Code Doc.docx

Scanned on: 4:50 November 12, 2022 UTC

Overall Similarity Score

Results Found

Total Words in Text

Identical Words	20
Words with Minor Changes	27
Paraphrased Words	0
Omitted Words	0

Code Doc.docx

Scanned on: 4:50 November 12, 2022 UTC

Results

Sources that matched your submitted document.

Building a Recommendation Engine in Python · GitHub *https://gist.github.com/Awuor87/f0b496f04dd200e05fa6c646009ac9ab*	2%

IDENTICAL

Identical matches are one to one exact wording in the text.

MINOR CHANGES

Nearly identical with different form, ie "slow" becomes "slowly".

PARAPHRASED

Close meaning but different words used to convey the same message.

Unsure about your report?

The results have been found after comparing your submitted text to online sources, open databases and the Copyleaks internal database. For any questions about the report contact us on support@copyleaks.com

Learn more about different kinds of plagiarism here

Code Doc.docx

Scanned on: 4:50 November 12, 2022 UTC

Scanned Text

Your text is highlighted according to the matched content in the results above.

IDENTICAL MINOR CHANGES PARAPHRASED

```
# DATA PRE-PROCESSING -
FILTERING BOOKS
import pandas as pd
import numpy as np
pdf=pd.read_csv('products.csv')
cdf=pd.read_csv('copurchase.csv')
pdf
pdf.info()
pdf.describe()
pdf['group'].unique()
new_pdf=pdf[pdf['group']=='Book']
new_pdf
new_pdf.group.unique()
book_ids=pdf[pdf['group']=='Book']
['id'].to_numpy()
book_ids
non_book_ids=pdf[pdf['group']!='Book']
['id'].to_numpy()
non_book_ids
len(new_pdf)
len(pdf)
cdf
len(cdf)
new_cdf=cdf[cdf.Source.isin(book_ids)]
new_cdf=new_cdf[new_cdf.Target.isin(boo
k_ids)]
new_cdf
len(new_cdf)
new_cdf.Source.isin(book_ids).unique()
new_cdf.Target.isin(book_ids).unique()
# All the edges and nodes are related with
BOOK
# producing book related datasets
# new_pdf.to_csv('books_info.csv')
```

```
# new_cdf.to_csv('books_copurchase.csv')
# no. of nodes
len(new_pdf)
# no. of edges
len(new_cdf)
in_d=pd.DataFrame(new_cdf.groupby(['Tar
get'])['Source'].size()).reset_index()
in_d.columns=['id','in_d']
out_d=pd.DataFrame(new_cdf.groupby(['So
urce'])['Target'].size()).reset_index()
out_d.columns=['id','out_d']
all_d=pd.merge(in_d,out_d,on='id',how='out
er')
all_d[np.isnan(all_d)]=0
all_d['all_d']=(all_d.in_d+all_d.out_d)
all_d=all_d.sort_values('all_d',ascending=Fa
lse)
all_d
# len(all_d)
# node with highest all_degree
new_pdf[new_pdf['id']==14849]
# BOOK RECOMMENDATION
# ## Importing the required modules
import string
import re
from nltk.corpus import stopwords
from stemming.porter2 import stem
import networkx
from operator import itemgetter
# ## Accessing the meta-data
Md_File = open('amazon-meta.txt', 'r',
encoding = 'utf-8', errors = 'ignore')
Amazon_Products = {}
(Id, ASIN, Title, Categories, Group,
Copurchased, SalesRank, TotalReviews,
AvgRating, DegreeCentrality,
ClusteringCoeff) = ("", "", "", "", "", "", 0,
0, 0.0, 0, 0.0)
for line in Md_File:
line = line.strip()
if(line.startswith("Id")):
Id = line[3:].strip()
elif(line.startswith("ASIN")):
ASIN = line[5:].strip()
elif(line.startswith("title")):
Title = line[6:].strip()
Title = ' '.join(Title.split())
elif(line.startswith("group")):
Group = line[6:].strip()
elif(line.startswith("salesrank")):
SalesRank = line[10:].strip()
elif(line. startswith("similar")):
ls = line.split()
Copurchased = ' '.join([c for c in
ls[2:]])
elif(line.startswith("categories")):
```

```
ls = line.split()
Categories = '
'.join((Md_File.readline()).lower() for i in
range(int(ls[1].strip())))
```

```
Categories = re.compile('[%s]' %
re.escape(string.digits +
string.punctuation)).sub(' ', Categories)
Categories = '
'.join(set(Categories.split()) -
set(stopwords.words("english")))
Categories = ' '.join(stem(word) for
word in Categories.split())
