**דו"ח פרויקט** – quick SVD, קירוב SVD שמתכנס יותר מהר בסגנון Conquer & Divide

תקציר - תיאור הבעיה:  
אלגוריתם SVD מיוחס למשפחת האלגוריתמים מסוג gradient descent , שכידוע מועד למינימום לוקלי. נהוג לאתחל את ערכי הווקטורים בערכים רנדומליים. אנו מעוניינים לבדוק האם:  
1. ניתן למצוא שיערוך טוב בזמן מהיר לערכי הווקטורים וסטיות התקן, במקום להריץ SVD מלא.  
2. האם השערוך מהווה אתחול טוב יותר מרנדומלי וישפר את קצב ההתכנסות של אלגוריתם ה- SVD המלא.  
נשווה בין שלושת השיטות האפשריות (1,2 והSVD- המלא) ונבין את הפשרות שכל שיטה כרוכה בהן, על מנת שבהינתן פלטפורמה מסוימת לה אנו רוצים להתאים מערכת המלצה, נתאים את שיטת ההמלצה המתאימה ביותר (מבחינת דיוק וזמני ריצה). לדוגמא- באפליקציות בעלות בסיס נתונים גדול מאוד בהן זמן בניית המודל יכול להימשך זמן רב, אולי נעדיף לשערך את ערכי המודל מבלי להריץ SVD מלא שיקר בזמן, גם אם זה פוגע בדיוק.

תיאור האלגוריתם:

הכנה לאלגוריתם:

א. קובעים גודל מינימלי של דירוגים כקלט (אחוז מגודל המאגר), עליו נרצה להריץ SVD רגיל בסוף רקורסיה.

ב. מחלקים את הדירוגים לרשימה שלn מאגרים בהתאם לגודל המינימלי של המאגר.

(n צריך לצאת מספר זוגי, מוצגת הודעה במקרה וזה לא כך).

קלט:   
- רשימת המאגרים

א. במקרה והרשימה בגודל 1 מריצים SVD רגיל כל המאגר שברשימה ומחזרים את מודל הSVD שנבנה.  
(נחזיר את ממוצע הווקטורים וה- bias שחזרו עבור כל משתמש/פריט מכל אחד מהחצאים).  
ב. אחרת: מחלקים את הרשימה ל2, על 2 החלקים מריצים רקורסיבית את האלגוריתם.  
ג. נאחד את המודלים שחזרו משני התהליכים, (במקרה שפריט/משתמש נמצא רק באחד משני המאגרים, ניקח את הווקטור היחיד שהתקבל עבורו).

ד. אם הגענו לשורש הרקורסיה:  
נריץ SVD מלא (עד התכנסות) מאותחל עם הווקטורים/bias שהתקבלו.

תיאור הניסוי:

- השערת העבודה:

א. שלמות: בסוף האלגוריתם לכל משתמש ולכל פריט יהיה וקטור ו bias כמו ב SVD הרגיל.

ב. נכונות (קרבה לתוצאות ה SVD המקורי): כל latent feature יחושב עבור כמה מאגרים קטנים ובסופו של התהליך יבחר הממוצע לכן בסוף התרומה של כל latent feature תהיה קרובה לערך מסוים אם הוא הופיע ביותר מכמה מאגרים ולכן סטטיסטית ברוב המקרים יתרום באותה מידה כאילו הורץ ה-SVD המקורי.

ג. זמן ריצה: מכיוון שמריצים את אלגוריתם ה-SVD  על מאגרים קטנים יותר, האלגוריתם יתכנס מהר יותר עבור חלקיו. כמו כן, חישוב SVD על תתי מאגרים זרים מאפשרת ריצה מבוזרת של אלגוריתם זה.

- בסיסי נתונים:  
1) מאגר הנתונים: yelp\_training\_set\_review שניתן לנו בעבודות ההגשה.

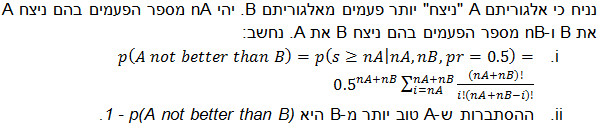
2) Movie Lens 1 - Million Records.

