דו"ח מיני פרוייקט: נושאים באנליזה של מידע

סטודנטים:
אלון מזרחי
רועי בן דב

מרצה:
ד"ר סיון סבתו

פברואר 27, 2020

תוכן עניינים

<u>סעיף 1</u>

קבוצת התנאים של גרסה 2:
קבוצה שמפרידה לפי *threshold* של פיקסל בודד, לפי מלבנים מגודל $2x6$, ולפי מספר פיקסלים כהים בשורות ועמודות. פירוט על כל סוג תנאי יופיע בהמשך.

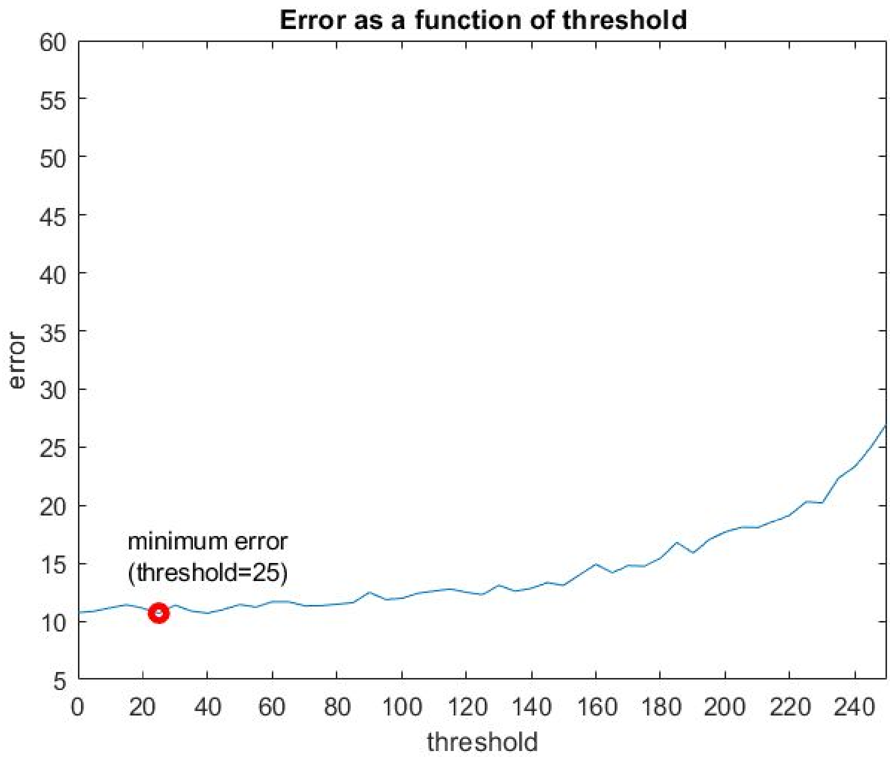
ראשית נציין כמה מסקנות מתהליך בחירת קבוצת התנאים החדשה:

1. כדי לבחור את התנאים הכי טובים, החלטנו לייצר סביבת הרצה קבועה על מנת שלא תיהיה תלות בין סביבת ההרצה לקבוצות התנאים, ולכן את הבדיקות עבור קבוצות התנאים השונות ערכנו עם פרמטר הקלט $L = 11$, ועבור גודל מדגם ולידיציה של 15% ממדגם האימון.
2. לחלק מקבוצות התנאים קיימים פרמטרים שנתונים לבחירתנו. למשל, עבור השאלה "האם הפיקסל במקום x, y כהה?" הפרמטר להגדרת המונח "כהה" הינו בר שינוי: ערך הפיקסל גדול מ- 10, 70, 128 וכו'. אם כך, לפני הוספת קבוצת תנאים מסויימת, נלמד את הפרמטר הטוב ביותר באמצעות שימוש במדגמי האימון והבדיקה.
3. יכול להיות שאחוז ההצלחה עבור קבוצת התנאים ישתנה כתלות בעוד קבוצת תנאים, כלומר, יכול להיות שקבוצת התנאים A לבדה היא בעלת אחוז הצלחה נמוך, וקבוצות התנאים B, C בעלות אחוזי הצלחה גבוהים, ואילו קבוצת התנאים $A \cup B$ מביאה אחוז הצלחה גבוה יותר ביחס ל- $B \cup C$.

   לכן עבור כל קבוצת תנאים, נציין תחילה את אחוז השגיאה כאשר היא קבוצת התנאים היחידה לפיה העץ מפצל את עליו, ולאחר מכן נבדוק אותה בתור קומבינציה שלה ושל הקבוצות שאגדנו עד כה.

החלטנו ללמוד את הפרמטר הכי טוב עבור קבוצת התנאים של גרסה 1 , כלומר את המספר (ה-threshold) שעבורו הפיקסל בקואורדינטה x, y שבדוגמה נחשב "כהה". בגרסה הראשונה ערך זה היה שווה 128 , כלומר שאלנו עבור כל פיקסל האם ערכו גדול מ- 128 . על מנת ללמוד את הפרמטר הרצנו באופן בלתי תלוי thresholds שונים (כאשר מדגמי האימון והוולידציה מוגרלים כל פעם באופן אחיד).

להלן גרף המתאר את השגיאה כפונקציה של ה-threshold, החל מ- $threshold = 0$ עד $threshold = 250$, בקפיצות של 5 .



קיבלנו כי עבור ערך threshold של 25 מתקבלת השגיאה הנמוכה ביותר לקבוצת תנאים זאתי לבדה - 10.73% .

נסמן: $CthresholdN = all\ x, y\ with\ threshold\ of\ N$

לדוגמה:

$Cthreshold128 = all\ x, y\ with\ threshold\ of\ 128$,

$Cthreshold25 = all\ x, y\ with\ threshold\ of\ 25$.

כעת, החלטנו לבדוק האם כדאי להוסיף את $Cthreshold25$ ביחד עם $Cthreshold128$ או להחליף ביניהן.

- אחוז שגיאה $Cthreshold25$: 10.71% .
- אחוז שגיאה $Cthreshold128$: 12.64% .
- אחוז שגיאה $Cthreshold25 \cup Cthreshold128$: 11.23% .

קיבלנו כי אחוז השגיאה עבור $Cthreshold25$ קטן מאחוז השגיאה עבור $Cthreshold25 \cup Cthreshold128$, ולכן עדיף להחליף את $Cthreshold128$ מהגרסה הראשונה ב-$Cthreshold25$.

