Graph Neural Network Case Study Walkthrough

SUGGESTING MOVIES TO VIEWERS BASED ON MOVIES THEY HAVE RATED

Introduction:

OTT platforms have become a major part of our lives, especially in the pandemic. Most of us have switched to digital platforms for entertainment. Here we have a case study that tries to solve the problem of suggesting movies to users of these OTT platforms based on what movies they have watched and how much they have rated. Users tend to like it more when their suggestions are more personalized to their taste. So let's take a look at how we achieve this using Graph Neural Networks (GNNs).

Topics:

1) Datasets
2) Creating our Graph Network
3) Traversing through our Graph
4) Creating the data for our Neural Network
5) Working of our Neural Network

Dataset:

So we have been given two data sets to work with here.

- One of them is the movies data set where we have the name of each movie and their unique ID and their Genre.
- The other data set contains the users and what movies they have watched and hence rated.
- First, we look at the "ratings" data set and we decide on a threshold for the ratings given to them by a certain user. We decide the threshold to be 5. So every movie rated below 5 will not be considered for our activity.

Creating our Graph Network:

- We define two dictionaries, item_frequency and pair_frequency. Then we group the "ratings" data set based on what movies have been watched by which user. Suppose user 1 has seen movie_1, movie_3,

movie_6, movie_47, and movie_50. And let's suppose user 2 has watched movie_1, movie_2, and movie_3. So the groups would be (movie_1, movie_3, movie_6, movie_47, movie_50), (movie_1, movie_2, movie_3) and so on.

- So each group represents the movies watched by each user that they rated over 5. Now coming to the item_frequency and pair_frequency, item_frequency represents how many users have watched that movie. The dictionary pair_frequency describes how many times, a certain pair of movies have appeared in the same group.

- We have used a "tqdm()" function with the for loops because we want to see the progress of iterations.

- Then we want to create a graph where the nodes would be the movies. Now to create edges, we will look at every pair of movies and we are looking at the product of their PMI index and their pairing frequency. If this product is greater than the minimum threshold of weight that we have assigned, then we create an edge between those two movies. Now, why are we creating this graph? We are trying to model what movies are frequently watched together based on all of the user data. To think of this more intuitively, the higher the weight of an edge between two movies A and B, the higher is the probability of movie B being suggested after you have watched movie A and vice versa.

Traversing through our Graph:

- Now that we have the graph, we build a function "next_step" that does the simple operation of traveling to the next node given you're currently on a node i.e. when you have watched a movie, what are the next movies you could consider.

- Since each node in the graph is likely to have more than one neighbor (judging from the average degree which was 57), we have to take a probabilistic approach. In other words, since we have more than one option for the next step, we assign probabilities to each edge arising from our current node/movie and then we make our random choice based on those probabilities.

- Now here we have two hyperparameters, p, and q, through which we can modify the probabilities a little. Now our neighbors can have an edge with a movie we have already watched (i.e previous). In that case, we would want to scale down the probability of re-visiting that node. If a neighbor has edges with the previous movie besides having an edge with the current movie, we will strongly want to visit that node. If a neighbor has an edge with the current node but not with the previous node, we want to keep their probability midway between the earlier two cases. So by adjusting the value of p and q, we can control the probabilities of these scenarios.

- Then we have a function called "random_walk". This function takes five arguments, namely graph, num_walks, num_steps, p, and q. We have discussed what p and q do and we will touch on this later, why we are calling these two hyperparameters in this function once again. The graph argument accepts the graph of movies that we have created. Now to explain what purpose the other two arguments serve, we need to look at what the function does altogether. This function extracts a certain number of random walks within the graph, starting from a random node. Now the argument 'num_steps' defines the number of steps in each one of those random walks and the number of such walks has been defined by the 'num_walks' argument. The "random_walk" function calls the "next_step" function

which helps choose the next step during the particular random walk. The p and q we have used as arguments in the “random_walk” function is used as arguments in the “next_step” function.

- So the output of this function would be an array of the nature:
  [[movie_1, movie_2, movie_67,movie_400,……. movie_90],

  [movie_3, movie_1, movie_60, movie _1000, …….. movie_2]
  .
  .
  .
  ]

Creating the data for our Neural Network:

- Then we have the “generate_exampes” function. Inside this function, we iterate through all our random walks and apply the skipgram function from keras.preprocessing library. Let’s throw a little light on how the skipgram function works.

- Suppose there is a sentence “I love to play cricket”. Now we apply skipgram function on this. It would return us some samples in the format
  { word_1 from the sentence, word_2} -> label. The word_1 is a random word from the sentence and the word_2 can be any random word from the total vocabulary. Now the value of the label becomes ‘1’, if the word_2 which has been randomly chosen from the vocabulary, happens to be a part of the input sentence as well. Otherwise, the value of the label becomes ‘0’. Here are some examples that can help explain it better.
  Input Sentence: ‘I love to play cricket’.
  Examples of output samples:
  { I, play} -> 1
  (since both words are in input sentence)
  {I, cricket} -> 1
  {love, play}-> 1
  {love, football} -> 0 (since football doesn’t exist in the input sentence)

- So instead of sentences, we are using our walks that we generated using the “random_walks” function. So, when we run skipgram over these random walks, we get tuples of movies and a corresponding label. If both of those movies have been a part of the random walk, they are labeled as ‘1’, otherwise ‘0’. And we have a dictionary called example_weights where we count the number of occurrences of those particular two movies.

Working of our Neural Network:

- The output of the "generate_examples" function becomes the frame of data on which we are going to train our neural network model. The first movie in the tuple is now called the context movie, the second movie is called target and our corresponding outputs are the labels. So, our neural network model takes

two inputs target and context. Then it converts them to their respective embeddings through its embedding layer.

- Embedding is a process in which we try to convert a word into a vector of a limited number of dimensions. The intuition behind embedding is that similar words will have similar embeddings. So if we take the dot product of two similar word's embeddings, it will give a higher value as opposed to that for two less similar words.

- Our Neural Network (as we can see in the create_model() function) has two layers. One of them is embedding- which takes the target and context words and converts them into target and context embeddings. Then we have a layer that takes the dot product of these two embeddings and gives either '1' or '0' as output.

- After this model is done training on the data we had generated through our "create_examples()" function, we extract the embedding layer of our model which is the only matrics we will be needing for our use case.

- So now that we have got the embedding layer after training, how do we use it? We take a set of movies and call them query movies. Our objective now is to find out the top 5 recommendations for each movie. So we convert each query to query embedding and then try to find out the top 5 similar movies from the embedding matrics that we extracted from the neural network after the training. Those 5 similar movies can be suggested to the user.

Additional Reading Materials

- For Word to Vector Embedding: https://jalammar.github.io/illustrated-word2vec/

- A comprehensive research paper on node embedding with implementation code: https://paperswithcode.com/task/graph-embedding