Medical Diagnosis using Knowledge Graphs and Deep Learning

Anmol Goel @ WellOwise, IIIT Delhi | Dr. Saher Mehdi

1. Problem Statement

Online symptom checkers have significant potential to improve patient care and bringing personalised medicine to a large population. Research and development of a robust triage and disease diagnostic system that could compare favourably with human doctors with respect to diagnostic accuracy is imperative.

- ➔ Reliable
- ➔ Simple
- ➔ Accurate

Project Overview

- ➔ Research of existing diagnosis methods
- ➔ Knowledge Graphs
- ➔ Node2vec algorithm
- ➔ API & frontend development
- ➔ Deployment

Previous Work

Bayesian Networks
Random Forests

Tip

There are many examples of a symptom checker service which have been deployed online with success. Some of these include http://symcat.com, https://webmd.com, etc.

2. Knowledge Graphs

➔ **What**

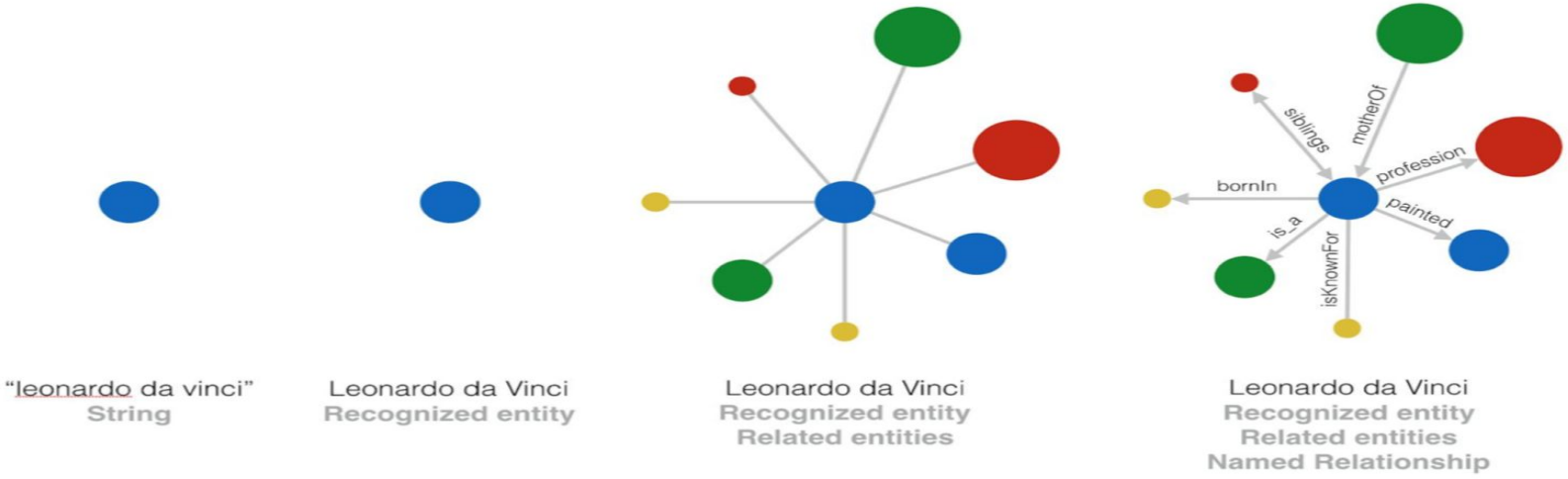
a knowledge graph acquires and integrates information into an ontology and applies a reasoner to derive new knowledge.

➔ **How**

Can be built on graph databases like Neo4j.

Knowledge Graphs intuition

Knowledge Graphs follow a simple principle: organizing information in a structured way by explicitly describing the relations among entities.



