Factorization Machines

IEEE 2010

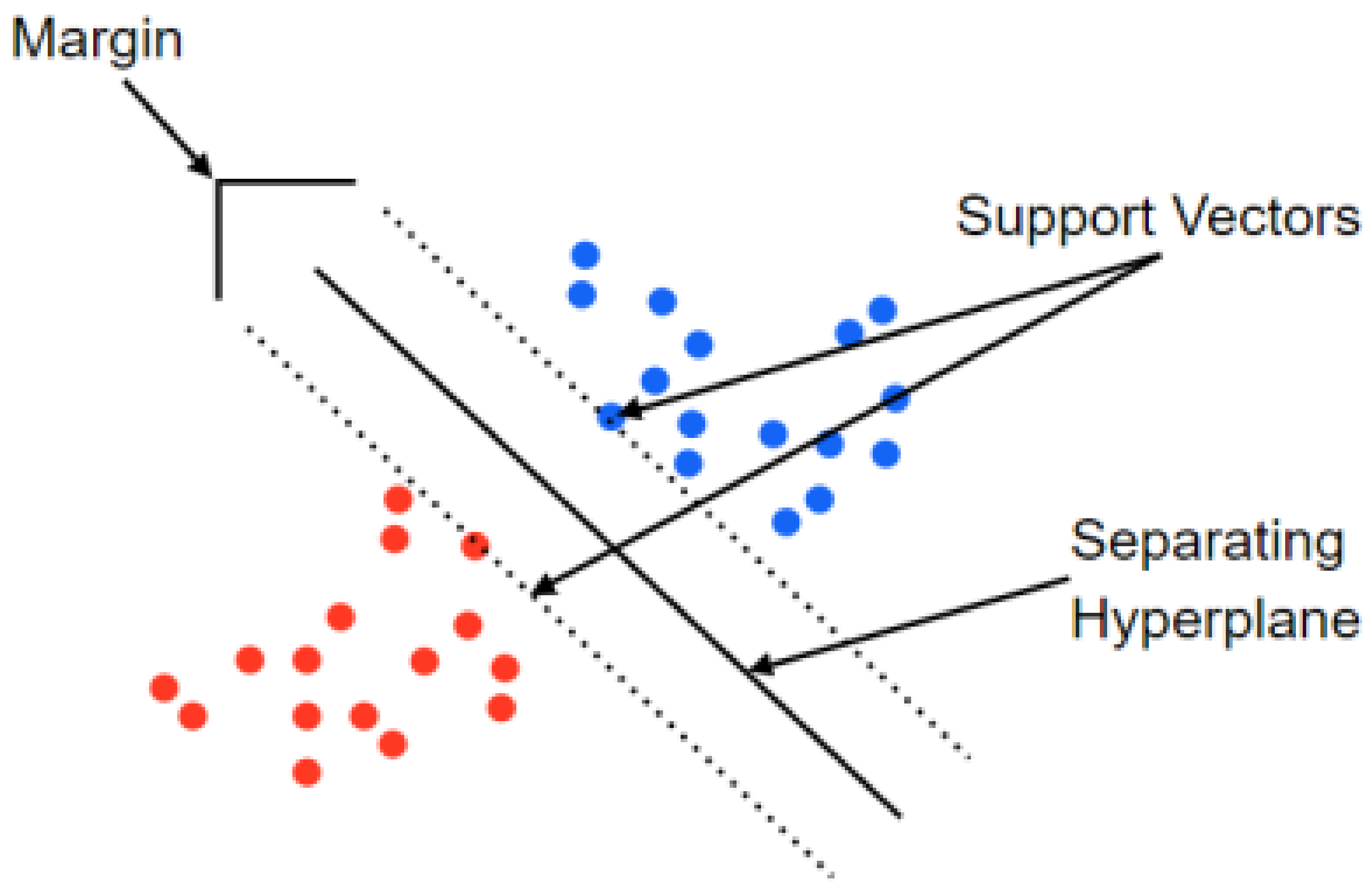
최은학
DSAIL Winter Intern

CONTENTS

Introduction

Support Vector Machine **(SVM)**

- 두 클래스를 가장 잘 구분짓는 선, 즉 초평면을 찾는 알고리즘
- 가장자리에 위치한 데이터간의 거리(Margin)가 가장 큰 구분선을 초평면으로 정의
- 선형 초평면을 기본으로 하지만, 커널 함수를 이용해 비선형 분류에 활용가능



Introduction

SVM → Sparse Data ✗

Fatorization Model → Standard Prediction Data ✗

Factorizaion Machine

- Sparse한 상황에 적용 가능
- Linear Complexity
- General Predictor

Prediction Under Sparsity

Dataset

	Feature vector **x**																					Target y	
	User				Movie					Other Movies rated					Time	Last Movie rated							
	A	B	C	...	TI	NH	SW	ST	...	TI	NH	SW	ST	...		TI	NH	SW	ST	...			
$\mathbf{x}^{(1)}$	1	0	0	...	1	0	0	0	...	0.3	0.3	0.3	0	...	13	0	0	0	0	...		5	$y^{(1)}$
$\mathbf{x}^{(2)}$	1	0	0	...	0	1	0	0	...	0.3	0.3	0.3	0	...	14	1	0	0	0	...		3	$y^{(2)}$
$\mathbf{x}^{(3)}$	1	0	0	...	0	0	1	0	...	0.3	0.3	0.3	0	...	16	0	1	0	0	...		1	$y^{(2)}$
$\mathbf{x}^{(4)}$	0	1	0	...	0	0	1	0	...	0	0	0.5	0.5	...	5	0	0	0	0	...		4	$y^{(3)}$
$\mathbf{x}^{(5)}$	0	1	0	...	0	0	0	1	...	0	0	0.5	0.5	...	8	0	0	1	0	...		5	$y^{(4)}$
$\mathbf{x}^{(6)}$	0	0	1	...	1	0	0	0	...	0.5	0	0.5	0	...	9	0	0	0	0	...		1	$y^{(5)}$
$\mathbf{x}^{(7)}$	0	0	1	...	0	0	1	0	...	0.5	0	0.5	0	...	12	1	0	0	0	...		5	$y^{(6)}$

