אוניברסיטת בן גוריון בנגב

הפקולטה למדעי ההנדסה

המחלקה להנדסת מערכות מידע

מערכות המלצה

תרגיל בית מספר 1

מערכות המלצה מבוססות מודל

מגישים:

דניאל קפון 305596587

יובל חפץ 302957139

פרופ' ברכה שפירא

ליאת אנטברג-פרידמן

תאריך הגשה: 27.4.2018

תוכן עניינים

1. מבוא

בעבודה זו מימשנו מערכת המלצה לדירוג סרטים ע"י משתמשים על מאגר הנתונים MovieLens ml-20m. המימוש התבצע באמצעות שלושה מודלים שונים – SVD, SVD++ ו- content based model. בנוסף התבצעה השוואה למודלים דומים מתוך החבילה turicreate.

2. שיטות הערכה:

2.1 **RMSE**: לפי הנוסחה- $\sqrt{\frac{\sum r_{u,i}(r_{u,i}-\hat{r}_{u,i})^2}{N}}$

בשיטה זו השתמשנו לטובת אלגוריתם ה-gradient descent אשר יפורט בהמשך, וכן לטובת הערכת הביצועים של המודלים השונים על ה- test set.

2.2 **Precision**: ישנן מספר שיטות להערכת הדיוק אשר מתוארות בספרות. חלקן מעריכות את המודל בהקשר של השגיאות בין הדירוגים החזויים לדירוגים האמתיים (RMSE, MAE וכו') וחלקן מתייחסות לדיוק המודל לפי הצלחתו להמליץ על מוצרים רלוונטיים לכל משתמש.

מדד ההערכה הנוסף שבחרנו לממש הוא מהסוג השני, של בדיקת דיוק המודל. בחרנו לממש מדד זה על מנת לקבל משוב שונה מהמשוב ש-

RMSE נותן על מנת שנוכל לבחון את הצלחת המודל מכיוון נוסף- יכולתו להמליץ על סרטים רלוונטיים למשתמשים. הנוסחה הבסיסית שבה השתמשנו היא - $P@N = m^{+,N}/N$ כאשר N הוא מספר סרטים, לבחירת הממדל, בעלי הדירוג החזוי הגבוה ביותר של המשתמש, ו-$m^{+,N}$ הוא מספר הסרטים הרלוונטיים למשתמש מתוך קבוצת N הסרטים בעלי הדירוג החזוי הגבוה ביותר. הרלוונטיות של הסרטים יכולה להיקבע ע"י הממדל. דוגמה לסרטים רלוונטיים היא סרטים אשר קיבלו דירוג 5 (בפועל). חיסרון לשיטה זו יכול להיות כאשר אין למשתמש מספיק דירוגים של 5 אם בכלל ולכן יכולים להיות לא מעט משתמשים שהבדיקה לא רלוונטית עבורם.

אנחנו בחרנו לשנות מעט את שיטת בדיקה זו ולהעריך את הדיוק של המודל כסוג של בעיית סיווג לפי הסיווגים אוהב/ לא אוהב – הדיוק יחושב לפי היחס בין כמות הסרטים שדורגו 4.0 ומעלה בפועל מתוך הקבוצה של הסרטים אשר הדירוג החזוי שלהם הוא 4.0 ומעלה. כלומר מה אחוז הסרטים שדורגו בפועל מעל 4.0 מתוך כל הסרטים שדורגו מעל 4.0 בחיזוי (עבור כל משתמש).

$$Precision = \frac{TP}{TP + FP}$$

3. מודל בסיסי - SVD

המודל מומש לפי האלגוריתם המתואר בפרק 3.3.1 בספר הקורס. המודל המתבסס על Matrix Factorization, ממפה את מרחב ה- latent factors המשותפים למשתמשים ולמוצרים (סרטים במקרה זה).

