Hafta 4 (Recommendation Systems)

@mebaysan

19/09/2021

Bu haftanın veri seti

- MovieLens 20M Dataset
- The Movies Dataset
- Online Retail Dataset

İlgili Okuma Listesi

- Öneri Sistemleri 101
- Öneri Sistemleri (Recommendation Systems)
- Introduction to recommender systems
- Recommendation Systems (Stanford EDU)
- Classifying Different Types of Recommender Systems
- Recommendation systems: Principles, methods and evaluation
- Matrix factorization (recommender systems)
- Singular Value Decomposition vs. Matrix Factorization in Recommender Systems
- Netflix Update: Try This at Home - Simon Funk
- Python Surprise Package for Matrix Factorization

Benim Yazdığım Yazılar

- What is Association Rule Learning? An Applied Example in Python
- What is Content-Based Filtering? An Applied Example in Python

Recommendation Systems (Tavsiye Sistemleri)

1990'lara kadar uzanmaktadır. Genel olarak gündemimize 2009 yılında Netflix'in yaptığı bir yarışma ile girmiştir. Bazı teknikleri kullanarak kullanıcılara tavsiyeler yapar. **Başlıca kullanım nedeni içerik bolluğudur. İçerik bol fakat kullanıcının ilgisini çeken küme elemanları daha az.** Bol içeriği kullanıcıya göre filtreleyerek tavsiye işlemi gerçekleşir.

Tavsiye Sistemleri Türleri

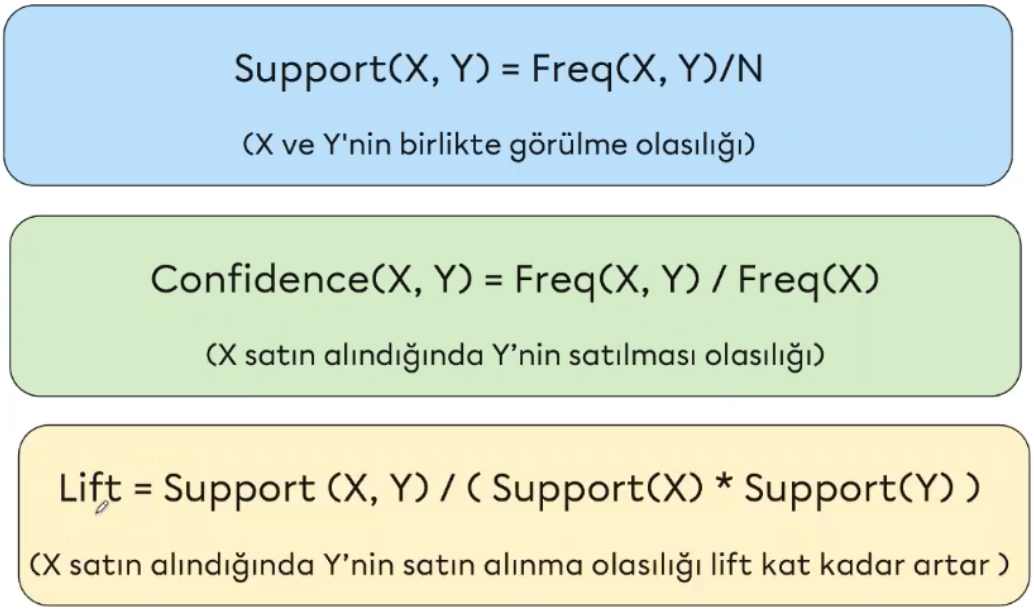
- Simple Recommender Systems (Basit Tavsiye Sistemleri)
  - İş bilgisi ya da basit tekniklerle yapılan genel örnekler
  - Kategorinin en yüksek puanlıları, trend olanlar, efsaneler vb.
- Association Rule Learning (Birliktelik Kuralı Öğrenimi)
  - Sepet analizi olarak da bilinir
  - Birliktelik analizi ile öğrenilen kurallara göre ürün önerileri
- Content Based Filtering (İçerik Temelli Filtreleme)