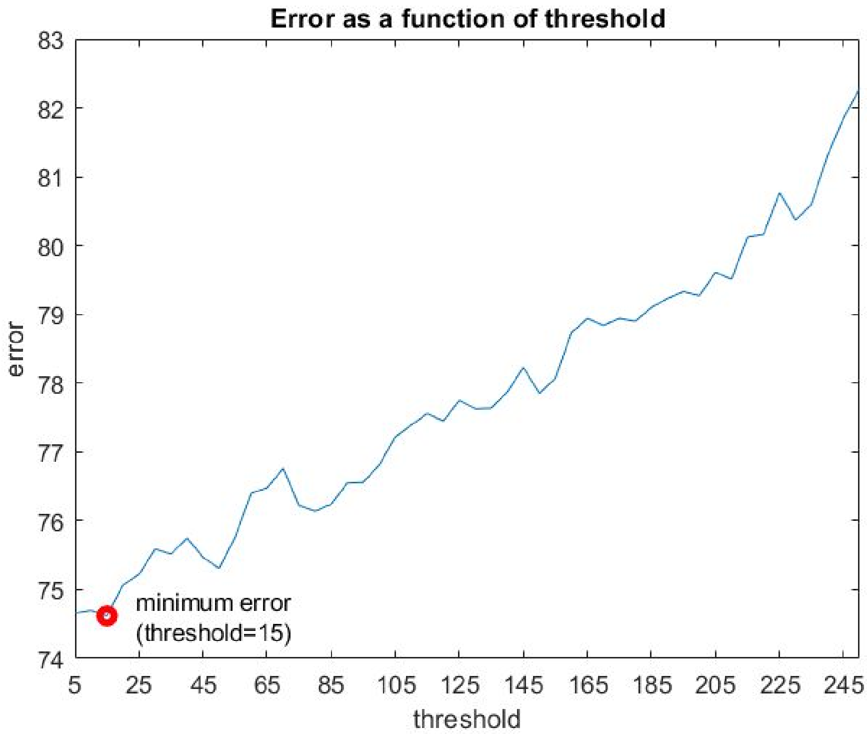
קבוצת התנאים הבאה שהחלטנו לחקור הינה אוסף השאלות: "האם מספר הפיקסלים הכהים בדוגמה גדול מ- k , כאשר $k = 1, ..., 28^2$?".

על מנת שנקבל זמן ריצה סביר, ומתוך מחשבה שאין יתרון משמעותי לקפיצות קטנות של k , החלטנו להריץ את k בקפיצות של 10 מהערכים 10 עד 28^2 .

נסמן את קבוצת התנאים האם מספר הפיקסלים "הכהים" בתמונה גדול מ- k ב- $Cdark$.

נשים לב כי גם פה אנו יכולים לבחור את הפרמטר $threshold$.

להלן גרף המתאר את השגיאה כפונקציה של ה-threshold, החל מ- $threshold = 5$ עד $threshold = 250$, בקפיצות של 5 .



ראשית נבחין כי אחוזי השגיאה גבוהים מאוד ביחס לקבוצות הקודמות, ומכך נסיק כי קבוצת תנאים זו, כשלעצמה, אינה טובה לנו. למרות זאת, נמשיך ונבדוק את אחוזי השגיאה שלה ביחד עם הקבוצות הקודמות. בנוסף, מנתוני הגרף, נבחר להגדיר פיקסל כהה עבור קבוצה זאת להיות פיקסל בעל ערך גדול מ- 15 . נשים לב שערך זה שונה מהערך עבור קבוצת התנאים $Cthreshold$.

- אחוז שגיאה $Cdark$: 74.62% .
- אחוז שגיאה $Ctreshold25 \cup Cdark$: 11.24% .
- אחוז שגיאה $Ctreshold128 \cup Cdark$: 12.97% .

מתוצאות אלו נסיק כי קבוצת תנאים זו אינה טובה לנו (היא כנראה כללית מדי ואינה מבדילה בין דוגמאות של ספרות בצורה טובה), ונבחר לא להשתמש בה.

קבוצת התנאים הבאה היא אוסף השאלות "האם הריבוע בגודל n המתחיל מפיקסל x, y (פיקסל זה נמצא בפינה השמאלית העליונה של הריבוע) מכיל k פיקסלים כהים?".

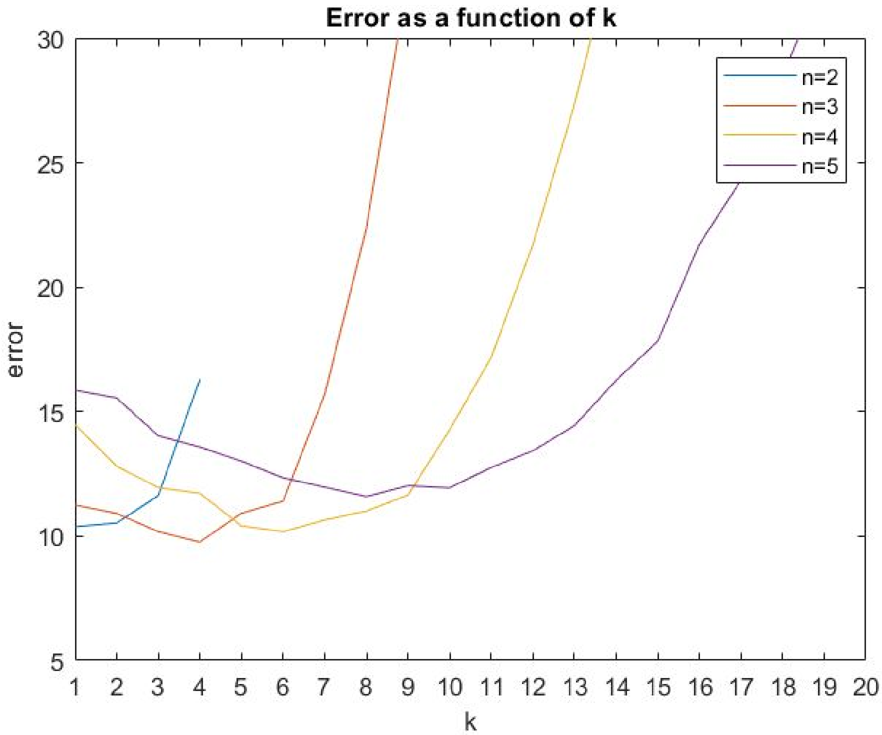
נסמן ב- $CblockN$ את קבוצת התנאים עבור ריבועים מגודל N.

במקרה זה ישנם שלושה פרמטרים אותם אנו יכולים לבחור: $n,\ k,\ threshold$.

על סמך התוצאות הקודמות, נבחר לקבע את הפרמטר $threshold$ לערך 25.

כעת הרצנו בדיקות על מנת למצוא את ה- k האופטימלי לכל $n \in \{2,3,4,5\}$.

להלן גרף של עקומות המתארות את השגיאה כפונקציה של k, כאשר כל עקומה היא עבור ערך n שונה.



מצאנו שעבור $n = 3,\ k = 4$ מתקבלת השגיאה הנמוכה ביותר - 9.77%.

