Nenegativni matrični rastavi

Josipa Marelja, Domina Sokol

10.06.2021.

Sadržaj

1 Uvod

Nenegativni matrični rastavi (*eng. non-negative matrix factorization, skr. NMF*) su grupa algoritama koji pripadaju polju multivarijabilne analize (*eng. multivariate analysis*) i polju linearne algebre. Cilj je početnu nenegativnu matricu prikazati kao umnožak dviju drugih nenegativnih matrica.

Uvjet nenegativnosti osigurava lakše promatranje matrica, tj. lakše provođenje matričnog računa. Također, za određene tipove podataka nenegativnost se podrazumijeva (npr. audio spektrogrami, aktivnost mišića...). Ponekad se radi s podacima za koje negativne vrijednosti jednostavno nemaju smisla.

Ovakva faktorizacija nije uvijek moguća pa joj se obično pristupa numerički umjesto analitički.

2 Teorijska pozadina NMF

2.1 Osnovna ideja

Počinje se od nenegativne matrice (nenegativnost se odnosi na činjenicu da matrica ne sadrži negativne elemente) koja se faktorizira na dvije druge matrice tako da nijedna opet ne sadrži negativne elemente.

Neka je V nenegativna matrica (proizvoljnog tipa). Cilj NMF je prikazati V kao umnožak nekih drugih nenegativnih matrica W, H. Formalno, tražimo rastav:

$$V = W \cdot H$$

gdje je

$$V \in M_{m,n}(\mathbb{R}), W \in M_{m,p}(\mathbb{R}), H \in M_{p,n}(\mathbb{R}).$$

Matrično množenje se implementira po retcima matrice V:

$$v_i = W \cdot h_i; \forall i \in \{1, \ldots, n\}$$

Ključno svojstvo matričnog množenja koje NMF iskorištava je proizvoljna veličina dimenzije p kod ulančavanja produkta W i H. Naime, m i n su određeni tipom početne matrice V, ali p može biti bilo koji prirodni broj. To znači da kod množenja matrica dimenzije faktora mogu biti puno manje nego dimenzija produkta (u smislu broja elemenata). NMF koristi ovo svojstvo tako da generira faktore značajno manjih dimenzija u usporedbi s originalnom matricom.

Primjer:
Neka je V tipa (10000×500).
$V \approx W \cdot H$ gdje je W tipa (10000×10), a H tipa (10×500).
V sadrži 5 milijuna elemenata! W i H, redom, sadrže 100000 i 5000 elemenata.

Na ovaj način može se intuitivno predstaviti jedan problem iz obrade prirodnog jezika. Konkretnije, problem pojavljivanja određenih riječi u tekstu (skupu tzv. dokumenata). Neka V predstavlja broj pojavljivanja riječi po dokumentima, na način da stupci označavaju dokumente (npr. rečenice), a retci riječi. Tada se u umnošku $W \cdot H$ matrica W može shvatiti kao matrica značajki (*eng. feature matrix*), a H kao matrica koeficijenata (*eng. coefficients matrix*).

Sada se svaki originalni dokument (redak matrice V) može shvatiti kao da je sagrađen od manjeg skupa sakrivenih značajki (*eng. features*). Uloga NMF je da generira tražene (korisne) značajke.

2.2 Primjene

Kao što je pokazano na prethodnom primjeru, NMF ima razne uloge u različitim znanstvenim poljima. Neka od polja gdje se često nailazi na NMF uključuju:

- astronomiju
  Kao metoda smanjenja dimenzionalnosti podataka koja se primjenjuje u spektroskopiji i procesuiranju te postprocesuiranju slika.
- statistiku
  Dopuna nedostajućih podataka (*eng. data imputation*).
- rudarenje teksta (*eng. text mining*)
  Nalaženje odnosa između dokumenata i riječi u tekstu.
- denoiziranje govora (*eng. speech denoising*)
  Uklanjanje šuma iz signala koji prenosi govor rastavom čitavog signala na šum i bitne podatke (govor).
- bioinformatiku
  Analiza gena i proteina.

2.3 Formalizacija

Budući da se radi o numeričkoj metodi, cilj faktorizacije je da što bolje aproksimira početnu matricu. To podrazumijeva minimizaciju funkcije greške u rastavu. Time pronalazak zadovoljavajućeg rastava prelazi u optimizacijski problem.