    - Ürün benzerliğine göre öneriler yapılan uzaklık temelli yöntemler
- Collaborative Filtering (İş Birlikli Filtreleme)
    - **Topluluğun kullanıcı ya da ürün bazında ortak kanaatlerini yansıtan yöntemler**
    - User-Based
    - Item-Based
    - Model-Based (Matrix Factorization)

Association Rule Learning (Birliktelik Kuralı Öğrenimi)

Veri içerisindeki örüntüleri (pattern, ilişki, yapı) bulmak için kullanılan kural tabanlı bir makine öğrenmesi tekniğidir

Apriori Algoritması



Sepet analizi yöntemidir. Ürün birlikteliklerini ortaya çıkarmak için kullanılır.

`Support(X,Y) = Freq(X,Y)/N` → X ve Y'nin birlikte görülme olasılığıdır.

`Confidence(X,Y) = Freq(X,Y)/Freq(X)` → X satın alındığında Y'nin satılması olasılığı

`Lift = Support(X,Y)/(Support(X)*Support(Y))` → X satın alındığında Y'nin satın alınma olasılığı lift kat kadar artar

ARL'de büyük ölçekli işlerde büyük problemlerden biri **sepet** tanımının yapılmasıdır.

Apriori Algoritması ile ARL Örneği

Ekmek satın alındığında süt satın alınması olayı.

İşlem	Ürünler
Transaction 1	Ekmek, Süt, Kola, Peynir
Transaction 2	Ekmek, Süt, Kola
Transaction 3	Ekmek, Süt
Transaction 4	Ekmek, Soğan
Transaction 5	Un, Süt, Kola, Peynir
Transaction 6	Un, Süt, Kola
Transaction 7	Un, Süt
Transaction 8	Un, Soğan

$$\text{Support(Ekmek)} = \frac{4}{8} \qquad \text{Support(Süt)} = \frac{6}{8}$$

$$\text{Support(Ekmek, Süt)} = \frac{3}{8}$$

$$\text{Confidence(Ekmek, Süt)} = \frac{\text{Freq(Ekmek, Süt)}}{\text{Freq(Ekmek)}} = \frac{3}{4}$$

$$\text{Lift(Ekmek, Süt)} = \frac{\text{Support (Ekmek, Süt)}}{\text{Support(Ekmek)} * \text{Support(Süt)}} = \frac{\frac{3}{8}}{\frac{4}{8} * \frac{6}{8}} = 1$$

Support(Ekmek) → Bütün işlemler içerisinde ekmeğin satılma oranını verir (8 işlemde 4 ekmek satılmış `4/8`)

Support(Süt) → Bütün işlemler içerisinde süt satılma oranını verir (8 işlemde 6 süt satılmış `6/8`)

Confidence(Ekmek, Süt) → Ekmeğin satıldığı tüm işlemlerde ekmek ve sütün beraber satılma olasılığını verir (ekmeğin satıldığı 4 işlem içerisinde toplam 3 adet işlemde ekmek ve süt beraber satılmış `3/4`)

Lift(Ekmek, Süt) → Ekmek satın alındığında süt satın alınma olasılığı lift kat kadar artar (tüm işlemler içerisinde ekmek ve sütün aynı anda satılması olasılığının, ekmek ve sütün ayrı ayrı satılma olasılıklarının çarpımına bölünmesidir. `(3/8) / ((4/8) * (6/8))`)