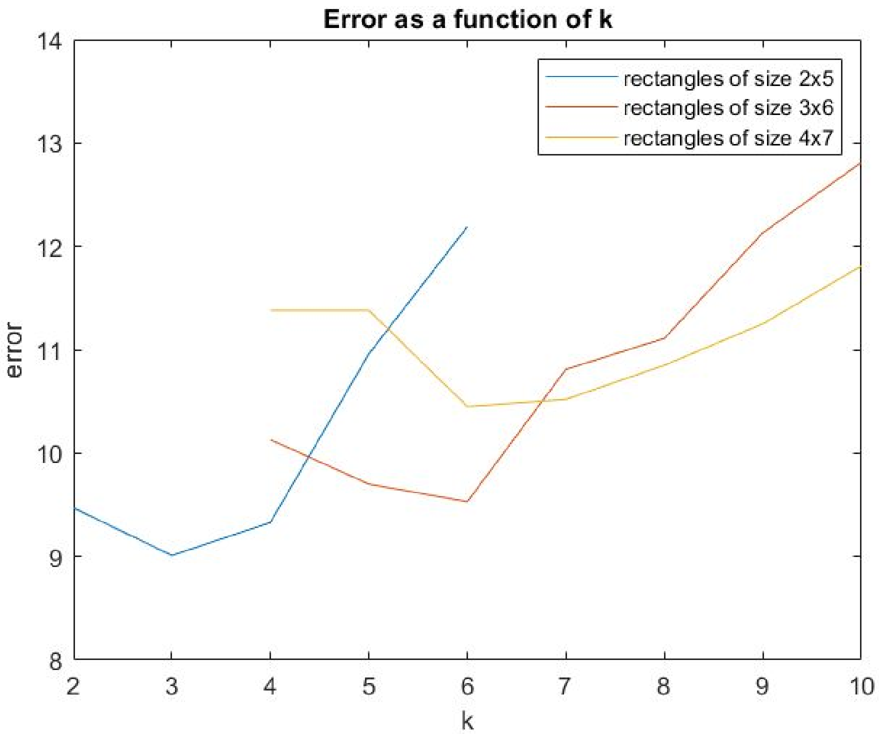
גילינו כי קבוצת התנאים $Cblock3$ הינה בעלת אחוז השגיאה הנמוך ביותר עד כה, וכעת נבדוק את אחוזי השגיאה כאשר מאחדים אותה עם קבוצות תנאים אחרות.

- אחוז שגיאה $Cblock3 \cup Cdark$: 10.08%.
- אחוז שגיאה $Cblock3 \cup Cthreshold25$: 9.88%.
- אחוז שגיאה $Cblock3 \cup Cthreshold25 \cup Cdark$: 9.98%.

אם כך, לעת עתה נשתמש בקבוצת התנאים $Cblock3$, שסיפקה את אחוז השגיאה הנמוך ביותר.

מכיוון ש- $Cblock3$ הינה קבוצת התנאים שהביאה לנו את השגיאה הנמוכה ביותר, חשבנו להכליל את הרעיון של קבוצת תנאים זו לקבוצה של מלבנים. הרעיון נובע מהעובדה שהספרות אינן ריבועים סימטרים, ונוטות יותר לצורות שניתנות להרכבה ממלבנים קטנים. נשיב לב שמלבנים בגדלים $28x1,\ 1x28$ מהווים שורות ועמודות בכל דוגמה, וכך ניתן גם לבצע את אותה שאלה על שורות ועמודות.

עבור קבוצת התנאים $CblockN$ ראינו כי המגמה בעבור ה- k האופטימלי הינה בערך 0.4 מגודל (שטח) הריבוע.
בעבור קבוצת התנאים של המלבנים, בחרנו שלושה מלבנים מהגדלים $2x5,\ 3x6,\ 4x7$ ובדקנו עבורם מיהו ה- k האופטימלי (נזכיר כי k הוא הסף עבור מספר הפיקסלים הכהים במלבן).
להלן התוצאות:



מכך נסיק כי עבור מלבנים "גדולים" ($m \cdot n > 20$) נרצה להגדיר $k = 0.25 \cdot mn$, ועבור מלבנים קטנים ($m \cdot n \leq 20$) נגדיר $k = \frac{1}{3}mn$ (עבור $m \cdot n < 10$ נעגל למעלה).

הרצנו בדיקות על כל המלבנים מהגדלים $m\ x\ n \quad s.t. \ \ m,n \in \{1,...,8\},\ m \neq n$, כאשר כל זוג m,n מכתיב מלבנים בגדלים $m\ x\ n,\ n\ x\ m$.
להלן הממצאים:

m \ n	1	2	3	4	5	6	7	8
1		9.55	10.86	9.92	9.08	12.06	12.36	11.94
2			10.08	9.68	8.91	8.68	8.80	10.83
3				10.09	9.86	9.45	9.55	9.81
4					10.59	10.76	10.52	11.01
5						12.49	12.35	11.98
6							14.07	13.87
7								16.13
8								

ניתן לראות כי המלבנים בגדלים $2x5,\ 2x6,\ 2x7$ הביאו את אחוזי השגיאה הנמוכים ביותר, כאשר אחוז השגיאה של $2x6$ הוא הנמוך ביותר.
נסמן ב- $CrectangleMxN$ את קבוצת התנאים של מלבנים מגודל MxN.

ממצאים העולים מקבוצת תנאים זו:

- אחוז שגיאה $Crectangle2x6$: 8.68%.
- אחוז שגיאה $Crectangle2x5 \cup Crectangle2x6 \cup Crectangle2x7$: 8.64%.
- אחוז שגיאה $Crectangle2x6 \cup Cblock3$: 8.83%.
- אחוז שגיאה $Crectangle2x6 \cup Cblock3 \cup Cthreshold25$: 9.27%.

בנוסף החלטנו לבדוק גם עבור מלבנים מגדלים שונים (גם אורך וגם רוחב), למרות שאינדיבידואלית הם לא הניבו את אחוזי השגיאה הנמוכים ביותר.

- אחוז שגיאה $Crectangle1x5 \cup Crectangle2x6 \cup Crectangle4x7$: 8.58%.

קיבלנו כי אחוז השגיאה הנמוך ביותר מתקבל עבור מלבנים מהגדלים $1x5,\ 2x6,\ 4x7$, אך שגיאה זו אינה רחוקה מהשגיאה שמתקבלת עבור $Crectangle2x6$ בלבד, וזמן הריצה משמעותית ארוך יותר.
אם כך, מבין כל קבוצות התנאים שבדקנו עד כה, נבחר את קבוצת התנאים $Crectangle2x6$ לבדה, שמהווה את האיזון הטוב ביותר בין זמן ריצה לאחוז שגיאה.

כפי שציינו קודם, ניתן גם להגיע לשאלות מהסוג "האם השורה / עמודה x מכילה לפחות k פיקסלים כהים?" באמצעות הצבה של $m = 28,\ n = 1$ ולהפך.

נסמן קבוצה זו ב- $ClinecolumnK$ כאשר K הוא סף הפיקסלים הכהים באותה שורה או עמודה.

לכל K, בקבוצה $ClinecolumnK$ ישנם בסה"כ 56 תנאים (28 לכל שורה ולכל עמודה). מספר תנאים זה הינו קטן יחסית ולכן הרשנו לבצע ריצה עבור איחוד של קבוצות התנאים $ClinecolumnK$ כאשר $1 \leq K \leq 20$, עם 1120 תנאים סה"כ.

נסמן קבוצת תנאים זו ב- $Clinecolumn1to20$.

עבור $Clinecolumn1to20$ קיבלנו אחוז שגיאה של 16.67, אך שמנו לב כי מעל 75% מהשאלות שנבחרו לצמתי העץ היו עבור ערכי K בין 1 ל- 10.

אם כך, כדי לצמצם בזמן ריצה כאשר נמזג בין קבוצה זו לקבוצות אחרות, נאחד אותן עם $Clinecolumn1to10$.

