Recommender System

author: 김영진 (2016025241)

Summary of algorithm

Training

1. 파라미터들을 초기화 한다.
   1. 모든 학습 데이터에서 평점의 평균을 계산한다. (μ 라고 하자.)
   2. 각 유저에 번호를 부여한다 (0번부터 $N_u - 1$까지) (N_u 는 유저의 개수)
   3. 각 아이템에 번호를 부여한다 (0번부터 $N_i - 1$까지) (N_i는 유저의 개수)
   4. 각 유저마다 길이가 k인 고유의 벡터를 생성하여 Random uniform 초기화한다. (생성된 각 벡터를 p_u라고 하자)
   5. 각 유저마다 길이가 k인 고유의 벡터를 하나 더 생성하여 Random uniform 초기화한다. (생성된 각 벡터를 x_u라고 하자)
   6. 각 유저마다 고유의 스칼라를 랜덤으로 생성한다 (생성된 각 스칼라를 t_u라고 하자)
   7. 각 유저마다 고유의 스칼라를 하나 더 생성한다 (생성된 각 스칼라를 α_u라고 하자)
   8. 각 유저마다 고유의 스칼라를 하나 더 생성한다 (생성된 각 스칼라를 b_u라고 하자)
   9. 각 아이템마다 길이가 k인 고유의 벡터를 생성하여 Random uniform 초기화한다. (생성된 각 벡터를 q_i라고 하자)
   10. 각 아이템마다 길이가 k인 고유의 벡터를 하나 더 생성하여 Random uniform 초기화한다. (생성된 각 벡터를 y_i라고 하자)
   11. 각 아이템마다 고유의 스칼라를 랜덤으로 생성한다 (생성된 각 스칼라를 b_i라고 하자)
   12. $N_i * N_t$ 개 만큼의 스칼라를 랜덤으로 생성한다 (생성된 각 스칼라를 $b_{i,Bin(t)}$라고 하자) (N_t는 하이퍼 파라미터이다)
   13. ($2k$, k) 크기의 행렬을 랜덤으로 초기화하여 W^u_{Att} 라고 하자.
   14. 길이가 k인 벡터 b^u_{Att}를 랜덤으로 생성한다.
2. 어떤 인풋 u, i, r, t, R_u, R_i, (유저 번호, 아이템 번호, 평점, 타임스탬프, 해당 유저가 구매한 적있는 모든 아이템들의 집합, 해당 아이템을 구매한 적 있는 모든 유저들의 집합)을 학습 데이터로 받았을 때, Loss 함수를 정의하자. 여기서 t는 해당 유저의 기록을 참고하여 normalize하고, $\text{Bin}(t)$의 경우 해당 아이템의 기록을 참고하여 N_t개의 Bin만큼 Discretize한다.

$$L(u,i,r,t,R_u,R_i) = \frac{1}{2}(r - \hat{r}(u,i,t,R_u,R_i))^2$$

$$\hat{r}(u,i,t,R_u,R_i) = \mu + (p_u + y)(q_i + \frac{1}{\sqrt{R_i}}\sum_{j \in R_i} x_j) + b_u + b_i + \alpha_u * \text{dev}_u(t) + b_{i,Bin(t)}$$

$$y = \sum_{j \in R_u} y_j \otimes \text{Softmax}(\text{Relu}((y_j, p_u)W^u_{\text{Att}} + b^u_{\text{Att}}))$$

$$\text{Relu}(x) = \max(0, x)$$

$$\text{Softmax}(x) = \frac{\exp x}{\sum_j \exp x_j}$$

$$\text{dev}_u(t) = \text{sign}(t - t_u) * |t - t_u|^{0.4}$$

1. 위와 같이 정의한 $L(u,i,r,t,R_u,R_i)$ 가 최소가 되도록 Gradient Descent 기반 옵티마이저를 사용해 최적화 한다. 이 실험에서는 Adam 옵티마이저를 사용하였다.

Inference

1. 어떤 인풋 u, i, t 가 주어졌을 때, 트레이닝 데이터를 참고하여 $\text{Bin}(t), R_u, R_i, t$ 를 생성한다. (여기서 t는 normalize된 값을 말한다)
2. $\hat{r}(u,i,t,R_u,R_i)$ 을 계산한다.