Python'da ARL

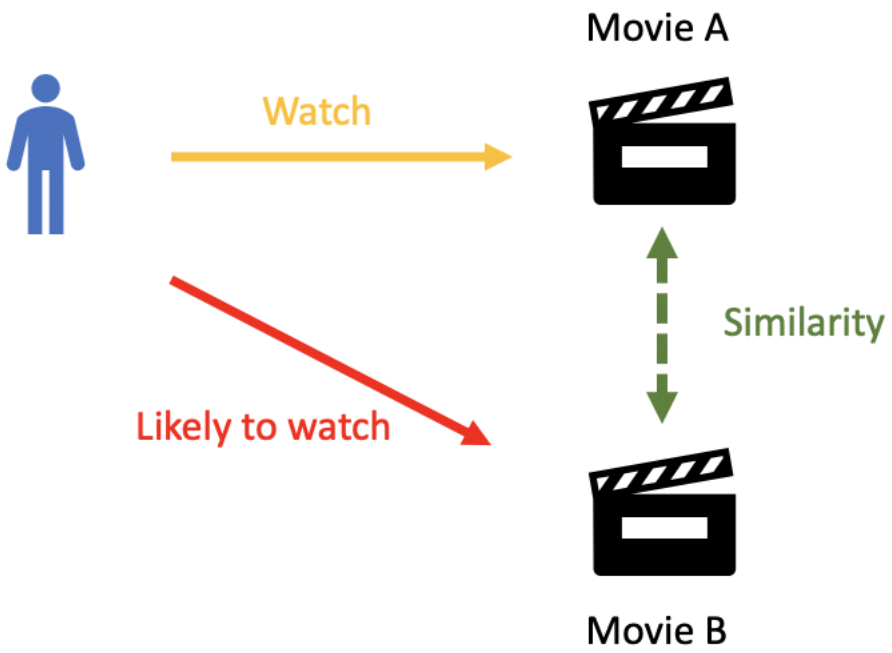
```
!pip install mlxtend
```

```
from mlxtend.frequent_patterns import apriori, association_rules
```

```
########################################
# ARL Veri Yapısını Hazırlama (Invoice-Product Matrix)
########################################

# Verinin gelmesini istediğimiz durum:

# Satırlar her bir işlemi (sepeti) temsil ederken, sütunlar ürünleri temsil eder. Hangi sepette hangi ürün var binary olarak temsil edilir.

# Description   NINE DRAWER OFFICE TIDY   SET 2 TEA TOWELS I LOVE LONDON    SPACEBOY BABY GIFT SET
# Invoice
# 536370                            0                               1                    0
# 536852                            1                               0                    1
# 536974                            0                               0                    0
# 537065                            1                               0                    0
# 537463                            0                               0                    1
```

Content-Based Filtering

Ürün içeriklerinin benzerlikleri üzerinden tavsiyeler geliştirilir.



- Ürün/film/blog içerikleri içerisindeki metinleri matematiksel olarak temsil ederiz. Metinleri vektörleştirme denir.
- Benzerlikleri Hesaplanır
  - En yaygın kullanılan yöntemler:
    - Count Vector (Word Count)
      - Bu yöntem frekansa bağlı olduğu için biraz yanlıdır
    - TF-IDF

	Word 1	Word 2	Word 3	.	.	.	Word n
Movie 1	1	2	0				4
Movie 2	5	1	3				3
Movie 3	3	5	3				2
.							
.							
.							
Movie m	3	5	4				1

Genel olarak kuracağımız yapı yukarıdaki şekildedir. Satırlarda filmler/ürünler, sütunlarda ise tüm filmlerdeki geçen tüm eşsiz kelimeler vardır. Kesişimler hangi kelimenin hangi filmde kaç tane olduğunu temsil eder.

Yukarıdaki örneğe bakarsak, Movie3 izleyen birine MovieM'i tavsiye ederdik. Sezgisel olarak bütün ikili kombinasyonların yakınlığına baktık.

	Word 1	Word 2	Word 3	.	.	.	Word n
Movie 1	1	2	0				4
Movie 2	5	1	3				3
Movie 3	3	5	3				2
.							
.							
.							
Movie m	3	5	4				1