Huge sparsity (average of x >> n)

- $X \subset \mathbb{R}^{M \times N}, n = |U| + |I| + |T| + \cdots$: feature data
- $x_i \subset \mathbb{R}^n \in D, i \in \{1, 2, \dots, m\}$: feature vector
- $y_i \in \mathbb{R}, i \in \{1, 2, 3, 4, 5\}$: target value(rating)
- $\hat{y}(x)$: predicted value

Prediction Under Sparsity

Dataset

	Feature vector **x**																					Target y	
	User				Movie					Other Movies rated					Time	Last Movie rated							
	A	B	C	...	TI	NH	SW	ST	...	TI	NH	SW	ST	...	Time	TI	NH	SW	ST	...			
$\mathbf{x}^{(1)}$	1	0	0	...	1	0	0	0	...	0.3	0.3	0.3	0	...	13	0	0	0	0	...		5	$y^{(1)}$
$\mathbf{x}^{(2)}$	1	0	0	...	0	1	0	0	...	0.3	0.3	0.3	0	...	14	1	0	0	0	...		3	$y^{(2)}$
$\mathbf{x}^{(3)}$	1	0	0	...	0	0	1	0	...	0.3	0.3	0.3	0	...	16	0	1	0	0	...		1	$y^{(2)}$
$\mathbf{x}^{(4)}$	0	1	0	...	0	0	1	0	...	0	0	0.5	0.5	...	5	0	0	0	0	...		4	$y^{(3)}$
$\mathbf{x}^{(5)}$	0	1	0	...	0	0	0	1	...	0	0	0.5	0.5	...	8	0	0	1	0	...		5	$y^{(4)}$
$\mathbf{x}^{(6)}$	0	0	1	...	1	0	0	0	...	0.5	0	0.5	0	...	9	0	0	0	0	...		1	$y^{(5)}$
$\mathbf{x}^{(7)}$	0	0	1	...	0	0	1	0	...	0.5	0	0.5	0	...	12	1	0	0	0	...		5	$y^{(6)}$

Prediction Under Sparsity

Dataset

	Feature vector **x**																					Target y	
	User				Movie					Other Movies rated					Time	Last Movie rated							
	A	B	C	...	TI	NH	SW	ST	...	TI	NH	SW	ST	...	Time	TI	NH	SW	ST	...			
$\mathbf{x}^{(1)}$	1	0	0	...	1	0	0	0	...	0.3	0.3	0.3	0	...	13	0	0	0	0	...		5	$y^{(1)}$
$\mathbf{x}^{(2)}$	1	0	0	...	0	1	0	0	...	0.3	0.3	0.3	0	...	14	1	0	0	0	...		3	$y^{(2)}$
$\mathbf{x}^{(3)}$	1	0	0	...	0	0	1	0	...	0.3	0.3	0.3	0	...	16	0	1	0	0	...		1	$y^{(2)}$
$\mathbf{x}^{(4)}$	0	1	0	...	0	0	1	0	...	0	0	0.5	0.5	...	5	0	0	0	0	...		4	$y^{(3)}$
$\mathbf{x}^{(5)}$	0	1	0	...	0	0	0	1	...	0	0	0.5	0.5	...	8	0	0	1	0	...		5	$y^{(4)}$
$\mathbf{x}^{(6)}$	0	0	1	...	1	0	0	0	...	0.5	0	0.5	0	...	9	0	0	0	0	...		1	$y^{(5)}$
$\mathbf{x}^{(7)}$	0	0	1	...	0	0	1	0	...	0.5	0	0.5	0	...	12	1	0	0	0	...		5	$y^{(6)}$

Prediction Under Sparsity

Dataset

Feature vector **x**

	User				Movie					Other Movies rated					Time	Last Movie rated					Target y	
	A	B	C	...	TI	NH	SW	ST	...	TI	NH	SW	ST	...		TI	NH	SW	ST	...		
$\mathbf{x}^{(1)}$	1	0	0	...	1	0	0	0	...	0.3	0.3	0.3	0	...	13	0	0	0	0	...	5	$y^{(1)}$
$\mathbf{x}^{(2)}$	1	0	0	...	0	1	0	0	...	0.3	0.3	0.3	0	...	14	1	0	0	0	...	3	$y^{(2)}$
$\mathbf{x}^{(3)}$	1	0	0	...	0	0	1	0	...	0.3	0.3	0.3	0	...	16	0	1	0	0	...	1	$y^{(2)}$
$\mathbf{x}^{(4)}$	0	1	0	...	0	0	1	0	...	0	0	0.5	0.5	...	5	0	0	0	0	...	4	$y^{(3)}$
$\mathbf{x}^{(5)}$	0	1	0	...	0	0	0	1	...	0	0	0.5	0.5	...	8	0	0	1	0	...	5	$y^{(4)}$
$\mathbf{x}^{(6)}$	0	0	1	...	1	0	0	0	...	0.5	0	0.5	0	...	9	0	0	0	0	...	1	$y^{(5)}$
$\mathbf{x}^{(7)}$	0	0	1	...	0	0	1	0	...	0.5	0	0.5	0	...	12	1	0	0	0	...	5	$y^{(6)}$

Factorization Machines

Model Equation

$$\hat{y}(\mathbf{x}) := w_0 + \sum_{i=1}^{n} w_i \, x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle \mathbf{v}_i, \mathbf{v}_j \rangle \, x_i \, x_j$$

- W_0: Global bias
- W_i: 개별 특성의 가중치
- <V_i, V_j>: i, j번째 변수간의 상호작용을 모델링 $< v_i, v_j > := \sum_{f=1}^{k} v_{i,f} \cdot v_{j,f}$