Detailed description of code

Model

```
def __init__(self,
        num_users,
        num_items,
        mu,
        k,
        reg_lambda,
        sz_related,
        n_time_bins,
        use_yp_attention=True,
        use_xq_attention=False,
        use_x=True,
        use_y=True,
        use_bti=True,
        use_btu=True
    ):
        super().__init__(self)
        self.num_users = num_users
        self.num_items = num_items
        self.embed_size = k
        self.mu = mu
        self.reg_lambda = reg_lambda
        self.sz_related = sz_related
```

```
        self.n_time_bins = n_time_bins
        self.use_yp_attention = use_yp_attention
        self.use_xq_attention = use_xq_attention
        self.use_x = use_x
        self.use_y = use_y
        self.use_bti = use_bti
        self.use_btu = use_btu


        '''
        Embeddings
        '''

        mask_zero = True

        self.user_embedding = tf.keras.layers.Embedding(self.num_users + 1,
self.embed_size, input_length=1,
            embeddings_regularizer=tf.keras.regularizers.L2(self.reg_lambda),
            embeddings_initializer='normal',
            mask_zero=mask_zero)
        self.user_x_embedding = tf.keras.layers.Embedding(self.num_users + 1,
self.embed_size, input_length=sz_related,
            embeddings_regularizer=tf.keras.regularizers.L2(self.reg_lambda),
            embeddings_initializer='normal',
            mask_zero=mask_zero)
        self.user_tu_embedding = tf.keras.layers.Embedding(self.num_users + 1, 1,
input_length=1,
            embeddings_regularizer=tf.keras.regularizers.L2(self.reg_lambda),
            embeddings_initializer='normal',
            mask_zero=mask_zero)
        self.user_alpha_embedding = tf.keras.layers.Embedding(self.num_users + 1, 1,
input_length=1,
            embeddings_regularizer=tf.keras.regularizers.L2(self.reg_lambda),
            embeddings_initializer='normal',
            mask_zero=mask_zero)
        self.item_embedding = tf.keras.layers.Embedding(self.num_items + 1,
self.embed_size, input_length=1,
            embeddings_regularizer=tf.keras.regularizers.L2(self.reg_lambda),
            embeddings_initializer='normal',
            mask_zero=mask_zero)
        self.item_y_embedding = tf.keras.layers.Embedding(self.num_items + 1,
self.embed_size, input_length=sz_related,
            embeddings_regularizer=tf.keras.regularizers.L2(self.reg_lambda),
            embeddings_initializer='normal',
            mask_zero=mask_zero)
        self.user_bias = tf.keras.layers.Embedding(self.num_users + 1, 1,
input_length=1,
            embeddings_regularizer=tf.keras.regularizers.L2(self.reg_lambda),
            embeddings_initializer=tf.keras.initializers.Constant(0.0),
```

```
            mask_zero=mask_zero)
        self.item_bias = tf.keras.layers.Embedding(self.num_items + 1, 1,
input_length=1,
            embeddings_regularizer=tf.keras.regularizers.L2(self.reg_lambda),
            embeddings_initializer=tf.keras.initializers.Constant(0.0),
            mask_zero=mask_zero)
        self.item_time_bin_bias = tf.keras.layers.Embedding((self.num_items + 1) *
self.n_time_bins, 1, input_length=1,
            embeddings_regularizer=tf.keras.regularizers.L2(self.reg_lambda),
            embeddings_initializer=tf.keras.initializers.Constant(0.0))
        self.yp_attention = Attention(k=self.embed_size)
        self.xq_attention = Attention(k=self.embed_size)
```