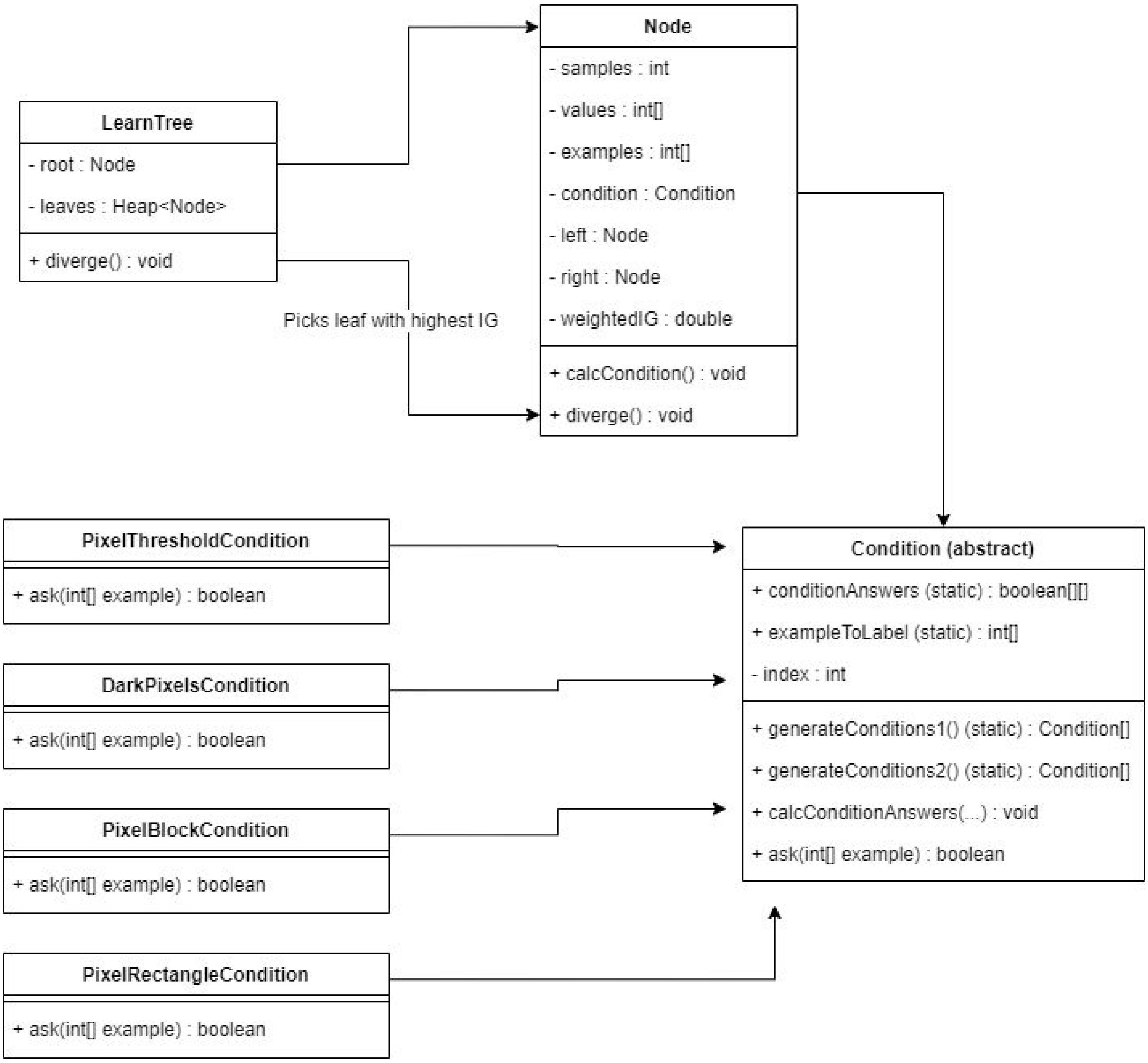
- אחוז שגיאה $Clinecolumn1to20$: 16.66%.
- אחוז שגיאה $Clinecolumn1to10$: 16.98%.
- אחוז שגיאה $Clinecolumn1to10 \cup Crectangle2x6$: 8.02%.
- אחוז שגיאה $Clinecolumn1to10 \cup Crectangle2x6 \cup Cthreshold25$: 7.63%.
- אחוז שגיאה $Clinecolumn1to10 \cup Crectangle2x6 \cup Cblock3$: 8.20%.
- אחוז שגיאה $Clinecolumn1to10 \cup Crectangle2x6 \cup Cblock3 \cup Cthreshold25$: 8.11%.

לסיכום, מבין כל קבוצות התנאים השונות שחקרנו, הסקנו כי קבוצת התנאים הטובה ביותר, שמאזנת הכי טוב בין אחוז שגיאה לזמן ריצה, היא $Clinecolumn1to10 \cup Crectangle2x6 \cup Cthreshold25$, כלומר קבוצה שמפרידה לפי $threshold$ של פיקסל בודד, לפי מלבנים מגודל $2x6$, ולפי מספר פיקסלים כהים בשורות ועמודות.

אחוז שגיאה של קבוצת תנאים זו: 7.63%.

סעיף 2

החלטנו לממש את הפרויקט ב-Java.
אופן מימוש האלגוריתם נשען על שלוש מחלקות מרכזיות: LearnTree, Node, Condition.
להלן תרשים UML של המחלקות עם השדות והשיטות המרכזיים (פרטים טכניים הושמטו לשם פשטות):



הסבר מפורט על כל מחלקה בהמשך.

LearnTree

- המחלקה המרכזית של העץ.
- מכילה שדה מסוג Node ששומר את שורש העץ.
- מכילה ערימת מקסימום (max heap) של עלים שממויינים לפי Information gain.
- השיטה diverge מוציאה מן הערימה את העלה בעל ה-Information gain הגדול ביותר (נמצא בראש הערימה), מפצלת אותו, ומכניסה את שני בניו החדשים לערימה.

Node

- מחלקה שמייצגת צומת (פנימי או עלה) בעץ.
- מכילה את השדות:
  - samples: מספר הדוגמאות שהגיעו לצומת זה.
  - values: מערך המכיל את כמות הדוגמאות שהגיעו עבור כל ספרה.
  - examples: הערך המכיל אינדקסים של הדוגמאות שהגיעו לצומת.
  - condition: התנאי הנבחר, לפיו העלה יפוצל בעתיד כאשר יקראו לשיטה diverge.
  - left / right: הבנים של הצומת. מקבלים ערך null אם הצומת הוא עלה.
  - weightedIG: שדה ששומר את ה-Information gain (המשוקלל לפי מספר הדוגמאות שהגיעו אליו) של העלה, במידה ויפוצל.
- השיטה calcCondition עוברת על כל התנאים הקיימים ומסמלצת פיצול של הצומת (במצב עלה) עם כל תנאי, ושומרת את התנאי (בשדה condition) שהביא את ה-Information gain הגדול ביותר. עם תנאי זה השדה יפוצל בעתיד, כאשר יקראו לשיטה diverge.
- השיטה diverge מפצלת את העלה לפי התנאי שנבחר ב-calcCondition ומסדרת את שדות הבנים כנדרש. השיטה הופכת את הצומת ממצב "צומת פנימי" למצב "עלה".

