­מסמך מלווה לפרויקט סיום בקורס כריית נתונים

---------------------------------------------------------------------------------

**חלק ראשון**

סעיף 1 –

ייצור הנתונים שיותאמו לclusters יתבצע באופן הבא:

1) בחירה אקראית של מספר הנקודות שבכל קבוצה כך שסכום הקבוצות יהיה שווה לn

2) K פעמים יתבצעו השלבים הבאים:

1. maxMumberInThisGroup- הגרלת מספר מ 5000 עד 10000 (שהוא ייצג את המספר המקסימלי שיכול להיות בקבוצה הזו) עבור כל מספר שהוגרל בסעיף קודם שהוגרל- יוגרלו dim מספרים מ maxMumberInThisGroupחלקי 2 ועד maxMumberInThisGroup וזה בעצם נקודה אחת מסך הנקודות
   1. סעיף זה יתבצע בפונקציית עזר **שמקבלת** את הK הרלוונטי ויוצאת נקודות ואת
   2. חזרה על סעיף 3 C פעמים

הסבר לסעיפים 2-3: סעיפים אלו התבצעו באופן הזה על מנת לאפשר לנקודות פוטנציאליות להיות באותו אשכול

סעיף 2 –

האלגוריתם שיתבצע הוא אשכול היררכי, שבו כל נקודה מתחילה כאשכול נפרד. במהלך החישוב, נבצע מספר לולאות: בלולאה הפנימית, נחשב את המרחקים בין כל נקודה לכל נקודה אחרת שלא באותו אשכול בהתחלה זה יהיה כולם עם כולם, ולאחר מכן, בשלבים יותר מתקדמים, נחשב ממוצע עבור כל אשכול כדי לייצג אותו כנקודה אחת.

לאחר מכן, נחשב את המרחק האוקלידי בין כל נקודת אשכול לנקודת אשכול אחרת. בסיום כל לולאה, נבחר את המרחק המינימלי בין האשכולות ונוודא שכל הנקודות יתאגדו לפי מיזוג על פי המרחק הקצר ביותר

לגבי ההחלטה על מספר האשכולות אם לא הוזן k מראש, נמשיך במיזוג האשכולות עד שכל הנקודות יהיו באותו אשכול.

הפונקציה h\_clustering תבצע clustering היררכי על נקודות ב-dim ממדים, כאשר הקלט הוא רשימת points שמכילה את הנקודות.

אם k לא הוזן, הפונקציה תמשיך עד שכל הנקודות יתאגדו לאשכול אחד.

dist הוא פונקציית המרחק (אם לא הוזן, נניח שמדובר במרחק אוקלידי), ו-clusts יהיה משתנה הפלט שבו יאוחסנו האשכולות.

סעיף 3 –

במסגרת המימוש של אלגוריתםK-Means , מקבלת את המשתנים הבאים:

dim מספר הממדים של כל נקודה.

מספר הקלאסטרים (או None אם לא מוגדר מראש). K

n מספר הנקודות.

points רשימת הנקודות.

Clusts רשימת הקלאסטרים שיכיל את הרשימה הסופית ובה כל תא מייצג אשכול

הפונקציה פועלת בשני מקרים שונים, בהתאם לערך של k.

כאשר k מוגדר:

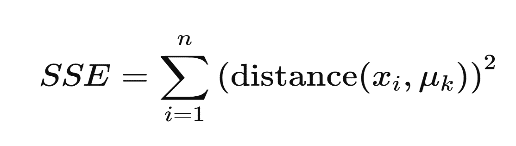
במקרה זה, אנחנו בוחרים באופן אקראי k נקודות מתוך מאגר הנקודות שלנו כמרכזים התחלתיים.

לאחר מכן, עבור כל נקודה, אנו מחשבים את המרחקים שלה לכל אחד מהמרכזים באמצעות מרחק אוקלידי.

הנקודות מסווגות לפי הקרבה למרכזים, ואז כל מרכז משתנה להיות ממוצע הנקודות שהוקצו לו. התהליך הזה חוזר על עצמו עד שהמרכזים מתייצבים.

