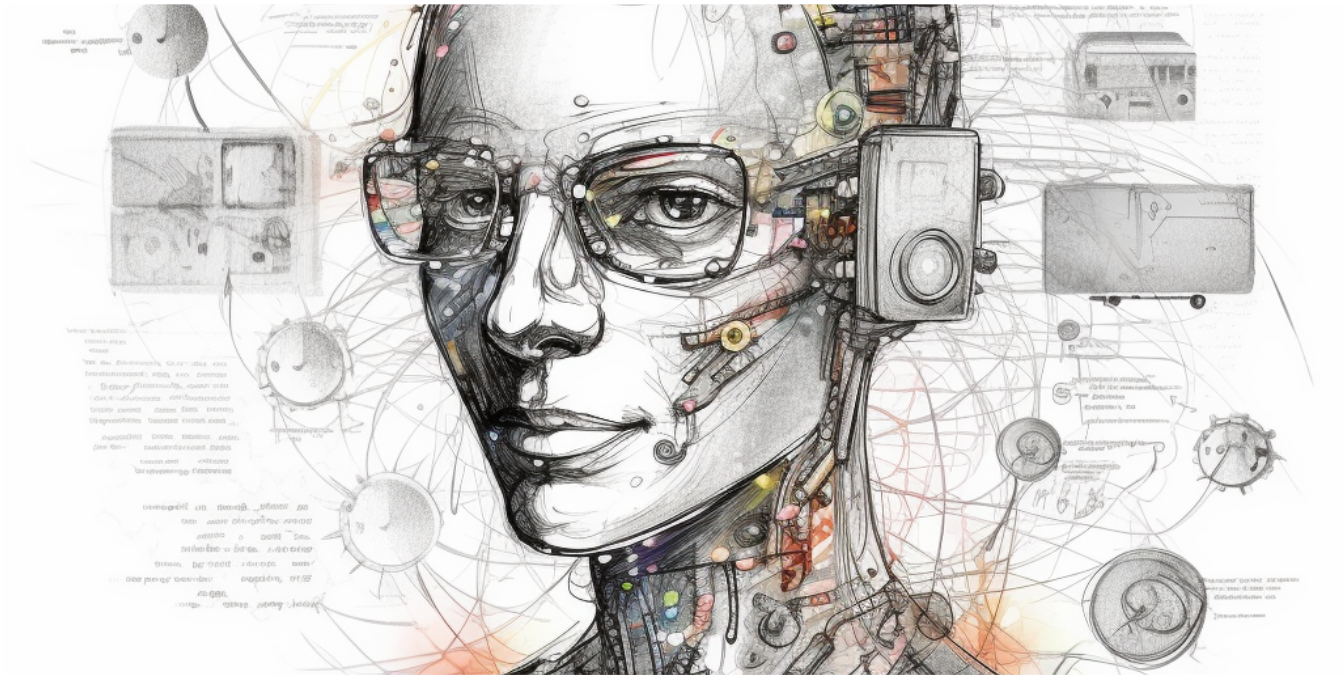
Matthew Mo

Graph Isomorphism Neural Networks—Drug Discovery Example

Molecular properties regressive modelling using Graph Isomorphism Networks

6 min read · Feb 27

14



Tomaz Bratanic in Neo4j Developer Blog

Knowledge Graphs & LLMs: Fine-Tuning Vs. Retrieval-Augmented Generation

What are the limitations of LLMs, and how to overcome them

12 min read · Jun 6