Factorization Machines

Expressiveness

- f가 충분히 크다면 Positive definite matrix W에 대해 $W = V \cdot V^T$을 만족하는 V가 존재

  => f가 클때, 어떠한 Interaction Matrix W도 표현할 수 있음

- Sparse한 상황에서는 복잡한 상호작용 계산 어려움

  => 작은 k를 사용해 일반화 성능을 높힘

Parameter Estimation Under Sparsity

Feature vector **x**

	User				Movie					Other Movies rated					Time	Last Movie rated					Target y	
	A	B	C	...	TI	NH	SW	ST	...	TI	NH	SW	ST	...		TI	NH	SW	ST	...		
$x^{(1)}$	1	0	0	...	1	0	0	0	...	0.3	0.3	0.3	0	...	13	0	0	0	0	...	5	$y^{(1)}$
$x^{(2)}$	1	0	0	...	0	1	0	0	...	0.3	0.3	0.3	0	...	14	1	0	0	0	...	3	$y^{(2)}$
$x^{(3)}$	1	0	0	...	0	0	1	0	...	0.3	0.3	0.3	0	...	16	0	1	0	0	...	1	$y^{(2)}$
$x^{(4)}$	0	1	0	...	0	0	1	0	...	0	0	0.5	0.5	...	5	0	0	0	0	...	4	$y^{(3)}$
$x^{(5)}$	0	1	0	...	0	0	0	1	...	0	0	0.5	0.5	...	8	0	0	1	0	...	5	$y^{(4)}$
$x^{(6)}$	0	0	1	...	1	0	0	0	...	0.5	0	0.5	0	...	9	0	0	0	0	...	1	$y^{(5)}$
$x^{(7)}$	0	0	1	...	0	0	1	0	...	0.5	0	0.5	0	...	12	1	0	0	0	...	5	$y^{(6)}$

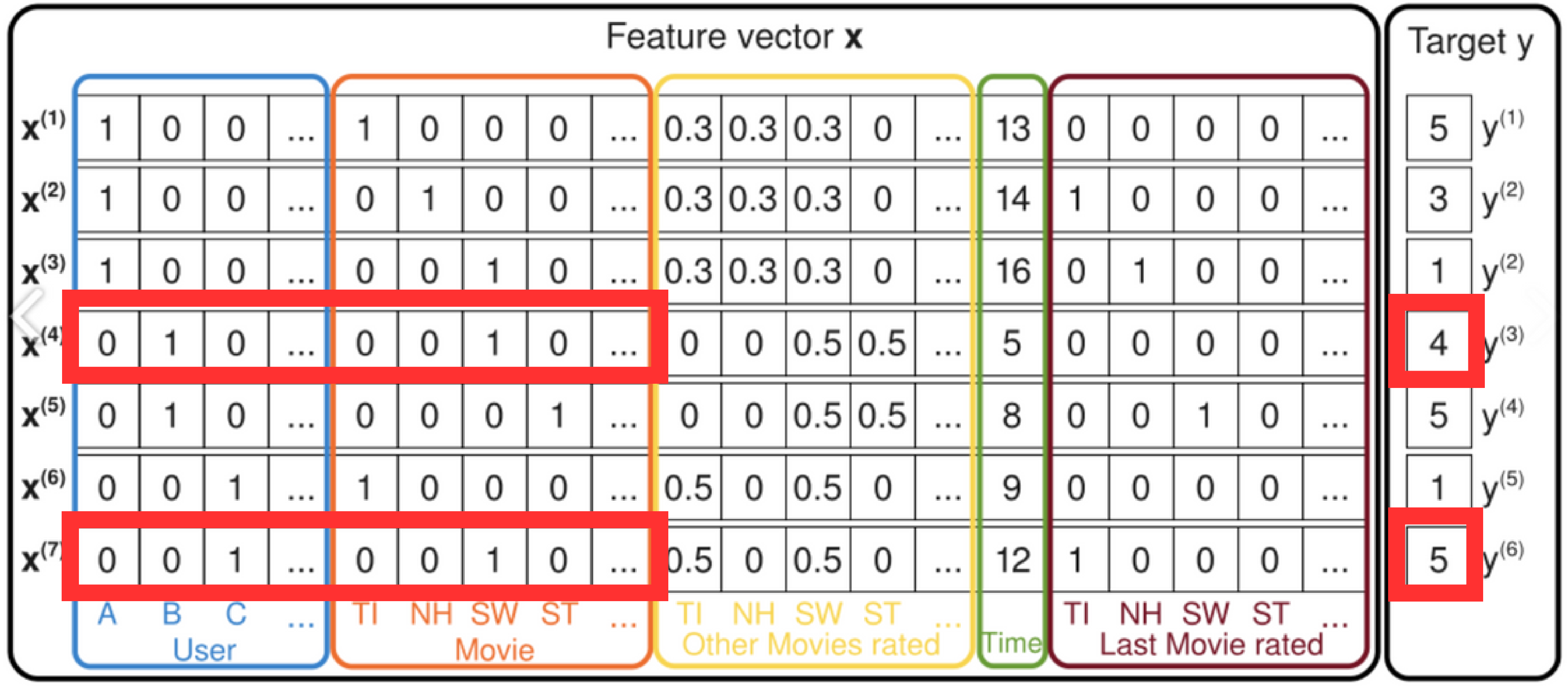
No interaction

Break the independence

factorized interaction

$<v_A, v_{ST}>$

Parameter Estimation Under Sparsity



	Feature vector x																				Target y	
	A	B	C	...	TI	NH	SW	ST	...	TI	NH	SW	ST	...	Time	TI	NH	SW	ST	...		
	User				Movie					Other Movies rated					Time	Last Movie rated						
$x^{(1)}$	1	0	0	...	1	0	0	0	...	0.3	0.3	0.3	0	...	13	0	0	0	0	...	5	$y^{(1)}$
$x^{(2)}$	1	0	0	...	0	1	0	0	...	0.3	0.3	0.3	0	...	14	1	0	0	0	...	3	$y^{(2)}$
$x^{(3)}$	1	0	0	...	0	0	1	0	...	0.3	0.3	0.3	0	...	16	0	1	0	0	...	1	$y^{(2)}$
$x^{(4)}$	0	1	0	...	0	0	1	0	...	0	0	0.5	0.5	...	5	0	0	0	0	...	4	$y^{(3)}$
$x^{(5)}$	0	1	0	...	0	0	0	1	...	0	0	0.5	0.5	...	8	0	0	1	0	...	5	$y^{(4)}$
$x^{(6)}$	0	0	1	...	1	0	0	0	...	0.5	0	0.5	0	...	9	0	0	0	0	...	1	$y^{(5)}$
$x^{(7)}$	0	0	1	...	0	0	1	0	...	0.5	0	0.5	0	...	12	1	0	0	0	...	5	$y^{(6)}$