Aproksimacija $V \approx WH$ se nalazi minimizacijom funkcije greške

$$\|V - WH\|_F; W, H \geq 0$$

gdje se koristi Frobeniusova norma (matrična Euklidska norma):

$$\|A\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2} = \sqrt{tr(A^*A)}$$

Intuitivno, poželjno je smanjiti udaljenost između aproksimacije WH i stvarne vrijednosti V.

Razlog zbog kojeg se koristi numerički pristup umjesto analitičkog je taj što problem nenegativne faktorizacije pripada NP (*non-deterministic polynomial*) klasi složenosti algoritama. Jednostavno rečeno, rezultat faktorizacije lako se provjerava (množenjem matrica W i H), ali nije "lagan" za pronaći. Točnije, nije dokazano da se traženi rezultat može izračunati u polinomijalnom vremenu.

Dakle, prethodni izraz $V \approx WH$ možemo zamijeniti sa

$$V = WH + U$$

gdje je U rezidual koji ne mora nužno biti nenegativan.

2.3.1 Algoritam Hadamardovog množenja

Najjednostavniji algoritam pronalaska W i H je algoritam koji koristi multiplikativno ažurirajuće pravilo (*eng. multiplicative update rule*). Za definiranje ažurirajućih pravila iteracija Hadamardovim produktom prvo će se fiksirati jedan faktor, npr. W te zatim minimizirati funkciju troška s obzirom na drugi faktor.

Funkcija troška može se zapisati u obliku:

$$\|V - WH\|_F^2 = \sum_{i=1}^{n} \|V_{:i} - WH_{:i}\|_2^2$$

odnosno rastaviti na n nezavisnih potproblema gdje se minimizira svaki stupac od H posebno. Tada se minimizira zapravo niz kvadratnih problema

$$\underset{h\geq 0}{min} F(h) = \underset{h\geq 0}{min} \|v - Wh\|_2^2$$

gdje su v, h stupci od V, H, redom.

Daljnjim računom, za fiksnu trenutnu aproksimaciju $\tilde{h} \geq 0$ se dobije da je gradijent funkcije troška F sljedeći:

$$\nabla_h F = W^T W h - W^T v + V_{\tilde{h}}(h - \tilde{h}).$$

Izjednačavanjem gradijenta s nulom se dobije izraz s minimumom $h*$:

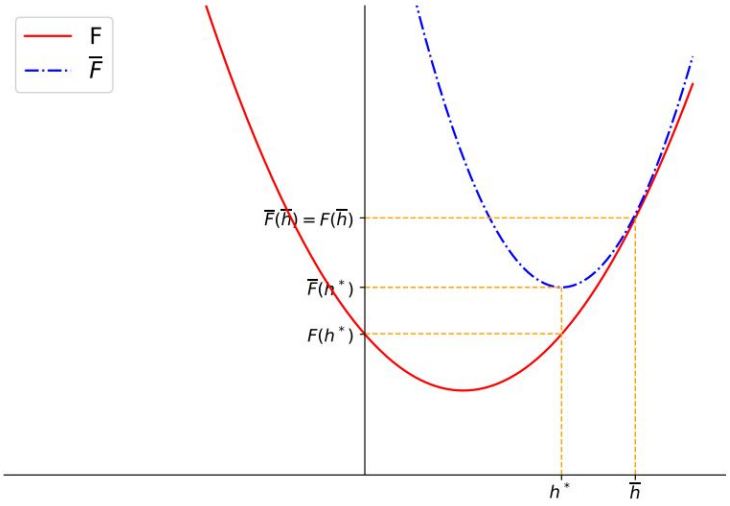
$$(W^T W + V_{\tilde{h}})h^* = W^T v - V_{\tilde{h}}\tilde{h}.$$

S obzirom da je $\tilde{h}$ globalni minimum funkcije troška u fiksiranoj iteraciji $\bar{F}$, vrijedi:

$$F(h^*) \leq \bar{F}(h^*) \leq \bar{F}(\tilde{h}) = F(\tilde{h})$$

što znači da postoji spust u funkciji troška. Ova pojava se može viziualizirati jednostavnim primjerom (Slika 1).

Pravilo ažuriranja za čitavu matricu H se dobije ponavljajući opisani postupak za svaki njen redak, a na sličan način se izvodi i pravilo ažuriranja za matricu W.



Slika 1: Nerastuća ciljna funkcija

1. Inicijalizacija W i H.
2. Računanje novih W i H:

$$H_{i,j}^{n+1} = H_{i,j}^{n} \frac{((W^n)^T V)_{i,j}}{((W^n)^T W^n H^n)_{i,j}}$$

$$W_{i,j}^{n+1} = W_{i,j}^{n} \frac{(V(H^{n+1})^T)_{i,j}}{(W^n H^{n+1}(H^{n+1})^T)_{i,j}}$$

dok se W i H ne stabiliziraju.

