데이터 사이언스

recommender system 구현.

2015004475 김태훈

1. 구동 환경

OS: Windows 10 64비트

Language: Python 3.7.10

2. 프로그램 구조

1. 데이터 읽어오기.
2. 읽어온 데이터 인덱싱.
3. Matrix factorization을 위해 Matrix U, V를 정의하고 이를 곱해 prediction matrix 정의.
4. cost와 optimizer를 정의하여 U, V를 학습.
5. prediction 결과 파일 저장.

3. 코드 설명

```
import os
import sys
import time
import numpy as np
import pandas as pd
import tensorflow as tf

# Hyper params
learning_rate = 0.001
training_epochs = 3000
feature_len = 100
```

필요 모듈 임포트와 학습을 위한 하이퍼 파라미터 설정.

learning_rate: adam optimizer에 적용할 learning rate로 자주 사용하는 값인 0.001 사용.

training_epochs: 학습 반복 epoch 수 설정.

feature_len: matrix factorization에서의 U, V 벡터의 shape를 정하는 파라미터. 학습 데이터들이 약 (1k, 1k) 정도의 shape를 가져 적절한 값으로 지정.

```
def read_file(train, test):
    df_train = pd.read_csv(train, sep='\t', names=['user', 'item', 'rate', 'time'])
    df_test = pd.read_csv(test, sep='\t', names=['user', 'item', 'rate', 'time'])

    return df_train, df_test


def write_file(path, data):
    try:
        os.remove(path)
    except FileNotFoundError:
        pass

    with open(path, 'a') as f:
        f.writelines(data)
```

read_file()

주어진 학습 데이터 파일과 테스트 데이터 파일을 읽어 pd dataframe 형태로 반환.

각 컬럼은 순서대로 user, item, rate, timestamp 데이터를 담고 있음.

write_file()

결과 파일 경로와 데이터를 받아 주어진 경로에 데이터 파일을 생성. 이미 존재하면 삭제하고 새로 생성.

```
def adam_optimizer(cost):
    train_step = tf.train.AdamOptimizer(
        learning_rate=learning_rate,
        beta1=0.9,
        beta2=0.999,
        epsilon=1e-08,
        use_locking=False,
        name='Adam'
    ).minimize(cost)

    return train_step
```

주어진 코스트를 받아 Adam optimizer를 통해 minimize 하도록 하는 train step을 반환.

```
def build_R_matrix(num_user, num_item):
    # Matrix factorization, R(u, i) = U(u, f) * V(f, i)
    U = tf.Variable(tf.random_uniform([num_user, feature_len]))
    V = tf.Variable(tf.random_uniform([feature_len, num_item]))
    result = tf.matmul(U, V)
    result_flatten = tf.reshape(result, [-1])

    return result, result_flatten
```

Matrix U, V가 곱해지면 원래 matrix R이 복구되어야 함. R(u, i) = U(u, f) * V(f, i) 로 u는 유저의 수, i는 아이템의 수, f는 사전 지정된 feature 벡터의 shape 지정 파라미터.

result는 U와 V를 곱한 복원 행렬이고 result_flatten은 결과 행렬을 1차원 선형 벡터 형태로 reshape 한 결과.

```
if __name__ == '__main__':
    start_time = time.time()

    train_file = sys.argv[1]
    test_file = sys.argv[2]
    result_file_path = train_file.split('/')[-1] + '_prediction.txt'

    df_train, df_test = read_file(train_file, test_file)
```

터미널에서 본 파이썬 스크립트를 사전 지정된 arguments와 함께 실행시킨 경우.

argument를 통해 읽어야 할 데이터 경로를 지정 후 결과 파일의 이름을 정의.

df_train, df_test에 각각 학습, 테스트 데이터를 읽어 dataframe 형태로 저장.

```
    # 인덱스가 1부터 시작하니 한 칸씩 당겨줌.
    user_indices = [x - 1 for x in df_train.user.values]
    item_indices = [x - 1 for x in df_train.item.values]
    # pre_preference = [1] * df_train.rate.values.size
    rates = df_train.rate.values

    num_user = max(df_train.user.max(), df_test.user.max())
    num_item = max(df_train.item.max(), df_test.item.max())

    result, result_flatten = build_R_matrix(num_user, num_item)
```

데이터 프레임으로 읽어온 데이터의 row 인덱스는 1부터 시작하기 때문에 0부터 시작하도록 조정.

rates에 유저의 레이팅을 저장.

num_user, num_item은 각각 학습 데이터와 테스트 데이터에서 등장하는 유저, 아이템의 id 중 가장 큰 숫자 값으로 해당 id 만큼의 유저 수, 아이템 수가 있다고 가정.