- מדדים לביצועי האלגוריתם:  
א. קירבה המודל המקורב לSVD המקורי: קירבה הוגדרה כסכום שני הערכים הנ"ל:

* סכום המרחק האוקלידי בין הווקטורים.
* סכום המרחק בין ערכי סטיית התקן של המשתמשים והפריטים.

נרצה שהקרבה תשאף ל 0.

ב. דיוק: RMSE.  
ג. ממוצע זמני ריצה של מספר קבוע של 3 ריצות עבור 2 הגדול ושל 10 ריצות עבור מאגר 1.

- מובהקות סטטיסטית: כאשר ניצח יהיה:   
א. זמן ריצה קצר יותר  
ב. כמות הפעמים שהאלגוריתם חזה דירוג יותר נכון מאשר SVD מלא.  
  
****

כאמור, המובהקות הסטטיסטית הינה https://lh5.googleusercontent.com/EZK1bROyE0conqvEpXlGUexSEplA2sF6Xp3ivhNHEHXT6i2qX1mwRtSMX4KkLyi4gAeYypjfSC_DrVkNSnWdVgQ8DydXfPuNS4YrDBAcu_WQ9bru5muHsi81jHLs-qD0DZMLnQM

תוצאות הניסוי:

במסמך המצורף, "Data.pdf" ניתן לראות את הנתונים שחישבנו.

הגרפים לקוחים מתוצאות אלה.

נתייחס תחילה להשערות העבודה שלנו:

א. שלמות: בסוף האלגוריתם לכל משתמש ולכל פריט יהיה וקטור ו bias כמו ב SVD הרגיל.

תוצאה:   
זוהי למעשה אינה השערה, אלא תכונה של האלגוריתם, בהינתן פריט/משתמש שקיים במאגר הגדול, כלומר יש דירוג שכולל את פריט/משתמש זה, אזי דירוג זה ינותב לאחד מהמאגרים הקטנים, שם ייווצר בכלל וקטור לפריט וגם למשתמש של דירוג זה ובפרט לפריט/משתמש הנתון, ע"י אלגוריתם ה SVD.  
לאחר מכן בשלב במיזוג וקטור זה יעבור (יכול להיות שישתנה, אך לא ימחק) במעלה הרקורסיה עד לשורש בו מודל ה quick SVD יכיל אותו.

ב. נכונות: כל latent feature יחושב עבור כמה מאגרים קטנים ובסופו של התהליך יבחר הממוצע לכן בסוף התרומה של כל latent feature תהיה קרובה לערך מסוים אם הוא הופיע ביותר מכמה מאגרים ולכן סטטיסטית ברוב המקרים יתרום באותה מידה כאילו הורץ ה-SVD המקורי.

תוצאה:   
מהנתונים שלנו, ניתן לראות באופן ברור שככל שמספר המאגרים עולה (מחלקים יותר את המאגר הראשי) המרחק של מודל ה quick SVD גדל מהרגיל אך מתמתן עם הזמן.

לשם הבנת ההשפעה של מרחק, אם נתייחס ל bias גם כ latent features, ניתן לחשב את בעזרת הנוסחה

לדוגמה:  
על המאגר הגדול כאשר חילקנו ל 2770 מאגרים קטנים (בו היו תוצאות ה RMSE הכי נמוכות) שזה 0.00036 מהמאגר הגדול המרחק היה 6675, ישנם 6000 משתמשים ו4000 פריטים

לכל משתמש/פריט 10 features ו bias, מכאן, הטעות לכל feature היא בממוצע

נתון יפה שניתן לראות מהגרף למטה, בו מוצג הפרש ה של המודלים מול הגדרת המרחק שלנו, הוא שיש התאמה של מה שנראה כקו ישר בין 2 הפרמטרים.

ג. זמן ריצה: מכיוון שמריצים את אלגוריתם ה-SVD  על מאגרים קטנים יותר, האלגוריתם יתכנס מהר יותר עבור חלקיו. כמו כן, חישוב SVD על תתי מאגרים זרים מאפשרת ריצה מבוזרת של אלגוריתם זה.

