GitHub User Classification Using Graph Convolution Network

```
import pandas as pd
from tqdm import tqdm
import json
import os
import umap
import numpy as np
import scipy.sparse as sp
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import f1_score, roc_auc_score,
average_precision_score, confusion_matrix


import stellargraph as sg
from stellargraph.mapper import FullBatchNodeGenerator
from stellargraph.layer import GCN

import warnings
import tensorflow as tf
from tensorflow.keras import backend as K
from tensorflow.keras import activations, initializers, constraints,
regularizers
from tensorflow.keras.layers import Input, Layer, Lambda, Dropout,
Reshape, Dense
from tensorflow.keras.callbacks import EarlyStopping

from tensorflow.keras import layers, optimizers, losses, metrics,
Model
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Read in edges, features, and targets

```
edges_path = 'datasets-master/git_web_ml/git_edges.csv'
targets_path = 'datasets-master/git_web_ml/git_target.csv'
features_path = 'datasets-master/git_web_ml/git_features.json'

# Read in edges
edges = pd.read_csv(edges_path)
edges.columns = ['source', 'target'] # renaming for StellarGraph
compatibility
display(edges.shape, edges.head())
```

```
(289003, 2)

   source  target
0       0   23977
1       1   34526
```

```
2       1    2370
3       1   14683
4       1   29982
```

```python
# Read in features
with open(features_path) as json_data:
    features = json.load(json_data)

max_feature = np.max([v for v_list in features.values() for v in
v_list])
features_matrix = np.zeros(shape = (len(list(features.keys())),
max_feature+1))

i = 0
for k, vs in tqdm(features.items()):
    for v in vs:
        features_matrix[i, v] = 1
    i+=1
```

```
100%|██████████| 37700/37700 [00:00<00:00, 57768.12it/s]
```

```python
node_features = pd.DataFrame(features_matrix, index = features.keys())
display(node_features.shape, node_features.head())
```

```
(37700, 4005)

   0     1     2     3     4     5     6     7     8     9     ...
3995  \
0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0  ...
0.0
1   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0  ...
0.0
2   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0  ...
0.0
3   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0  ...
0.0
4   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0  ...
0.0

   3996  3997  3998  3999  4000  4001  4002  4003  4004
0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
1   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
2   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
3   0.0   0.0   0.0   0.0   0.0   0.0   0.0   1.0   0.0
4   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0

[5 rows x 4005 columns]
```

```python
# Read in targets
targets = pd.read_csv(targets_path)
targets.index = targets.id.astype(str)
```

```
targets = targets.loc[features.keys(), :]
display(targets.shape, targets.head(),
targets.ml_target.value_counts(normalize=True))
```

```
(37700, 3)
```

```
    id          name  ml_target
id
0    0        Eiryyy          0
1    1    shawflying          0
2    2   JpMCarrilho          1
3    3     SuhwanCha          0
4    4  sunilangadi2          1
```

```
0    0.741671
1    0.258329
Name: ml_target, dtype: float64
```

```
G = sg.StellarGraph(node_features, edges.astype(str))
```

```
print(G.info())
```

```
StellarGraph: Undirected multigraph
 Nodes: 37700, Edges: 289003

 Node types:
  default: [37700]
    Features: float32 vector, length 4005
    Edge types: default-default->default

 Edge types:
    default-default->default: [289003]
        Weights: all 1 (default)
        Features: none
```

```
train_pages, test_pages = train_test_split(targets, train_size=200)
val_pages, test_pages = train_test_split(test_pages, train_size=200)
```

```
train_pages.shape, val_pages.shape, test_pages.shape
```

```
((200, 3), (200, 3), (37300, 3))
```

Pre-processing

Target pre-processing

```
target_encoding = LabelBinarizer()

train_targets =
target_encoding.fit_transform(train_pages['ml_target'])
val_targets = target_encoding.transform(val_pages['ml_target'])
test_targets = target_encoding.transform(test_pages['ml_target'])
```

Graph Data Pre-processing