elif(line.startswith("reviews")):
ls = line.split()
TotalReviews = ls[2].strip()
AvgRating = ls[7].strip()
elif(line == ""):
try:
md = {}
if (ASIN != ""):
Amazon_Products[ASIN] = Md
md['Id'] = Id
md['Title'] = Title
md['Categories'] = '
'.join(set(Categories.split()))
md['Group'] = Group
md['Copurchased'] = Copurchased
md['SalesRank'] = int(SalesRank)
md['TotalReviews'] =
int(TotalReviews)
md['AvgRating'] = float(AvgRating)
md['DegreeCentrality'] =
DegreeCentrality
md['ClusteringCoeff'] =
ClusteringCoeff
except NameError:
continue
(Id, ASIN, Title, Categories, Group,
Copurchased, SalesRank, TotalReviews,
AvgRating, DegreeCentrality,
ClusteringCoeff) = ("", "", "", "", "", "", 0, 0,
0.0, 0, 0.0)
Md_File.close()
# ## Filtering out data related to book
records
Amazon_Books = {}
for asin, md in Amazon_Products.items():
if (md['Group'] == 'Book'):
Amazon_Books[asin] =
Amazon_Products[asin]
for asin, md in Amazon_Books.items():
Amazon_Books[asin]['Copurchased'] = '
'.join([cp for cp in md['Copurchased'].split()
if cp in Amazon_Books.keys()])
Amazon_Books
# ## Adding recommendation property
Copurchase_Graph = networkx.Graph()
for asin, md in Amazon_Books.items():
Copurchase_Graph.add_node(asin)
for a in md['Copurchased'].split():
Copurchase_Graph.add_node(a.strip())
similarity = 0
n1 = set((Amazon_Books[asin]
['Categories']).split())
n2 = set((Amazon_Books[a]
['Categories']).split())
n1In2 = n1 & n2
n1Un2 = n1 | n2
if(len(n1In2)) > 0:
similarity = round(len(n1In2) /
```

```
len(n1Un2), 2)
Copurchase_Graph.add_edge(asin,
a.strip(), weight = similarity)
dc = networkx.degree(Copurchase_Graph)
for asin in
networkx.nodes(Copurchase_Graph):
md = Amazon_Books[asin]
md['DegreeCentrality'] = int(dc[asin])
ego =
networkx.ego_graph(Copurchase_Graph,
asin, radius = 1)
md['ClusteringCoeff'] =
round(networkx.average_clustering(ego), 2)
Amazon_Books[asin] = md
Amazon_Books_File = open('amazon-
books.txt', 'w', encoding = 'utf-8', errors =
'ignore')
Amazon_Books_File.write("Id\t" + "ASIN\
t" + "Title\t" + "Categories\t" + "Group\t" +
"Copurchased\t" + "SalesRank\t" +
"TotalReviews\t" + "AvgRating\t"
"DegreeCentrality\t" +
```

```
"ClusteringCoeff\n")
for asin, md in Amazon_Books.items(): #
converting the meta-data into txt file
Amazon_Books_File.write(md['Id'] + "\t"
+ asin + "\t" +
md['Title'] + "\t" +
md['Categories'] + "\t" +
md['Group'] + "\t" +
md['Copurchased'] + "\t" +
str(md['SalesRank']) + "\t" +
str(md['TotalReviews']) + "\t" +
str(md['AvgRating']) + "\
t" + \

str(md['DegreeCentrality']) + "\t" + \
str(md['ClusteringCoeff'])
+ "\n")
Amazon_Books_File.close()
# writing the adjacency edge list
Amazon_Books_File = open("amazon-
books-copurchase.edgelist", 'wb')
networkx.write_weighted_edgelist(Copurch
ase_Graph, Amazon_Books_File)
Amazon_Books_File.close()
# ## Reading the text file
Books_File = open('amazon-books.txt', 'r',
encoding = 'utf-8', errors = 'ignore')
Books = {}
Books_File.readline()
for line in Books_File:
cell = line.split("\t")
Md = {}
Md['Id'] = cell[0].strip()
ASIN = cell[1].strip()
Md['Title'] = cell[2].strip()
Md['Categories'] = cell[3].strip()
Md['Group'] = cell[4].strip()
Md['Copurchased'] = cell[5].strip()
```

```
Md['SalesRank'] = int(cell[6].strip())
Md['TotalReviews'] = int(cell[7].strip())
Md['AvgRating'] = float(cell[8].strip())
md['DegreeCentrality'] =
int(cell[9].strip())
md['ClusteringCoeff'] =
float(cell[10].strip())
Books[ASIN] = md
Books_File.close()
Books_File = open("amazon-books-
copurchase.edgelist", "rb")
Copurchase_Graph =
networkx.read_weighted_edgelist(Books_Fi
le)
Books_File.close()
# ## Giving Book id as input for
recommending books
print("Looking for Recommendations for
Customer Purchasing this Book: ")
print("---------------------------------------------
----------------")
Purchased_ASIN = '0805047905'
