Dupli Checker

PLAGIARISM SCAN REPORT

3%
Plagiarized

Date	2022-11-11
Words	1021
Characters	4672

Content Checked for Plagiarism

```
# -*- coding: utf-8 -*-
"""indian_bipartite.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1LTPqNd9TkDet8GVAbj95JxIlg1cJg37o

#importing required libraries
"""


Import pandas as pd
Import networkx as nx
Import numpy as np
From networkx.algorithms import bipartite
Import matplotlib.pyplot as plt
From networkx.drawing.layout import bipartite_layout

!pip install mlxtend

From google.colab import drive
Drive.mount('/content/drive')

Df1 = pd.read_csv('/content/drive/MyDrive/Recipe (1).csv')

"""#preprocessing of dataset"""

Df1.head()

Df1.rename(columns = {'0':'Recipe','1':'Ingr'}, inplace = True)

Df1 = df1.apply(lambda x: x.str.lower())

Df1.replace(to_replace = 'clarified butter', value = 'ghee', inplace=True)

Df1.replace(to_replace = 'garam masala powder', value = 'garam masala', inplace=True)
```

```
Df1 = df1[df1.Ingr != "nan"]

Df1.reset_index(inplace = True)

Df1.head(10)

Df1.tail()

Df1.drop(df1[df1['Recipe'] == "imarti"].index, inplace = True)

Df1.drop(df1[df1['Recipe'] == "bhatura"].index, inplace = True)

Df1.drop(df1[df1['Recipe'] == "sevai"].index, inplace = True)

Df1.drop(df1[df1['Recipe'] == "khaman"].index, inplace = True)

Df1.drop(df1[df1['Recipe'] == "boondi"].index, inplace = True)

Df1.drop(df1[df1['Recipe'] == "brown rice"].index, inplace = True)

Df1.reset_index(inplace = True)

Df1.to_csv('recipes_bipartite.csv')

Df1.info()

"""#creating bipartite graph"""

Set1 = []
Set1 = df1['Recipe'].tolist()
Set1 = set(set1)
Set1 = list(set1)

Len(set1)

Set2 = []
Set2 = df1['Ingr'].tolist()
Set2 = set(set2)
Set2 = list(set2)

Len(set2)

Set(set1).intersection(set(set2))

Recipe_graph = nx.Graph()
Recipe_graph.add_nodes_from(set1,bipartite=1)
Recipe_graph.add_nodes_from(set2,bipartite=0)

For I in range(len(df1)):
  K1 = df1[‘Recipe’][i]
  K2 = df1[‘Ingr’][i]
  Recipe_graph.add_edges_from([(k1,k2)])

Bipartite.is_bipartite(recipe_graph)

Fig = plt.figure(1, figsize=(200, 80), dpi=60)
Nx.draw(recipe_graph,with_labels=True)
Plt.savefig(“graph.png”)

X, Y = bipartite.sets(recipe_graph)
Pos = dict()
Pos.update( (n, (1, i)) for I, n in enumerate(X) )
Pos.update( (n, (2, i)) for I, n in enumerate(Y) )
Plt.figure(3,figsize=(300,300))
Nx.draw(recipe_graph, pos=pos, with_labels=True,node_size=100)
Plt.savefig(“graph2.png”)

“””#Association rule mining”””

Data = pd.read_excel(“/content/drive/MyDrive/indian_food.xlsx”)
Data

Diet = input(“enter your prefered diet “)
Flavor = input(“enter your prefered flavor profile “)
Course = input(“enter your prefered course “)

Filtered = data.loc[data[‘diet’]==diet]
Filtered = data.loc[data[‘flavor_profile’]==flavor]
Filtered = data.loc[data[‘course’]==course]
Display = pd.DataFrame({‘name’:filtered[‘name’],’ingredients’:filtered[‘ingredients’]})
Display

Count = 0
Basket = []
For I in display[‘ingredients’]:
    String = str(i)
    Temp = [x.strip() for x in string.split(‘,’)]
    Basket.append(temp)
Basket

From mlxtend.preprocessing import TransactionEncoder

Transact = TransactionEncoder()
Transact_ary = transact.fit(basket).transform(basket)
Data = pd.DataFrame(transact_ary, columns=transact.columns_)
Data

From mlxtend.frequent_patterns import apriori
```

```
Apriori(data,min_support=0.02,use_colnames=True)

“””Projection of Bipartite graph w.r.t Recipes”””

Projection_recipes = bipartite.projected_graph(recipe_graph,set1)
Projection_recipes.edges()

Projection_ingredients = bipartite.projected_graph(recipe_graph,set2)
Projection_ingredients.edges()

“””#Health Suggestions

How many common ingredients between two recipes
"""

Common_ingredients = bipartite.generic_weighted_projected_graph(recipe_graph,set1)
Print(“--- Similarity Scores between two recipes ---“)
Similar = list(common_ingredients.edges(data=‘weight’))

Import requests
Def get_healthscores(recipe):
    url = https://calorieninjas.p.rapidapi.com/v1/nutrition

    querystring = {“query”:recipe}

    headers = {
     “X-RapidAPI-Key”: “f45a109323msh0d7a964746c0587p169d05jsnc19e066174bc”,
     “X-RapidAPI-Host”: “calorieninjas.p.rapidapi.com”
    }

    Response = requests.request(“GET”, url, headers=headers, params=querystring)

    Return response.text

Recipe = input(“Enter recipe “)

Print(get_healthscores(recipe))

For x in similar:
  If x[0] == recipe and x[2]>=2:
    Temp = get_healthscores(x[1])
    # print(len(temp))

    If(len(temp)==13):
      Continue
    Else:
      Print(x[1])
      Print(temp)