חיזוי הדירוג של סרט i ע"י משתמש u מתבצע לפי הנוסחה:

$$\hat{r}_{ui} = \mu + b_i + b_u + q_i^T p_u \,.$$

כאשר q_i הינו ווקטור בגודל מס' ה-latent factors (k) המשויך למוצר i ומבטא את מידת השייכות של המוצר לכל factor, ו-p_u הינו ווקטור בגודל מס' ה-latent factors המשויך למוצר u ומבטא את מידת השייכות של המשתמש לכל factor.

μ הינו הדירוג הממוצע, bi הינו bias המזוהה עם מוצר i ו- bu מזוהה עם משתמש u.

למידת פרמטרי המודל מתבצעת ע"י הבאת הנוסחה הבאה למינימום:

$$\min_{b_*, q_*, p_*} \sum_{(u,i)\in\mathcal{K}} (r_{ui} - \mu - b_i - b_u - q_i^T p_u)^2 + \lambda_4 (b_i^2 + b_u^2 + \|q_i\|^2 + \|p_u\|^2) \,.$$

ישנן מספר שיטות למציאת הפרמטרים. במימוש שלנו השתמשנו בשיטת gradient descent, אשר מעדכנת בצורה איטרטיבית את הפרמטרים בכיוון הפוך לגרדיאנט הפונקציה. העדכון מתבצע באמצעות הנוסחאות –

$$
\begin{aligned}
b_u &\leftarrow b_u + \gamma \cdot (e_{ui} - \lambda_4 \cdot b_u) \\
b_i &\leftarrow b_i + \gamma \cdot (e_{ui} - \lambda_4 \cdot b_i) \\
q_i &\leftarrow q_i + \gamma \cdot (e_{ui} \cdot p_u - \lambda_4 \cdot q_i) \\
p_u &\leftarrow p_u + \gamma \cdot (e_{ui} \cdot q_i - \lambda_4 \cdot p_u)
\end{aligned}
$$

γ הינו פרמטר קבוע המבטא את קצה הלמידה, λ הינו פרמטר רגולריזציה למניעת overfitting. כל הפרמטרים הנלמדים אותחלו לערכים קטנים, חיוביים ושליליים בהתפלגות אחידה (*-0.05 - 0.05*).

האימון התבצע על הנתונים בקובץ ה-ratingTrains.csv. מאגר הנתונים אורגן במטריצה שבה מספר השורות הינו מספר הדירוגים הקיימים בקובץ, והעמודות הן מספרי ה-*ID* של המשתמשים והסרטים ועמודה של *rating*, כל רשומה מייצגת דירוג של סרט ע"י משתמש. הנתונים בקובץ חולקו ל-*train Set* ול- *validation Set* ביחס של **80-20**, בהתאם לנהוג בספרות. בכל איטרציה התבצע מעבר על כל הרשומות שב- *train set*, כך שעבור כל רשומה התבצע חיזוי הדירוג לפי הפרמטרים העדכניים, חושבה השגיאה, e_{ui}, ובהתאם אליה עודכנו הפרמטרים לפי נוסחאות ה- *gradient descent*. לאחר סיום האיטרציה חושבה הטעות הממוצעת *RMSE* על ה- *validation set*. אם התוצאה נמוכה מהתוצאה באיטרציה הקודמת, האלגוריתם ממשיך הלאה לאיטרציה הבאה, ואם לא האלגוריתם עוצר.

לאחר מכן ניתן לבדוק את איכות המודל על קובץ הנתונים *ratingsTest* עליו מחושבת הטעות הממוצעת *RMSE* ומדד ה- *precision* שתואר בסעיף 1.2. בקובץ זה קיימים סרטים שאין עליהם כלל נתוני דירוג בקובץ האימון. בעיה זו מוכרת כבעיית *Cold Start*, ויש לה פתרונות רבים ומגוונים, כגון שימוש במודל *content based* למילוי ראשוני של הדירוג. במסגרת סעיף זה בחרנו לחשב את הדירוג הראשוני לפי ממוצע הדירוגים ועוד החותך של המשתמש המדרג: $\mu + b_u$.