Movie1, Movie2 = (1-5)^2 + (2-1)^2 + (0-3)^2 + .. + (4-3)^2 = 27

Movie1, Movie3 = (1-3)^2 + (2-5)^2 + (0-3)^2 + .. + (4-2)^2 = 26

Movie1, Moviem = (1-3)^2 + (2-5)^2 + (0-4)^2 + .. + (4-1)^2 = 38

Movie2, Movie3 = (5-3)^2 + (1-5)^2 + (3-3)^2 + .. + (3-2)^2 = 21

Movie2, Moviem = (5-3)^2 + (1-5)^2 + (3-4)^2 + .. + (3-1)^2 = 25

Movie3, Moviem = (3-3)^2 + (5-5)^2 + (3-4)^2 + .. + (2-1)^2 = 2

Yukarıda gördüğümüz gibi, her movie çifti için Öklid uzaklığı ile tüm Movie'lerin uzaklığını hesaplıyoruz. Bu uzaklık temelli bir yöntemdir.

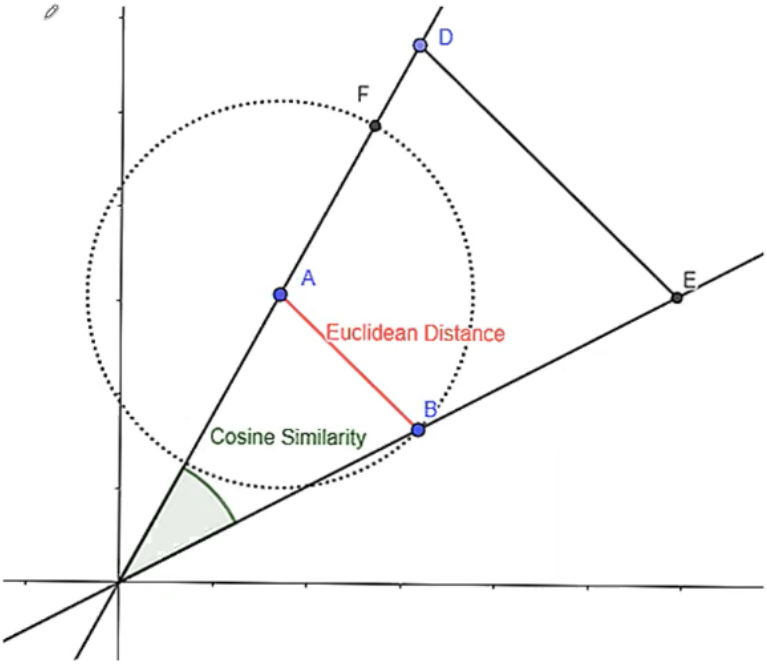
$$d(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

Cosine Similarity (Kosinüs Benzerliği)

Farklı bir vektör uzaklığı hesaplama işlemidir. Benzerlik temellidir.

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}$$

Öklid Uzaklığı ile Kosinüs Benzerliği



TF-IDF

- TF-IDF = TF(t) * IDF(t)
- ADIM 1: TF(t) = (Bir t teriminin ilgili dokümanda gözlenme frekansı) / (Dokümandaki toplam terim sayısı) (term frequency)
- ADIM 2: IDF(t) = 1 + log_e((Toplam doküman sayısı + 1) / (İçinde t terimi olan doküman sayısı + 1) (inverse document frequency)
- ADIM 3: TF-IDF = TF(t) * IDF(t)
- ADIM 4: TF-IDF Değerlerine L2 normalization.
  - L2 normalization işlemi satır bazında olur (document).
  - Bir satırdaki her bir gözlemin karesi alınarak toplanır ve toplamın karekökü alınır.
  - Satırdaki tüm gözlemler karekök sonucunda elde edilen değerlere bölünür.

Python'da Content Based Filtering

```
# !pip install sklearn
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import TfidfVectorizer
```

Collaborative Filtering (İş Birlikçi Filtreleme)

Elimizde sağlıklı user-based veya item-based veri yoksa, content-based filtering yapmalıyız.