Similar interaction

$$<v_B, v_{SW}> \approx <v_C, v_{SW}>$$

$$v_B \approx v_C$$

Factorization Machines

Computation

$O(kn^2)$

$$\sum_{i=1}^{n}\sum_{j=i+1}^{n}\langle \mathbf{v}_i,\mathbf{v}_j\rangle\, x_i\, x_j$$

$$= \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\langle \mathbf{v}_i,\mathbf{v}_j\rangle\, x_i\, x_j - \frac{1}{2}\sum_{i=1}^{n}\langle \mathbf{v}_i,\mathbf{v}_i\rangle\, x_i\, x_i$$

$$= \frac{1}{2}\left(\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{f=1}^{k} v_{i,f}\, v_{j,f}\, x_i\, x_j - \sum_{i=1}^{n}\sum_{f=1}^{k} v_{i,f}\, v_{i,f}\, x_i\, x_i\right)$$

$$= \frac{1}{2}\sum_{f=1}^{k}\left(\left(\sum_{i=1}^{n} v_{i,f}\, x_i\right)\left(\sum_{j=1}^{n} v_{j,f}\, x_j\right) - \sum_{i=1}^{n} v_{i,f}^2\, x_i^2\right)$$

$$\boxed{= \frac{1}{2}\sum_{f=1}^{k}\left(\left(\sum_{i=1}^{n} v_{i,f}\, x_i\right)^2 - \sum_{i=1}^{n} v_{i,f}^2\, x_i^2\right)}$$

$O(kn)$

d-way FM

$$\hat{y}(x) := w_0 + \sum_{i=1}^{n} w_i\, x_i$$
$$+ \sum_{l=2}^{d}\sum_{i_1=1}^{n}\cdots\sum_{i_l=i_{l-1}+1}^{n}\left(\prod_{j=1}^{l} x_{i_j}\right)\left(\sum_{f=1}^{k_l}\prod_{j=1}^{l} v_{i_j,f}^{(l)}\right)$$

Factorization Machines

as Predictor

- Factorization Machine은 Regression, Binary classification, Ranking 가능
- 최적화 목표에 정규화항 추가

Learning

Gradient Descent

$$\hat{y}(x) = w_0 + \sum_{i=1}^{n} w_i x_i + \frac{1}{2}\sum_{f=1}^{k}\left(\left(\sum_{i=1}^{n} v_{if} x_i\right)^2 - \sum_{i=1}^{n} v_{if}^2 x_i^2\right)$$

$$\frac{\partial}{\partial\theta}\hat{y}(\mathbf{x}) = \begin{cases} 1, & \text{if } \theta \text{ is } w_0 \\ x_i, & \text{if } \theta \text{ is } w_i \\ x_i \sum_{j=1}^{n} \boxed{v_{j,f} x_j} - v_{i,f} x_i^2, & \text{if } \theta \text{ is } v_{i,f} \end{cases} \qquad (4)$$

i와 독립 >> Precompute

FM vs SVM

Polynomial SVM

$$\hat{y}(\mathbf{x}) = w_0 + \sqrt{2}\sum_{i=1}^{n} w_i\, x_i + \sum_{i=1}^{n} w_{i,i}^{(2)} x_i^2 + \sqrt{2}\sum_{i=1}^{n}\sum_{j=i+1}^{n} w_{i,j}^{(2)}\, x_i\, x_j \quad (9)$$