U ovom postupku se koristi Hadamardovo množenje (po koordinatama) umjesto "običnog" matričnog množenja. Zbog toga se ovaj algoritam ponekad naziva i algoritam Hadamardovog produkta. Ovo je najpopularniji i najstariji algoritam nenegativnih matričnih rastava kojeg su prvi put objavili Lee i Seung u svojem radu. Primijetimo da su faktori

$$\frac{W^T V}{W^T W H}, \frac{V H^T}{W H H^T}$$

matrice neutralnog elementa za Hadamardovo množenje

$$\begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix}$$

kad je $V = WH$ (točno jednako).

Bitno je napomenuti da ova faktorizacija nije jedinstvena. Naime, faktore W i H moguće je transformirati u drugačije faktore pomoću odgovarajuće matrice i njenog inverza:

$$WH = WBB^{-1}H = \tilde{W}\tilde{H}$$

gdje je B nenegativna generalizirana permutacijska matrica.

Ovi uvjeti su potrebni da bi se moglo garantirati nenegativnost novonastalih $\tilde{W}$ i $\tilde{H}$. Generalizirane permutacijske matrice su permutacijske matrice čiji nenula elementi ne moraju biti jedinice nego bilo koji nenula realni brojevi. Ovdje se dodatno zahtijeva uvjet nenegativnosti za B, pa bi jedan primjer takve matrice bio npr.:

$$\begin{bmatrix} 0 & 2 & 0 \\ 3 & 0 & 0 \\ 0 & 0 & 4 \end{bmatrix}.$$

Ostaje osigurati nenegativnost novonastalih W^{k+1}, H^{k+1}, uz uvjet da su W^k, H^k nenegativne. Zapravo se ovime želi izbjeći da algoritam "zaglavi" u nekoj graničnoj točki koja nije stacionarna, tj. da nije zadovoljen KKT uvjet nego vrijedi $\nabla_{H_{ij}} F < 0$.

Ako se pretpostavi da su početne W, H pozitivne, vrijedi:

$$\nabla_{H_{ij}^k} F < 0$$

odnosno

$$\left[(W^k)^T W^k H^k\right]_{ij} - \left[(W^k)^T V\right]_{ij} < 0$$

iz čega primjenom iteracije Hadamardovim produktom slijedi

$$H_{ij}^{k+1} = H_{ij}^k \frac{\left[(W^k)^T V\right]_{ij}}{\left[(W^k)^T W^k H^k\right]_{ij}} > H_{ij}^k > 0.$$

2.3.2 Algoritmi alternirajućih najmanjih kvadrata

Algoritam alternirajućih najmanjih kvadrata (eng. alternating least squares - ANLS) prvi put su predložili Paatero i Tapper u svom radu. Motivacija iza ovog algoritma je konveksnost funkcije minimizacije rastava u slučaju fiksiranja jednog od faktora V ili W. Fiksiranjem jednog faktora problem rastava se svodi na problem nenegativnih najmanjih kvadrata (eng. nonnegative least squares - NNLS).

U svakom koraku ažuriranja matrica H i W rješavaju se dva potproblema najmanjih kvadrata. Ovdje je dana osnovna ideja iza tog postupka:

1. Inicijalizacija $W^0 \geq 0, H^0 \geq 0$
2. Ponavljaj dok nije zadovoljen uvjet zaustavljanja:

   Riješi $W_{k+1} = \underset{W \geq 0}{argmin} \|V - WH^k\|_F^2$

   Riješi $H_{k+1} = \underset{H \geq 0}{argmin} \|V - W^{k+1}H\|_F^2$

Budući da metoda najmanjih kvadrata u svakom koraku daje optimalno rješenje za trenutnu iteraciju potproblema, to znači da svaka iteracija algoritma smanjuje grešku više od prethodno viđenog (osnovnog) algoritma. Loša strana toga je da je algoritam sporiji za izvođenje i teža za implementaciju.

Kao poboljšanje ove ideje osmišljen je algoritam alternirajućih najmanjih kvadrata (eng. alternating least squares - ALS) koji je neegzaktna verzija algoritma nenegativnih alternirajućih najmanjih kvadrata. Različito je to što se ne zahtijeva nenegativnost kod rješavanja problema najmanjih kvadrata u potproblemima. Budući da je egaktno rješenje zamijenjeno projekcijom rješenja neograničenog problema najmanjih kvadrata kako bi se naknadno dobila nenegativnost, gubi se svojstvo kovergencije. Međutim, ovaj postupak ubrzava izvođenje algoritma.