print("ASIN = ", Purchased_ASIN)
print("Title = ", Books[Purchased_ASIN]
['Title'])
print("SalesRank = ",
Books[Purchased_ASIN]['SalesRank'])
print("TotalReviews = ",
Books[Purchased_ASIN]['TotalReviews'])
print("AvgRating = ",
Books[Purchased_ASIN]['AvgRating'])
print("DegreeCentrality = ",
Books[Purchased_ASIN]
['DegreeCentrality'])
print("ClusteringCoeff = ",
Books[Purchased_ASIN]['ClusteringCoeff'])
n = Purchased_ASIN
ego =
networkx.ego_graph(Copurchase_Graph, n,
radius = 1)
Ego_Graph = networkx.Graph(ego)
threshold = 0.5 # finding the nodes having
similarity measure based on category above
the threshold value
Ego_Trim_Graph = networkx.Graph()
for a, b, c in Ego_Graph.edges(data =
True):
if e['weight'] >= threshold:
Ego_Trim_Graph.add_edge(a, b)
Neighbors =
Ego_Trim_Graph.neighbors(Purchased_ASI
N)
ASIN_Meta = []
for asin in Neighbors:




ASIN = asin
Title = Amazon_Books[ASIN]['Title']
SalesRank = Amazon_Books[ASIN]
['SalesRank']
TotalReviews = Amazon_Books[ASIN]
['TotalReviews']
```

```
AvgRating = Amazon_Books[ASIN]
['AvgRating']
DegreeCentrality =
Amazon_Books[ASIN]['DegreeCentrality']
ASIN_Meta.append((ASIN, Title,
SalesRank, TotalReviews, AvgRating,
DegreeCentrality, ClusteringCoeff))
# ## Showing Top 5 Recommendations
Top5_ByAbgRating_ThenByTotalReviews
= sorted(ASIN_Meta, key = lambda x:
(x[4], x[3]), reverse = True)[:5]
print("Top 5 Recommendations By
AvgRating Then By TotalReviews for Users
Purchased The Book: ")
print("----------------------------------------------
-----------------------------------------")
print('ASIN\t', 'Title\t', 'SalesRank\t',
'TotalReviews\t', 'AvgRating\t',
'DegreeCentrality\t', 'ClusteringCoeff')
for asin in
Top5_ByAbgRating_ThenByTotalReviews:
print(asin)
# LINK PREDICTION
# importing required modules
import pandas as pd
import networkx as nx
import matplotlib.pyplot as plt
from tqdm import tqdm
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import
train_test_split, GridSearchCV
from sklearn import svm
from sklearn.metrics import
classification_report
from sklearn.linear_model import
LogisticRegression
from sklearn.metrics import
confusion_matrix, classification_report
# ## Reading Dataset and Visualizing
Network Graph
# reading and loading the dataset
Dataset = pd.read_csv('Final_Network.csv')
# graph object construction
Graph = nx.from_pandas_edgelist(Dataset,
'Source', 'Target', create_using = nx.Graph())
plt.figure(figsize = (20, 20))
pos = nx.random_layout(Graph, seed = 30)
nx.draw(Graph, with_labels = False, pos =
pos, node_size = 40, alpha = 0.3, edge_color
= 'green', node_color = 'blue')
plt.show()
# ## Finding Missing Edges from the
Network
Source = set(Dataset['Source'].unique())
Target = set(Dataset['Target'].unique())
Nodes = list(Source.union(Target))
Number_of_Nodes = len(Nodes)
Adjacency_Matrix =
nx.to_numpy_array(Graph, nodelist =
Nodes)
Missing_Nodes = []
Number = 0
for i in tqdm(range(Number_of_Nodes)):
for j in range(Number,
Number_of_Nodes):
```

```
if(i != j):
try:
if nx.shortest_path_length(Graph,
Nodes[i], Nodes[j]) <= 2:
if Adjacency_Matrix[i,j] == 0:

Missing_Nodes.append([Nodes[i],
Nodes[j]])
except:
continue
Number += 1
# ## Sampling a Part of Connected Edges
for Training
Removed_Edges = []
DF = Dataset.copy()
N = 0
for i in tqdm(Dataset.index.values):
Current =
nx.from_pandas_edgelist(DF.drop(index =
i), 'Source', 'Target', create_using =
nx.Graph())

if(nx.number_connected_components(Curre
nt) == 1 and len(Current.nodes) ==
Number_of_Nodes):
N += 1
if(N >= 2):
N = 0
continue
Removed_Edges.append(i)
DF = DF.drop(index = i)
# ## Combining Data of Connected and
Missing Edges for Training
Positive_Samples =
Dataset.loc[Removed_Edges]
Positive_Samples['Label'] = 1
Positive_Samples =
Positive_Samples.drop(labels = ["Unnamed:
0"], axis = 1)
Positive_Samples
