Лабораториска вежба 2
Мрежни вгнездувања

GEM Python библиотека

Palash Goyal и Emilio Ferrara имаат објавено истражување за методите за мрежните вгнездувања (линк - arxiv). За целите на ова истражување имаат имплементирано неколку типови на методи за мрежни вгнездувања, кои се споредени во трудот. Имплементациите се комбинирани со библиотека наречена GEM (Graph Embedding Methods) која е јавно достапна преку интернет на следната GitHub страна.

Во оваа библиотека се имплементирани многу најсовремени техники за мрежни вгнездувања вклучувајќи ги Locally Linear Embedding, Laplacian Eigenmaps, Graph Factorization, HOPE, SDNE и node2vec. Рамката имплементира неколку функции за да го оцени квалитетот на добиеното вградување вклучувајќи реконструкција на графови, предвидување на врски, визуелизација и класификација на јазли. Поддржува многубројни резултати за реконструкција на рабови, вклучувајќи ја и косинусната сличност и евклидовото растојание.

Во оваа лабораториска вежба потребни се методите Laplacian Eigenmaps, node2vec и SDNE. За да се креира класа инстанца од оваа класа со употреба на GEM имплементациите се користи следниот код:

```
from gem.embedding.sdne import SDNE
from gem.embedding.node2vec import node2vec
from gem.embedding.lap import LaplacianEigenmaps

method1 = LaplacianEigenmaps(d=2)


method2 = node2vec(d=2, max_iter=1, walk_len=80, num_walks=10,
                   con_size=10, ret_p=1, inout_p=1)


method3 = SDNE(d=2, beta=5, alpha=1e-5, nu1=1e-6, nu2=1e-6, K=3,
               n_units=[50, 15, ], n_iter=50, xeta=0.01,
n_batch=500,
               modelfile=['enc_model.json', 'dec_model.json'],
               weightfile=['enc_weights.hdf5', 'dec_weights.hdf5'])
```

Разгледајте ја документацијата на конструкторот на секоја класа за да знаете кој параметар треба да се зададе со соодветното име. Откако ќе ја креирате инстанцата од соодветниот метод, со повик на функцијата

```
method.learn_embedding(graph=graph, edge_f=None,
                       is_weighted=False, no_python=True)
```

можете да ги креирате векторите. Излезот од оваа функција е numpy матрица во која се сместени векторите, подредени според редоследот на јазлите кој го дава **`graph.nodes()`**.

Податочно множество – Cora

Податочното множество Cora се состои од 2 708 научни публикации класифицирани во една од седум класи. Мрежата се состои од 5 429 врски. Секој јазол претставува научен труд, а рабовите помеѓу јазлите претставуваат цитирана врска помеѓу двата документи. Секоја публикација во множеството е опишана преку вектор со 0/1 вредности кој покажува отсутност/присутност на соодветниот збор од речникот во насловот на публикацијата. Речникот се состои од 1 433 уникатни зборови. Податочното множество може да се преземе од следниот линк. Датотеката README обезбедува повеќе детали.

Задачи

Задача 1 (3 поени)

Во оваа задача треба да се имплементира класификација на јазлите во неколку чекори. Најпрво потребно е графот да се вчита, да се одреди тренирачко и тестирачко множество на јазли и да се енкодираат класите во соодветни векторски репрезентации. Кодот за овој дел е даден во како почетен код во скриптата ***`lab2_starter_code.py`***. За класификација на јазлите треба да се дефинираат и испробаат неколку пристапи. Евалуацијата на различните пристапи е дадена во скриптата, и се изведува преку пресметување на вредностите на accuracy, F1-micro и F1-macro.