1. Inicijalizacija $W^0 \geq 0, H^0 \geq 0$
2. Ponavljaj dok nije zadovoljen uvjet zaustavljanja:

   Riješi $W_{k+1} = \underset{W \geq 0}{argmin} \|V - WH^k\|_F^2$

   $W^{k+1} = [W^{k+1}]_+$

   Riješi $H_{k+1} = \underset{H \geq 0}{argmin} \|V - W^{k+1}H\|_F^2$

   $H^{k+1} = [H^{k+1}]_+$

Ovdje je nenegativnost postignuta na najjednostavniji mogući način, zamjenom negativnih elemenata nulom. Ovime se postiže i dodatno dobro svojstvo matrica - rijetkost (eng. sparsity). Iako, za razliku od algoritma Hadamardovog množenja, ovdje se nula u matrici kasnije može zamijeniti nenula elementom u daljnjim iteracijama.

Razlika između ALS i ANLS je u tome što kod ANLS uvijek dolazi do spusta u funkciji minimizacije i ima bolju aproksimacijsku grešku, ali treba značajno više vremena za provedbu. Zato se ANLS rjeđe koristi u praksi. Obično se koriste obje metode u hibridnom obliku algoritma, gdje se započinje s ALS jer je brži i potiče rijetkost matrica, a dovršava s ANLS jer on uvijek konvergira.

Postoji i treća opcija, algoritam hijerarhijskih alternirajućih najmanjih kvadrata (eng. hierarchical alternating least squares - HALS) koji rješava potprobleme nenegativnih najmanjih kvadrata pomoću metode egzaktnog koordinatnog spusta. Svaki put se ažurira jedan stupac matrice W odnosno redak matrice H. Fiksira se $r-1$ stupaca (redaka) i minimizira se j-ti stupac (redak):

$$minJ_j(W_{:j}, H_{j:}) = \|R^{(j)} - W_{:j}H_{j:}\|_F^2$$

gdje je $R^{(j)}$ j-ti rezidual

$$R^{(j)} = V - \sum_{i \neq j}^{r} W_{:i}H_{i:}.$$

Sada se nađe stacionarne točke računajući gradijent u $W_{:j}$ i $H_{j:}$

$$0 = \frac{\partial J_j}{\partial W_{:j}} = W_{:j}H_{j:}H_{j:}^T - R^{(j)}H_{j:}^T$$

$$0 = \frac{\partial J_j}{\partial H_{j:}} = H_{j:}^T W_{:j}^T W_{:j} - (R^{(j)})^T W_{:j}$$

Slijedi da će ažurirajuća pravila za j-te komponente W i H biti

$$W_{:j} \leftarrow \left[\frac{R^{(j)}H_{j:}^T}{H_{j:}H_{j:}^T}\right]_+$$

$$H_{j:} \leftarrow \left[\frac{R^{(j)}W_{:j}}{W_{:j}^T W_{:j}}\right]_+$$

Rezidual se može zapisati u sljedećem obliku:

$$R^{(j)} = V - \sum_{i \neq j}^{r} W_{:i}H_{i:} = V - WH + W_{:j}H_{:j}$$

što možemo supstituirati u prethodne izraze za nove W i H čime se dobiva konačno pravilo ažuriranja W i H:

1. Inicijalizacija $W^0 \geq 0, H^0 \geq 0$
2. Ponavljaj dok nije zadovoljen uvjet zaustavljanja:

   za $j = 1, \ldots, r$ radi

   $$W_{j:}^{(k+1)} = \left[W_{:j} + \frac{[XH]_{:j}^T - W\left[HH^T\right]_{:j}}{[HH^T]_{jj}}\right]_+$$

   $$H_{j:}^{(k+1)} = \left[H_{j:} + \frac{\left[X^TW\right]_{:j}^T - H^T\left[W^TW\right]_{:j}}{[W^TW]_{jj}}\right]_+$$

   $k = k + 1$

Gore opisani algoritam HALS konvergira mnogo brže nego algoritam Hadamardovog množenja, pri čemu ima sličnu kompleksnost izvođenja. HALS je najbolja opcija i za rijetke i za guste matrice.