Temelde 3 Collaborative Filtering Tipi Vardır

- Item-Based Collaborative Filtering
  - Memory Based olarak karşımıza çıkabilir
  - Temelinde ürünlerin beğenilme alışkanlıklarını ifade eder
  - X filmi ile benzer beğenilme yapısına sahip filmler önerilir
- User-Based Collaborative Filtering
  - Memory Based olarak karşımıza çıkabilir
  - Temelinde kullanıcıların beğenme alışkanlıklarını ifade eder
  - Öneri yapılmak istenen kullanıcı ile aynı filmleri izleyen ve aynı beğenme korelasyonuna sahip kullanıcıların izlediği filmler önerilir
- Model-Based Collaborative Filtering (Matrix Factorization)
  - Latent Factor Model olarak karşımıza çıkabilir
  -

Item-Based Collaborative Filtering

Temelinde ürünlerin beğenilme alışkanlıklarını ifade eder

X filmi ile benzer beğenilme yapısına sahip filmler önerilir

Bu sefer oluşturulacak matriste odağımız sütunlar olacaktır

	Movie 1	Movie 2	Movie 3	.	.	.	Movie n
User 1	1	4	1				4
User 2	5	2	5				3
User 3	3	5	2				5
.							
.							
.							
User m	3	1	4				1

Örnek olarak Movie2'i izleyen bir X kişisine hangi filmi önerirdik?

> Bu kişiye sezgisel olarak MovieN'i tavsiye ederdik. Bu sefer sütunlar bazında düşünürüz. MovieN'in beğenilme pattern'i Movie2 gibidir ([4-4][2-3][5-5][1-1]).

Python'da Item-Based Collaborative Filtering

Aslında bunu yapmak için ek bir kütüphaneye ihtiyaç duymayız. Korelasyon'a dayalı bir teknik olduğundan `corrwith` fonksiyonu ile yaparız.

```
# user movie df'inin oluşturulması.
user_movie_df = common_movies.pivot_table(index=["userId"], columns=["title"], values="rating")
user_movie_df.shape # bu dataframe'de satırlar kullanıcıları, sütunlar filmleri temsil eder. Kesişimlerde ise verilen ratingler var
user_movie_df.head(10)
user_movie_df.columns
len(user_movie_df.columns)
common_movies["title"].nunique()


#####################################
# Adım 3: Item-Based Film Önerilerinin Yapılması
#####################################


movie_name = "Matrix, The (1999)"
movie_name = user_movie_df[movie_name]
user_movie_df.corrwith(movie_name).sort_values(ascending=False).head(10)

movie_name = "Ocean's Twelve (2004)"
movie_name = user_movie_df[movie_name]
user_movie_df.corrwith(movie_name).sort_values(ascending=False).head(10)

# rastgele film seçimi
movie_name = pd.Series(user_movie_df.columns).sample(1).values[0]
movie_name = user_movie_df[movie_name]
user_movie_df.corrwith(movie_name).sort_values(ascending=False).head(10)
```

User-Based Collaborative Filtering

Temelinde kullanıcıların beğenme alışkanlıklarını ifade eder

Öneri yapılmak istenen kullanıcı ile aynı filmleri izleyen ve aynı beğenme korelasyonuna sahip kullanıcıların izlediği filmler önerilir

Burada da Item-Based filtering de kullandığımız veri yapısını kullanırız

Python'da User-Based Filtering

Aslında bunun için de `corrwith` fonksiyonunu kullanıyoruz. Fakat burada veri yapısını oluşturmak biraz daha zahmetli.

- Önce User-Movie Matrisi (item-based için yaptığımız gibi) oluşturacağız
- Öneri yapacağımız kullanıcının izlediği filmleri belirliyoruz
- Hedef kullanıcımız ile aynı filmleri izleyen kullanıcılara erişiyoruz
- Öneri yapacağımız kullanıcı ile en benzer davranışlı kullanıcıları belirleyeceğiz
- Ardından seçtiğimiz hedef kullanıcı ile aynı filmleri izlemiş kişilerin korelasyonlarını hesaplayıp en yüksek korelasyona sahip olanları önereceğiz

Model-Based Filtering (Matrix Factorization)

Burada da Item-Based filtering de kullandığımız veri yapısını kullanırız.