מספר ה - latent factors שבחרנו לאחר מספר בדיקות של המודל עם ערכים שונים הוא 30. שאר הפרמטרים הקבועים נבדקו לפי ערכים המומלצים בספר הקורס ובמקומות אחרים בספרות אך לא נבדקו בצורה מקיפה ואיטרטיבית אלא ידנית, בשל זמן הריצה הגבוה של אימון המודל. הערכים שנתנו את התוצאות הטובות ביותר הם אלו המומלצים בספר הקורס: $\lambda = 0.02$, $\gamma = 0.005$.

<u>תוצאות המודל על סט הוולידציה:</u> **RMSE = 0.781**

תוצאות המודל על סט הטסט: **RMSE = 0.801 Precision=0.77**

4. מודל משופר SVD++

בסעיף זה התבקשנו לממש שיפור למודל ה-SVD הבסיסי. בחרנו לממש את מודל ה-SVD++ המתואר בסעיף 3.3.2 בספר הקורס.

מודל זה דומה מאוד ל-SVD, אך הוא מביא לידי ביטוי גם פידבק משתמע (implicit feedback). המודל משתמש במידע על קבוצת הסרטים שכל משתמש דירג בכדי ללמוד על המשתמש, ללא קשר לדירוגים בפועל שהוא נתן לסרטים. הוא עושה זאת באמצעות פקטור נוסף, ווקטור y_i בגודל מספר ה-latent factors, המשויך לסרט i. ניתן לאפיין כל משתמש לפי סט הסרטים שהוא דירג, תוך עדכון הפקטור yi של הסרטים המדורגים על ידו. חיזוי הדירוג עבור משתמש וסרט מסוימים מתבצע לפי הביטוי:

$$\hat{r}_{ui} = \mu + b_i + b_u + q_i^T \left(p_u + |\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} y_j \right)$$

כאשר *R(u)* הינה קבוצת הסרטים שמשתמש u דירג.

לימוד הפרמטרים מתבצע גם כאן באמצעות gradient descent לפי הנוסחאות הבאות:

$$\begin{aligned}
&b_u \leftarrow b_u + \gamma \cdot (e_{ui} - \lambda_5 \cdot b_u) \\
&b_i \leftarrow b_i + \gamma \cdot (e_{ui} - \lambda_5 \cdot b_i) \\
&q_i \leftarrow q_i + \gamma \cdot (e_{ui} \cdot (p_u + |\mathrm{R}(u)|^{-\frac{1}{2}} \textstyle\sum_{j \in \mathrm{R}(u)} y_j) - \lambda_6 \cdot q_i) \\
&p_u \leftarrow p_u + \gamma \cdot (e_{ui} \cdot q_i - \lambda_6 \cdot p_u) \\
&\forall j \in \mathrm{R}(u) : y_j \leftarrow y_j + \gamma \cdot (e_{ui} \cdot |\mathrm{R}(u)|^{-\frac{1}{2}} \cdot q_i - \lambda_6 \cdot y_j)
\end{aligned}$$

הערה: כל העדכונים מבוצעים עבור כל רשומות הדירוג של הצמדים u,i . העדכון האחרון ברשימה זו מתבצע בכל מעבר על רשומה כמספר הסרטים שהמשתמש u דירג, מה שגורם לזמן ריצה ארוך במיוחד.

בכל איטרצייה, האלגוריתם רץ על כל המשתמשים, ועבור כל משתמש מחשב את הסכום $\sum_{j \in \mathrm{R}(u)} y_j$, רץ על כל הסרטים אותם המשתמש דירג, מחשב עבור כל סרט כזה את הדירוג החזוי, את השגיאה מהדירוג האמתי ומעדכן בהתאם את הפרמטרים.

גם כאן מספר ה- latent factors הוא 30, אך ערכי המקדמים שסיפקו את התוצאות הטובות ביותר שונים מאלו שמופיעים בספר הקורס ונקבעו להיות:

$\gamma = 0.005$, $\lambda_5 = \lambda_6 = 0.02$.