      Print(‘\n’)
```

```
“””#Square Clustering

Square clustering – The probability that two neighbors of node v share a common neighbor different from v
“””

Nx.square_clustering(recipe_graph,nodes=set1)

“””#Degree Analysis

Each recipe and ingredients degree
“””

Ingredients, recipe = bipartite.degrees(recipe_graph, set1)

Ingredients

“””Visualizing the degrees”””

Import matplotlib.pyplot as plt
From collections import Counter

Ingredients_dist = Counter(sorted(dict(ingredients).values()))
Recipe_dist = Counter(sorted(dict(recipe).values()))

Ingredients_dist

Def plot_twonode_dist(top_dist,bottom_dist):
  Fig = plt.figure(figsize=(12,4),dpi=500)
  Ax = [fig.add_subplot(1,2,i+1) for I in range(2)]

  Ax[0].plot(list(top_dist.keys()),list(top_dist.values()),'bo',linestyle='-',label = 'Ingredients size')
  Ax[1].plot(list(bottom_dist.keys()),list(bottom_dist.values()),'bo',linestyle='-',label = 'Recipe size')

  For axis in ax:
    Axis.set_ylabel('Frequency',fontsize=10)
    Axis.tick_params(axis='both',which='major',labelsize=16)
    Axis.legend(loc='upper right',fontsize=12,ncol=1,frameon=True)

  Ax[0].set_xlabel('Degree',fontsize=10)
  Ax[1].set_xlabel('Degree',fontsize=10)

  Plt.tight_layout()
  Plt.show()

Plot_twonode_dist(ingredients_dist,recipe_dist)

Gs = bipartite.projected_graph(recipe_graph,set1)
Gm = bipartite.projected_graph(recipe_graph,set1,'MultiGraph')

Popularity_distribution = Counter(sorted(dict(Gs.degree()).values()))
```

```
Strength_distribution =   Counter(sorted(dict(Gm.degree()).values()))

Def plot_proj_dist(pop_dist,str_dist):
  Fig,ax = plt.subplots()

  Ax.plot(list(pop_dist.keys()),list(pop_dist.values()),'bo',linestyle='-',label='Popularity')
  Ax.plot(list(str_dist.keys()),list(str_dist.values()),'ro',linestyle='-',label='Strength')

  Ax.set_xlabel('Degree',fontsize=1)
  Ax.set_ylabel('Frequency',fontsize=1)
  Ax.tick_params(axis='both',which='major',labelsize=16)
  Ax.legend(loc='upper right',fontsize=12,ncol=1,frameon=True)

  Plt.tight_layout()
  Plt.show()

Plot_proj_dist(popularity_distribution,strength_distribution)

Def get_highest(degree_dict,degree_type):
  Q = Counter(dict(degree_dict))
  Print('Highest '+ degree_type)
  For x,y in q.most_common(10):
    Print('%s : %i'% (x,y))

"""Most used Ingredients"""

Get_highest(ingredients,'Ingredients')

"""Recipes that uses most ingredients"""

Get_highest(recipe,'Recipes')

Get_highest(Gs.degree(),'Popularity')

Get_highest(Gm.degree(),'Strength')

"""#Community Detection

Weighted projected graph – The specified nodes with weights representing the number of shared neighbors
"""

Gw1 = bipartite.weighted_projected_graph(recipe_graph,set1)

Import community
From community import community_louvain
Comm_dict = community_louvain.best_partition(Gw1)
Print(comm_dict)

Comm_set = set(list(comm_dict.values()))
Print(comm_set)
```

```
Comm_org = {comm:[] for comm in list(comm_set)}
For node, comm in comm_dict.items():
  Comm_org[comm].append(node)
Print(comm_org)

Gw2 = bipartite.weighted_projected_graph(recipe_graph,set2)

Import community
From community import community_louvain
Comm_dict = community_louvain.best_partition(Gw2)
#comm_dict = community.best_partition(Gw2)
Print(comm_dict)

Comm_set = set(list(comm_dict.values()))
Print(comm_set)

Comm_org = {comm:[] for comm in list(comm_set)}
For node, comm in comm_dict.items():
  Comm_org[comm].append(node)
Print(comm_org)

“””#Hubs and Authorities “””

Hubs, authorities = nx.hits(projection_ingredients, max_iter = 50, normalized = True)

H = []
For I in hubs:
    h.append([I,hubs[i]])
h.sort(reverse = True, key=lambda x:x[1])
print(‘HUBS:’)
for I in h[:5]:
    print(i)

a = []
for I in authorities:
    a.append([I,authorities[i]])
a.sort(reverse = True, key=lambda x:x[1])
print(‘\nAUTHORITIES:’)
for I in a[:5]:
    print(i)

“””#Page Rank

To give importance score for each ingredient according to the network
“””

Pr = nx.pagerank(projection_ingredients)
Pagerank = []
For I in pr:
    Pagerank.append([I,pr[i]])
Pagerank.sort(reverse = True, key = lambda x:x[1])
Print('PANGERANK of ingredients:')
For I in pagerank[:25]:
    Print(i)

"""#Matching

Maximum cardinality matching – matching is considered as a maximum cardinality matching if it contains the largest possible number of edges. As each edge will cover exactly two vertices, it is equivalent to finding a matching that covers as many vertices as possible.
"""

My_matching = bipartite.matching.hopcroft_karp_matching(recipe_graph, set1)

My_matching
```