Condition

- מחלקה אבסטרקטית שמטרתה להיות מורחבת ע"י מחלקות של תנאים אמיתיים, למשל PixelThresholdCondition.
- מכילה שדה סטטי בשם conditionAnswers ששומר את התשובות של כל התנאים על כל הדוגמאות. השדה הוא מסוג מערך דו מימדי של booleans, כלומר לכל דוגמה יש את התשובות של כל התנאים עליה (אם עונים "כן" או "לא" על הדוגמה). שדה זה מיועד לאופטימיזציה (יוסבר בהמשך).
- מכילה שדה סטטי בשם exampleToLabel מסוג מערך של integers, ששומר את התווית של כל דוגמה. שדה זה מיועד לאופטימיזציה (יוסבר בהמשך).
- לכל אובייקט מסוג Condition יש אינדקס. באמצעות האינדקס ניגשים למערך conditionAnswers ביחד עם אינדקס של דוגמה מסויימת עליה אנו רוצים לבדוק את התנאי.
- השיטות הסטטיות generateConditions1, generateConditions2 מחזירות מערך של תנאים לפי כל גירסה (1 או 2), תלוי בקלט התוכנית.
- השיטה calcConditionsAnswers מקבלת כקלט את ה-data set ומחשבת מראש את המערך conditionsAnswers, לשימוש ע"י התוכנית בהמשך.
- השיטה האבסטרקטית ask מקבלת דוגמה (תמונה של ספרה) ומחזירה boolean בהתאם לתשובה של התנאי על הדוגמה. כל מחלקה שתרחיב את Condition צריכה לממש שיטה זו.

- כפי שהוסבר בסעיף 1, ישנן 4 מחלקות מרחיבות את המחלקה Condition:
  - PixelThresholdCondition
  - DarkPixelsCondition
  - PixelBlockCondition
  - PixelRectangleCondition

  כל מחלקה כזאת מממשת לפי הלוגיקה המיוחדת שלה את השיטה ask.

PixelThresholdCondition

- מקבל קואורדינטה של פיקסל x,y וערך $threshold$.
- בשיטה ask, בהינתן דוגמה, בודק האם ערך הפיקסל x,y גדול מהערך $threshold$.
- מבוצע באצעות גישה ישירה למערך הדוגמה.

DarkPixelsCondition

- מקבל מספר num וערך $threshold$.
- בודק האם קיימים בדוגמה לכל הפחות num פיקסלים בעלי ערך גדול מ- $threshold$.
- מבוצע באמצעות לולאה שעוברת על כל הפיקסלים בדוגמה.

PixelBlockCondition

- מקבל קואורדינטה של פיקסל x,y , גודל בלוק n , מספר k , וערך $threshold$.
- בודק האם הריבוע בגודל n המתחיל מהפיקסל x,y מכיל לכל הפחות k פיקסלים בעלי ערך גדול מ- $threshold$.
- מבוצע באמצעות שתי לולאות שמתחילות מהפיקסל x,y וסופרות את מספר הפיקסלים הכהים בריבוע.

PixelRectangleCondition

- ממומש כמו PixelBlockCondition רק שמקבל ערכי $width$, $height$ במקום n .

באמצעות אופן מימוש זה, הבדל מימוש שתי הגרסאות הסתכם רק בהוספת מחלקות חדשות של תנאים המרחיבות את המחלקה Condition למערך התנאים של התוכנית בלבד.

אתגרים:
בחירת מבני הנתונים בהם השתמשנו היוו מרכיב חשוב עבור זמן הריצה של התוכנית.
ראשית השתמשנו בערימת מקסימום שמכילה עלים וממיינת אותם לפי ערך ה-information gain שלהם. כך יכולנו לשלוף את העלה בעל ה-information gain הגדול ביותר אותו אנו רוצים לפצל ב- $O(\log n)$, ולהכניס את שני בניו לאחר הפיצול גם ב- $O(\log n)$ (n הוא מספר העלים בעץ).
ראינו כי אין צורך במיון רגיל של רשימת העלים, אלא רק ידיעה של המקסימלי.
בנוסף, לפני בניית העץ, החלטנו לבצע עיבוד מקדים שכולל חישוב התשובות של כל התנאים על כל הדוגמאות (חישוב זה יקרה בכל מקרה במהלך בניית העץ, בדרך זו ביצענו אותו רק פעם אחת), ושמרנו את התוצאות במערך בוליאני דו-מימדי. מהלך זה איפשר לנו להימנע מחישוב תשובות התנאים על הדוגמאות מחדש כל פעם שרצינו לפצל עלה, וחסך לנו כמות גדולה של פעולות בזמן הריצה. החיסרון היחיד של פעולה זו הוא בסיבוכיות המקום, אך בדקנו וראינו שחיסרון זה לא מהווה בעיה.
בנוסף, בכל עלה שמרנו רק את הדוגמאות (לפי אינדקסים) שהגיעו אליו, כך שבעת פיצול נצטרך לעבור רק עליהן ולא על המאגר המלא של הדוגמאות.

עוד כללי אצבע טכניים שהנחו אותנו במהלך המימוש והאיצו את התוכנית:

- שימוש במערכים פרימיטיביים ($int[]$) במקום מערכים דינמיים (ArrayList). פעולה זו חוסכת מספר קטן של פעולות בכל גישה למערך. פעולות אלו עלולות להצטבר ולהאט את התוכנית.
- הימנעות מגישה ישירה ל-data set במהלך פיצול העץ.
גודל ה-data set הוא עצום (מערך בגודל $60000 \cdot 28^2$) ולכן הגישה אליו (אפילו לאלמנט הראשון שמייצג את התווית של כל דוגמה) מאטה מאוד את התוכנית, בגלל איך שהמחשב עובד. מסיבה זו יצרנו את המערך exampleToLabel ששומר את התוויות של כל הדוגמאות ב-data set. בנוסף גם איננו משתמשים כלל ב-data set במהלך בניית העץ מכיוון שכבר חישבנו את תשובות התנאים על כל דוגמה מבעוד מועד.
- הימנעות מכתיבה מוגזמת לפי עקרונות OOP.
אמנם כתיבה בסגנון OOP מקלה עלינו, אך כאשר זמן הריצה חשוב לנו עדיף להימנע ממנה, או לפחות משימוש גדול בה. מסיבה זו בין היתר המחלקה Node ייצגה גם צמתים פנימיים וגם עלים, ולא יצרנו מחלקה שונה לכל סוג. נמנענו כשיכולנו גם מ-Polymorphism כיוון שימוש בו לעיתים מאט את התוכנית (גישה ל-virtual table בזמן ריצה וכו').