כאשר k הוא None:

במקרה זה, אנחנו מריצים את אלגוריתם K-Means מספר פעמים עבור ערכים שונים של k למשל, מ-1 עד k\_max

עבור כל ערך של k, אנו בודקים איזה K יש לו שגיאה יותר נמוכה, עבור כל קבוצה, נחשב את המרחקים בין כל נקודה במרכז שלה, ואז נרצה למצוא את הk שמניב את השגיאה הקטנה ביותר – SSE

בסיום, אנו בוחרים את הערך של k שמניב את התוצאה האופטימלית ומבצעים את אלגוריתם K-Means בפעם האחרונה עם הערך האופטימלי של k.  
בחרנו שבדיקת K אופטימלי תהיה בלולאה מk=2 עד עשירית מהK על מנת לצמצם פיזור יתר (למנוע מצבים שבה כל נקודה באשכול נפרד)

עבור סעיף 4 לא נדרשנו להסביר

---------------------------------------------------------------------------------

**חלק שני**

סעיף 1 –

איך הקוד מבצע את המשימה? פונקציית CreateMaxDate:

מגרילים מספר אקראי בין 10 ל-21 נקרא לו X שמציין את מספר הפעמים שבהן תופעל הפונקציה generate\_data.

לאחר מכן, הקוד מריץ לולאה שמבצעת את הפונקציה generate\_data X פעמים.

בתוך כל חזרה בלולאה, currentN (מספר הנקודות) נבחר בצורה אקראית בין 30 ל-1000,currentK שהוא מספר הקבוצות מחושב על פי נוסחה currentN // 10. כלומר 10 אחוז

פונקציית generate\_data:

פונקציה זו אחראית על יצירת נקודות רנדומליות, חלוקתם לקבצות, וכתיבתם לקובץ ה-CSV.

המשתנה arrayOfGroupSizes מחלק את ה-N שזה מספר הנקודות באופן אקראי בין k קבוצות.

לאחר מכן, עבור כל קבוצה, הפונקציה יוצרת קבוצת נקודות בעזרת הפונקציה points\_gen שזו פונקציה חיצונית שנמצאת בתוך part1, ומבצע את יצירת הנקודות בפועל

לבסוף, הקוד כותב את הנקודות שנוצרו לקובץ ה-CSV.

הפונקציה המרכזית שמבצעת את המשימה היאgenerate\_data, שמייצרת את הנקודות ומחלקת אותן לקבוצות. הפונקציה CreateMaxDate אחראית לקרוא לפונקציה זו מספר פעמים אקראי, כך שנוצרים נתונים חדשים כל פעם.

סעיף 2 –

כשבנינו את האלגוריתם, המטרה שלנו הייתה לבצע אשכול תעזור לי בצורה יעילה תוך שמירה על ניהול זיכרון חכם והתאמה טובה של מרחקים בין נקודות. החלטנו להשתמש במרחק מהלנוביס כי הוא מאפשר לנו להתחשב בשונות המשתנים ולמנוע השפעה של הבדלים בקנה המידה של הנתונים.

לגבי אתחול קלאסטרים ראשוניים:

בהתחלה, בחרנו נקודות אקראיות כמרכזי קלאסטרים. החלטנו ללכת על גישה אקראית כי היא נותנת חלוקה ראשונית מגוונת יותר של הנתונים. וגם אין בעצם מצב שבו קלאסטר יהיה ריק מהסיבה שנקודה אחת בוודאות תהיה חלק מקלאסטר מסוים שהוא הקלאסטר שהיא נקודת המרכז שלו.

לגבי ייצוג קלאסטר בצורה חכמה:

כדי לייצג קלאסטר, השתמשנו בווקטור שמכיל מספר נקודות, סכום כל הנקודות, וסכום הריבועים שלהן. זה מאפשר לנו לחשב בקלות את המרכז והסטייה התקנית בהמשך.

לגבי איחוד קלאסטרים בצורה יעילה:

כשהבנו ששני קלאסטרים קרובים מספיק, השתמשנו בפונקציה לאיחוד קלאסטרים. הדרך הזו חוסכת לנו חישובים מיותרים ומשתמשת בנתונים שכבר חישבנו.