```
# Get the adjacency matrix
A = G.to_adjacency_matrix(weighted=False)

# Add self-connections
A_t = A + sp.diags(np.ones(A.shape[0]) - A.diagonal())

# Degree matrix to the power of -1/2
D_t = sp.diags(np.power(np.array(A.sum(1)), -0.5).flatten(), 0)

# Normalise the Adjacency matrix
A_norm = A.dot(D_t).transpose().dot(D_t).todense()

# Define the function to get these indices
def get_node_indices(G, ids):
    # find the indices of the nodes
    node_ids = np.asarray(ids)
    flat_node_ids = node_ids.reshape(-1)

    flat_node_indices = G.node_ids_to_ilocs(flat_node_ids) # in-built
function makes it really easy
    # back to the original shape
    node_indices = flat_node_indices.reshape(1, len(node_ids)) # add 1
extra dimension

    return node_indices

train_indices = get_node_indices(G, train_pages.index)
val_indices = get_node_indices(G, val_pages.index)
test_indices = get_node_indices(G, test_pages.index)

# Expand dimensions
features_input = np.expand_dims(features_matrix, 0)
A_input = np.expand_dims(A_norm, 0)

y_train = np.expand_dims(train_targets, 0)
y_val = np.expand_dims(val_targets, 0)
y_test = np.expand_dims(test_targets, 0)
```

GCN Model

```
from stellargraph.layer.gcn import GraphConvolution, GatherIndices

# Initialise GCN parameters
kernel_initializer="glorot_uniform"
bias = True
bias_initializer="zeros"
n_layers = 2
layer_sizes = [32, 32]
dropout = 0.5
```

```
n_features = features_input.shape[2]
n_nodes = features_input.shape[1]
```

First of all, let's initialise the Input layers with the correct shapes to receive our 3 inputs:

1. Features matrix
2. Train/Val/Test indices
3. Normalised adjacency matrix

```
x_features = Input(batch_shape=(1, n_nodes, n_features))
x_indices = Input(batch_shape=(1, None), dtype="int32")
x_adjacency = Input(batch_shape=(1, n_nodes, n_nodes))
x_inp = [x_features, x_indices, x_adjacency]
x_inp
```

```
[<KerasTensor: shape=(1, 37700, 4005) dtype=float32 (created by layer
'input_1')>,
 <KerasTensor: shape=(1, None) dtype=int32 (created by layer
'input_2')>,
 <KerasTensor: shape=(1, 37700, 37700) dtype=float32 (created by layer
'input_3')>]
```

```
x = Dropout(0.5)(x_features)
x = GraphConvolution(32, activation='relu',
                     use_bias=True,
                     kernel_initializer=kernel_initializer,
                     bias_initializer=bias_initializer)([x,
x_adjacency])
x = Dropout(0.5)(x)
x = GraphConvolution(32, activation='relu',
                     use_bias=True,
                     kernel_initializer=kernel_initializer,
                     bias_initializer=bias_initializer)([x,
x_adjacency])

x = GatherIndices(batch_dims=1)([x, x_indices])
output = Dense(1, activation='sigmoid')(x)

model = Model(inputs=[x_features, x_indices, x_adjacency],
outputs=output)
model.summary()
```

```
Model: "model"
______________________________________________________________________
____________________________
Layer (type)                    Output Shape         Param #
Connected to
======================================================================
============================
input_1 (InputLayer)            [(1, 37700, 4005)]   0

______________________________________________________________________
```

```
________________________________________________________________
dropout_2 (Dropout)             (1, 37700, 4005)     0
input_1[0][0]
______________________________________________________________________________________________________________
________________________________________________________________
input_3 (InputLayer)            [(1, 37700, 37700)]  0

