Graph Gurus 32 demo instructions

This exercise from Graph Gurus 32 demonstrates the k-Nearest Neighbor (kNN) algorithm for classifying nodes. Given a dataset where some of the vertices are classified (labeled), it predicts the class or classification of an unlabeled vertex by comparing it to those that are labeled. It can be considered a meta-algorithm, because it requires that the similarity between vertices has already been determined. It can also be used as a supervised machine learning technique, to find the optimal value of k.

These instructions should be distributed along with a GraphStudio solution file (export_GG32_classification.tar.gz) and data folder containing 4 files. Optionally, it may include several queries; the queries are included in the solution file.

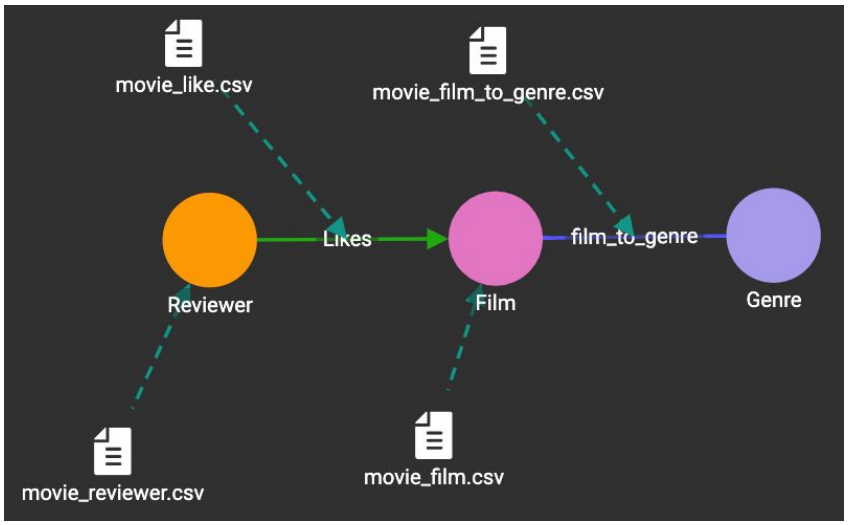
Schema

The schema is a simple movie recommendation network with 3 vertex types (Reviewer, Film, and Genre) and two edge types to connect them. The edges connecting Reviewers to Films are weighted, representing how much the Reviewer likes the Film.

Dataset

The data is a small synthetic graph with only a handful of instances:

- 8 Film vertices
- 6 Reviewer vertices
- 10 Genre vertices
- 15 Like edges
- 16 movie_to_genre edges