לגבי חישוב מרכז וסטיית תקן במהירות:

במקום לחשב כל פעם מחדש, אנחנו משתמשים בפונקציה לחישוב מרכז וסטיית תקן כדי לעדכן את המרכז והסטייה התקנית על בסיס הנתונים שנאספו עד כה.

לגבי מיזוג קלאסטרים לפי מרחק:

לפני שאנחנו מאחדים קלאסטרים, אנחנו בודקים את המרחק ביניהם. אם המרחק קטן מהסף שהגדרנו, אנחנו מחברים אותם כדי למנוע ריבוי קלאסטרים קטנים מדי.

לגבי הפונקציה הראשית - ניהול כל התהליך:

בפונקציה המרכזית הרכבנו את כל השלבים יחד:

1. טעינת נתונים בבלוקים - כדי לא להעמיס על הזיכרון, אנחנו טוענים את הנתונים במקטעים.
2. חישוב קלאסטרים ראשוניים - אם אין מספיק קלאסטרים מוכנים מראש, אנחנו יוצרים אותם מהנקודות הראשונות.
3. שיוך נקודות לקלאסטרים - כל נקודה מקבלת מרחק לכל קלאסטר ומצטרפת לזה שהכי קרוב אליה.
4. מיזוג קלאסטרים קרובים - אם קלאסטרים קרובים מדי, אנחנו מחברים אותם לפי תנאי הסף שקבענו.
5. שמירת התוצאה - בסוף התהליך, אנחנו כותבים את התוצאות לקובץ, כולל מזהה קלאסטר לכל נקודה.

סעיף 3 –

הסבר על מימוש אלגוריתם BFR

מטרה

מימוש אלגוריתם BFR (Bradley-Fayyad-Reina) לצורך אשכול כמות גדולה של נקודות ממרחב בעל ממדים רבים. האלגוריתם מיועד לעבודה בזיכרון מוגבל תוך טעינת בלוקים של נתונים.

פרטי המימוש

האלגוריתם ממומש בפונקציה bfr\_cluster, אשר מקבלת את הפרמטרים הבאים:

dim: מספר הממדים של כל נקודה.

k: מספר האשכולות הסופי הרצוי (אם לא מוגדר, מבוצעת בחירה אוטומטית לפי שיטה נלמדת).

n: מספר הנקודות הכולל בקובץ.

block\_size: גודל הבלוקים לטעינה בכל איטרציה.

in\_path: נתיב לקובץ CSV עם כל הנקודות.

out\_path: נתיב קובץ הפלט שבו תישמר כל נקודה עם מזהה האשכול שלה.

תהליך העבודה של האלגוריתם

טעינת הבלוק הראשון:

מתבצע K-Means ראשוני עם k אשכולות על הבלוק הראשון.

נוצרים אשכולות ה-DS (Discard Set) ומיוצגים בצורה דחוסה (N, SUM, SUMSQ).

הנקודות והאשכולות נרשמים בקובץ הפלט עם מזהי אשכולות.

כלומר, כל הבלוק הראשון מתווסף לds

איטרציות טעינה נוספות:

בכל איטרציה נטען בלוק נוסף בגודל block\_size.

כל נקודה נבחנת מול אשכולות ה-DS, ואם היא בטווח סף (Mahalanobis < 2) היא מתווספת לאשכול המתאים.

אם נקודה לא מתאימה ל-DS, נעשה בדיקה נוספת שהיא בדיקה עבור הקלאסטרים שנמצאים כבר בcs

אם גם לשם היא לא נמצאת מתאימה היא מתווספת לrs

בסיום המעבר על הבלוק נעשים הצעדים הבאים

עיבוד ה-RS:

כאשר ה-RS גדל מעל סף 2 \*K מופעל עליו K-Means.

אשכולות עם יותר מנקודה נכנסים ל-CS (Compression Set).

אשכולות ה-CS נבדקים למיזוג לפי מרחק מהלנוביס.

ואז נעשה MERGE בקבוצת CS

מיזוגים סופיים:

לאחר סיום קריאת כל הבלוקים:

מבוצע K-Means נוסף על ה-RS למעבר אשכולות ל-CS.

כל אשכול ב-CS נבדק למיזוג ל-DS לפי מרחק מהלנוביס.

אשכולות CS שלא מוזגו מתווספים כ-DS חדשים.

