Projekt 1 - Metody klasyfikacji i redukcji wymiaru

Klaudia Weigel

1 Specyfikacja problemu

Mając dany zbiór ocen filmów mamy podzielić go na dwa podzbiory i na podstawie pierwszego podzbioru spróbować odgadnąć brakujące oceny. Drugi podzbiór służy do przetestowania naszych prognoz. Zaimplementowane algorytmy do przewidywania ocen to NMF oraz dwa warianty SVD. Dane mają być podzielone tak, aby 90% ocen każego z użytkowników znajdowało się w zbiorze treningowym, czyli tym na którym uruchamiany jest algorytm.

2 Wczytanie i podział danych

Oryginalny plik z danymi zawiera oceny 610 użytkowników, którzy ocenili 9724 filmy. Łącznie 100836 ocen. Format pliku to:

```
userId,movieId,rating,timestamp
1,1,4.0,964982703
1,3,4.0,964981247
1,6,4.0,964982224
1,47,5.0,964983815
1,50,5.0,964982931
...
```

Gdzie `userId` to identyfikator użytkownik, `movieId` to identyfikator filmu, a `rating` to ocena.
Dane do testowania programu zostały wczytane za pomocą modułu pandas i podzielone na zbiory testowy i treningowy przy użyciu funkcji groupby oraz sample. Dane zostały pogrupowane ze względu na użytkownika, a następnie z każdej grupy zostało losowo wybranych około 90% ocen.

```
df = pd.read_csv('ratings.csv', usecols = ['userId', 'movieId', 'rating'])
# Grupujemy dane ze względu na użytkownika i z każej grupy
# wybieramy losowo około 90% ocen
train_df = df.groupby('userId').apply(lambda x: x.sample(frac=0.9))
train_df.to_csv('training_ratings.csv', index = False)

# Łączymy razem oryginalne dane i zbiór treningowe, następnie
# usuwamy elementy występujące podwójnie
test_df = pd.concat([df, d2]).drop_duplicates(keep = False)
test_df.to_csv('test_ratings.csv', index = False)
```

3 Uruchomienie i opis działania programu

Program został napisany w Pythonie wersji 3.8.2. Uruchamiany jest poleceniem:

```
python3 NAZWA_PROGRAMU --train TRAIN_FILE --test TEST_FILE --alg ALG --
    result RESULT_FILE
```

Gdzie `TRAIN_FILE` to nazwa pliku treningowego, `TEST_FILE` to nazwa pliku testowego, `RESULT_FILE` to nazwa pliku do którego zapisany będzie wynik (jeli plik nie istnieje zostanie utworzony, kolejne wyniki są dopisywane), a `ALG` to nazwa algorytmu, dostępne warianty to `nmf`, `svd1` i `svd2`. Program po wczytaniu danych z pliku treningowego i testowego, tworzy macierze $\boldsymbol{Z}$ i $\boldsymbol{V}$ rozmiaru $n \times d$, gdzie n jest ilością użytkowników, a d ilością filmów i których elementami są oceny. Macierz $\boldsymbol{Z}$ przybliżana jest wybranym algorytmem, a następnie sprawdzane jest jak dobrze zostały przewidziane oceny ze zbioru testowego. Posłuży do tego miara RMSE:

$$RMSE = \sqrt{\frac{1}{|T|} \sum_{(u,m) \in T} (\boldsymbol{Z'}[u,m] - \boldsymbol{V}[u,m])^2}$$

Gdzie T jest zbiorem indeksów (użytkownik, film), które należą do zbioru testowego, a $\boldsymbol{Z'}$ przybliżoną macierzą. Program wyliczy RMSE i zapisze wynik do pliku `RESULT_FILE`.