תוצאות המודל על סט הוולידציה: **RMSE = 0.779**

תוצאות המודל על סט הטסט: **RMSE = 0.798 Precision=0.771**

5. מודל Content Based

למימוש המודל, ביצענו שימוש הן בקובץ- movies.csv והן בקובץ tagsTrain.csv.
לשם אימון הנתונים, בחרנו ליצור דמיון מבוסס תוכן בין סרטים שונים, על ידי יצירת bag of words המורכב מכלל המילים המופיעות בכלל התיוגים אשר קיבל הסרט מכלל המשתמשים בסט האימון, וכן הז'אנרים המאפיינים את הסרט. כלל המילים נוקו מסימונים שונים, הוסרו stop words וכלל המילים נכתבו באותיות קטנות.
כל מילה בסרט קיבלה משקול tf*idf על ידי שימוש בחבילת skleran.TfIdfVectorizer (נעשה שימוש בחבילה מכיוון שמדובר בחומר מקורס אחזור מידע ולכן לא הרגשנו צורך לממש את המשקולות בעצמנו).
לאחר מכן, יצרנו מילון ובו עבור כל סרט, מוחזקת רשימת 1,000 הסרטים הדומים לו ביותר על ידי חישוב cosine similarity, כאשר כל איבר ברשימה הינו tuple ובו מספר הסרט הדומה, וחישוב הדמיון.
על מנת לחזות את הדירוגים של צמדי המשתמש-סרט בסט המבחן, עברנו על רשימת הצמדים, עבור כל סרט נשלפה מהמילון רשימת הסרטים הדומים לו, והוצלבה עם רשימת הסרטים אותם דירג המשתמש. במידה וההצלבה החזירה רשימה ריקה, החיזוי יהיה הדירוג הממוצע של המשתמש. במידה והרשימה הינה בעלת איבר אחד או יותר, הדירוג החזוי יהיה ממוצע הדירוגים של המשתמש עבור עד 5 הסרטים בעלי הדמיון הגבוה ביותר לסרט שאנו מנסים לחזות את הדירוג עבורו.
לאחר מתן התחזית, מודפסים למסך ערך הRMSE וערך ה- precision עבור סט המבחן.

תוצאות המודל על סט המבחן: **RMSE = 1.005 Precision=0.67**

יש לציין כי התוצאות שהתקבלו אינן טובות באופן מיוחד, וזאת על פי החשד מפני שתיוגי המשתמשים אינם מאפיינים את הסרט באופן אחיד ולכן יוצרים רעש גדול מן הרצוי, ויחס האות לרעש הינו נמוך במקרה זה. נעדיף להשתמש במודלים מבוססים תוכן, על תוכן אובייקטיבי אודות הסרטים.

6. השוואה בין תוצאות המודלים והסבר על ההשוואה

	RMSE	Precision
SVD	0.801	0.77
SVD++	0.798	0.771

Content base	1.005	0.67

טבלה 1 - השוואה בין תוצאות המודלים השונים על ה-test set

<u>הערה:</u> עקב זמני ריצה ארוכים התוצאות לא נבחנו בשיטת k-fold validation כנהוג במצב של השוואה בין מודלים.

לאחר הרצת כלל המודלים, ניכרת עליונותו של מודל הSVD על סט נתונים זה, בדגש על יכולתו לחזות בדיוק רב יותר את הדירוג אותו ייתן המשתמש לסרט מסוים, שזו הינה משימת התחזית העיקרית עבור סט נתונים זה.

עם זאת, מדד ה-precision מציג תוצאות דומות.