______________________________________________________________________________________________________________
________________________________________________________________
graph_convolution_2 (GraphConvo (1, 37700, 32)       128192
dropout_2[0][0]

input_3[0][0]
______________________________________________________________________________________________________________
________________________________________________________________
dropout_3 (Dropout)             (1, 37700, 32)       0
graph_convolution_2[0][0]
______________________________________________________________________________________________________________
________________________________________________________________
graph_convolution_3 (GraphConvo (1, 37700, 32)       1056
dropout_3[0][0]

input_3[0][0]
______________________________________________________________________________________________________________
________________________________________________________________
input_2 (InputLayer)            [(1, None)]          0

______________________________________________________________________________________________________________
________________________________________________________________
gather_indices (GatherIndices)  (1, None, 32)        0
graph_convolution_3[0][0]

input_2[0][0]
______________________________________________________________________________________________________________
________________________________________________________________
dense (Dense)                   (1, None, 1)         33
gather_indices[0][0]
======================================================================
============================
Total params: 129,281
Trainable params: 129,281
Non-trainable params: 0
______________________________________________________________________________________________________________
________________________________________________________________
```

```
model.compile(
    optimizer=optimizers.Adam(lr=0.01),
    loss=losses.binary_crossentropy,
    metrics=["acc"],
)
```

```
es_callback = EarlyStopping(monitor="val_loss", patience=10,
restore_best_weights=True)

history = model.fit(
    x = [features_input, train_indices, A_input],
    y = y_train,
    batch_size = 32,
    epochs=200,
    validation_data=([features_input, val_indices, A_input], y_val),
    verbose=1,
    shuffle=False,
    callbacks=[es_callback],
)
```

```
Epoch 1/200
1/1 [==============================] - 198s 198s/step - loss: 0.6941 -
acc: 0.4400 - val_loss: 0.6409 - val_acc: 0.7750
Epoch 2/200
1/1 [==============================] - 95s 95s/step - loss: 0.6443 -
acc: 0.7200 - val_loss: 0.5901 - val_acc: 0.7750
Epoch 3/200
1/1 [==============================] - 98s 98s/step - loss: 0.6011 -
acc: 0.7200 - val_loss: 0.5447 - val_acc: 0.7750
Epoch 4/200
1/1 [==============================] - 98s 98s/step - loss: 0.5575 -
acc: 0.7200 - val_loss: 0.5129 - val_acc: 0.7750
Epoch 5/200
1/1 [==============================] - 109s 109s/step - loss: 0.5320 -
acc: 0.7200 - val_loss: 0.5031 - val_acc: 0.7750
Epoch 6/200
1/1 [==============================] - 105s 105s/step - loss: 0.5229 -
acc: 0.7200 - val_loss: 0.5026 - val_acc: 0.7750
Epoch 7/200
1/1 [==============================] - 100s 100s/step - loss: 0.5115 -
acc: 0.7200 - val_loss: 0.4909 - val_acc: 0.7750
Epoch 8/200
1/1 [==============================] - 107s 107s/step - loss: 0.4825 -
acc: 0.7200 - val_loss: 0.4701 - val_acc: 0.7750
Epoch 9/200
1/1 [==============================] - 108s 108s/step - loss: 0.4673 -
acc: 0.7200 - val_loss: 0.4464 - val_acc: 0.7750
Epoch 10/200
1/1 [==============================] - 100s 100s/step - loss: 0.4274 -
acc: 0.7250 - val_loss: 0.4277 - val_acc: 0.7850
Epoch 11/200
1/1 [==============================] - 121s 121s/step - loss: 0.4138 -
acc: 0.7650 - val_loss: 0.4151 - val_acc: 0.8150
Epoch 12/200
1/1 [==============================] - 107s 107s/step - loss: 0.3863 -
acc: 0.8250 - val_loss: 0.4040 - val_acc: 0.8300
Epoch 13/200
```