유저 수와 아이템 수를 통해 result, result_flatten에 행렬 U, V 곱 매트릭스 형태를 저장.

```
  # rating
  # result_flatten에서 rate를 가져옴. 2차원 행렬에서 (user, item)의 값이 1차원 벡터의 아래 indices 파라미터 값에 해당함.
  R = tf.gather(result_flatten, user_indices * tf.shape(result)[1] + item_indices)

  # SAE
  # cost_pre = tf.reduce_sum(tf.abs(R - pre_preference))
  cost = tf.reduce_sum(tf.abs(R - rates))

  # train_step_pre = adam_optimizer(cost_pre)
  train_step = adam_optimizer(cost)
```

result_flatten은 레이팅 정보를 담은 result의 1차원 형태인데, (A, B) shape의 2차원 행렬에서 I, j 위치의 레이팅 값은 1차원 벡터의 인덱스 i * B + j 위치의 값과 같다.

cost는 예측 결과와 실제 결과의 Sum Absolute Error로 정의하였다. SAE와 MSE 두 loss를 실험해 본 결과 SAE의 성능이 훨씬 좋게 나왔는데, sparse matrix의 factorization을 수행하며 부정확한 예측치, outlier, 가 많이 나오는 특성상 outlier에 좀 더 robust한 SAE가 더 나은 것으로 생각된다.

Adam optimizer를 이용해 Training step을 정의하였다. adam의 하이퍼 파라미터는 많이 사용하는 well known 값으로 이용.

```
# pre_ckpt_path = "output/pre_preference/"
ckpt_path = f"output/{train_file.split('/')[-1]}/"
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    print("학습시작")

    for epoch in range(training_epochs):
        c, _ = sess.run([cost, train_step])
        if epoch % 100 == 0:
            print(f"Epoch: {epoch}, cost: {c}")

    saver = tf.train.Saver()
    saver.save(sess, ckpt_path)
    saver.restore(sess, ckpt_path)
```

학습 모델 체크포인트 저장 경로 지정. 세션 가동 후 tf 변수 초기화, 지정된 epoch 수만큼 모델 학습.

학습된 모델을 체크포인트로 저장, 복원.

```
    print('Training data restoration...')
    r_hat = np.clip(sess.run(result), 1, 5)

    for u, v, r in df_train[['user', 'item', 'rate']].values[:10]:
        print(f'Rating for user: {str(u)} for item {str(v)}: {str(r)}, prediction: {str(r_hat[u-1][v-1])}')

    print('Test data prediction...')
    output = []
    for u, v, r in df_test[['user', 'item', 'rate']].values:
        line = (str(u) + '\t' + str(v) + '\t' + str(r_hat[u-1][v-1]) + '\n')
        output.append(line)

    write_file(result_file_path, output)
    print(f'Time: {time.time() - start_time}')
```

학습 데이터의 복원 정도를 보기 위해 10개 샘플에 대해 예측 결과 출력.

테스트 데이터에 지정된 유저-아이템-레이팅 세트를 출력 문자열 형태로 만들어 결과 파일에 출력.

4. 소스 컴파일 가이드

python 3.7.10으로 작성된 스크립트.

conda2를 이용해 가상환경을 구축하고

```
numpy~=1.20.2
pandas~=1.2.4
tensorflow~=1.13.1
```

requirements.txt에 포함된 모듈을 인스톨.

실행 명령 예시) data가 "프로젝트 루트/data-2/u1.base", "프로젝트 루트/data-2/u1.test" 인 경우

```
(recommender_system) D:\수업\데사\실습\2021_ite4005_2015004475\recommender>python recommender.py data-2/u1.base data-2/u1.test
```

결과 파일은 프로젝트 루트 경로에 u#.base_prediction.txt 파일로 저장됨.