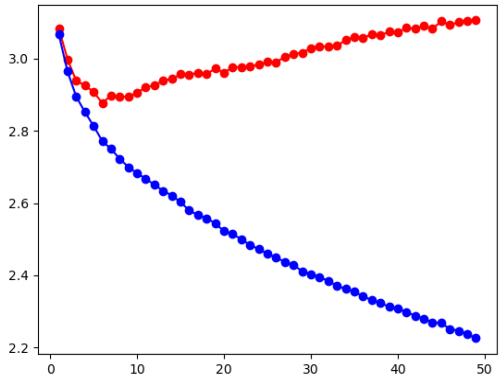
4 Algorytm NMF

Algortm NMF polega na przybliżeniu oryginalnej macierzy, iloczynem $\boldsymbol{W}\boldsymbol{H}$, gdzie $\boldsymbol{W}$ jest macierzą rozmiaru $n \times r$, a $\boldsymbol{H}$ macierzą rozmiaru $r \times d$. W Pythonie do wyznaczenia tych macierzy użyjemy funkcji NMF biblioteki `scikit-learn`. Ponieważ algorytm NMF biblioteki scikit-learn działa tylko na pełnej macierzy zostało przetestowane wypełnienie pustych miejsc zerami, średnią po rzędach (średnią ocen użytkownika) oraz średnią po kolumnach (średnią ocen filmu).

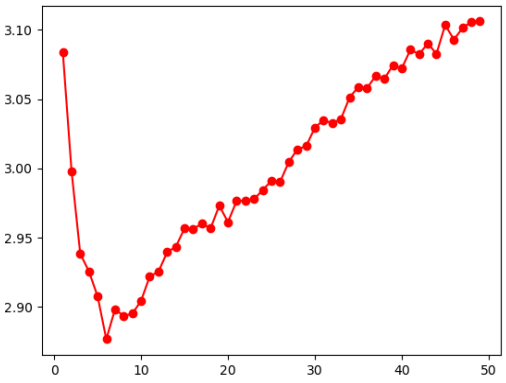
4.1 Wypełnienie pustych miejsc zerami.

Puste miejsca w macierzy treningowej zostały wypełnione zerami, a następnie algorytm został przetestowany dla $r \in [1, 50)$
```
r = 1, rmse = 3.093982
r = 2, rmse = 3.007912
r = 3, rmse = 2.949374
r = 4, rmse = 2.926061
r = 5, rmse = 2.910239
...
r = 47, rmse = 3.103491
r = 48, rmse = 3.107112
r = 49, rmse = 3.110596
```

RMSE dla tego wariantu wynosi około 3. Na wykresie zostało przedstawione także RMSE dla zbioru treningowego. Widać, że m większy parametr r, tym bardziej dane dopasowywane są do zbioru treningowego, dając tym samym coraz gorsze RMSE.



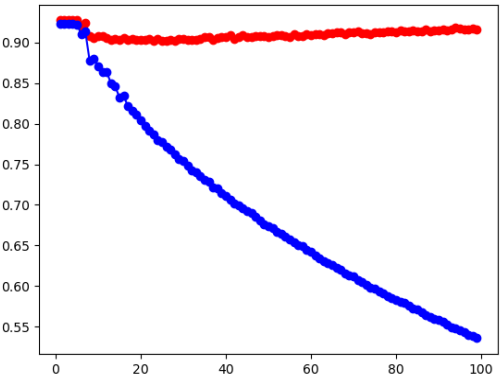
(a) Wykres testowego(czerwony) i treningowego(niebieski) RMSE