```
1/1 [==============================] - 109s 109s/step - loss: 0.3743 -
acc: 0.8500 - val_loss: 0.3936 - val_acc: 0.8300
Epoch 14/200
1/1 [==============================] - 116s 116s/step - loss: 0.3418 -
acc: 0.8800 - val_loss: 0.3862 - val_acc: 0.8350
Epoch 15/200
1/1 [==============================] - 112s 112s/step - loss: 0.3133 -
acc: 0.9100 - val_loss: 0.3833 - val_acc: 0.8450
Epoch 16/200
1/1 [==============================] - 108s 108s/step - loss: 0.3177 -
acc: 0.8900 - val_loss: 0.3837 - val_acc: 0.8450
Epoch 17/200
1/1 [==============================] - 102s 102s/step - loss: 0.3038 -
acc: 0.9100 - val_loss: 0.3865 - val_acc: 0.8450
Epoch 18/200
1/1 [==============================] - 113s 113s/step - loss: 0.2697 -
acc: 0.9150 - val_loss: 0.3878 - val_acc: 0.8500
Epoch 19/200
1/1 [==============================] - 106s 106s/step - loss: 0.2693 -
acc: 0.9100 - val_loss: 0.3852 - val_acc: 0.8600
Epoch 20/200
1/1 [==============================] - 114s 114s/step - loss: 0.2485 -
acc: 0.9150 - val_loss: 0.3828 - val_acc: 0.8600
Epoch 21/200
1/1 [==============================] - 112s 112s/step - loss: 0.2391 -
acc: 0.9150 - val_loss: 0.3839 - val_acc: 0.8600
Epoch 22/200
1/1 [==============================] - 103s 103s/step - loss: 0.2421 -
acc: 0.9050 - val_loss: 0.3910 - val_acc: 0.8650
Epoch 23/200
1/1 [==============================] - 108s 108s/step - loss: 0.2298 -
acc: 0.9150 - val_loss: 0.4014 - val_acc: 0.8650
Epoch 24/200
1/1 [==============================] - 115s 115s/step - loss: 0.2101 -
acc: 0.9200 - val_loss: 0.4156 - val_acc: 0.8650
Epoch 25/200
1/1 [==============================] - 112s 112s/step - loss: 0.1987 -
acc: 0.9050 - val_loss: 0.4329 - val_acc: 0.8650
Epoch 26/200
1/1 [==============================] - 110s 110s/step - loss: 0.2120 -
acc: 0.9300 - val_loss: 0.4421 - val_acc: 0.8650
Epoch 27/200
1/1 [==============================] - 119s 119s/step - loss: 0.1976 -
acc: 0.9300 - val_loss: 0.4433 - val_acc: 0.8600
Epoch 28/200
1/1 [==============================] - 108s 108s/step - loss: 0.2059 -
acc: 0.9250 - val_loss: 0.4445 - val_acc: 0.8550
Epoch 29/200
1/1 [==============================] - 113s 113s/step - loss: 0.1641 -
acc: 0.9350 - val_loss: 0.4507 - val_acc: 0.8650
```

```
Epoch 30/200
1/1 [==============================] - 118s 118s/step - loss: 0.1655 -
acc: 0.9350 - val_loss: 0.4608 - val_acc: 0.8550
```

Model Evaluation

Now that we have the trained model, let's evaluate its accuracy on the test set we've set aside.

```
test_preds = model.predict([features_input, test_indices, A_input])
```

```
def evaluate_preds(true, pred):
    auc = roc_auc_score(true, pred)
    pr = average_precision_score(true, pred)
    bin_pred = [1 if p > 0.5 else 0 for p in pred]
    f_score = f1_score(true, bin_pred)
    print('ROC AUC:', auc)
    print('PR AUC:', pr)
    print('F1 score:', f_score)
    print(confusion_matrix(true, bin_pred, normalize='true'))

    return auc, pr, f_score



auc, pr, f_score =
evaluate_preds(test_targets.ravel(),test_preds[0].ravel())
```