2.3.3 Gradijentni spust

Metoda projiciranog gradijentnog spusta (eng. projected gradient descent - PGD) može se koristiti zahvaljujući postignutoj nenegativnosti zamijenjivanjem negativnih elemenata nulom. Postupak se sastoji od nalaska gradijenta $\nabla F(h^k)$, odabira veličine koraka spusta a^k te projekcije ažuriranog rješenja na nenegativni dio $\mathbb{R}_+^n$:

$$h^{k+1} = \left[h^k - a_k \nabla F(h^k)\right]_+ .$$

Za poboljšanje brzine konvergencije gradijentnog spusta se često umjesto fiksne veličine koraka koristi neka metoda odabira koraka koja osigurava varijabilnost njegove veličine. Jedna takva metoda odabira koraka a^k je korištenjem tzv. Armijo pravila:

1. Inicijalizacija $W^0 \geq 0, H^0 \geq 0, 0 < \beta < 1, 0 < \sigma < 1, k = 1$
2. Ponavljaj dok nije zadovoljen uvjet zaustavljanja:

   $W^{k+1} = \left[W^k - a_k \nabla_W F(W^k, H^k)\right]_+$

   $a_k = \beta^{t_k}$ gdje je t_k najmanji $t \in \mathbb{N}$ t.d. je ispunjeno **(*)**

$$H^{k+1} = \left[H^k - a_k \nabla_H F(W^{k+1}, H^k)\right]_+$$
$a_k = \beta^{t_k}$ gdje je t_k najmanji $t \in \mathbb{N}$ t.d. je ispunjeno **(**)**

Armijo pravilo koristi izraze:
(*)

$$(1-\sigma)\left\langle \nabla_W F(W^k, H^k), W^{k+1} - W^k \right\rangle + \frac{1}{2}\left\langle W^{k+1} - W^k, (W^{k+1} - W^k)(H^k(H^k)^T) \right\rangle \leq 0$$

(**)

$$(1-\sigma)\left\langle \nabla_H F(W^{k+1}, H^k), H^{k+1} - H^k \right\rangle + \frac{1}{2}\left\langle H^{k+1} - H^k, (W^{k+1})^T W^{k+1}(H^{k+1} - H^k) \right\rangle \leq 0$$

kako bi se osiguralo dovoljan spust u svakoj iteraciji algoritma. Zbog ovoga je dobivena bolja konvergencija nego da se koristi fiksni korak spusta a.

2.3.4 Inicijalizacija algoritma i uvjet zaustavljanja

Budući da problem nenegativne matrične faktorizacije nije konveksan, za očekivati je da postoje lokalni minimumi. Iz tog razloga je bitno dobro inicijalizirati početne matrice kako bi se smanjila šansa "zaglavljivanja" algoritma u lokalnom minimumu umjesto pronalaska globalnog.

Nasumičnom inicijalizacijom je moguće započeti daleko od stacionarne točke što onda povlači duže vrijeme izvršavanja zbog potrebe za većim brojem iteracija kako bi algoritam kovergirao. Zato postoje neke metode inicijalizacije koje su bolje:

- Poboljšana metoda nasumične inicijalizacije - dodatnim korakom se inicijalna točka poboljšava na način da se nađe optimalni faktor za skaliranje koraka
- metode za klasteriranje - korištenjem centroida nađenim pomoću metoda za klasteriranje se inicijaliziraju stupci od W i zatim se inicijalizira H u odgovoru na nađeni W
- dekompozicija na singularne vrijednosti - svaki faktor ranga 1 najbolje aproksimacije ranga r od V može sadržavati negativne i pozitivne elemente. Svaki faktor ranga 1 se zamijeni s odgovarajućim pronađenim nenegativnim faktorima ranga 1, uz uvjet na maksimalnu moguću normu.

Kriterij zaustavljanja također ima nekoliko vrsta, a razlika je u tome na čemu se baziraju - razvoju funkcije troška, uvjetima optimalnosti ili razlici funkcije troška između dvije iteracije. Ovi kriteriji se uglavnom kombiniraju s dodatnim uvjetom poput maksimalnog broja iteracija ili vremenskim ograničenjem da bi se osiguralo da program u jednom trenutku staje, konvergirao ili ne.

3 Kod

Za samostalnu implementaciju je odabran osnovni algoritam Hadamardovog množenja, odnosno multiplikativnog ažurirajućeg pravila. Na njemu se lijepo mogu pokazati glavne ideje iteracija svih algoritama. U svakoj iteraciji se računa sa stupcima odnosno retcima odgovarajućih matrica.