סיום והפלט:

כל הנקודות עודכנו עם מזהי אשכולות.

כתיבה סופית של כלל הנקודות לקובץ out\_path.

החלטות מימוש חשובות:

הסף למרחק מהלנוביס נקבע כ־2 .

מיזוג אשכולות מתבצע רק כאשר מתקיים מרחק מהלנוביס קטן מהסף.

לשמירה על הזיכרון, RS נשמר עד הגעה לגבול ומעובד באמצעות K-Means.

כל נקודה נרשמת תמיד עם מזהה אשכול עדכני למעקב מלא.

הקוד בנוי כך שלא מאחסן את כל הנקודות ב-DS ו-CS אלא רק את הייצוג הדחוס (כפי שדורש האלגוריתם).

סעיף 4 –

אלגוריתם CURE – שלבים מסודרים וברורים

* שלב 0 – דגימה ו-Clustering ראשוני

דגימת תת-קבוצה מהדאטה הראשי – נבחר תת-קבוצה של נקודות מהדאטה כדי לבצע clustering ראשוני.

ביצוע HAC (Hierarchical Agglomerative Clustering) – נאחד נקודות לקלאסטרים על בסיס קרבה היררכית.

מציאת נקודות מייצגות לכל קלאסטר:

נחשב סנטרואיד (מרכז הכובד) של כל קלאסטר.

נמצא את הנקודות הרחוקות ביותר מהסנטרואיד – הנקודות שמכסות בצורה מקסימלית את הקלאסטר.

מספר הנקודות המייצגות יהיה פרופורציונלי לגודל הקלאסטר (למשל, אם קלאסטר מכיל 100 נקודות, ניקח 10 נקודות רחוקות).

* שלב 1 – הזזת הנקודות המייצגות לכיסוי טוב יותר

חישוב נקודות אמצע – לכל נקודה מייצגת נחשב נקודת אמצע עם הסנטרואיד. פעמיים, כלומר, פעם אחת נחשב אמצע בין הסנטרואיד לבין הנקודות המייצגות ואז שוב פעם בין התוצאה שהייתה מהחישוב הקודם לנקודה המייצגת

ככה בעצם נבצע הזזה נוספת של 25% נוספים לכיוון הסנטרואיד, כך שהנקודה המייצגת תהיה קרובה יותר למרכז הקלאסטר אך עדיין תשמור על גיוון.

* שלב 2 – שיוך נקודות חדשות לקלאסטרים

סריקה מחדש של כל הדאטה

לכל נקודה חדשה בדאטה נחפש את הקלאסטר הקרוב ביותר:

המרחק יימדד לפי הנקודות המייצגות של כל קלאסטר

לכל נקודה נחשב את המרחק לכל אחת מהנקודות המייצגות בכל קלאסטר.

הקלאסטר עם המרחק המינימלי לנקודה ייבחר להיות הקלאסטר של הנקודה.  
בסוף השלב הזה, כל הנקודות בקובץ הנתונים שובצו לקלאסטרים!

---------------------------------------------------------------------------------

חלק 3 –

יצירת הקבצים נעשתה בעזרת פונקציית generate\_data מהחלק הראשון שמבצע תהליך של יצירת נתונים הכולל את השלבים הבאים:

1. חלוקת הנתונים באופן אקראי ל-K קבוצות, כך שסכום הקבוצות יהיה שווה ל-n.
2. עבור כל אשכול (סה"כ K פעמים), מבוצעים השלבים הבאים:

* בחירה באופן אקראי מספר מקסימלי של נקודות באותו אשכול (maxMumberInThisGroup) בטווח שבין 5000 ל-10000.
* עבור כל מספר שהוגרל, ייבחרו dim (ממד) מספרים בטווח שבין חצי מהמספר המקסימלי לבין המספר המקסימלי, כך שיתקבלו נקודות אקראיות שמתאימות לאותו האשכול.

המטרה היא ליצור את הנתונים באופן שמבטיח חלוקה אקראית אך עדיין שומרת על האשכולות, כך שהנקודות בתוכם לא יהיו מופרדות באופן טריוויאלי, אלא יתחלקו בצורה טבעית בתוך החלוקה שנבחרה.