תוספות:

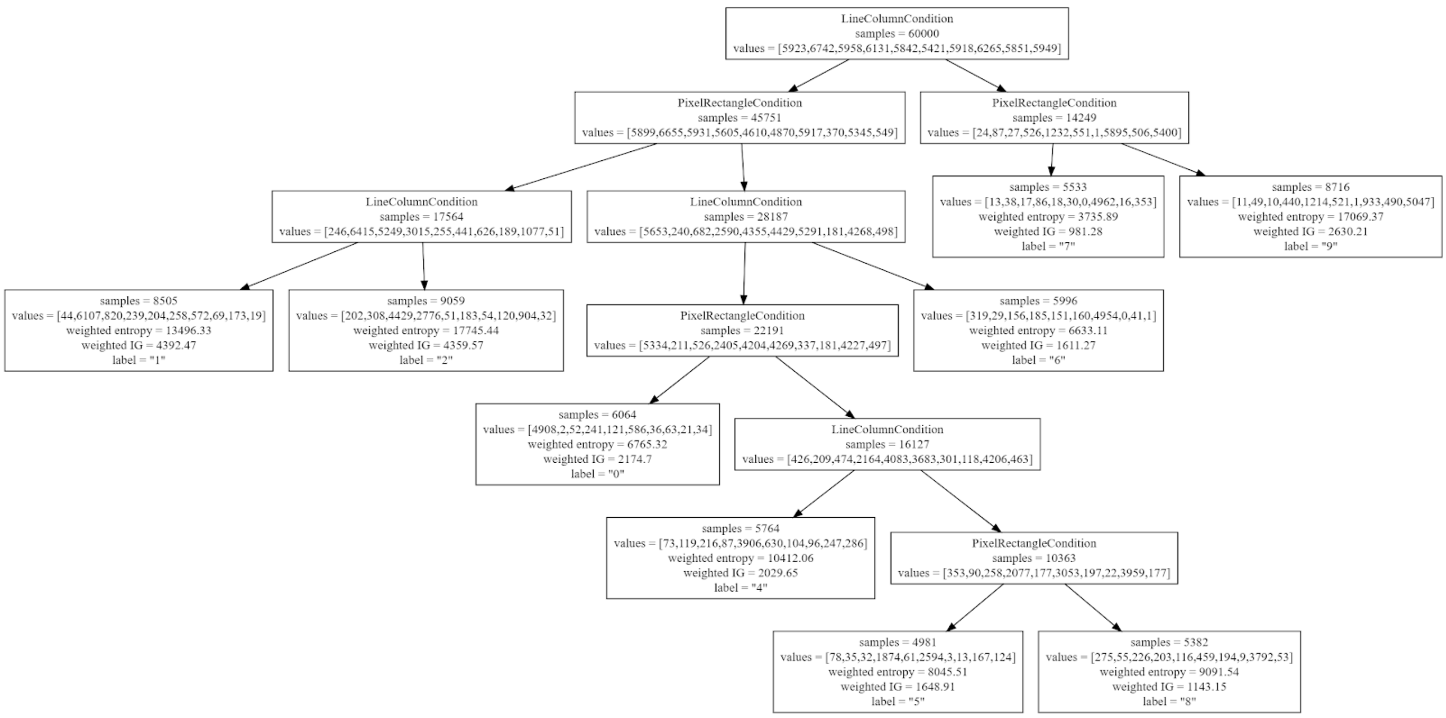
על מנת לבדוק את עצמנו ואת תכונות העצים שנוצרו הוספנו אפשרות לראות בכמה תנאים מכל סוג קבוצת תנאים העץ השתמש. השימוש משתמש בתכונות ה-reflection של Java ואינו דורש שינוי קוד בעת כתיבת תנאים חדשים.

לדוגמה, עבור קבוצת התנאים הסופית בסעיף 1 קיבלנו את המידע הבא לגבי מספר התנאים מכל סוג קבוצה:

Conditions usage:
LineColumnCondition : 45 / 256
PixelThresholdCondition : 51 / 256
PixelRectangleCondition : 160 / 256

עוד פיצ'ר שהוספנו שמשרת את אותה מטרה הוא האפשרות לראות ויזואלית את העץ שנוצר. הוספנו מחלקה חדשה בשם TreeVisualizer שמקבלת כקלט עץ ומחזירה קוד בשפת dot (שפה שנועדה להצגת תרשימים וגרפים). מהקוד שחוזר ניתן לקבל עץ באמצעות אתר שמציג שפת dot, למשל http://viz-js.com.

להלן דוגמה עבור עץ שנוצר עם פרמטר $L = 3$.



<u>סעיף 3</u>

a) הרצנו את כל הבדיקות הבאות עם הפרמטרים $L = 10,\ P = 15\%$.

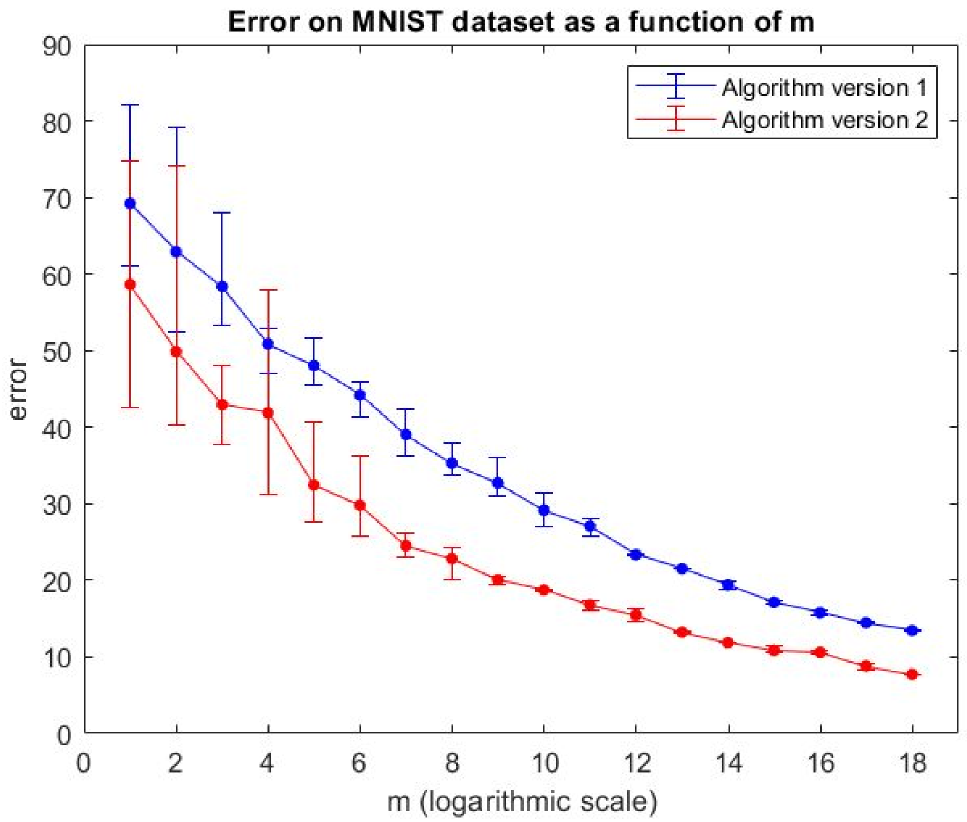
לכל גודל מדגם m , הרצנו את האלגוריתם לפחות פעמיים ולכל היותר 10 פעמים, כאשר הגורם המשפיע על כמות הריצות עבור אותו הגודל הינו האם שגיאת הבדיקה הנוכחית נמצאת בטווח של 0.5% מממוצע השגיאות של כל הבדיקות הקודמות. בדיקה זו נראית לנו יותר סבירה מבדיקה עבור שתי בדיקות סמוכות, שנשמעת פחות יציבה.

