Deep Learning and Data Science (CSE5851) **Assignment7**

Department of Statistics and Data Science **2020321803 Song minsoo**

April 22, 2021

1. [Binary Classification] Perform a classification task based on **logistic regression** using the set of embedding vectors as input. Use one of the files "karate_club.adjlist" or "karate_club.edgelist", which correspond to the adjacency list and edge list, respectively, so that you work on the dataset for the Zachary's karate club network. **Sample** a portion of the labeled nodes at random, and use them as training data, where the rest of the nodes are used as test data. **Find the average F1 score by repeating your experiments 10 times** for a given setting. **Provide a table** that shows the results according to different portions of the **labeled nodes varying from 5% to 65% in increment of 10%** (i.e., {5,15, ,65}%). **Make discussions** on resulting trends.

- Note 1: You should use three embedding methods including matrix factorization, DeepWalk with softmax, and DeepWalk with hierarchical softmax.
- Note 2: Set hyperparameters arbitrarily.
- Note 3: Refer to the file "karate_label" to see the label of each node. (Also, refer to the figure below.)

- First, Let's see the embedding result.
  - I think that the DeepWalk with softmax has the best embedding vectors. And Matrix Factorization has the worst one.
  - Lets' check with the classification task using embedding vectors.

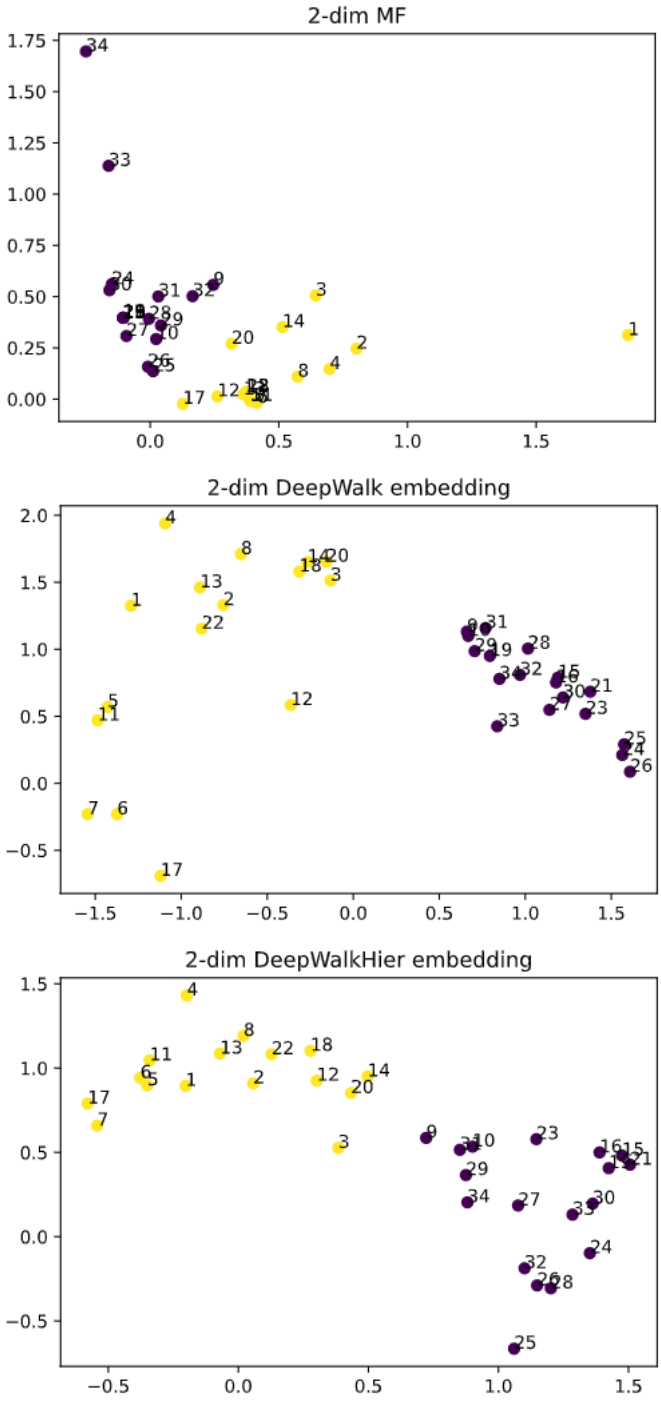
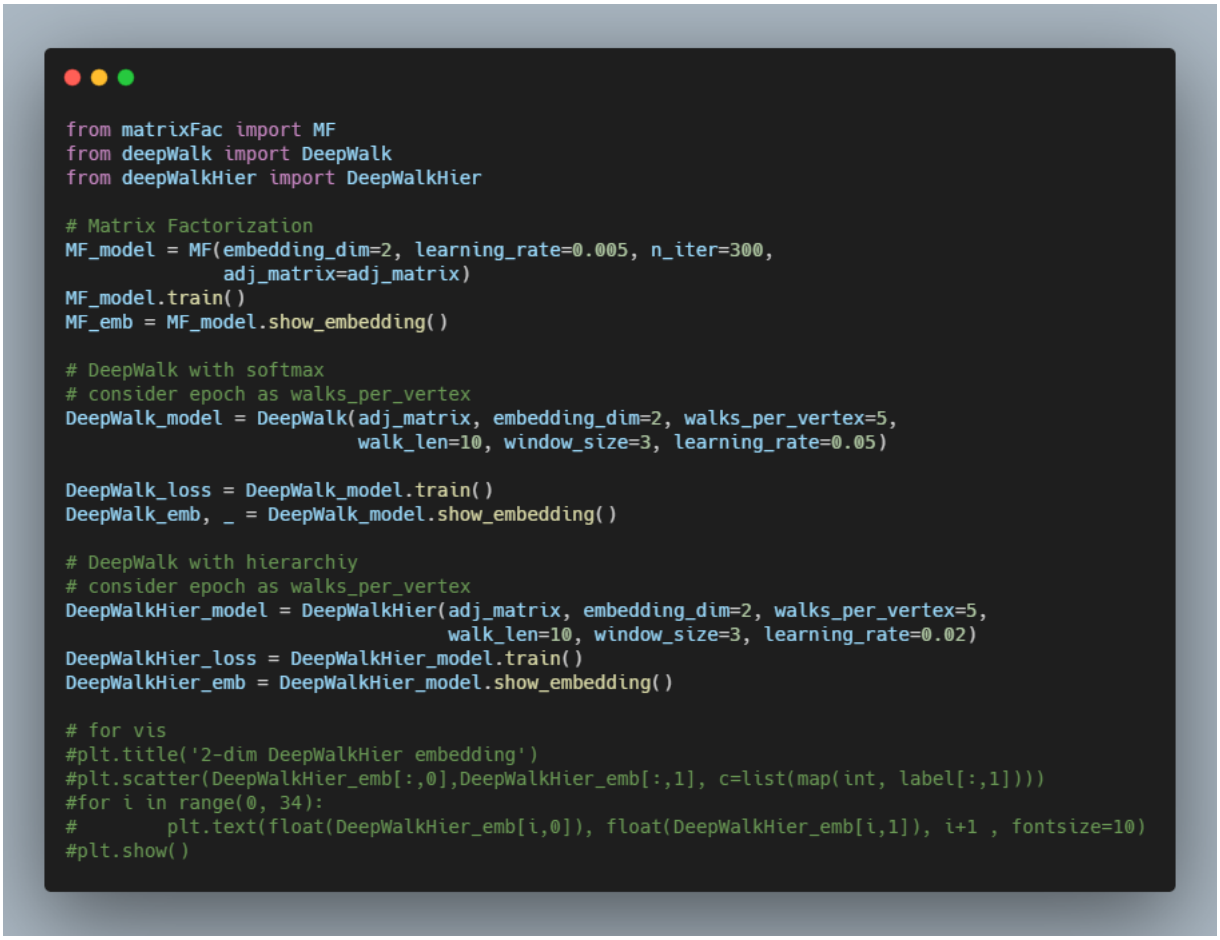