אנו מעריכים כי הדבר קורה ממספר סיבות:

- מודל SVD מעדכן משקולות תוך כדי ריצה על סט ולידציה, לומד כל העת ומשתפר בזמן אמת, בעוד מודל מבוסס נתונים מהסוג סוג בחרנו להשתמש הינו יחסית סטטי והשיפור בו יחול משינוי הנתונים המוכנסים אליו.
- מודל SVD מספק חלוקה של המשתמשים ושל הסרטים לקונספטים משותפים בעודו מסתמך על הדירוגים הקיימים בלבד, ולעומתו מודל content based מסתמך על נתונים חיצוניים שמסופקים על הסרטים ולכן תלוי באיכותם.
- הנתונים שקיימים בסט הנתונים הנתון אודות הסרטים השונים אינם מספקים. מדובר בשם הסרט, הז'אנר ותיוגים אשר ניתנו על ידי המשתמשים השונים ואין הכרח שהם תואמים למציאות, כלומר, אינם אובייקטיביים. על מנת לספק מודל מבוסס נתונים שמדייק בצורה טובה יותר, יש להכניס נתונים אובייקטיביים נוספים כגון משתתפי הסרט, הבימאי או הבימאית, ועוד, דברים שלרוב הינם בעלי חשיבות בעת בחירת סרט. כמו כן, מודל מבוסס נתונים הינו מודל המתאים יותר למתן רשימת המלצות מאשר לחיזוי מדויק של דירוג, ואכן נראה כך מהתוצאות.
- משימת החיזוי העיקרית לשמה נועד מודל content based היא יצירת המלצה על "top N" סרטים, לעומת משימת החיזוי שאליה מוכוון מודל SVD שהיא חיזוי דירוג. על כן להערכתנו מודל ה-SVD מציג תוצאות טובות יותר במדד ה-RMSE.

כצפוי מודל ה- SVD++ מציג תוצאות מעט טובות יותר מהמודל הבסיסי במדד RMSE, אך לא באופן משמעותי (שיפור של פחות מ- 0.5%). המסקנה היא שהמשתנים המשתנים המשתמעים לא מספקים מידע רב בנוסף לזה הקיים. ביחס לזמן הריצה הארוך של ה- SVD++ ולשיפור המועט שאותו הוא מספק, בסט נתונים זה אין הצדקה להשתמש במודל זה.

<u>הערה:</u> במסגרת הניסיונות לייצר מודל המבוסס על SVD נעשה ניסיון להשתמש בחתימת הזמן של הדירוגים (time stamp), באמצעות מודל SVD Time-stamp אך מודל זה לא הציג תוצאות טובות יותר מהמודל הבסיסי. לאחר בחינת הפרמטר, התברר כי עבור מרבית המשתמשים, כמעט כל הדירוגים של כל משתמש התרחשו באותו יום, ובנוסף מרבית הדירוגים של כלל המשתמשים התרחשו במספר ימים מצומצם. עובדה זו יכולה להסביר מדוע המודל לא הראה תוצאות משופרות.

