Longterm Project : Recommendation System

2017029970 우원진

1. Summary of my Algorithm

Recommendation System을 만들때 여러가지 접근 방법이 있지만, Collaborative Filtering을 써야겠다고 생각했다. 그 중에서도 Machine Learning을 이용한 Matrix Factorization을 사용했다. Matrix Factorization을 위해 p(user수 x 12), q(movie수 x 12)로 나누는데 초기값은 모두 0과 1 사이의 랜덤한 값을 주었다.(여기서 왜 column의 수가 12인가는 여러값을 사용해보고 경험상 12로 정했다.) 학습을 위한 목적 함수는 $L = \sum (r_{ij} - \mu - b_i - b_j - p_i \cdot q_j)^2 + \lambda (\|p_i\|^2 + \|q_j\|^2 + b_i^2 + b_j^2)$ 를 사용했다. 여기서 rij는 i번째 유저가 j번째 영화에 rating한 실제값, M은 전체 rating된 값의 평균, bi는 user bias의 i번째 row, bj는 movie bias의 j번째 row, pi는 유저 매트릭스의 i번째 row, qj는 영화 매트릭스의 j번째 row이고, lambda는 hyper parameter로 0.1로 두고 학습하였다. 이 또한 여러번의 실험으로 정한 값이다. 이제 목적 함수를 정했으니 먼저 Uninteresting을 찾기로 했다. 유저가 rating을 매긴 영화들에 대해서는 평점이 낮든 높든 interesting한 것이기 때문에 interesting하다는 1로 바꿨다. 그리고 gradient descent를 이용하여 학습을 했다. 여기서 Epoch는 50번, learning-rate는 0.25로 설정했다. 학습을 할때 Training Set을 Scikit learn의 train_test_split을 이용해 training set과 Validation 셋으로 0.95 : 0.5 비율로 나눴다. 그리고 매번 epoch마다 validation set에 대한 rmse를 계산해 만약 rmse가 올라가면 overfitting이 발생한것이라고 생각해 patience를 4로 설정해 만약 4번이나 rmse가 높아진다면 학습을 멈추도록 설정했다. 그리고 학습이 끝나면 만들어진 p, q, b_p, b_q, mean을 가지고, np.dot(p, q.transpose()) + mean + b_p + b_q.transpose() 를 통해 모든 값이 차있는 matrix를 만들었다. 이것은 uninteresting을 찾기 위한 matrix로 threshold를 0.8로 잡아 만약 값이 0.8보다 작으면 이는 uninteresting값이라고 생각하여 해당위치의 index에 해당하는 원래 matrix에 0을 집어넣었다. 이러면 원래의 sparse matrix에 값이 몇개라도 더 채워지는 것이다. 이렇게 조금 더 채워진 matrix를 가지고 위의 학습 과정을 또 반복했다. 그리고 만들어진 matrix에서 실제 rating은 1보다 작거나, 5보다 큰 것이 없으므로 1보다 작은 값들은 1로, 5보다 큰 값들은 5로 바꿨다.

2. Detailed description of code

```
train_data, val_data = train_test_split(dataframe, test_size=0.05, random_state=123)
```

기존의 트레이닝 데이터를 train_data와 Validation을 위한 데이터로 나누는 코드이다. 여기서 Validation 데이터의 비율은 0.05이다.

```
df_table = train_data.pivot("user_id", "item_id", "rating")
for i in range(1, last_movie+1):
    if not i in df_table.columns:
        df_table[i] = np.NaN
df_table = df_table[[i for i in range(1, last_movie+1)]]
print(df_table)
```

이 코드는 만약 가장큰 movie_id가 1900인데 movie_id가 1800인 영화를 본 유저가 한명도 없을때 그래도 계산을 위해 추가 해주기 위한 코드이다.

```
def mean_squared_error_with_bias(y, p, q, b_p, b_q, mean):
    return ((y-mean-b_p-b_q-np.dot(p, q))**2+_lambda*np.sqrt(np.sum(np.square(p))) + _lambda*np.sqrt(np.sum(n
```