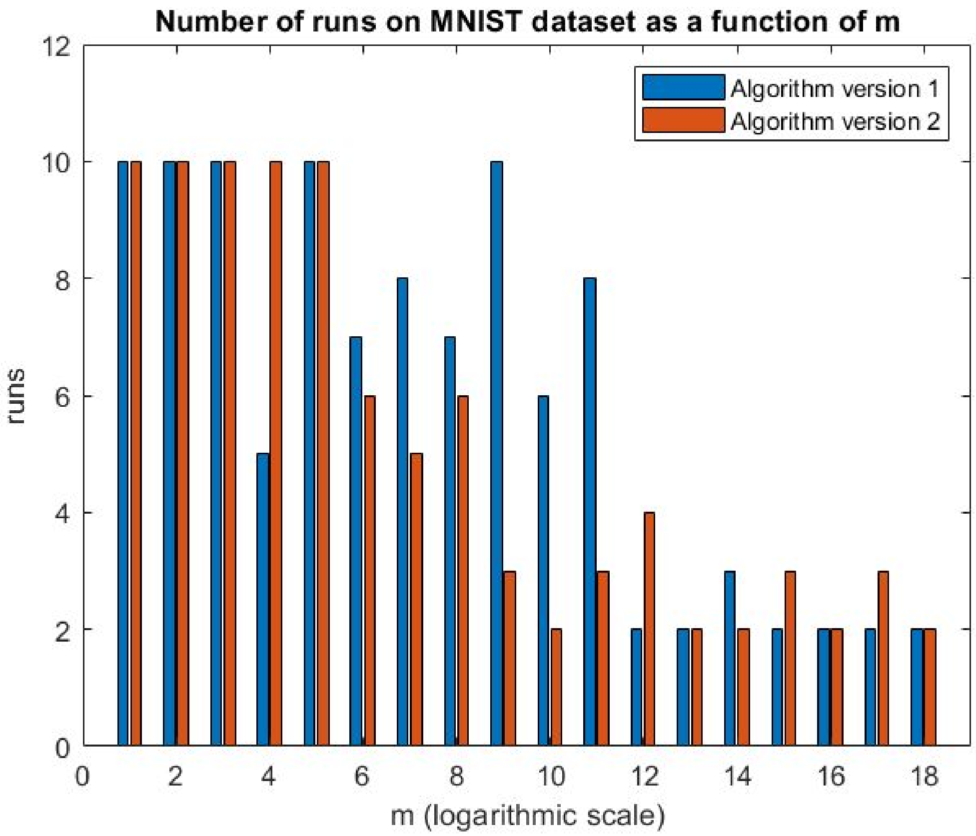
גודל המדגם ההתחלתי שבדקנו הוא 0.1% מהגודל המלא, ועולה בכפולות של 1.5 . לדוגמה: $m = 60, 90, 135, 202, \ldots$.

השגיאה המתוארת עבור כל גודל מדגם היא ממוצע השגיאות של כל ההרצות לגודל זה. כנ"ל גם לגבי מספר ההרצות. נציג את התוצאות ב-logarithmic scale.

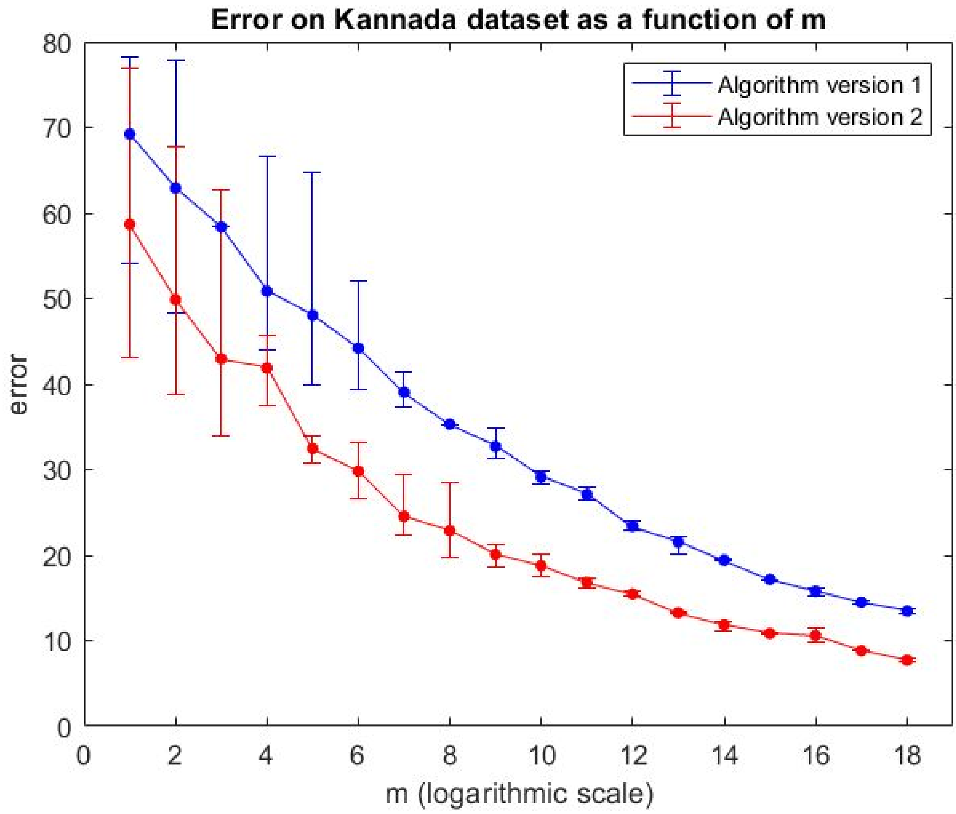
התוצאות מוצגות בעמודים הבאים.