```
ROC AUC: 0.8855380533388334
PR AUC: 0.7430231163326726
F1 score: 0.6831320233159065
[[0.95231726 0.04768274]
 [0.41025109 0.58974891]]
```

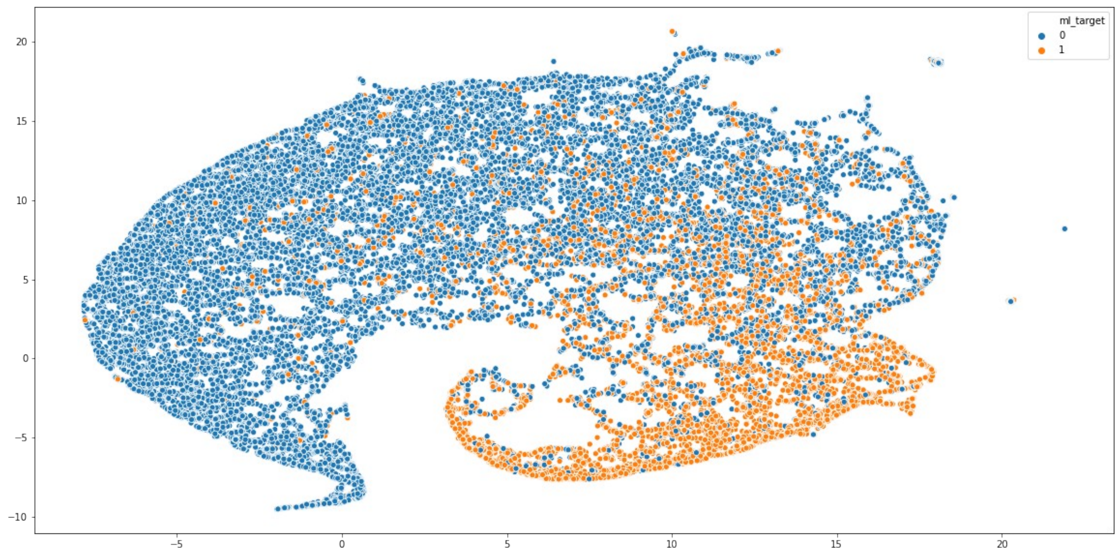
We're getting a ROC AUC score of 0.89 with just 200 labelled examples, not bad at all. Let's visualise what the model has learned by accessing the embeddings before the classification layer.

```
embedding_model = Model(inputs=x_inp, outputs=model.layers[-2].output)
all_indices = get_node_indices(G, targets.index)
emb = embedding_model.predict([features_input, all_indices, A_input])
emb.shape
```

```
(1, 37700, 32)
```

```
u = umap.UMAP(random_state=42)
umap_embs = u.fit_transform(emb[0])
```

```
plt.figure(figsize=(20,10))
ax = sns.scatterplot(x = umap_embs[:, 0], y = umap_embs[:, 1], hue =
targets['ml_target'])
```



As you can see, two classes are quite distinctly clustered in the opposite sides of the graph. Yet, there's some degree of mixing in the center of the plot, which can be expected because ML and web developers still have a lot in common.

We can also put the results of GCN into perspective by training and evaluating a Random Forest model trained with the same samples.

```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier()

rf.fit(node_features.loc[train_pages.index, :], train_targets.ravel())

test_preds = rf.predict_proba(node_features.loc[test_pages.index, :])
[:, 1]
evaluate_preds(test_targets.ravel(), test_preds)
```

```
ROC AUC: 0.8409632000657918
PR AUC: 0.7091492821465811
F1 score: 0.5649756298482873
[[0.96987212 0.03012788]
 [0.57215637 0.42784363]]

(0.8409632000657918, 0.7091492821465811, 0.5649756298482873)
```

Using the GCN we get about 0.04 increase to the ROC-AUC and PR-AUC which means that the graph data indeed adds useful information to the classification problem. Let's see what now what would be the performance if we had more labelled data avaliable.

Adding More Data

To make these experiments faster and less complicated, let's now use the StellarGraph API fully. Since you understand what's happening under the hood, there's nothing wrong with making your life easier! We're going to run the experiment with 1000 labelled nodes but feel free to choose your own parameters here.