Prvi program je pisan u Julia programskom jeziku. Tu postoji opcija pozivanja modula LinearAlgebra u kojem se nalaze implementirane neke ključne korištene funkcije. Na slici se nalazi čitava funkcija, s komentarima i primjerom ulaza s odgovarajućim izlazom.

3.1 Primjer algoritma u Julia programskom jeziku

Implementacija algoritma Hadamardovog produkta s multiplikativnim ažurirajućim pravilom (u Juliji):

```
using LinearAlgebra

function nnmf(X::Array{Int64,2},
              r::Integer;
              maxiter::Integer=1000,
              tol::Number=1e-4,
              V::Matrix{Float64}=rand(Float64, size(X, 1), r),
        #matrica V tipa float64 , dimenzija X*1 i uzimamo r slučajno izgeneriranih brojeva tipa Float
              W::Matrix{Float64}=rand(Float64, r, size(X, 2)))

    L = 0
    row, col = size(X)
```

Slika 2: Ulazni parametri funkcije

```
for iter in 1:maxiter
    #Prvi korak
    b = V * W
    #pogledamo umnozak matrica prije ulaska u petlju
    # Drugi korak
    V_new = copy(V)
    for k in 1:r
        for i in 1:row
            #uzimamo i-ti redak od X i pomoću apostrofa dobivamo adjungiranu matricu od itog retka
            #množimo je sa k-tim redkom matrice W i djelimo s (umnoškom
            #i-tog retka adjunktne matrice umnoška V i W->b i k-tog retka matrice W)
            weight = (X[i, :]' * W[k, :]) / (b[i, :]' * W[k, :])
            #izračunom dobivamo težinu koju pomnožimo s vrijednosti matrice na poziciji uređenog para (i,k)
            V_new[i, k] *= weight
        end
    end
```

Slika 3: Ažuriranje matrice V

Slika 4: Postupak množenja

```
b_new = V_new * W
# Četvrti korak
W_new = copy(W)
for j in 1:col
    for k in 1:r
        #uzimamo adjungiranu matricu j-tog stupca matrice X i množimo sa k-tim stupcem nove matrice V_new
        #podjelimo sa adjunktnom matricom j-tog stupca matrice b_new i pomnožimo s k-tim stupcem matrice V_new
        weight = (X[:, j]' * V_new[:, k]) / (b_new[:, j]' * V_new[:, k])
        W_new[k, j] *= weight
    end
end
```

Slika 5: Ažuriranje matrice W

```
        V, W = V_new, W_new;
        #računamo normu razlike prvobitne matrice X i umnoška njenog rastava
        L_new = norm(X - V * W)^2
        #gledamo razliku
        rel_diff = abs(L_new - L) / (abs(L) + 1)
        if rel_diff <= tol
            break
        end
        # ažuriramo vrijednosti od L
        L = L_new
        println("Iteration: $iter, Relative Difference: $rel_diff")
    end
    return V, W
end
```

Slika 6: Krajnja ažuriranja i izlaz funkcije

3.2 Primjena NMF na problem obrade prirodnog jezika

Kao još jedan program odabrana je primjena NMF na konkretan problem modeliranja i izvlačenja tema teksta iz zadanog korpusa. Ovaj primjer je odabran specifično jer se NMF često koristi u ove svrhe i vrlo je efektivan. Također, ovaj primjer je detaljnije objašnjen na početku seminara gdje je specificirano čemu odgovaraju retci odnosno stupci faktora koji se traže.

Ovaj program implementiran je u Python programskom jeziku zbog toga što postoji ugrađena funkcija koja izvršava NMF u modulu istog imena. U pozadini je algoritam NMF s minimizacijom Frobeniusove norme razlike aproksimacije i ulazne matrice. Koristi se nenegativna dvostruko-jednostruka dekompozicija singularnih vrijednosti (eng. Nonnegative Double Singular Value Decomposition) koji je baziran na dva SVD procesa: jedan aproksimira matricu podataka, a drugi aproksimira pozitivne dijelove rezultantnih parcijalnih SVD faktora koji koriste algebarsko svojstvo matrica jediničnog ranga.