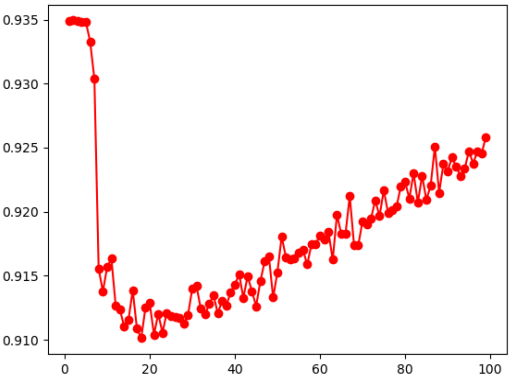
(b) Wykres testowego RMSE

Figure 1: NMF wypełnienie zerami

4.2 Wypełnienie pustych miejsc średnią ocen użytkownika



(a) Wykres testowego (czerwony) i treningowego (niebieski) RMSE



(b) Wykres testowego RMSE

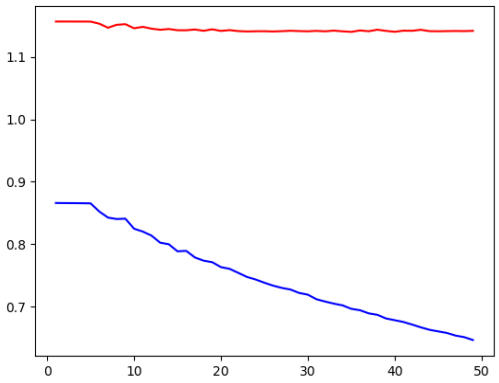
Figure 2: NMF - średnia ocen użytkownika

RMSE zbioru testowego jest w okolicach 0.9. Sprawdźmy dla jakiego r osiągniemy najniższe RMSE. Aby to zrobić przetestujemy algorytm na dziesięciu różnych parach zbiorów:

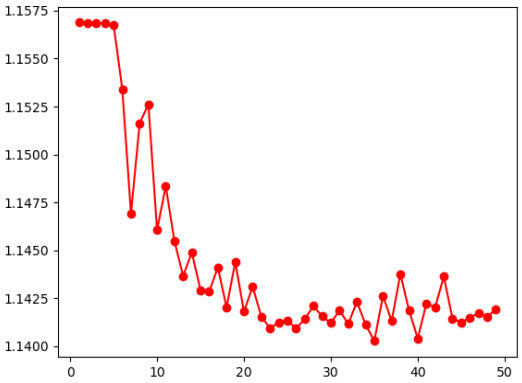
```
1.  Rmse min:  0.910120, r:  18
2.  Rmse min:  0.916684, r:  21
3.  Rmse min:  0.901820, r:  25
4.  Rmse min:  0.898073, r:  28
5.  Rmse min:  0.894653, r:  21
6.  Rmse min:  0.911509, r:  14
```

```
7.  Rmse min:  0.910604, r:  28
8.  Rmse min:  0.904042, r:  18
9.  Rmse min:  0.910724, r:  22
10.  Rmse min:  0.907746, r:  17
```

4.3 Wypełnienie pustych miejsc średnią ocen filmu



(a) Wykres testowego (czerwony) i treningowego (niebieski) RMSE



(b) Wykres testowego RMSE

Figure 3: NMF - średnia ocen filmu

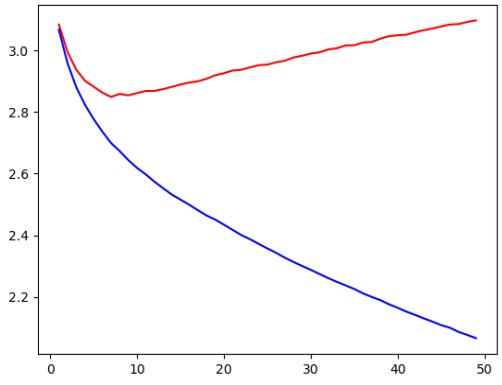
RMSE jest w okolicy 1, co jest gorszym wynikiem niż kiedy w puste miejsca wstawiana jest średnia ocen danego użytkownika.

W ostatecznej implementacji puste miejsca zapełniane są średnią ocen użytkownika, a $r = 21$.