모든 $w_{i,j}$ 는 independent

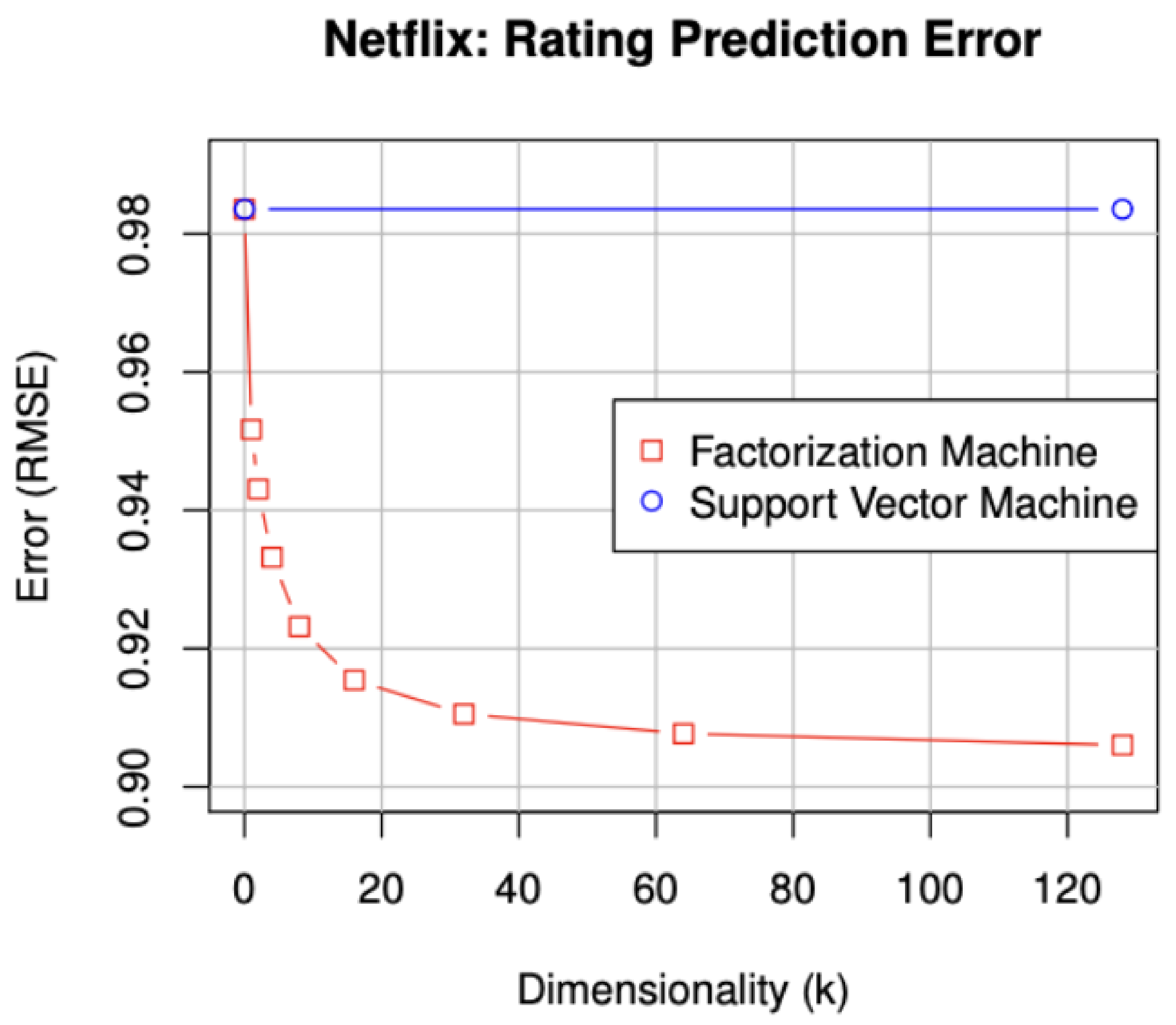
↓

feature간의 상호작용이 없으면

estimation 어려움

FM vs SVM

Summary



- SVM은 직접적인 상호작용의 관찰이 필요하지만, FM은 직접적인 상호작용이 없어도 추정 가능 (under sparsity)
- SVM과 다르게 FM은 바로 학습 가능
- SVM은 특정 학습 데이터에 의존하지만, FM은 학습데이터에 무관

FM vs Other Factorization Models

FM →

- Matrix and Tensor Factorization
- SVD++
- PITF for Tag Recommendation
- Factorized Personalized Markov Chains

FM can mimic many of these models by using the right input data

FM vs Other Factorization Models

PITF for Tag Recommendation

users U, items I, tags T (binary indicator)

$$n := |U \cup I \cup T|, \quad x_j := \delta\,(j = i \vee j = u \vee j = t) \quad (13)$$

Right Input Data

$$\hat{y}(\mathbf{x}) = w_0 + w_u + w_i + w_t + \langle \mathbf{v}_u, \mathbf{v}_i \rangle + \langle \mathbf{v}_u, \mathbf{v}_t \rangle + \langle \mathbf{v}_i, \mathbf{v}_t \rangle$$

Used for ranking between two tags

$$\hat{y}_{u,i,t} = \sum_f \hat{u}_{u,f} \cdot \hat{t}^U_{t,f} + \sum_f \hat{i}_{i,f} \cdot \hat{t}^I_{t,f}$$

PITF equation

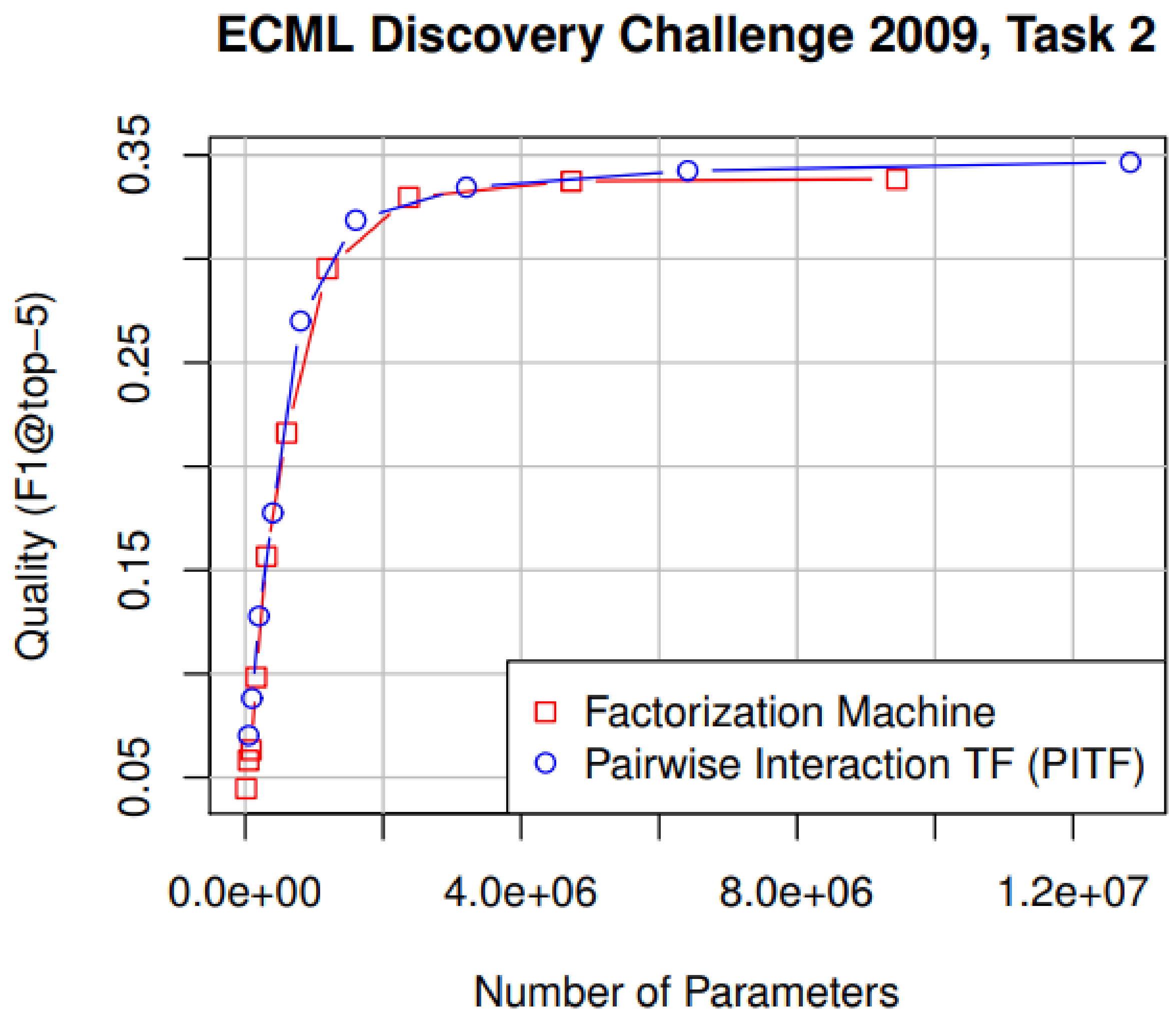
$$\hat{y}(\mathbf{x}) := w_t + \langle \mathbf{v}_u, \mathbf{v}_t \rangle + \langle \mathbf{v}_i, \mathbf{v}_t \rangle$$