이 함수는 위에서 설명한 목적 함수를 나타낸 함수이다.

```
def mf_uninteresting(index_arr, validation_set):
    p = np.random.rand(row, attr_num)  # number of parameter : row * 3
    q = np.random.rand(col, attr_num)  # number of parameter : col * 3
    b_p = np.random.rand(row, 1)
    b_q = np.random.rand(col, 1)
    min_rmse_error = 9999999
    best_p = None
    best_q = None
    best_b_p = None
    best_b_q = None
    not_improved_cnt = 0
    for _ in range(k):
        d_p = [[np.zeros(attr_num), 0] for _ in range(row)]
        d_q = [[np.zeros(attr_num), 0] for _ in range(col)]
        d_b_p = [[0, 0] for _ in range(row)]
        d_b_q = [[0, 0] for _ in range(col)]
        error = 0
        for i, j in index_arr:
            p_i = p[i]
            q_j = q[j]
            b_p_i = b_p[i][0]
            b_q_j = b_q[j][0]
            a = np.dot(p_i, q_j)
            # values of derivation
            d_p[i][0] += _lambda*p_i - (1-b_p_i-b_q_j-1-a)*q_j
            d_p[i][1] += 1  # cnt
            d_q[j][0] += _lambda*q_j - (1-b_p_i-b_q_j-1-a)*p_i
            d_q[j][1] += 1
            d_b_p[i][0] += _lambda*b_p_i - (1-b_p_i-b_q_j-1-a)
```

이 함수는 위에서 설명한 uninteresting을 찾기 위해 Training하는 함수이다.

```
pre_use_mat_result = np.dot(uninteresting_p, uninteresting_q.transpose()) + 1
pre_use_mat_result += uninteresting_b_p
pre_use_mat_result += uninteresting_b_q.transpose()
print(pre_use_mat_result)
uninteresting_rating_index = np.argwhere(pre_use_mat_result < 0.8)
print(uninteresting_rating_index.shape)
```

이부분은 uninteresting을 찾기위해 트레이닝 한 후 찾아진 matrix를 사용해 0.8보다 작은 값들의 index를 찾는 부분이다.

```
original_rating_mat = df_table.fillna(-1).values
for index in uninteresting_rating_index:
    original_rating_mat[index[0]][index[1]] = 0
result_rated_indexes = np.argwhere(original_rating_mat >= 0)
rating_sum = 0
cnt = 0
for index in result_rated_indexes:
    rating_sum += original_rating_mat[index[0]][index[1]]
    cnt += 1
rating_mean = rating_sum / cnt
```

이 부분은 계산에 필요한 mean값도 찾고, uninteresting한 index에 대해 실제 matrix에 0 값

을 넣는 부분이다.

```
def mf_training(index_arr, data_arr, validation_set, mean):
    p = np.random.rand(row, attr_num)  # number of parameter : row * attr_num
    q = np.random.rand(col, attr_num)  # number of parameter : col * attr_num
    b_p = np.random.rand(row, 1)
    b_q = np.random.rand(col, 1)
    min_rmse_error = 9999999
    best_p = None
    best_q = None
    best_b_p = None
    best_b_q = None
    not_improved_cnt = 0
    for _ in range(k):
        d_p = [[np.zeros(attr_num), 0] for _ in range(row)]
        d_q = [[np.zeros(attr_num), 0] for _ in range(col)]
        d_b_p = [[0, 0] for _ in range(row)]
        d_b_q = [[0, 0] for _ in range(col)]
        acc_error = 0
        rmse_error = 0
        for i, j in index_arr:
            p_i = p[i]
            q_j = q[j]
```

이 함수는 uninteresting한 부분에 대해 0값이 채워진 orginal matrix에 대해 Matrix Factorization을 위해 학습하는 함수이다.

```
recommend_result = np.dot(p, q.transpose()) + rating_mean
recommend_result += b_p
recommend_result += b_q.transpose()
recommend_result = np.where(recommend_result < 1.0, 1, recommend_result)
recommend_result = np.where(recommend_result > 5.0, 5, recommend_result)
print(recommend_result)
```

이 부분은 Matrix Factorization을 통해 구한 matrix에 대해 1보다 작은 값은 1, 그리고 5보다 큰 값은 5로 바꿔주는 부분이다.

3. Compile and Run

```
uwonjin-ui-MacBookPro:Recommendation System woowonjin$ python3 recommender.py u1.base u1.test
```

이와 같이 명령어를 입력하면 된다. Scikit-learn 라이브러리를 사용했기 때문에 라이브러리 설치도 필요하다.

4. Others

목적 함수를 처음에는 bias를 넣지 않고 사용했는데 이때 rmse값이 약 1.02 정도 나왔다. 그리고 bias를 넣고 난 후에 약 0.98 정도가 나와서, 어떻게 하면 성능을 높일수 있을까라는 생각에 Hyper Parameter들을 계속 바꿔주다 보니 약 0.94 ~ 0.95정도의 성능까지 올라갔다. 그리고 모든 weight를 실행때마다 랜덤하게 정해주기 때문에 실행마다 약간의 성능 차이를 보인다.