SmallSEOTools

PLAGIARISM SCAN REPORT

Words	564	Date	April 15,2021
Characters	6778	Excluded URL	

0% Plagiarism	100% Unique	0 Plagiarized Sentences	17 Unique Sentences

Content Checked For Plagiarism

```
#Installing networkx Library
!pip install network
#Importing Required Libraries
import networkx #For handling Network graphs
import random
import csv
import numpy as np
import pandas as pd
from tensorflow.keras.layers import Dense,Input,Activation
from tensorflow.keras.models import Sequential
from sklearn.preprocessing import Normalizer
from tensorflow.keras.losses import categorical_crossentropy
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from tensorflow.keras.optimizers import Adam
from keras.utils import np_utils
#stroing dataset path
path='/content/drive/MyDrive/SENA/facebookdata.txt'
#Loading dataset into dataset as graph in variable
with open(path,'rb') as file:
graph=networkx.read_edgelist(file)
networkx.draw_spectral(graph,with_labels = True)
--------------------------------------------------------------------------------------------
def Generate_False_Graph(Graph,neg_no):
"""
Input : a graph as Networkx class
number of negative samples
Output :neg_no number of false edges (edges which does not exist in the graph)
and build a new graph using negative edges
"""
Negative_graph= networkx.Graph()
count=0
while (count [source,target]=random.sample(Graph.nodes(),2) # pick random two nodes in the graph
try :
short_path=networkx.shortest_path_length(Graph,source,target)
if(short_path>=2):# there does not exist a direct edges betwwen them
Negative_graph.add_edge(source,target, positive="False") # adding the false edges in the new graph created
count += 1
except:
```

```
pass
return Negative_graph
------------------------------------------------------------------------------------------------
def Extract_Sample(Graph,pos_no,neg_no):
"""
Input : a graph as Networkx class
Output : List of Positive and Negative samples
"""
no_edges=Graph.number_of_edges()
# Poistive samples
pos_graph = networkx.Graph()
positves=random.sample(Graph.edges(),pos_no)

pos_graph.add_edges_from(positves, positive="True")
networkx.write_edgelist(pos_graph, "positive_graph.txt", data=['positive'])
# Negative Samples
neg_graph=Generate_False_Graph(Graph,neg_no)
networkx.write_edgelist(neg_graph, "negatvie_graph.txt", data=['positive'])
return pos_graph.edges(),neg_graph.edges()
------------------------------------------------------------------------------------------------
def Extract_feature(Graph,positives,negatives):
"""
Input: Graph as network class
Positive dataset(Nodes having edges among them)
Negative Dataset(Nodes having shortest path greater than 2)
Output:Features of the two nodes in the given dataset
Features: Common Neighbours, Jacaard Coefficient, Resource allocation index, Adamic Adar Coefficient, Preferential
Attachment
"""
data = []
feature_name = [common_neighbors,networkx.resource_allocation_index,networkx.jaccard_coefficient,
networkx.adamic_adar_index, networkx.preferential_attachment]
label = ["label"] + ["1" for i in range(len(positives))] + ["0" for i in range(len(negatives))]
for func in feature_name:
preds = func(Graph, positives)
feature = [func.name] + [i[2] for i in preds]
preds = func(Graph, negatives)
feature = feature + [i[2] for i in preds]
data.append(feature)
data.append(label)
col_names = []
records = []
for col in data:
col_names.append(col[0])
records.append(col[1:])
records = np.array((records)).T
csvfile = pd.DataFrame(records, columns = col_names).to_csv('features.csv')
#Extracting 5000 samples for each dataset
x,y=Extract_Sample(graph,5000,5000)
Extract_feature(graph, x, y)
dataset=pd.read_csv("features.csv")
x=dataset.iloc[:,:-1].values
y=dataset.iloc[:,-1].values
#Normalizing the dataset
t = Normalizer().fit(x)
x = t.transform(x)
------------------------------------------------------------------------------------------------
def rand_hexcolor():
"""
Input :None
Output : Random color code in hexadecimal format
Process:pick random number and convert ot hexadecimal
"""

rgb = ""
```

```
for _ in "RGB": # creating color for each channel
i = random.randrange(0, 2**8)
rgb += i.to_bytes(1, "big").hex()
return '#'+rgb
k=[0,1]
numbers=set(k.copy())
numbercol={i:rand_hexcolor() for i in numbers } # giving each number a color

from sklearn.manifold import TSNE # tsne converts the image to dimensional feature vector which is used for plotting
# dimensionality reduction for viewing 400 dimesion hyperspace in 2d space
tsne = TSNE(metric='cosine') # we use cosine metric which finds similarty between two feature vectors(pixels of the image)
embed_tsne = tsne.fit_transform(np.array((x))) # converting 400 feature
plt.figure(figsize=(35, 35)) # assigning size for each scatter plot
for idx in range(len(x)):# iterating through all training data
plt.scatter(*embed_tsne[idx, :],c=numbercol[y[idx]]) # each point is marked in the plot with tnse feature considering as coordinate with color chosen for that label corresponding to the record
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.33, random_state=42)
y_train = np_utils.to_categorical(y_train).astype('float32')
y_test = np_utils.to_categorical(y_test).astype('float32')
y_train.shape
#Building the base model
model = Sequential()
model.add(Input(len(x[0])))
model.add(Dense(32, activation='relu'))
model.add(Dense(2, activation='sigmoid'))
#Compiling the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
#Model training
model.fit(X_train, y_train, batch_size=200, epochs=1000)
model.evaluate(X_test,y_test)
model.save('/content/drive/MyDrive/SENA/model.h5')
from tensorflow.keras.models import load_model
model = load_model('/content/drive/MyDrive/SENA/model.h5')
X = X_test[3000]
X = np.expand_dims(X, axis=0)
np.argmax(model.predict(X))
y_test[3000]
```

Sources	Similarity