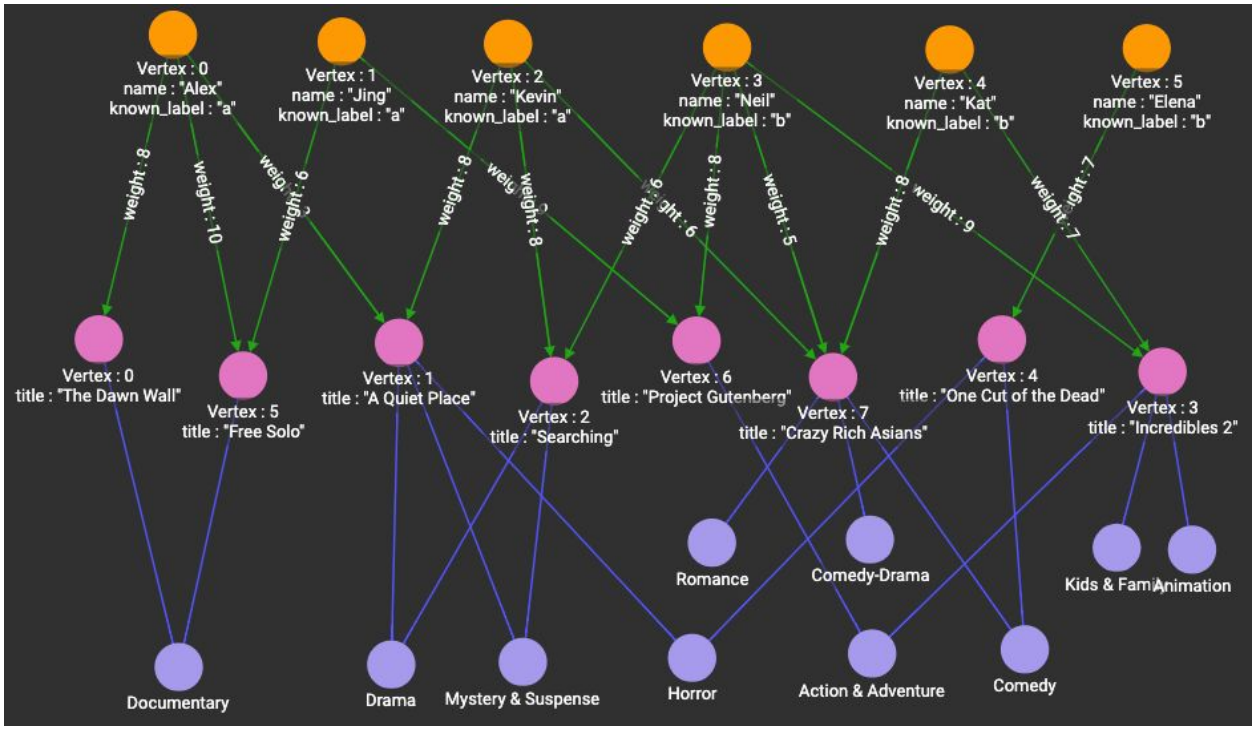
The graph data comes from four files, as shown above. The genres are all mentioned in the movie_film_to_genre.csv file. The genres do not have any attributes, so it is no necessary for them to have their own file.

Install the solution file called export_GG32_classification.tar.gz, which bundles the schema, loading jobs, and queries used in this exercise. **Follow these steps:**

(1) Start on the main page of GraphStudio. Import the solution file export_GG32_classification.tar.gz.
(2) Go to the Map Data to Graph page.
    (a) In the top menu, click on the Add Data File icon. One at a time, click the (+) icon to add each of the 4 movie* data files. Within the Add Data File panel, click each file name, select the Has header checkbox, and click ADD at the bottom right, to load the data file.
    (b) Note that this will place a duplicate file icon for each file on the schema diagram. You can delete these duplicates (the ones which are not linked to a vertex or edge type).
    (c) Finally, click the Publish Data Mapping up-arrow button.

(3) Go to the Load Data page. Wait a few seconds for the Load (right arrow) button to become active. Click it and wait for the data to be loaded.
(4) Go to the Write Queries page. Click the Install All Queries button, to the left of the "GSQL Queries" heading.

The figure below shows the full dataset. This view is saved in the solution. You can view it by going to the Explore Graph page and then clicking the Open Exploration History icon (far left on the top menu).



Classification Queries

We are using the knn_cosine algorithm which is in our GSQL Graph Algorithm Library: https://github.com/tigergraph/gsql-graph-algorithms

We have taken the generic algorithms in the library and set the vertex types and edge types to fit our example graph.

This algorithm has 3 forms, each for a slightly different use case. All of them use cosine neighborhood as the method for measuring similarity.

- **knn_cosine_ss** (VERTEX source, INT topK)
  Predict the label of the *source* vertex, using its *topK* most similar neighbors.
- **knn_cosine_all** (INT topK, BOOL includeLabeled=true)
  predict the label of each vertex in the graph, using their *topK* most similar neighbors. If *includeLabeled* is true, make predictions for the labeled vertices; otherwise, skip them.
- **knn_cosine_cv** (INT min_k, INT max_k):
  For each value of k in the range min_k to max_k, predict the label of each vertex in the graph. Compute the percentage of correct predictions for each value of k, and report the value of k which gives the most accurate predictions.
- knn_cosine_cv_sub is a subquery called by knn_cosine_cv. In the graph algorithm library, both queries are saved in one file. GraphStudio, however, does not allow two queries in one file, so they have been separated into two queries.

knn_cosine_ss

Try running knn_cosine_ss with vertex type = Reviewer, id = 0, and k = 3.
The result is:


```
{
  "predicted_label": "a"
}
```


If you repeat this for each id from 0 to 5, you should get the following. The correct answers are also shown.

id	predicted_label	correct label
0	a	a
1	a	a
2	b	a
3	a	b
4	b	b

5	<blank>	b

Let's dig deeper.
Q: Why did it not make a prediction for id=5?
A: If you look at the figure of the data, you'll see that Reviewer 5 reviewed only one film, "One Cut of the Dead", but no one else reviewed that film. Since there is no film shared between Reviewer 5 and the other Reviewers, it has no similar neighbors.