Првиот пристап е класификација на јазлите од податочното множество Cora преку насловот на публикациите. Секој наслов е претставен како вектор од 1 433 збора кој има вредност 0 доколку даденият збор не се наоѓа во насловот, и вредност 1 доколку зборот се наоѓа во насловот. Искористете го RandomForest методот за да истренирате класификатор (или некој друг класификатор по ваша желба), кој на влез го добива векторот со насловот за дадена публикација, а излезот е предвидената класа на публикацијата. Користете 100 како параметар за бројот на дрва во овој класификатор:

```
classifier = RandomForestClassifier(n_estimators=100,
random_state=0)

classifier.fit(train_features, train_targets)

predictions = classifier.predict(test_features)
```

Направете нови модели на класификација за множеството кои на влез ги добиваат карактеристиките на даденият јазел и карактеристиките на неговите N соседи (вредноста N определете ја според степенот на јазлите во даденият граф). Тестирајте ги моделите со тестирачкото множество и споредете ги со претходниот модел. За тестирање на методот користете ги јазлите за тестирање и пресметајте ги мерките за евалуација имплементирани во функцијата **`calculate_metrics(test_targets, predictions)`**. Дали се добиваат подобри вредности со вклучување на карактеристиките на соседните јазли?

Задача 2 (3 поени)

Со употреба на имплементациите од GEM креирајте мрежни вгнездувања добиени од методите Laplacian Eigenmaps, node2vec и SDNE. Зачувајте ги пресликаните вектори во фајлови за секој метод посебно. За зачувување и вчитување на векторите може да се користат следните функции:

```
def save_embeddings(file_path, embs, nodes):
    """Save node embeddings

    :param file_path: path to the output file
    :type file_path: str
    :param embs: matrix containing the embedding vectors
    :type embs: numpy.array
    :param nodes: list of node names
    :type nodes: list(int)
    :return: None
    """
    with open(file_path, 'w') as f:
        f.write(f'{embs.shape[0]} {embs.shape[1]}\n')
        for node, emb in zip(nodes, embs):
            f.write(f'{node} {" ".join(map(str, emb.tolist()))}\n')

def read_embeddings(file_path):
    """ Load node embeddings

    :param file_path: path to the embedding file
    :type file_path: str
    :return: dictionary containing the node names as keys
             and the embeddings vectors as values
    :rtype: dict(int, numpy.array)
    """
    with open(file_path, 'r') as f:
```

```
        f.readline()
        embs = {}
        line = f.readline().strip()
        while line != '':
            parts = line.split()
            embs[int(parts[0])] = np.array(list(map(float,
parts[1:])))
            line = f.readline().strip()
    return embs
```

При генерирањето на репрезентациските вектори со методот node2vec, во оваа лабораториска вежба користете ги следните вредности за параметрите:

- d = 50 – големина на репрезентациските вектори
- k = 20 – должина на случајните патеки
- n = 10 – број на случајни патеки за секој јазел
- c = 10 – големина на контекстот за оптимизација
- p = 4 – параметарот за враќање
- q = 1 – влез-излез параметар
- e = 3 – број на епохи на оптимизацијата

За да изградите 50-димензионални мрежни вектори со SDNE во оваа лабораториска вежба користете ги следните параметри:

- d = 50 – големина на репрезентациските вектори
- α = 1 – тежина за целта која ја регулира блискоста од прв ред
- β = 5 – казна во целта за блискоста од прв ред
- n = [100, 50] – енкодерот и декодерот се состојат од 2 скриени слоеви со големини 100 и 50.
- nu1 = 0, 000001 – L1 регуларизација
- nu2 = 0,000001 – L2 регуларизација

Задача 3 (4 поени)

Истренирајте го моделите за класификација со влезни вектори со секој од методите за пресликување на јазлите од претходната задача, а потоа направете споредба на методите со тестирачкото множество според мерките за евалуација. За тестирање на методот користете ги јазлите за тестирање и пресметајте ги мерките за евалуација имплементирани во функцијата **`calculate_metrics(test_targets, predictions)`**. Какви се резултатите во споредба со моделот кој на влез ги добива карактеристиките на тековниот јазел и моделот кој дополнително ги вклучува карактеристиките на соседните јазли?