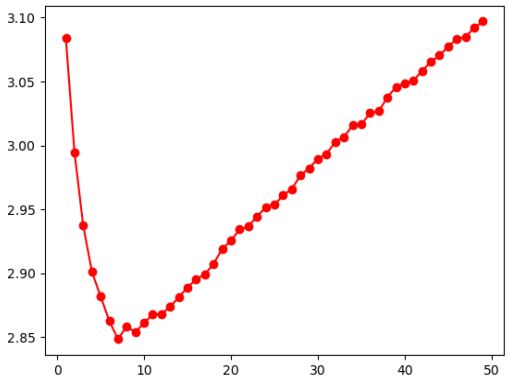
5 Algorytm SVD - wariant 1

Niech $\boldsymbol{U\Lambda V^T}$ będzie rozkładem SVD macierzy $\boldsymbol{Z}$. Nasz algorytm przybliży $\boldsymbol{Z}$ iloczynem $\boldsymbol{U_r\Lambda_r V_r^T}$, gdzie $\boldsymbol{U_r}$ to pierwsze r kolumn macierzy $\boldsymbol{U}$, $\boldsymbol{V_r}$ to pierwsze r kolumn $\boldsymbol{V}$, a $\boldsymbol{\Lambda_r}$ to obcięta do pierwszy r kolumn i r wierszy macierz $\boldsymbol{\Lambda}$. Do wyznaczenia tego iloczynu użyta została funkcja `TruncatedSVD` biblioteki `scikit-learn`. Podobnie jak NMF, TruncatedSVD działa tylko dla pełnej macierzy, zostały więc zastosowane takie same jak dla algorytmu NMF metody wypełnienia pustych miejsc.

5.1 Wypełnienie pustych miejsc zerami



(a) Wykres testowego (czerwony) i treningowego (niebieski) RMSE



(b) Wykres testowego RMSE

Figure 4: SVD1 wypełnienie zerami

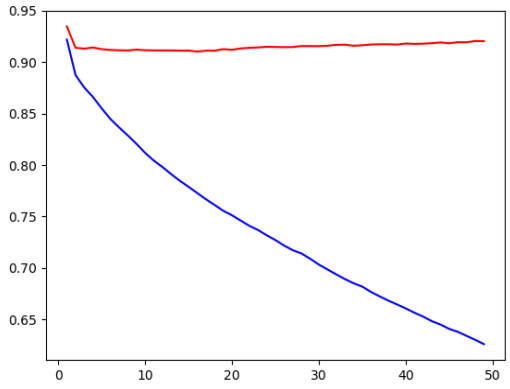
Podobnie jak w przypadku NMF, RMSE jest w okolicach trzech. Przetestujmy dla jakiego r osiągamy najmniejszy RMSE:

```
1.  Rmse min:  2.848548, r:  7
2.  Rmse min:  2.862318, r:  7
3.  Rmse min:  2.859439, r:  7
4.  Rmse min:  2.853045, r:  7
```

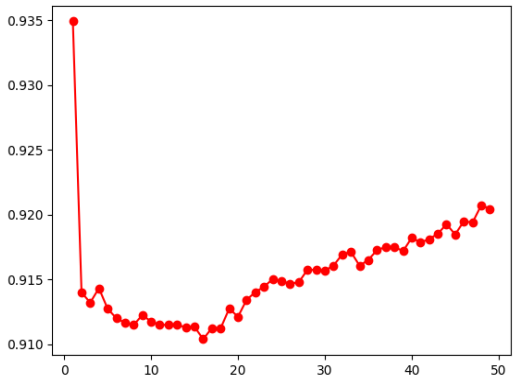
5.2 Wypełnienie pustych miejsc średnią ocen użytkownika

Dla dziesięciu różnych kombinacji zbiorów testowego i treningowego zostały obliczone minimalne $RMSE$:

```
1.  Rmse min:  0.910403, r:  16
2.  Rmse min:  0.917305, r:  14
3.  Rmse min:  0.899972, r:  11
4.  Rmse min:  0.897587, r:  13
5.  Rmse min:  0.895063, r:  9
6.  Rmse min:  0.911182, r:  14
7.  Rmse min:  0.910495, r:  13
8.  Rmse min:  0.903356, r:  11
9.  Rmse min:  0.910731, r:  6
10.  Rmse min:  0.895245, r:  11
```



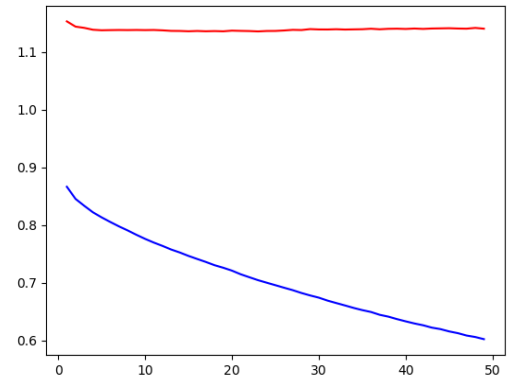
(a) Wykres testowego (czerwony) i treningowego (niebieski) RMSE