Summary에서 언급된 파라미터들을 모두 초기화한다.

```
class Attention(Layer):
    def __init__(self, k, element_wise=True):
        super(Attention, self).__init__()
        self.k = k
        att_out = k if element_wise else 1

        w_init = tf.random_normal_initializer()
        self.att_w = tf.Variable(
            initial_value=w_init(shape=(2*k, att_out), dtype=tf.float32),
            trainable=True,
        )
        self.att_b = tf.Variable(
            initial_value=w_init(shape=(att_out,), dtype=tf.float32),
            trainable=True,
        )

    def call(self, y, p):
        k = self.k

        p = tf.reshape(p, (-1, 1, k))
        p = tf.repeat(p, y.shape[1], axis=1)
        yp = tf.concat([y, p], axis=-1)
        yp = tf.matmul(yp, self.att_w) + self.att_b
        yp = tf.nn.relu(yp)
        yp = tf.nn.softmax(yp, axis=-1)

        return tf.reduce_sum(y * yp, axis=1)
```

Attention의 파라미터 초기화를 포함한 구현은 위와 같다. y, p를 concatenate하여 다시 k 차원으로 전사한 후에, 소프트맥스를 통해 어떤 y에 집중할지 정한다. 소프트맥스 결과를 통해 y 들의 weighted sum을 구한다.

일반적으로 FC를 사용하는 Attention은 concatenate결과를 스칼라 값으로 전사하여 사용하지만, 여기서는 실험 결과가 더 좋게 나오기 때문에 k차원으로 각 요소마다 따로 Attention하여 사용한다.

여기서 각 벡터나 스칼라들 중 0번 인덱스의 것들은 모두 0으로 초기화하고 업데이트 하지 않도록 한다. 이것은 이후 Inference 단계에서 처음 보는 아이템이나 유저를 위해 사용될 것이다.

```
def call(self, inputs):
        user_input, item_input, user_times, item_time_bins, user_relateds,
item_relateds = inputs

        p = self.user_embedding(user_input)
        p = tf.reshape(p, (-1, self.embed_size,))
        # p = self.user_inference(p)

        if self.use_y:
            y = self.item_y_embedding(user_relateds)
            if self.use_yp_attention:
                y = self.yp_attention(y, p)
            else:
                y = tf.reduce_sum(y, axis=1)
                y = y / tf.maximum(1.0,
tf.reshape(tf.sqrt(tf.math.count_nonzero(user_relateds, 1, dtype=tf.float32)), (-1,
1)))
        else:
            y = 0

        q = self.item_embedding(item_input)
        q = tf.reshape(q, (-1, self.embed_size,))
        # q = self.item_inference(q)

        if self.use_x:
            x = self.user_x_embedding(item_relateds)
            if self.use_xq_attention:
                x = self.xq_attention(x, q)
            else:
                x = tf.reduce_sum(x, axis=1)
                x = x / tf.maximum(1.0,
tf.reshape(tf.sqrt(tf.math.count_nonzero(item_relateds, 1, dtype=tf.float32)), (-1,
1)))
        else:
            x = 0

        pq = tf.reduce_sum((p + y) * (q + x), axis=-1)

        bu = self.user_bias(user_input)
        bu = tf.reshape(bu, (-1,))

        if self.use_btu:
            tu = self.user_tu_embedding(user_input)
            tu = tf.reshape(tu, (-1,))
            dt = tf.reshape(user_times, (-1,)) - tu
            dev_ut = tf.sign(dt) * tf.pow(tf.abs(dt), 0.4)
```

```
            alpha_u = self.user_alpha_embedding(user_input)
            alpha_u = tf.reshape(alpha_u, (-1,))
            btu = alpha_u * dev_ut
        else:
            btu = 0

        bi = self.item_bias(item_input)
        bi = tf.reshape(bi, (-1,))

        if self.use_bti:
            bti = self.item_time_bin_bias(item_input * self.n_time_bins +
item_time_bins)
            bti = tf.reshape(bti, (-1,))
        else:
            bti = 0

        return self.mu + pq + bu + btu + bi + bti
```

Summary에 언급된 $\hat{r}$ 을 그대로 구현한다.