7. תוצאות הרצה בחבילת turicreate

להלן קוד הריצה עבור מודל Matrix Factorization:

```
In [1]: import turicreate

/home/dk/anaconda2/lib/python2.7/site-packages/h5py/__init__.py:36: FutureWarning: Conversion of the second argument of i
ssubdtype from `float` to `np.floating` is deprecated. In future, it will be treated as `np.float64 == np.dtype(float).ty
pe`.
  from ._conv import register_converters as _register_converters

In [2]: turi_df_train = turicreate.SFrame.read_csv('/home/dk/ml-20m/ratingsTrain.csv')

Finished parsing file /home/dk/ml-20m/ratingsTrain.csv

Parsing completed. Parsed 100 lines in 4.62043 secs.

------------------------------------------------------
Inferred types from first 100 line(s) of file as
column_type_hints=[int,int,int,float,str]
If parsing fails due to incorrect types, you can correct
the inferred type list above and pass it to read_csv in
the column_type_hints argument
------------------------------------------------------

Read 1012866 lines. Lines per second: 221824

Read 4915869 lines. Lines per second: 487770

Read 8800679 lines. Lines per second: 569196

Read 12610416 lines. Lines per second: 610862

Finished parsing file /home/dk/ml-20m/ratingsTrain.csv

Parsing completed. Parsed 15945812 lines in 24.24 secs.

In [6]: turi_df_test = turicreate.SFrame.read_csv('/home/dk/ml-20m/ratingsTest.csv', column_type_hints=[int, int, int, float, str])

Read 981905 lines. Lines per second: 450528

Finished parsing file /home/dk/ml-20m/ratingsTest.csv

Parsing completed. Parsed 4054451 lines in 4.49782 secs.
```

```
In [7]: model = turicreate.factorization_recommender.create(turi_df_train, target = 'rating', user_id='userId', item_id='movieId')

| 46      | 16m 43s      | 0.289201          | 0.533753              | 0.0488281   |
| 47      | 17m 4s       | 0.286072          | 0.530822              | 0.0488281   |
| 48      | 17m 21s      | 0.289167          | 0.533701              | 0.0488281   |
| 49      | 17m 45s      | 0.285059          | 0.529815              | 0.0488281   |
| 50      | 18m 3s       | 0.276037          | 0.521206              | 0.0488281   |
+---------+--------------+-------------------+----------------------+-------------+
Optimization Complete: Maximum number of passes through the data reached.
Computing final objective value and training RMSE.
       Final objective value: 0.229757
       Final training RMSE: 0.474737
```

```
In [9]: ratings = model.predict(turi_df_test)

In [11]: import pandas as pd

In [12]: pdf_test = pd.read_csv('/home/dk/ml-20m/ratingsTest.csv')

In [13]: pdf_test['predRating'] = ratings

In [14]: import numpy as np

In [20]: np.sqrt(((pdf_test['predRating']-pdf_test['rating'])**2).mean())
Out[20]: 0.9306259374763113

In [23]: pdf_test.loc[pdf_test['predRating']<4.0, 'predRating']=0
         pdf_test.loc[pdf_test['predRating']>=4.0, 'predRating']=1
         pdf_test.loc[pdf_test['rating']<4.0, 'rating']=0
         pdf_test.loc[pdf_test['rating']>=4.0, 'rating']=1

In [34]: ate_count = pdf_test[pdf_test['rating']==1.0][['userId', 'rating']].groupby("userId").agg({'rating':'count'}).reset_index()

In [35]: valid_users = list(rate_count.loc[rate_count['rating']>4, 'userId'].astype('int64'))

In [36]: prec_df = pdf_test[pdf_test['userId'].isin(valid_users)]

In [38]: sum_pred_true = prec_df['rating']+prec_df['predRating']

In [39]: tp = sum_pred_true[sum_pred_true==2].count()

In [40]: fp = prec_df.loc[(prec_df['predRating']==1) & (prec_df['rating']==0) ,'predRating'].count()

In [41]: precision = tp/float(tp+fp)
         precision
Out[41]: 0.7631731980906682
```

תוצאות עבור סט המבחן: **RMSE: 0.93, precision = 0.76**

להלן קוד הריצה עבור מודל Item similarity:

- יש לציין, כי השתמשנו במודל זה מכיוון שמודל הKNN של חבילת turicreate ביצע ב- 21 שעות ריצה רק 4.5% מהתחזיות ולכן לא היה פיזיבילי להמשיך להריצו. לכן, השתמשנו במודל - item similarity, עם פונקציית דמיון מסוג פירסון.

```
def TuriKnn(self):

    train = tc.SFrame({'user_id': self.train_ratings_mat[:, 0].astype(int), 'item_id': self.train_ratings_mat[:, 1].
                       astype(int), 'rating': self.train_ratings_mat[:, 2]})
    test = tc.SFrame({'user_id': self.test_ratings_mat[:, 0].astype(int), 'item_id': self.test_ratings_mat[:, 1].
                      astype(int), 'rating': self.test_ratings_mat[:, 2]})
    model = tc.item_similarity_recommender.create(train, target='rating', similarity_type='pearson')
    pred_ratings = model.predict(test)
    n = len(pred_ratings)
    self.test_ratings_mat = np.c_[self.test_ratings_mat, np.zeros(n), np.zeros(n)]
    self.test_ratings_mat[:, 4] = np.array(pred_ratings)
    self.test_ratings_mat[:, 5] = self.test_ratings_mat[:, 2] - self.test_ratings_mat[:, 4]

    rmse = RMSE(self.test_ratings_mat, n)
    prec = precision(self.test_ratings_mat, n)

    print("KNN RMSE: " + str(rmse) + "\n")
    print("KNN precision: " + str(prec) + "\n")
```

```
Finalizing lookup tables.
Generating candidate set for working with new users.
Finished training in 157.769s
KNN RMSE: 0.946

KNN precision: 0.705
```