"leonardo da vinci"
String

Leonardo da Vinci
Recognized entity

Leonardo da Vinci
Recognized entity
Related entities

Leonardo da Vinci
Recognized entity
Related entities
Named Relationship

Medical Knowledge Base

This is an example of the knowledge graph we have built for this project. It encodes relationships between a) symptom-symptom and b) symptom-disease. The edge weights are assigned using a value of connectivity between 2 nodes (from the scraped data).

For illustration purposes only


```python
from py2neo import Graph

graph = Graph("bolt://localhost:7687", auth=("id", "pwd"))

def simSym(sym):
    query = f"""
    MATCH (sym:BaseSym {{name: '{sym}'}})-[r:SIMILAR_TO]-(target)
    RETURN target.name AS name, toInteger(r.score) AS weight
    ORDER BY toInteger(r.score) DESC
    """
    print(query)
    res = graph.run(query).data()
```

Node2vec algorithm

Node embeddings are a recent technique aimed at finding d-dimensional embeddings of nodes in a graph such that similar nodes in the graph have embeddings that are close together.

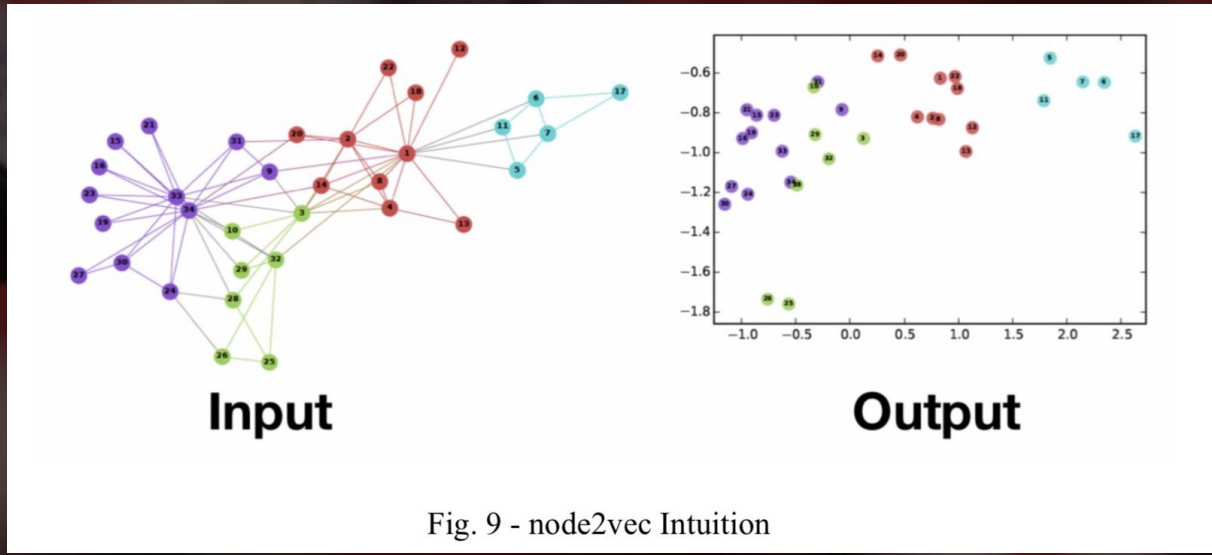
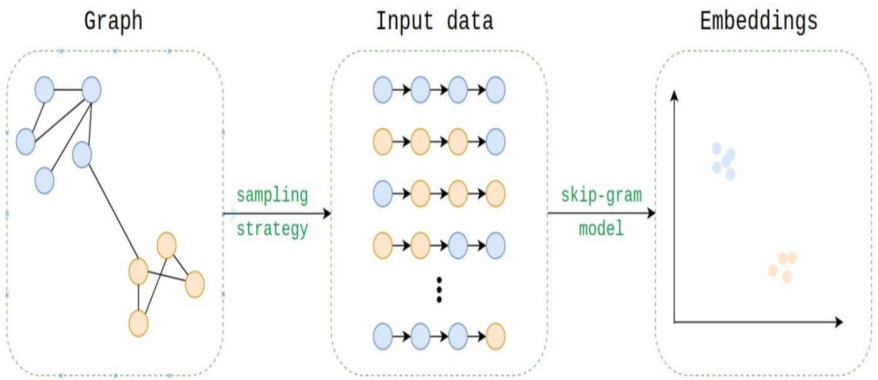
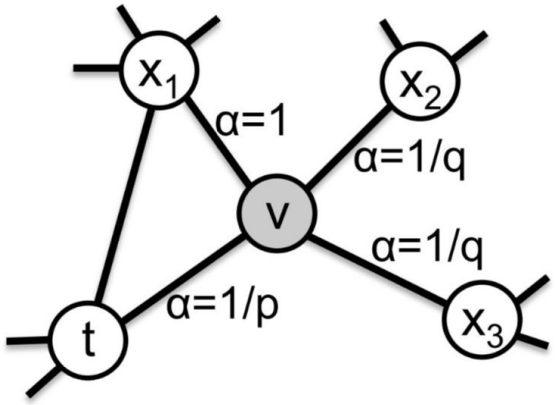