Positive_Samples.to_csv('Positive_Samples.
csv')
Negative_Samples =
pd.DataFrame(list(Missing_Nodes),
columns = ['Source', 'Target'])
Negative_Samples['Label'] = 0
Negative_Samples =
Negative_Samples.head(4330)
Negative_Samples
Negative_Samples.to_csv('Negative_Sample
s.csv')
Train_Dataset =
pd.concat([Positive_Samples,
Negative_Samples], axis = 0)
Train_Dataset
Train_Dataset.to_csv('Train_Dataset.csv')
X_Dataset = Train_Dataset[['Source',
'Target']]
Y_Dataset = Train_Dataset['Label']
X_Train, X_Test, Y_Train, Y_Test =
```

```
train_test_split(X_Dataset, Y_Dataset,
test_size = 0.3, random_state = 10)
# ## Training Samples using Support Vector
Classfier
kernels = ['rbf', 'linear']
c_values = [0.001, 0.01, 0.1, 1]
param_grid = {'kernel':kernels, 'C':c_values}
SVM_Model = GridSearchCV(cv = 10,
estimator = svm.SVC(), param_grid =
param_grid)
SVM_Model.fit(X_Train, Y_Train)
print(classification_report(SVM_Model.best
_estimator_.predict(X_Test), Y_Test))
print(confusion_matrix(SVM_Model.best_e
stimator_.predict(X_Test), Y_Test))
# COMMUNITY DETECTION
# # Importing Libraries
import networkx as nx
from
networkx.algorithms.community.centrality
import girvan_newman
import numpy as np
from matplotlib import pyplot as plt
import pandas as pd
import random
get_ipython().run_line_magic('matplotlib',
'inline')
# # Loading the Dataset containing Links
# First 5000 edges are considered from the
original dataset
edge_df=pd.read_csv('Book_copurchase_5k
_edges_connected_graph.csv')
edge_df.head()
# # Constructing Graph and Finding
Communities among the Co-Purchages
# ## 1. Using NetworkX
G=nx.Graph()
G.add_edges_from(edge_df.values.tolist())
# Finding the communities using the Girvan
Newman algorithm
communities = girvan_newman(G)
node_groups = []
for com in next(communities):
node_groups.append(list(com))
node_groups
len(node_groups) # Total number of
communities or clusters found is 51
color_map = []
# Taking only 7 groups out of 51
for node in G:



if node in node_groups[0]:
color_map.append('red')
elif node in node_groups[1]:
color_map.append('blue')
elif node in node_groups[2]:
color_map.append('green')
elif node in node_groups[3]:
color_map.append('yellow')
elif node in node_groups[4]:
color_map.append('brown')
```

```
elif node in node_groups[5]:
color_map.append('pink')
elif node in node_groups[6]:
color_map.append('cyan')
else:
color_map.append('#FF000000')
# Visualizing Communities represented by
different colors
nx.draw(G, node_color=color_map,)
plt.show()
# ## 2. Finding Communities using igram
import igraph
from igraph import *
# igram indices start from 0, so edge list is
transformed
edge_list_igraph = [[y-1 for y in x] for x in
edge_list]
edge_list_igraph
# defining the graph and adding edges &
vertices; stored in adjancency list
g2 = Graph()
g2.add_vertices(5000)
g2.add_edges(edge_list_igraph)
print(g2)
# Nodes with zero degree are removed from
adjacency list
to_delete_ids = [v.index for v in g2.vs if
v.degree() == 0]
g2.delete_vertices(to_delete_ids)
print(g2)
# created commmunities based on edge
betweeness
v =
g2.community_edge_betweenness(directed=
True)
print(v)
# groupping the nodes into clusters with
cluster indices
cluster1 = v.as_clustering()
print(cluster1)
# modularity value
cluster1.modularity
# membership list : which cluster each node
belongs
cluster1.membership
# Assigning differnt colors to the nodes
belonging to same cluster
color_palletes =
igraph.drawing.colors.ClusterColoringPalett
e(len(cluster1))
g2.vs['color'] =
color_palletes.get_many(cluster1.membershi
p)
# Visualizing the communities present in the
graph
visual_style["layout"] =
g2.layout_fruchterman_reingold()
visual_style["bbox"] = (400, 400)
visual_style["margin"] = 20
fig, ax = plt.subplots()
fig.set_figwidth(400)
fig.set_figheight(400)
igraph.plot(g2,**visual_style, target=ax)
```