	Movie 1	Movie 2	Movie 3	.	.	.	Movie n
User 1	1	2	BLANK				4
User 2	5	BLANK	3				3
User 3	3	5	5				BLANK
.							
.							
.							
User m	BLANK	2	1				1

Fakat burada matristeki boş hücreleri doldurmamız gerekmektedir. Matematiksel bir model kullanarak bu boş hücreleri dolduracağız.

	Movie 1	Movie 2	Movie 3	.	.	.	Movie n
User 1	1	2	BLANK				4
User 2	5	BLANK	3				3
User 3	3	5	5				BLANK
.							
.							
.							
User m	BLANK	2	1				1

	Movie 1	Movie 2	Movie 3	.	.	.	Movie n
User 1	1	2	3				4
User 2	5	2	3				3
User 3	3	5	5				1
.							
.							
.							
User m	5	2	1				1

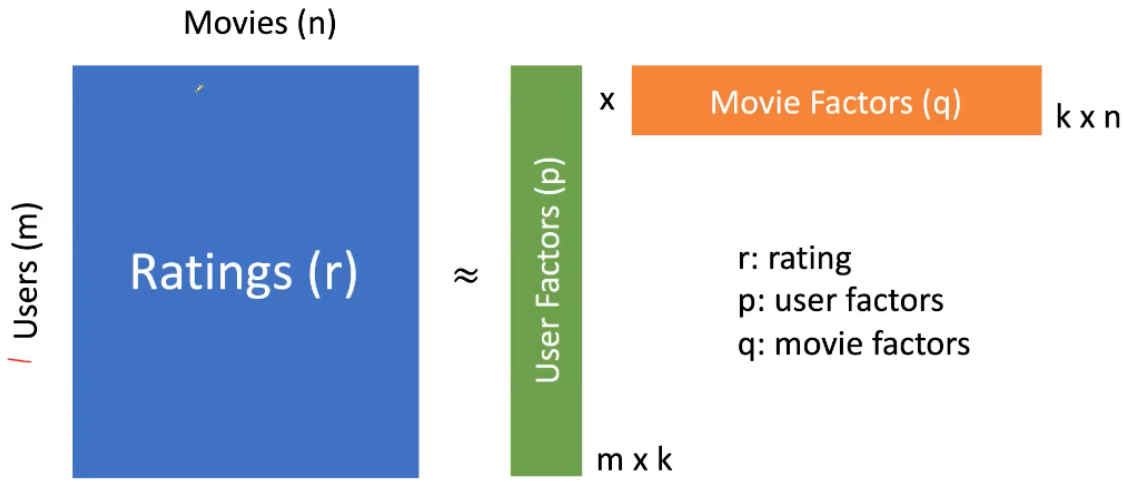
Bu User bu Movie'ye bu puanı verir diyeceğiz.

Boşlukları doldurmak için user'lar ve movie'ler için var olduğu varsayılan latent (gizli) feature'ların ağırlıkları, var olan veri üzerinden bulunur ve bu ağırlıklar ile var olmayan gözlemler için tahmin yapılır.

Matrix Factorization:

- User-Item matrisini 2 tane daha az boyutlu matrise ayrıştırır

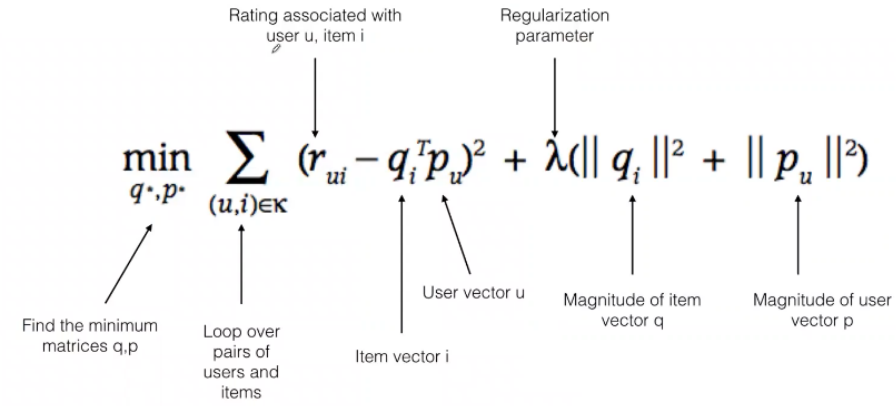
- 2 matristen User-Item matrisine gidişin latent factor'ler ile gerçekleştiği varsayımında bulunur
- Dolu olan gözlemler üzerinden latent factor'lerin ağırlıklarını bulur
- Bulunan ağırlıklar ile boş olan gözlemler doldurur