Figure 1: Embedding result of 3 models

- code



```python
from matrixFac import MF
from deepWalk import DeepWalk
from deepWalkHier import DeepWalkHier

# Matrix Factorization
MF_model = MF(embedding_dim=2, learning_rate=0.005, n_iter=300,
              adj_matrix=adj_matrix)
MF_model.train()
MF_emb = MF_model.show_embedding()

# DeepWalk with softmax
# consider epoch as walks_per_vertex
DeepWalk_model = DeepWalk(adj_matrix, embedding_dim=2, walks_per_vertex=5,
                          walk_len=10, window_size=3, learning_rate=0.05)

DeepWalk_loss = DeepWalk_model.train()
DeepWalk_emb, _ = DeepWalk_model.show_embedding()

# DeepWalk with hierarchiy
# consider epoch as walks_per_vertex
DeepWalkHier_model = DeepWalkHier(adj_matrix, embedding_dim=2, walks_per_vertex=5,
                                  walk_len=10, window_size=3, learning_rate=0.02)
DeepWalkHier_loss = DeepWalkHier_model.train()
DeepWalkHier_emb = DeepWalkHier_model.show_embedding()

# for vis
#plt.title('2-dim DeepWalkHier embedding')
#plt.scatter(DeepWalkHier_emb[:,0],DeepWalkHier_emb[:,1], c=list(map(int, label[:,1])))
#for i in range(0, 34):
#        plt.text(float(DeepWalkHier_emb[i,0]), float(DeepWalkHier_emb[i,1]), i+1 , fontsize=10)
#plt.show()
```

Figure 2: making embedding code

- code
  - I got the embedding vectors using 3models which is completed before assignments.
  - This is code checking the embedding vectors' quality with logistic regression.

```python
# to use stratified split
# change ratio from 0.05 to 0.07
# because 34 * 0.05 < 2
train_ratio = [0.07, 0.15, 0.25, 0.35, 0.45, 0.55, 0.65]

f1_result = pd.DataFrame()
f1_result['ratio'] = train_ratio
f1_result['MF'] = 0
f1_result['DeepWalk'] = 0
f1_result['DeepWalk hier'] = 0

emb_dic = {}
emb_dic['MF'] = MF_emb
emb_dic['DeepWalk'] = DeepWalk_emb
emb_dic['DeepWalk hier'] = DeepWalkHier_emb

from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score
from sklearn.linear_model import LogisticRegression

# make f1 result table!
for model in ['MF','DeepWalk','DeepWalk hier']:
    test_result = []
    f1_scores = [0]*10
    for tr in train_ratio :
        # try 10 times
        for i in range(0,10):
            X_train, X_test, y_train, y_test = train_test_split(
                emb_dic[model], label[:,1],
                train_size=tr,
                stratify=label[:,1] # use stratified split
            )

            cls_model = LogisticRegression()
            cls_model.fit(X_train, y_train)
            y_pred = cls_model.predict(X_test)

            f1_scores[i] = f1_score(y_test, y_pred, average='binary')
        # average f1 score for catch the randomness
        test_result.append(np.mean(f1_scores))
    f1_result[model] = test_result
```

Figure 3: making result table code

- f1 score result
  - As train ratio is larger, overall F1 is larger.
  - F1 score (Expected result!)
    * DeepWalk $\geq$ DeepWalk Hierarchical softmax $>$ MF
    * We usually use train data more than valid(test) data, so DeepWalk (with softmax) is the best choice.

	ratio	MF	DeepWalk	DeepWalk hier
0	0.07	0.866476	0.934597	0.942415
1	0.15	0.304226	0.915911	0.920234
2	0.25	0.975264	0.970909	0.988000
3	0.35	0.960476	0.995238	0.990476
4	0.45	0.976471	1.000000	0.994118
5	0.55	0.958095	1.000000	0.973333
6	0.65	0.963636	1.000000	1.000000

Figure 4: f1 score result screenshot