Q: Can we see the labels of the top k neighbors, so we can see how the predictions were made?
A: You can made a simple edit to the knn_cosine_ss query. Add the line
PRINT kNN;
before the RETURN statement. That is, the last 3 lines should look like the following:

```
PRINT predicted_label;
PRINT kNN;
 RETURN predicted_label;
```

Save and Install the revised algorithm.
Now run knn_cosine_2 (Reviewer 2, 3) again.


```
 {
  "predicted_label": "b"
 },
 {
  "kNN": [
   {
    "v_id": "4",
    "v_type": "Reviewer",
    "attributes": {
     "name": "Kat",
     "known_label": "b",
     "predicted_label": "",
     "@similarity": 0.3526,
     "@numerator": 48,
     "@norm2": 113
    }
   },
   {
    "v_id": "3",
    "v_type": "Reviewer",
    "attributes": {
     "name": "Neil",
     "known_label": "b",
     "predicted_label": "",
     "@similarity": 0.42436,
     "@numerator": 78,
     "@norm2": 206
```

```
      }
    },
    {
      "v_id": "0",
      "v_type": "Reviewer",
      "attributes": {
        "name": "Alex",
        "known_label": "a",
        "predicted_label": "",
        "@similarity": 0.14248,
        "@numerator": 24,
        "@norm2": 173
      }
    }
  ]
}
```


So we see that it used neighbors 0, 3, and 4, and their labels are "a", "b", and "b" respectively. So "b" wins. If we look at the graph, we see that indeed Reviewer 2 reviewed films also reviewed by 0, 3, and 4. If we set k = 3 or higher, it will simply pick all three co-reviewers.

Note that we labeled our vertices arbitrarily, so if there is any correlation between films liked and labels, it is purely coincidental!

In theory, as you try different values of k, the predictions could change.

knn_cosine_all

Now let's run knn_cosine_all, using k=3. We expect to get the same answers as we did in the previous section, running knn_cosine_ss, for each of the 6 Reviewers, with k=3.

However, for Reviewer 1, the prediction has changed:

id	predicted_label from knn_cosine_ss	predicted_label from knn_cosine_all	correct label
0	a	a	a
1	a	b	a
2	b	b	a
3	a	a	b
4	b	b	b
5	<blank>	<blank>	b

% correct	50%	33.3%	

What happened? Is there a bug in one of the algorithms? No, actually, it's because Reviewer 1 has only two similar neighbors, Reviewer 0 with label "a" and Reviewer 3 with label "b". So it's a tie. It just happens that one will favor "a" and the other will favor "b". Actually, it's possible that you are getting different answers than we are. Reviewer 4 also has an even number of similar neighbors. Reviewers 0, 2, and 3 each have three similar Reviewers, so they can't have a tie. We could alter the algorithms so that the neighbors with the higher similarity scores will break the tie, but kNN in general doesn't require that we do this.

knn_cosine_cv

This is the cross validation algorithm which is considered machine learning. This time we provide a range of k values. The query will run the equivalent of knn_cosine_all for each value of k in the range [min_k, max_k] and then report which value of k gave the highest percentage of correct predictions. In the case of a tie, it will pick the higher value of k.

Try knn_cosine_cv for the range [2, 5]. You should get answers like this:


```
{
  "@@correct_rate_list": [
    0.5,
    0.33333,
    0.33333,
    0.33333
  ]
},
{
  "best_k": 2
}
```


The four scores in @@correct_rate_list are the fraction of correct predictions for k = 2, 3, 4, and 5. k=2 has the highest accuracy, so it is selected as the best_k.

Now, if you were to run knn_cosine_ss, it is recommended that you use k=2 to get the best answers in general.

###