User-Item matrisi (solda)

Kullanıcıların Item'ları seçerken etki eden factor'ler matrisi (sağda)

Matrix Factorization der ki; Bir kullanıcı bir item seçerken göz önünde bulundurduğu bazı gizli (latent) faktörler vardır. Aslında bir kullanıcı bir item seçtiğinde var olan bu gizli faktörlerin match'i söz konusudur ve bunlar ortaya puanı çıkarır.

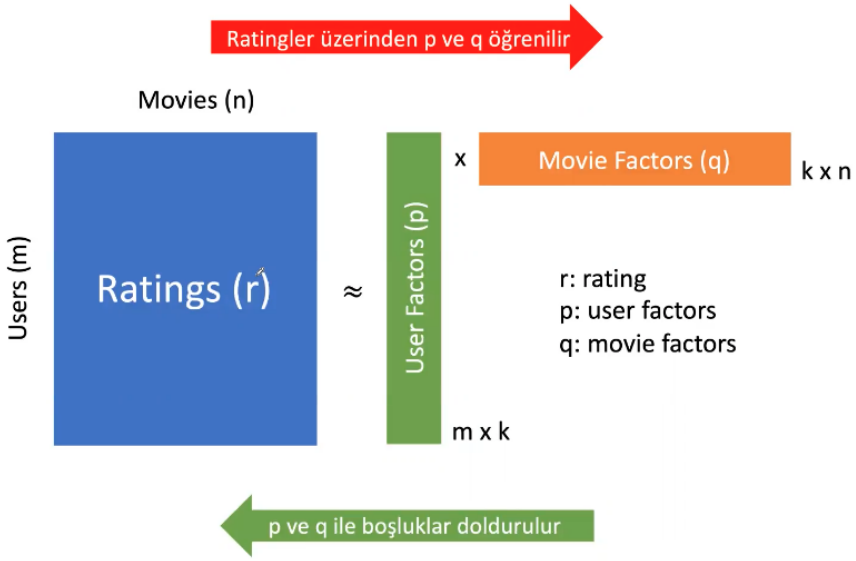


Boşlukları doldurmak için user'lar ve movie'ler için var olduğu varsayılan latent feature'ların ağırlıkları **var olan veri üzerinden bulunur** ve bu ağırlıklar ile var olmayan gözlemler için tahmin yapılır.

- Rating matrisinin iki factor matrisinin çarpımı (dot product) ile oluştuğu varsayılır
- Factor matrisler? user latent factors, movie latent factors
- Latent factors? Latent features? Gizli faktörler ya da değişkenler
- Kullanıcıların ve filmlerin latent feature'lar için skorlara sahip olduğu düşünülür
- Bu ağırlıklar (skorlar) önce var olan veri üzerinden bulunur ve sonra BLANK bölümler bu ağırlıklara göre doldurulur

Örnek olarak bazı latent factor olabilecek değişkenler:

- Filmin türü
- Belirli bir oyuncunun olup olmaması
- Filmin süresi vs.