תוצאות:

כפי שניתן לראות מהתוצאות, quick SVD ללא השלב הסופי (אימון ל SVD מלא) אכן רץ יותר מהר בבטחון של כ 96.8%, או ליתר דיוק עבור חלוקה למספיק תתי מאגרים תמיד ירוץ יותר מהר.

ככל שמעלים את מספר המאגרים האלגוריתם ללא החלק הסופי מסיים יותר מהר אך עם האימון הסופי עדיין יותר איטי מה SVD הרגיל...

תובנות:

1. האלגוריתם במצבו הנוכחי לא משפר את זמן הריצה עבור האלגוריתם SVD המלא, אך נותן קירוב בזמן יותר מהיר מאשר הרצת האלגוריתם המלא, אולם, יכול להיות שאת קירוב זה ניתן להשיג ע"י הרצת אלגוריתם SVD לזמן קצוב (למספר איטרציות קצוב בזמן) ובכך להשיג תוצאות דומות.
2. קיימות כמה בעיות באלגוריתם במצבו הנוכחי שמאטות את התכנסות ה quick SVD שניתן לתקן.  
   1. בגלל הרקורסיה קיים ריבוי thread-ים שרוב הזמן ממתינים לסיום ילדיהם בשביל לבצע את פעולת המיזוג.  
   ריבוי זה מיותר ומכביד על המעבד (שמוגבל במספר הליבות) וניתן לבטלו.  
     
   2. אם נתייחס לדרגת הרקורסיה כמספר מ 1 עד n וכל קדקוד זה thread של האלגוריתם שרץ, אזי,   
    בדרגה k<n-1 ניתן לבצע פעולות מיזוג לכל זוג קדקודים מדרגה k+1, ולכן רקורסיה אינו אלגוריתם יעיל מכיוון שכאן עבור אותו k יש רק 2 קדקודים אליהם נוכל לגשת.
3. בעיה במימוש הקיים שניתן לשפרה היא שמודל SVD שומר את ווקטורי המשתמשים/פריטים וסטיית התקן שלהן וגם את מאגר הנתונים שנועד בשביל "לאמן" את הווקטורים.  
   מאגר זה גם עובר פעולת המיזוג, דבר שמיותר באלגוריתם עצמו אך נועד כדי שנוכל להשוות את ביצועי האלגוריתמים.  
   למעשה: הצורך שלנו לשמור מידע בשביל לבדוק את ביצועי האלגוריתם הוסיפה זמן לריצתו כך שהתוצאות מוטות מבדיקה זאת.

יש צורך בעבודת המשך להבין כיצד לבטל את זמן מיזוג מאגרי הנתונים מהחישוב (כלומר לכלול רק את מיזוג הווקטורים) וליישם את השיפורים שהצענו בסעיף ב' ויכול להיות שזמן הריצה של האלגוריתם quick SVD, עוד יעקוף את ה SVD הרגיל שכן ההבדל ביניהם הוא מזערי.

הוראות הרצת הפרויקט:

ישנם 2 גרסאות להרצת הניסוי, אחת לבסיס נתונים הגדול של movieLens ואחת לבסיס נתונים הרגיל של yelp\_training\_set\_review.

עבור הרצה עם yelp\_training\_set\_review יש:

- לשנות את פרמטר: filename בקובץ Runner.cs להיות המסלול המלא בו קובץ ה json נמצא.

- להריץ.  
  
עבור MovieLens, יש:

- להוריד את המאגר נתונים.

- להריץ עליו את הממיר python המצורף בקוד מקור.

- לשנות את פרמטר: filename בקובץ Runner.cs להיות המסלול המלא בו קובץ ה json נמצא.

- לשנות את פרמטר versionTwo להיות True.

- להריץ.

הערה: הפלט יוצא במקור ל output, להפניית הפלט לקובץ יש להריץ מ cmd ולהוסיף > filename.txt כסיומת.