FM equation

W_t, Interaction의 독립성을 제외하면 매우 비슷

FM vs Other Factorization Models

Summary

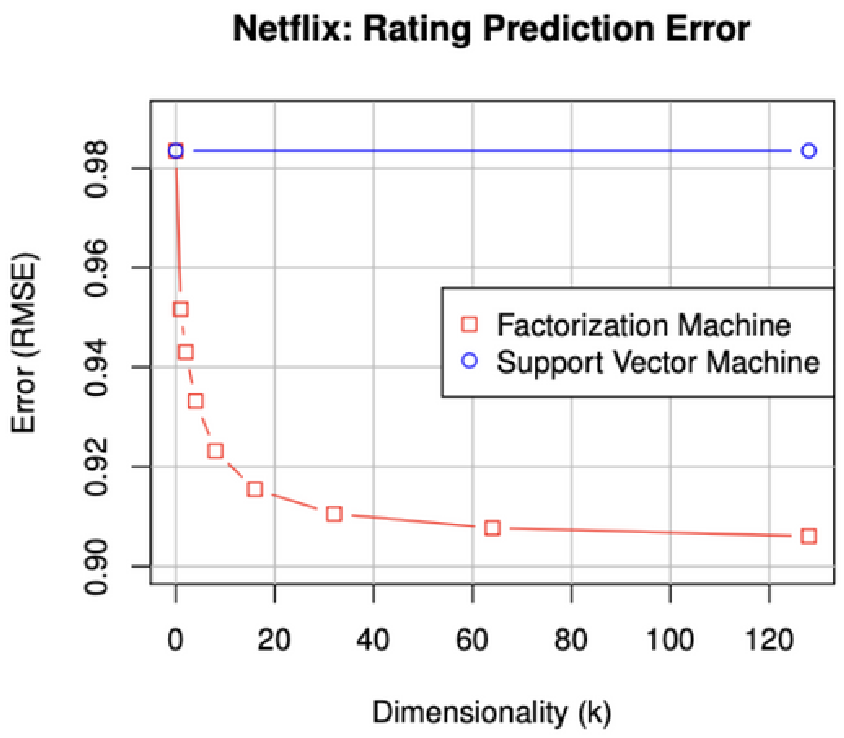


- MF와 다르게 FM은 general prediction model
- FM은 더 쉽게 적용이 가능하고 성능은 비슷

Conclusion

In contrast to SVM

- 매우 sparse한 상황에서도 estimation 가능
- 선형 모델 방정식을 통해 직접 최적화 가능

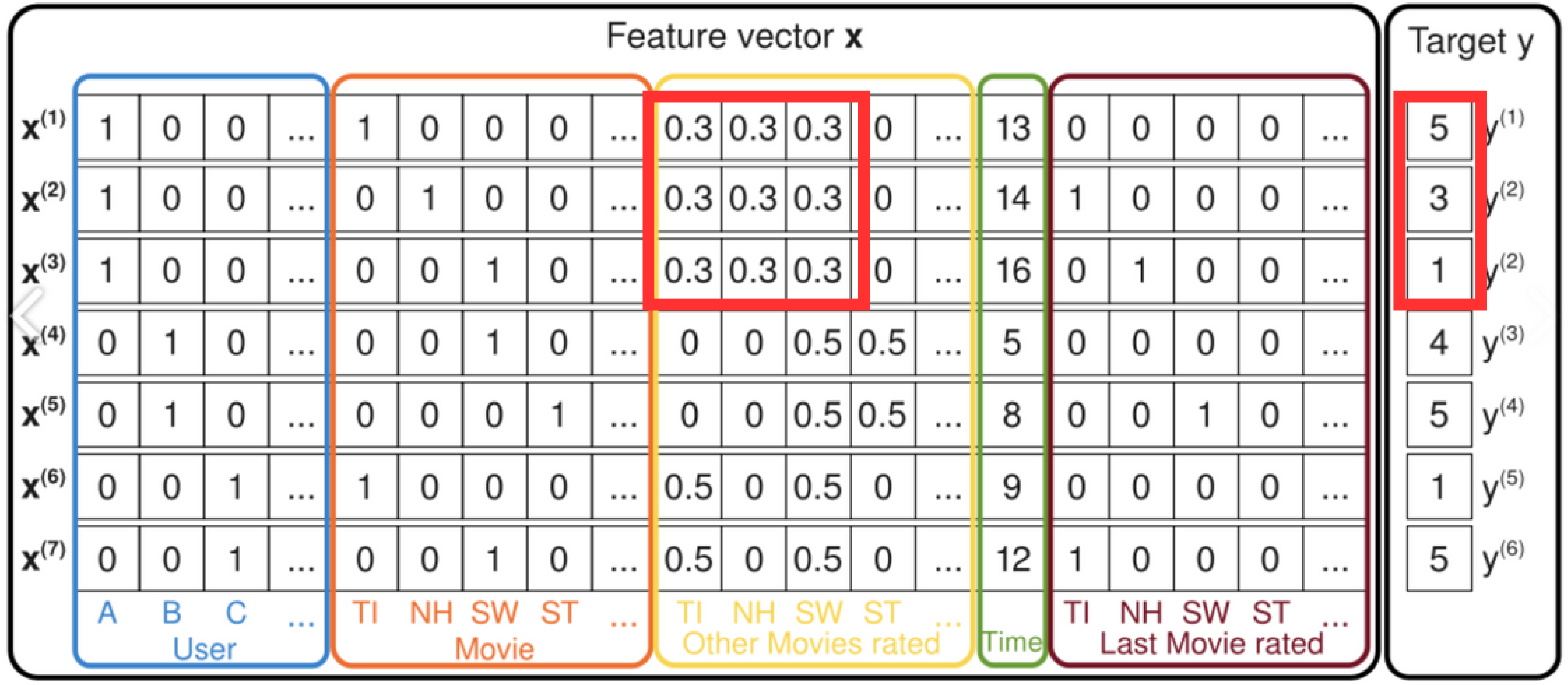


Discussion

- 일반화된 성능을 위해 작은 k를 선택했다고 하는데 그래프를 보면 k가 증가할수록 성능이 향상

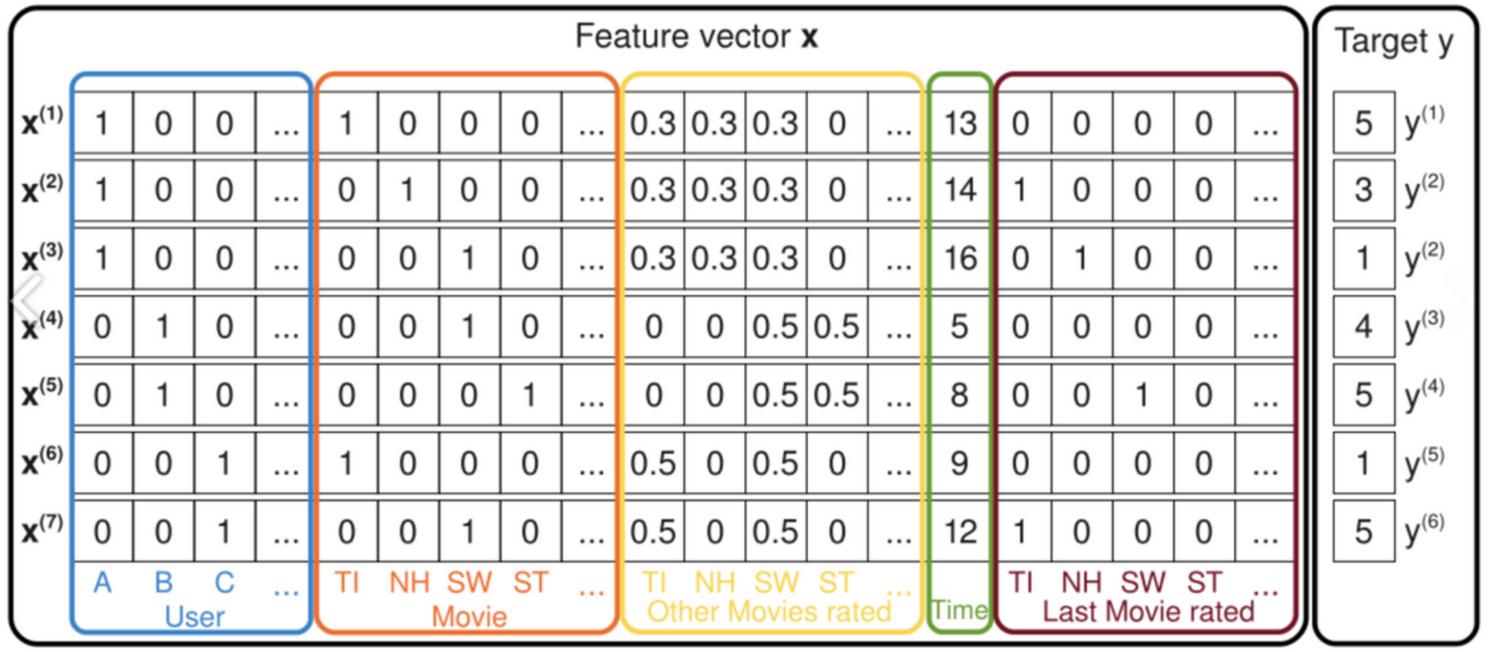
- 시간, 최근 평가한 영화 변수가 추가되었는데 이후 시점의 영화도 input data에 포함

>> [1 | 0 | 0], [0.5 | 0.5 | 0], [0.3 | 0.3 | 0.3]?



	User				Movie					Other Movies rated					Time	Last Movie rated					Target y	