NMF može početne matrice inicijalizirati na neke nasumične vrijednosti i prikladno ih skalirati. Model se također, po želji, može regularizirati dodatno koristeći standardne metode L_1 i L_2 norme. Ako se ova mogućnost ne iskoristi, regularizacija se svejedno vrši nad W i H.

	publish_date	headline_text
0	20030219	aba decides against community broadcasting lic...
1	20030219	act fire witnesses must be aware of defamation
2	20030219	a g calls for infrastructure protection summit
3	20030219	air nz staff in aust strike for pay rise
4	20030219	air nz strike to affect australian travellers
...	...	...
1103658	20171231	the ashes smiths warners near miss liven up bo...
1103659	20171231	timelapse: brisbanes new year fireworks
1103660	20171231	what 2017 meant to the kids of australia
1103661	20171231	what the papodopoulos meeting may mean for ausus
1103662	20171231	who is george papadopoulos the former trump ca...

1103663 rows × 2 columns

Slika 7: Izgled podataka

```
from sklearn.feature_extraction.text import TfidfVectorizer

vect = TfidfVectorizer(min_df=50, stop_words='english')

X = vect.fit_transform(documents.headline_text)
```

```
print(X)
```

```
  (0, 5904)     0.38150132624074345
  (0, 1502)     0.5178795390810941
  (0, 2273)     0.30728041415686325
  (0, 2810)     0.47191378105745035
  (0, 227)      0.5187809281395711
  (1, 2829)     0.5424693206334905
```

Slika 8: Vektorizacija

```
from sklearn.decomposition import NMF

model = NMF(n_components=10, random_state=5) # dimenzija na koju smanjujemo određuje broj tema

model.fit(X)

nmf_features = model.transform(X)
```

```
X.shape, nmf_features.shape, model.components_.shape # X, W, H
```

```
((1103663, 11213), (1103663, 10), (10, 11213))
```

Slika 9: Nenegativna matrična faktorizacija

Dodajemo imena stupcima matrice model.components_ (koja predstavlja H) jer stupci predstavljaju terme (riječi) u dokumentima.

```
components_df = pd.DataFrame(model.components_, columns=vect.get_feature_names())

components_df
```

	000	01	02	03	04	05	06	07	08	09	...	zimbabwean	zimbabwes	zinc	zinifex	z
0	0.000515	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000243	0.000000	0.000000	0.000000	0.0013
1	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000019	0.000000	0.000000	...	0.000000	0.000009	0.000601	0.000000	0.0013
2	0.000639	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.001682	0.000187	0.000000	0.000000	0.0001
3	0.000050	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000337	0.000537	0.000042	0.0000
4	0.000603	0.002029	0.011030	0.009208	0.010110	0.004755	0.004782	0.004933	0.005841	0.011264	...	0.000000	0.000763	0.000000	0.000000	0.0001
5	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000240	0.000548	0.000030	0.0003
6	0.000158	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000112	0.000331	0.000000	0.0000
7	0.001223	0.000000	0.000000	0.000000	0.000521	0.000105	0.000087	0.000790	0.000000	0.000000	...	0.004573	0.000989	0.000000	0.000786	0.0016
8	0.000022	0.028027	0.053097	0.042028	0.036621	0.038969	0.034321	0.025582	0.033360	0.037268	...	0.000000	0.000000	0.000320	0.000000	0.0001
9	0.002293	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.002326	0.002482	0.002140	0.001680	0.0014

10 rows × 11213 columns

Slika 10: Imenovanje dobivenih komponenti

Sada nalazimo najvažnije riječi za svaku temu.

```
for topic in range(components_df.shape[0]):
    tmp = components_df.iloc[topic]
    print(f'For topic {topic+1} the words with the highest value are:')
    print(tmp.nlargest(10))
    print('\n')
```

```
For topic 1 the words with the highest value are:
man         8.396817
charged     3.117248
murder      1.367349
jailed      0.891809
missing     0.880894
stabbing    0.727232
guilty      0.637090
arrested    0.600157
death       0.587411
sydney      0.532504
Name: 0, dtype: float64


For topic 2 the words with the highest value are:
interview    7.471284
extended     0.393083
michael      0.383856
david        0.226665
john         0.222362
```

Slika 11: Izvlačenje najbitnijih riječi za svaku dobivenu temu

Nalaženje teme dokumenta. Uzmimo npr. 105. dokument.

```
my_document = documents.headline_text[105]
my_document
```

```
'national gallery gets all clear after'
```

```
pd.DataFrame(nmf_features).loc[105]
```

```
0    0.000351
1    0.000000
2    0.000000
3    0.001151
4    0.019754
5    0.000000
6    0.000000
7    0.000000
8    0.000000
9    0.001774
Name: 105, dtype: float64
```

```
pd.DataFrame(nmf_features).loc[105].idxmax()
```

```
4
```