Örnek

Ratings (r)

	M1	M2	M3	M4
U1	1	2		4
U2	5		3	3
U3	3	5	5	
U4		2	1	1

m x n

≈

p (user factors)

	F1	F2
U1	p_{11}	p_{12}
U2	p_{21}	p_{22}
U3	p_{31}	p_{32}
U4	p_{41}	p_{42}

m x k

x

q (movie factors)

	M1	M2	M3	M4
F1	q_{11}	q_{12}	q_{13}	q_{14}
F2	q_{21}	q_{22}	q_{23}	q_{24}

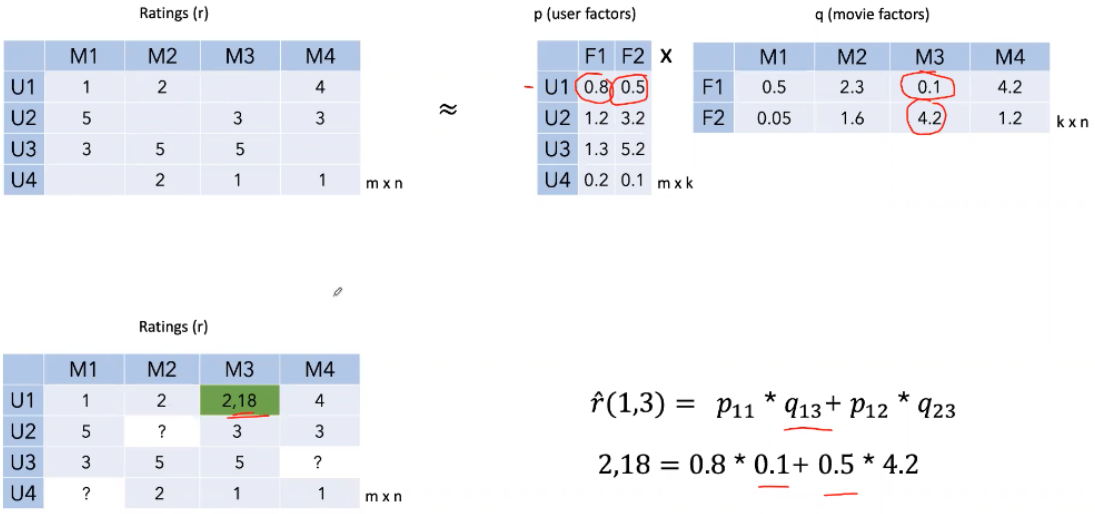
k x n

$$r_{11} = p_{11} * q_{11} + p_{12} * q_{21} \qquad r_{21} = p_{21} * q_{11} + p_{22} * q_{21} \qquad r_{33} = p_{31} * q_{13} + p_{32} * q_{23}$$

$$1 = p_{11} * q_{11} + p_{12} * q_{21} \qquad 5 = p_{21} * q_{11} + p_{22} * q_{21} \qquad 5 = p_{31} * q_{13} + p_{32} * q_{23}$$

- Var olan değerler üzerinden iteratif şekilde tüm p ve q'lar bulunur ve sonra kullanılır
- Başlangıçta rastgele p ve q değerleri ile rating matrisindeki değerler tahmin edilmeye çalışılır

- Her iterasyonda hatalı tahminler düzenlenerek rating matristeki değerlere yaklaşılmaya çalışılır
- Örneğin bir iterasyonda 5'e 3 dendiyse (tahmin) sonrakinde 4 sonrakinde 5 denir (tahmin)
- Böylece belirli bir iterasyon sonucunda p ve q matrisleri doldurulmuş olur
- Var olan p ve q'lara göre boş gözlemler için tahmin yapılır



Büyük veri ortamında ALS kullanılması gerekmektedir. Yukarıdaki matrix factorization yöntemi gibidir fakat p ve q ağırlıklarını bulurken uyguladığı ufak bir optimizasyon farklılığı vardır.

ALS: Spark ALS Documentation

NOT: Bu matrix factorization işlemi bazı kaynaklarda SVD olarak da anlatılmaktadır. SVD (Singular Value Decomposition) bir boyut indirgeme işlemidir (PCA gibi). Matrix Factorization ile SVD aynı işlem gibi düşüneceğiz. Detaylı okumalar için:

- Singular Value Decomposition vs. Matrix Factorization in Recommender Systems

Python'da Matrix Factorization

Python Surprise Package for Matrix Factorization