תוצאות עבור סט המבחן: **RMSE: 0.946, precision = 0.705**

השוואת התוצאות לתוצאות המודלים הקודמים שהוצגו:

	RMSE	Precision
SVD	0.801	0.77
SVD++	0.798	0.771
Content base	1.005	0.67
Turi – SVD	0.93	0.76
Turi - Similarity	0.946	0.705

טבלה 2 - השוואה לתוצאות בחבילה turicreate

ניתן לראות כי מודל ה-SVD שמומש בעבודה זו הציג תוצאות טובות יותר מבחינת מדד RMSE אך תוצאות דומות במדד precision.

8. בונוס: יצירת מודל משולב

לצורך יצירת המודל המשותף, למעשה ביצענו אימון בנפרד של SVD ושל מודל content based. לאחר מכן, ביצענו תחזיות עבור כל אחד מהמקרים, כאשר אם נדרש חיזוי עבור סרט שאינו קיים בסט האימון, ינתן חיזוי מסוג content based.

הדירוג הסופי משוקלל באופן הבא:

- באם המשתמש לו אנו מנסים לחזות דירוג, דירג פחות מ20 סרטים בסט האימון: $predicted\ rating = 0.6 * SVD_{rating} + 0.4 * Content\_rating$
- אחרת: $predicted\ rating = 0.9 * SVD_{rating} + 0.1 * Content\_rating$

החלטות אלה קיבלנו משום שמודל הSVD הניב תוצאות טובות יותר ורצינו לשמרו, ועם זאת, להרגשתנו, קשה לבסס פרופיל דירוגים על סמך מעט דירוגים של משתמש, ולכן ישנה סבירות שדווקא מודל מבוסס תוכן יניב תוצאות טובות יותר עבור מקרי "cold start".

תוצאות על סט המבחן: RMSE: 0.9, precision – 0.73

9. תיאור הקוד:

- הקבצים נטענים בהפעלת הפונקציה Load אשר טוענת את הקבצים ומחזירה שתי מטריצות, של אימון ושל טסט, כאשר העמודות שלה הן userId, movieId, ratings.

- המודלים בנויים במחלקה RecommenderSystem. ביצירת מופע של המחלקה השיטה prepareData מופעלת. השיטה יוצרת מילון אינדקסים ויוצרת מתוך נתוני האימון train set ו- validation set, תוך הקפדה על כך שכל משתמש יהיה בשני הסטים.
- השיטה trainBaseModel מאמנת מודל SVD בסיסי. הפרמטרים הנלמדים מאותחלים רנדומלית למספרים קטנים חיוביים ושליליים. הפרמטרים מסודרים במטריצות:

  p(numOfUsers **X** k_latents) **q**(numOfMovies **X** k_latents)

  bu(numOfUsers) **bi**(numOfMovies)