	A	B	C	...	TI	NH	SW	ST	...	TI	NH	SW	ST	...	Time	TI	NH	SW	ST	...		
$x^{(1)}$	1	0	0	...	1	0	0	0	...	0.3	0.3	0.3	0	...	13	0	0	0	0	...	5	$y^{(1)}$
$x^{(2)}$	1	0	0	...	0	1	0	0	...	0.3	0.3	0.3	0	...	14	1	0	0	0	...	3	$y^{(2)}$
$x^{(3)}$	1	0	0	...	0	0	1	0	...	0.3	0.3	0.3	0	...	16	0	1	0	0	...	1	$y^{(2)}$
$x^{(4)}$	0	1	0	...	0	0	1	0	...	0	0	0.5	0.5	...	5	0	0	0	0	...	4	$y^{(3)}$
$x^{(5)}$	0	1	0	...	0	0	0	1	...	0	0	0.5	0.5	...	8	0	0	1	0	...	5	$y^{(4)}$
$x^{(6)}$	0	0	1	...	1	0	0	0	...	0.5	0	0.5	0	...	9	0	0	0	0	...	1	$y^{(5)}$
$x^{(7)}$	0	0	1	...	0	0	1	0	...	0.5	0	0.5	0	...	12	1	0	0	0	...	5	$y^{(6)}$