Preprocessing

```
def _build_data(self):
        users, items = self._encode_user_item()
        data = np.array(self.observation)
        data[:, 0] = users.flatten()
        data[:, 1] = items.flatten()

        self.mu = np.mean(data[:, col_rating])

        user_times = self._normalize_time(data, col_user, self.num_users)
        item_times = self._normalize_time(data, col_item, self.num_items)
        self.user_times = user_times[0]
        self.user_min_times = user_times[1]
        self.user_max_times = user_times[2]
        self.item_times = item_times[0]
        self.item_min_times = item_times[1]
        self.item_max_times = item_times[2]

        self._build_relateds(data)

        return data
```

1. 각 유저와 아이템들에 번호를 부여
2. 평균 평점을 계산
3. 각 유저별 기록들의 최소 시간, 최고 시간을 계산
4. 각 아이템별 기록들의 최소 시간, 최고 시간을 계산

5. 각 유저, 아이템 간 한번이라도 연관되었는 지를 미리 정리

```
@staticmethod
def _normalize_time(data, by, num_by):
  '''
  normalize time by user
  '''

  inf = 0x3f3f3f3f
  min_times = np.full(num_by, inf)
  max_times = np.zeros(num_by)
  times = np.array(data[:, col_time], dtype=np.float32)

  for row, time in zip(data, times):
    pivot = row[by]
    min_times[pivot] = min(min_times[pivot], time)
    max_times[pivot] = max(max_times[pivot], time)

    for i, row in enumerate(data):
      pivot = row[by]
      min_time = min_times[pivot]
      max_time = max_times[pivot]
      interval = 1.0 if min_time == max_time else max_time - min_time
      times[i] = (times[i] - min_time) / interval

      return times, min_times, max_times
```

특정 컬럼의 값이 같은 것들 끼리 모아서 최소 시간과 최대 시간을 구한 후에, [0.0, 1.0] 사이로 interpolate한다.

```
def _build_relateds(self, data):
        n = data.shape[0]
        user_relateds = [list() for _ in range(self.num_users)]
        item_relateds = [list() for _ in range(self.num_items)]

        for user, item, _, _ in data:
            user_relateds[user].append(item + 1)
            item_relateds[item].append(user + 1)

        for i in range(self.num_users):
            user_relateds[i] = np.unique(user_relateds[i])
        for i in range(self.num_items):
            item_relateds[i] = np.unique(item_relateds[i])

        self.user_relateds = np.zeros((self.num_users, self.sz_related),
dtype=np.int64)
        self.item_relateds = np.zeros((self.num_items, self.sz_related),
dtype=np.int64)

        for index, row in enumerate(self.user_relateds):
```

```
            relateds = user_relateds[index]
            m = min(self.sz_related, len(relateds))
            row[:m] = np.array(relateds[:m])

        for index, row in enumerate(self.item_relateds):
            relateds = item_relateds[index]
            m = min(self.sz_related, len(relateds))
            row[:m] = relateds[:m]
```

각 유저와 아이템마다 관련된 아이템, 유저들을 모아서 중복되지 않도록 관리한다. 이 때 `self.sz_related` 보다 많을 경우 차례대로 그 이상의 것들은 무시한다.

여기서 각 기록의 모음을 정확히 (n, sz_related) 크기의 행렬로 표현하기 위해 빈 자리는 0으로, 나머지 자리는 원래 유저, 아이템 번호보다 1 크게 하여 저장한다.

Training

```
def fit(self, observation, batch_size=2000, epochs=0, validation_data=None):
        '''
        Build data, model, and Fit

        :param observation: numpy array which has shape (-1, 4).
            each columns must represent (user_id, item_id, time, rating)
        '''

        assert(len(observation.shape) == 2)
        assert(observation.shape[-1] == 4)

        self.observation = observation

        self.data = self._build_data()
        self.model = self._build_model()

        users = self.data[:, col_user] + 1
        items = self.data[:, col_item] + 1
        user_times = self.user_times
        user_relateds = self.user_relateds[self.data[:, col_user]]
        item_relateds = self.item_relateds[self.data[:, col_item]]
        item_time_bins = np.int64(np.minimum(self.item_times * self.n_time_bins,
self.n_time_bins - 1))
        ratings = self.data[:, col_rating]
        X = [users, items, user_times, item_time_bins, user_relateds, item_relateds]
        Y = ratings

        if validation_data is not None:
            validation_data_X = self._preprocess_prediction_data(validation_data)
            validation_data_Y = validation_data[:, col_rating]
            validation_data = (validation_data_X, validation_data_Y)
```

```
        epoch = 0
        prev_val_loss = 10000000000000000000
        while epochs == 0 or epoch < epochs:
            history = self.model.fit(X, Y,
                batch_size=batch_size,
                epochs=1,
                validation_data=validation_data)
            val_loss = history.history['val_loss'][0]
            if epochs == 0 and val_loss > prev_val_loss:
                break
            prev_val_loss = val_loss
            epoch += 1
```