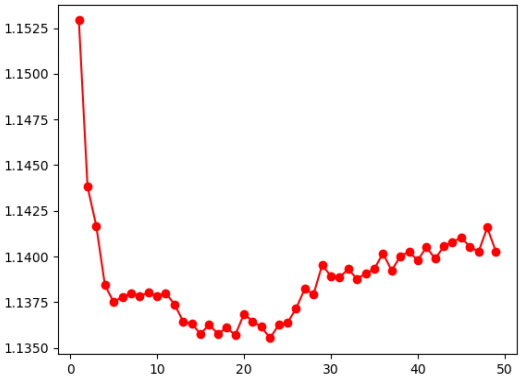
(b) Wykres testowego RMSE

Figure 5: SVD1 średnia ocen użytkownika

5.3 Wypełnienie pustych miejsc średnią ocen filmu



(a) Wykres testowego (czerwony) i treningowego (niebieski) RMSE



(b) Wykres testowego RMSE

Figure 6: SVD1 średnia ocen filmu

Podobnie jak w przypadku NMF, RMSE jest w okolicy 1.

W ostatecznej implementacji puste miejsca zapełniane są średnią ocen użytkownika, a $r = 11$.

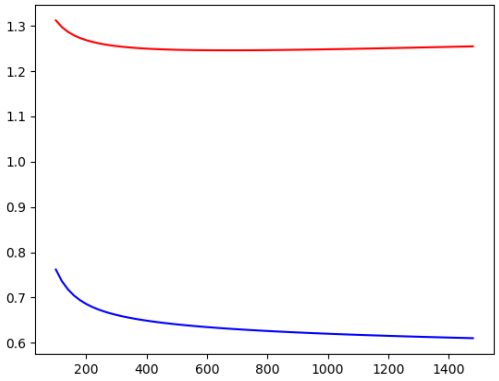
6 Algorytm SVD - wariant 2

Puste miejsca w macierzy $\boldsymbol{Z}$ wypełniane są zerami, a macierz $\boldsymbol{Z}^{(n)}$ definiujemy następująco:

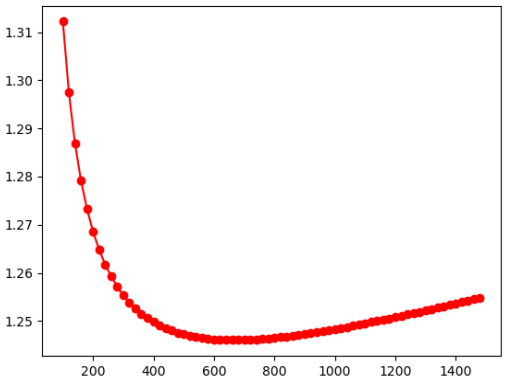
$$\boldsymbol{Z}^{(0)} = SVD_r[\boldsymbol{Z}]$$

$$\boldsymbol{Z}^{(n+1)} = SVD_r\left[\begin{cases} \boldsymbol{Z} & gdzie\ z_{ij} > 0 \\ \boldsymbol{Z}^{(n)} & wpp \end{cases}\right]$$

Gdzie SVD_r jest rozkładem SVD opisanym powyżej.



(a) Wykres testowego (czerwony) i treningowego (niebieski) RMSE



(b) Wykres testowego RMSE

Figure 7: SVD2 Wykres RMSE(n) dla r=7

Sprawdźmy dla jakiego n = N osiągane jest najmniejsze RMSE:

```
1.  Rmse min:  1.276479, N: 490
2.  Rmse min:  1.287496, N: 470
3.  Rmse min:  1.264424, N: 530
4.  Rmse min:  1.249645, N: 560
5.  Rmse min:  1.266998, N: 510
6.  Rmse min:  1.253367, N: 420
```

W implementacji $N = 500$.