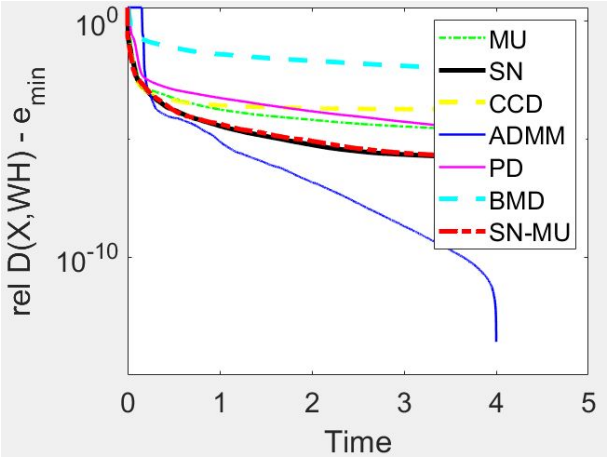
Slika 12: Primjer pronalaska teme za zadanu rečenicu

3.3 Primjer koda u Matlab programskom jeziku

Kao posljednji primjer, dan je program pisan u Matlab programskom jeziku. Ovaj kod nije napisan samostalno i nađen je na internetu. Ovdje je uključen jer vizualno prikazuje usporedbu vremena izvođenja različitih NMF algoritama. U programu su implementirani sljedeći algoritmi:

- NoLips algoritam - osmišljen za konveksne funkcije i proširen za nekonveksne. Radi se o metodi gradijentnog spusta dizajniranog za minimizaciju funkcije koja zadovoljava općenitiji uvjet od Lipschitzove neprekidnosti gradijenta.
- Scalar Newton - gradijentni spust koristeći Newtonov korak
- MU - multiplikativno ažurirajuće pravilo
- CCD - skraćeno za Cyclic Coordinate Descent - ciklički koordinatni gradijentni spust. Varijacija koordinatnog gradijentnog spusta spomenutog ranije.
- ADMM - skraćeno za Alternating Direction Method of Multiplier - metoda faktora alternirajućeg smjera. Radi se o metodi konveksne optimizacije koja dijeli probleme na manje dijelove za lakše nalaženje rješenja uvođenjem pomoćnih varijabli.
- Primal dual - izraz primarno-dualno se odnosi na činjenicu da se ne radi o čistoj minimizaciji nego o kombinaciji minimizacije i maksimizacije funkcije troška.
- SN-MU - Sparse Nonnegative Multiplicative Update - rijetko nenegativno ažurirajuće pravilo. Radi se o nadogradnji na metodu ažurirajućeg pravila koja uz očuvanje nenegativnosti potiču i rijetkost matrica.

Budući da su ovi algoritmi kompleksnije nadogradnje na osnovne algoritme, ovdje nisu detaljno objašnjeni, uostalom i zbog toga što je potrebno više pojmova da bi se prikladno objasnili. Ipak, odabrano je da ih se uključi u ovaj seminar da bi se demonstriralo koliko je tema NMF popularna, korisna i koliko se na njoj još uvijek radi.



Slika 13: Vremena izvođenja

4 Zaključak

U ovom seminarskom radu dana je osnovna ideja nenegativne matrične faktorizacije uz detaljnija objašnjenja klasičnih algoritama multiplikativnog ažurirajućeg pravila, alternirajućih najmanjih kvadrata i metode pomoću gradijentnog spusta.

Nabrojane su neke najčešće primjene u znanosti, i detaljnije je prikazana primjena u obradi prirodnog jezika uz što je ponuđen i primjer implementacije u Python programskom jeziku koji je danas široko korišten.

Nenegativna matrična faktorizacija je dobro istražena tema, ali i dalje je aktualna. Poboljšanja se i dalje nalaze čime je gotovo osigurano da svatko kome zatreba ova metoda može naći prikladnu implementaciju, u vidu korištenih iterativnih metoda, ili vremenske složenosti izvođenja.

Literatura

[1] `https://perso.uclouvain.be/paul.vandooren/ThesisHo.pdf`

[2] `https://scikit-learn.org/stable/modules/decomposition.html#nmf`

[3] `https://www.di.ens.fr/~aspremon/PDF/SymNMF.pdf`

[4] `https://www.cs.purdue.edu/homes/dgleich/conf/slides/mmm2012/dhillon.pdf`

[5] `https://www.di.ens.fr/~aspremon/PDF/SymNMF.pdf`

[6] `https://www.cs.toronto.edu/~cuty/Fast_NMF.pdf`