1. 같은 아이템인 기록별로 normalize된 time을 기반으로 N_t을 곱하여 discretize한다.
2. 미리 정리한 관련 아이템, 유저 정보를 바탕으로 R_u, R_i를 정리한다.
3. 주어진 `epochs` 번 만큼 트레이닝을 진행한다. 만약 `epochs`가 0일 경우 validation loss가 이전보다 증가할 때 까지 진행한다.

Inference

```
def _preprocess_prediction_data(self, data):
        n = data.shape[0]
        rel_pad = np.zeros(self.sz_related)
        users = np.empty(n, dtype=np.int64)
        items = np.empty(n, dtype=np.int64)
        user_times = np.empty(n, dtype=np.float32)
        item_time_bins = np.empty(n, dtype=np.int64)
        user_relateds = np.empty((n, self.sz_related))
        item_relateds = np.empty((n, self.sz_related))
        for i, row in enumerate(data):
            user = row[col_user]
            item = row[col_item]
            time = row[col_time]

            user_exist = user in self.users
            item_exist = item in self.items

            user = int(self.user_encoder.transform([[user]])[0, 0] if user_exist else
-1)
            item = int(self.item_encoder.transform([[item]])[0, 0] if item_exist else
-1)

            def norm(v, m, M):
                return (v-m)/(M-m) if m != M else 0.0

            user_times[i] = norm(time, self.user_min_times[user],
self.user_max_times[user]) if user_exist else 0.0
```

```
            item_time = norm(time, self.item_min_times[item],
self.item_max_times[item]) if item_exist else 0.0
            item_time_bins[i] = max(0, min(self.n_time_bins - 1, int(item_time *
self.n_time_bins)))

            user_relateds[i, :] = self.user_relateds[user] if user_exist else rel_pad
            item_relateds[i, :] = self.item_relateds[item] if item_exist else rel_pad

            users[i] = user + 1
            items[i] = item + 1

        X = [users, items, user_times, item_time_bins, user_relateds, item_relateds]

        return X
```

입력으로 주어진 데이터는 여지껏 보지 못한 유저 번호나 아이템 번호를 가지고 있을 수 있다. 이 경우 관련된 아이템이나 유저가 없다고 표시하고, 초기화했던 벡터들 중 0번 인덱스의 것을 사용한다. (0번 인덱스의 벡터는 zero mask되어 학습되지 않는다)

Instructions for compiling

Python dependencies

numpy, sklearn, tensorflow, pandas

Usage

```
$ ./recommender.py -h
usage: recommender [-h] [--epochs EPOCHS] [--batch-size BATCH_SIZE]
                   [--dim DIM] [--reg-lambda REG_LAMBDA] [--lr LR]
                   [--sz-related SZ_RELATED] [--n-time-bins N_TIME_BINS]
                   [--augment] [--seed SEED]
                   base test

positional arguments:
  base                  base filepath
  test                  test filepath

optional arguments:
  -h, --help            show this help message and exit
  --epochs EPOCHS, -e EPOCHS
                        epochs
  --batch-size BATCH_SIZE, -b BATCH_SIZE
                        batch size
  --dim DIM, -d DIM     The size of hidden factor
  --reg-lambda REG_LAMBDA
                        regularization factor
  --lr LR, -r LR        learning rate
  --sz-related SZ_RELATED
                        the maximum number of related objects
```

```
  --n-time-bins N_TIME_BINS
                        the number of item time bins
```

Example

```
./recommender.py ./data/u1.base ./data/u1.test
```

Any other specifications