Fig. 9 - node2vec Intuition



Node2vec Sampling Strategy

Sampling strategy involves a number of hyperparameters which includes number of walks, length of each walk, window size, number of iterations.



$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

Random Walk

Random walks are generated from each node of the graph which generates the required corpus on which the algorithm can perform embedding retrieval.

In a nutshell

- ➔ Whenever a new user enters their symptoms, a new node is created in the graph.
- ➔ Root node is the user while the connected nodes are symptoms entered by the user.
- ➔ Node2vec is applied on this new graph. These embeddings contain hidden semantic relationships between nodes.
- ➔ Using the embeddings, we obtain top-k similar nodes to the user node using cosine similarity.
- ➔ After returning results to the user, the user node and the graph embeddings are deleted from the server.

API and Frontend

API is developed using Flask framework in Python 3.7

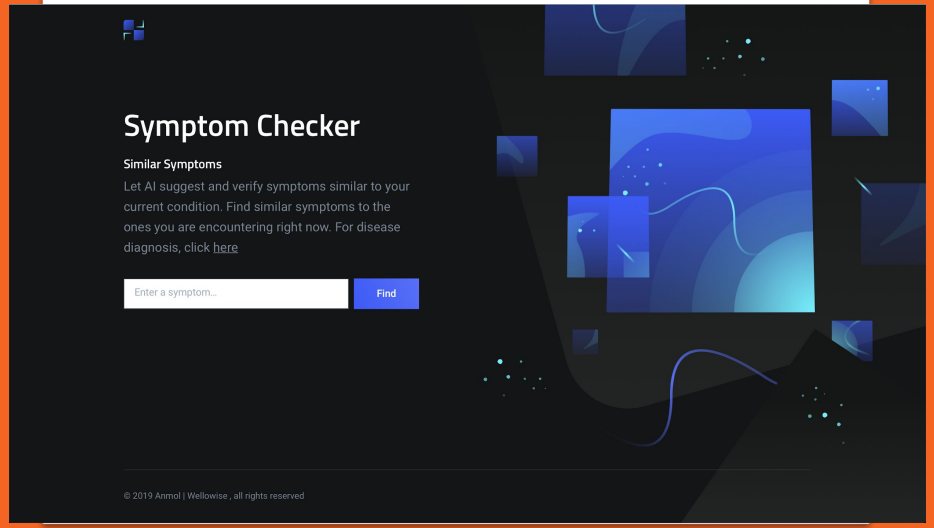
It has 2 endpoints -

a) similar symptom

b) diagnose

Frontend

Frontend is constructed using HTML, CSS and JavaScript. It fetches required information from the backend server by making REST API calls



Deployment

API and frontend is deployed using AWS EC2 instances. The application is served using Docker containers and automated deployment is done using Jenkins.