- Baseline



- Adding feature

user의 성별, 나이, 위치 추가

movie의 장르 추가

Implementation Baseline

```
ratings_df['Timestamp'] = pd.to_datetime(ratings_df['Timestamp'], unit='s')
ratings_df = ratings_df.sort_values(['UserID', 'Timestamp'])
ratings_df.reset_index(drop=True, inplace=True)
```

UserID, Timestamp기준으로 sorting (Last Movie 반영)

```
# Last Movie rated 추가
ratings_df['Last_MovieID'] = ratings_df.groupby('UserID')['MovieID'].shift(1)
```

Last MovieID feature 추가

	UserID	MovieID	Rating	Timestamp	Last_MovieID
182	3	593	3	2000-12-31 21:10:18	NaN
183	3	2858	4	2000-12-31 21:10:39	593.0
184	3	3534	3	2000-12-31 21:11:08	2858.0
185	3	1968	4	2000-12-31 21:11:08	3534.0
186	3	1431	3	2000-12-31 21:11:35	1968.0

Implementation Baseline

```python
ratings_df['Year'] = ratings_df['Timestamp'].dt.year
ratings_df['Month'] = ratings_df['Timestamp'].dt.month

ratings_df['Month_Num'] = (ratings_df['Year'] - 2000) * 12 + ratings_df['Month'] - 11

month_num = ratings_df['Month_Num'].reset_index(drop=True)
month_num.index = ratings_df['MovieID']
```

Month

```python
def Encoding(df = ratings_df, column='UserID', index='MovieID'):
    #원-핫 인코딩
    encoded_data = pd.get_dummies(df[column], prefix=column)
    encoded_data.index = df[index]
    encoded_data = encoded_data.astype(int)

    return encoded_data
```

User, Movie, Last Movie rated

```python
def create_other_movies_rated(df = ratings_df):

    user_movie_ratings = df.pivot_table(index='UserID', columns='MovieID', values='Rating')
    user_movie_ratings.columns = ['other_' + str(col) for col in user_movie_ratings.columns]
    user_movie_ratings.fillna(0.0, inplace=True)

    # binary indicator
    user_movie_ratings = user_movie_ratings.applymap(lambda x: 1 if x >= 1 else 0)

    # 정규화
    user_movie_ratings = user_movie_ratings.div(user_movie_ratings.sum(axis=1), axis=0)

    # Create other movies rated matrix
    Other_Movie = df['UserID'].apply(lambda x: user_movie_ratings.loc[x])
    Other_Movie.index = df['MovieID']

    return Other_Movie
```

Other Movie rated

Implementation Baseline

```python
class FM(keras.Model):
    def __init__(self, n_features, n_factor=10, regularization_factor=0.01):
        super().__init__()

        self.w_0 =  tf.Variable([0.0])
        self.w = tf.Variable(tf.zeros(shape = [n_features]))
        self.v = tf.Variable(tf.random.normal(shape = (n_features, n_factor)))
        self.regularization_factor = regularization_factor

    def call(self, inputs):

        # linear_term
        degree_1 = tf.reduce_sum(tf.multiply(self.w,inputs),axis= 1)

        # interaction_term
        degree_2 = 0.5 * tf.reduce_sum(
          tf.math.pow(tf.matmul(inputs,self.v),2)
            -tf.matmul(tf.math.pow(inputs,2),tf.math.pow(self.v,2))
            ,1
            ,keepdims=False
        )

        predict = self.w_0 + degree_1 + degree_2  # Regression은 그대로, binary classification은 sigmoid로

        return predict

    def compute_loss(self, y_true, y_pred):
        mse_loss = tf.reduce_mean(tf.square(y_true - y_pred))
        rmse_loss = tf.sqrt(mse_loss)
        l2_loss = self.regularization_factor * (tf.nn.l2_loss(self.w) + tf.nn.l2_loss(self.v))
        total_loss = rmse_loss + l2_loss
        return total_loss

    def compute_loss_val(self, y_true, y_pred):
        mse_loss = tf.reduce_mean(tf.square(y_true - y_pred))
        rmse_loss = tf.sqrt(mse_loss)
        total_loss = rmse_loss
        return total_loss


    def train_step(self, data):
        x, y_true = data
        with tf.GradientTape() as tape:
            y_pred = self(x, training=True)
            loss = self.compute_loss(y_true, y_pred)

        gradients = tape.gradient(loss, self.trainable_variables)
        self.optimizer.apply_gradients(zip(gradients, self.trainable_variables))
        return {'loss': loss}

    def test_step(self, data):
        x, y_true = data
        y_pred = self(x, training=False)
        loss = self.compute_loss_val(y_true, y_pred)
        return {'loss': loss}
```

Result

k	RMSE
2	0.5503
5	0.5361
10	0.5287
20	0.5456
40	0.5396

user 수: 500

lr: 0.005, epochs: 30

Implementation Adding feature

```python
# 첫 번째 자리 분리
users_df['Zip_1'] = users_df['Zip-code'].str[0]

# 첫 두 자리 분리
users_df['Zip_2'] = users_df['Zip-code'].str[:2]
```

```python
def Encoding_addfeature(df, column, index):
    encoded_data = pd.get_dummies(df[column], prefix=column)
    data = ratings_df['UserID'].apply(lambda x: encoded_data.loc[x-1])
    data.index = ratings_df['MovieID']

    return data
```

User 성별, 직업, 지역

```python
## 영화 장르 원-핫 인코딩
encoded_genre = movies_df['Genres'].str.get_dummies(sep='|')
encoded_genre.index = movies_df['MovieID']

genre = ratings_df['MovieID'].apply(lambda x: encoded_genre.loc[x])
genre.index = ratings_df['MovieID']
```

Movie 장르

Implementation Adding feature

Result

k	RMSE
2	0.657
5	0.6855
10	0.6853
20	0.6863
40	0.6767

user 수: 500

lr: 0.005, epochs: 30

- feature를 추가하니 성능이 감소
- n 대신 m(x)에 대해서 연산해야하는데 구현하지 못함
- 전체적을 성능이 너무 높게 나옴
- 여러번 반복하지 못함