```
# 1000 training examples
train_pages, test_pages = train_test_split(targets, train_size=1000)
val_pages, test_pages = train_test_split(test_pages, train_size=500)

train_targets =
target_encoding.fit_transform(train_pages['ml_target'])
val_targets = target_encoding.transform(val_pages['ml_target'])
test_targets = target_encoding.transform(test_pages['ml_target'])
```

```
# Initialise the generator
generator = FullBatchNodeGenerator(G, method="gcn")

# Use the .flow method to prepare it for use with GCN
train_gen = generator.flow(train_pages.index, train_targets)
val_gen = generator.flow(val_pages.index, val_targets)
test_gen = generator.flow(test_pages.index, test_targets)
```

```
Using GCN (local pooling) filters...
```

Building the GCN model is also extremely easy with `stellargraph`.

```
# Build necessary layers
gcn = GCN(
    layer_sizes=[32, 32], activations=["relu", "relu"],
generator=generator, dropout=0.5
)

# Access the input and output tensors
x_inp, x_out = gcn.in_out_tensors()

# Pass the output tensor through the dense layer with sigmoid
predictions = layers.Dense(units=train_targets.shape[1],
activation="sigmoid")(x_out)
```

```
model = Model(inputs=x_inp, outputs=predictions)
model.compile(
    optimizer=optimizers.Adam(lr=0.01),
    loss=losses.binary_crossentropy,
    metrics=["acc"],
)
```

You can see that the `stellargraph` integrates with Keras very seamlessly which makes workng with it so straightforward. Now, we can train the model in the same way we did before. The only difference is that we don't need to worry about providing all the inputs to the model, as the `generator` objects take care of it.

```
history = model.fit(
    train_gen,
```

```
    epochs=200,
    validation_data=val_gen,
    verbose=2,
    shuffle=False,  # this should be False, since shuffling data means
shuffling the whole graph
    callbacks=[es_callback],
)
```

```
Epoch 1/200
1/1 - 4s - loss: 0.2291 - acc: 0.9020 - val_loss: 0.2877 - val_acc:
0.9020
Epoch 2/200
1/1 - 4s - loss: 0.2227 - acc: 0.9110 - val_loss: 0.2937 - val_acc:
0.9040
Epoch 3/200
1/1 - 4s - loss: 0.2170 - acc: 0.9180 - val_loss: 0.2942 - val_acc:
0.9040
Epoch 4/200
1/1 - 4s - loss: 0.2086 - acc: 0.9200 - val_loss: 0.2935 - val_acc:
0.9000
Epoch 5/200
1/1 - 3s - loss: 0.1944 - acc: 0.9280 - val_loss: 0.2978 - val_acc:
0.9020
Epoch 6/200
1/1 - 3s - loss: 0.2048 - acc: 0.9220 - val_loss: 0.2989 - val_acc:
0.9020
Epoch 7/200
1/1 - 4s - loss: 0.1881 - acc: 0.9220 - val_loss: 0.3019 - val_acc:
0.9040
Epoch 8/200
1/1 - 3s - loss: 0.1781 - acc: 0.9330 - val_loss: 0.3110 - val_acc:
0.9000
Epoch 9/200
1/1 - 3s - loss: 0.1698 - acc: 0.9380 - val_loss: 0.3133 - val_acc:
0.8980
Epoch 10/200
1/1 - 4s - loss: 0.1720 - acc: 0.9370 - val_loss: 0.3074 - val_acc:
0.8940
Epoch 11/200
1/1 - 3s - loss: 0.1752 - acc: 0.9300 - val_loss: 0.3114 - val_acc:
0.8920
```

```
new_preds = model.predict(test_gen)
```

```
auc, pr, f_score =
evaluate_preds(test_targets.ravel(),new_preds[0].ravel())
```

```
ROC AUC: 0.8967718089804773
PR AUC: 0.7643095230146493
F1 score: 0.7103994490358128
```

```
[[0.92937979 0.07062021]
 [0.33722425 0.66277575]]
```

The test scores have imporved as expected, so adding more data can still lead to a better model.