  בכל איטרציה מתבצע מעבר על כל רשומות ה-trains set, הדירוג החזוי והשגיאה מחושבים ונשמרים בעמודות נוספות במטריצת ה- train set (החיזוי מתבצע באמצעות השיטה predictRating) ולאחר מכן הפרמטרים מעודכנים לפי הנוסחאות המתוארות בסעיף 2. בסוף האיטרציה הטעות הממוצעת RMSE מחושבת בפונקציה RMSE, על ה-validation set. האיטרציות נעצרות כאשר RMSE לא קטן יותר.
- השיטה trainImprovedModel מאמנת באופן דומה את המודל SVD++ אך יחד עם הפקטור הנוסף yi. הפרמטרים הנוספים לאלו שבמודל SVD ושמאותחלים גם הם רנדומלית הם:

  y(numOfMovies **X** k_latents) **sum_vecs**(numOfUsers **X** k_latents)

  sqrGroupSize(numOfUsers)

  בשורה u במטריצה sum_vecs שמור ווקטור סכומי הפרמטרים y_j כך ש-j הם הסרטים אותם משתמש u דירג. $\sum_{j \in R(u)} y_j$

  בווקטור sqrGroupSize שמור בתא u הערך $|R(u)|^{-0.5}$

  האלגוריתם דומה לזה המתואר ב-SVD ומפורט בסעיף 3.
- השיטות testBaseModel ו- testImprovedModel מחשבות את מדדי ה- RMSE וה- precision על מטריצת הטסט ratingsTest, כפי שמתואר בסעיפים 2 ו-3.
- השיטה predictRating מחשבת את הדירוג החזוי של צמד (משתמש, סרט) בהתאם לסוג המודל האחרון שאומן.
- הפונקציה TrainContentModel מקבלת את מטריצת הסרטים וכן את מטריצת המשקולות tf*idf, ומחשבת את הדמיון לפי cosine similarity בין כל זוג סרטים, כאשר עבור כל סרט נשמרים רק 1,000 הסרטים הדומים לו ביותר. הסרטים נשמרים במילון לפי מספר הסרט כמפתח, והמילון מוחזר למשתמש

- המתודה PredictRatingContent מקבלת את סט האימון וסט המבחן של הדירוגים וכן את מילון הדמיון בין הסרטים. המתודה עוברת על כל צמד של משתמש-סרט וחוזה דירוג עבורו, על בסיס התוכן שאומן במסגרת אימון המודל. המתודה מחזירה רשימה של tuples כאשר כל tuple מורכב ממספר המשתמש, מספר הסרט והדירוג החזוי.
- המתודה create_tfidf_movie_matrix מקבלת את הנתיב לקובץ ml-20m.zip, ובונה את ה- bag of words. לאחר מכן הפונקציה מחשבת את ערך הtf*idf עבור כל מילה בכל סרט, ומחזירה מטריצה זו למשתמש.
- המתודה trainCombinedModel למעשה מאמנת הן את מודל הSVD והן את מודל הcontent, ומחזירה את מטריצת הtfidf, מטריצת הדמיון ואת מטריצת הסרטים.

10. הסבר להרצת הקוד:

הקוד כתוב בסביבת עבודה python 3 ומשתמש בחבילות הבסיסיות pandas, zipfile, sklearn ו- numpy.

- בהרצת המודל, המשתמש מתבקש להזין את הנתיב המלא של קובץ ה-zip בשם ml-20m.zip כולל שם הקובץ.
- בתפריט המופיע לאחר מכן, יש לבחור את סוג המודל אותו רוצים לאמן. ניתן לאמן רק מודל אחד בכל הרצה (עקב בעיות זיכרון).
- לאחר סיום אימון המודל יופיע ערך ה- RMSE הסופי על סט הוולידציה (בעת בחירה בSVD או SVD++).
- מיד לאחר האימון יופיע תפריט נוסף בו ניתן לבחור לבחון את האופן בו תינתן התחזית על בסיס המודל שאומן (SVD, SVD++, Content based או מודל משולב). ניתן לבחור באפשרות של קבלת דירוג בודד על ידי הכנסת מספר משתמש ומספר סרט, או לחילופין בחירה באפשרות של חיזוי על סט מבחן, כאשר הפלט יהיה ערכי ה- RMSE וה-precision שהתקבלו לאחר החיזוי.
- ליציאה מהקוד יש להקיש 99 כאשר מופיע התפריט בשנית.