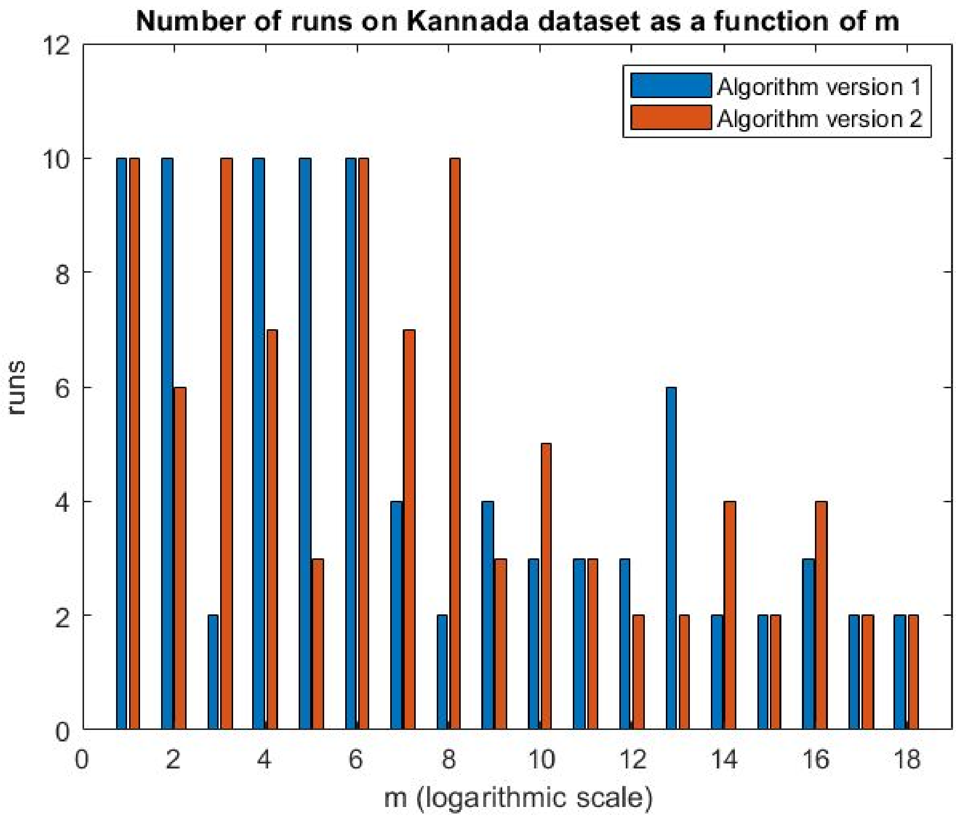
MNIST dataset





Kannada dataset





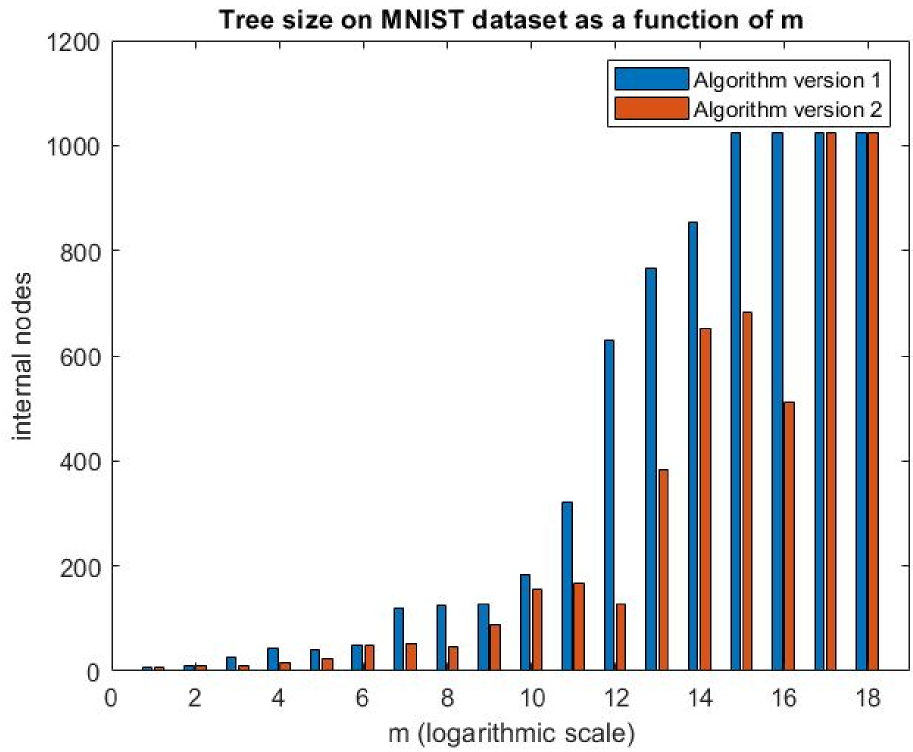
b) ההבדל העיקרי בין שתי הגרסאות הינו שהשגיאה היא יותר קטנה בגרסה השניה. נשים לב שבשני הדטה סטים השגיאה יורדת ככל שגודל המדגם עולה עבור שתי הגרסאות, שזו ההתנהגות לה ציפינו. בנוסף גם ה-error bars קטנים ככל ש- m עולה גם עבור 2 הגרסאות.
נשים לב כי עבור קנדה-דטה סט נראה שהשונות יותר קטנה (כלומר ככל ש- m עולה מספר ההרצות עבור אותו m קטן מהר יותר).

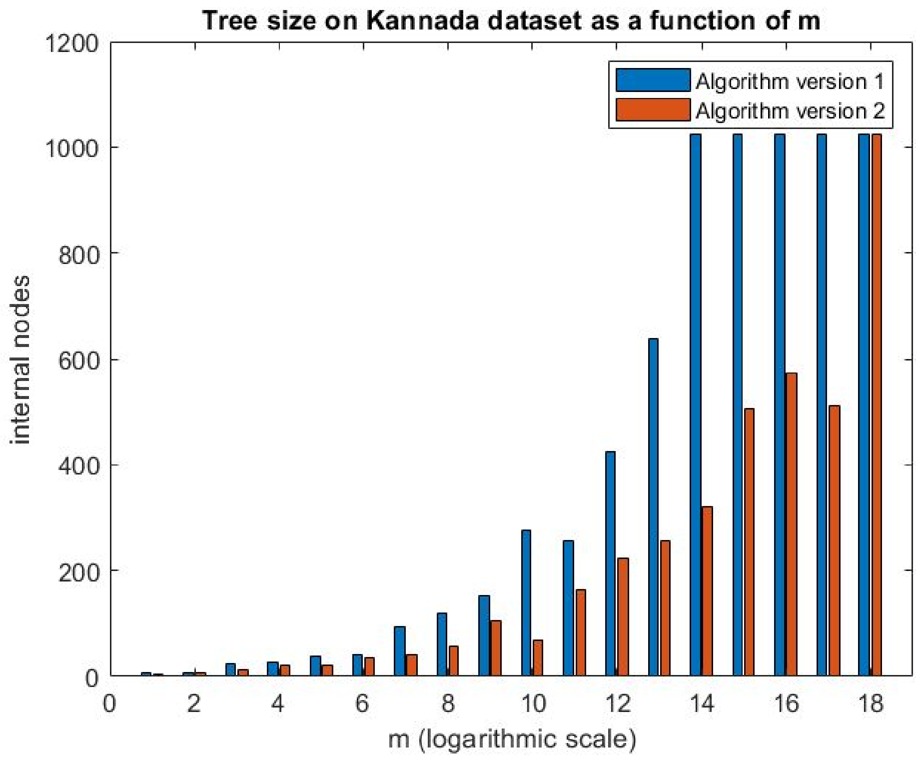
c) קיבלו את ההבדלים הללו מכיוון שבגרסה השניה אופי השאלות מאפשר לאלגוריתם למצוא בצורה יותר טובה איך לסווג את הספרות. נציין שמכך שגם עבור Kannada דטה סט קיבלנו תוצאה דומה ניתן להסיק כי האלגוריתם של גרסה 2 לא יצר overfitting עבור הספרות המערביות.
ככל שהמדגם גדל אנחנו מתקרבים יותר למציאת המסווג הטוב ביותר עבור אותו העץ ולכן בהסתברות גבוהה יותר המסווגים שלנו יחזרו תוצאות דומות, ולכן מספר ההרצות שלנו על אותו גודל מדגם קטן. התנהגות זו קוראת בשתי הגירסאות ובשני הדטה סטים.

סעיף 4

a) נציג את גודל העצים כתלות ב- m . הערך שנציג לכל m הינו ממוצע גדלי העצים על פני כל ההרצות עבור אותו m . נזכיר שמכיוון שהגדרנו פרמטר $L = 10$, נקבל לכל היותר $2^{10} = 1024$ צמתים פנימיים לכל m .

להלן התוצאות:





b) התצפיות המרכזיות שניתן להסיק הינן כי:

1) עבור כל דטה-סט ועבור כל גרסה של האלגוריתם, ככל שגודל המדגם עולה אז גודל העץ המוחזר עולה גם הוא.
2) בשני הדטה-סטים הגרסה הראשונה מחזירה עץ בגודל לפחות כמו בגרסה השניה. ככל שהמדגם גדל כך הפער גדל גם הוא.

c)

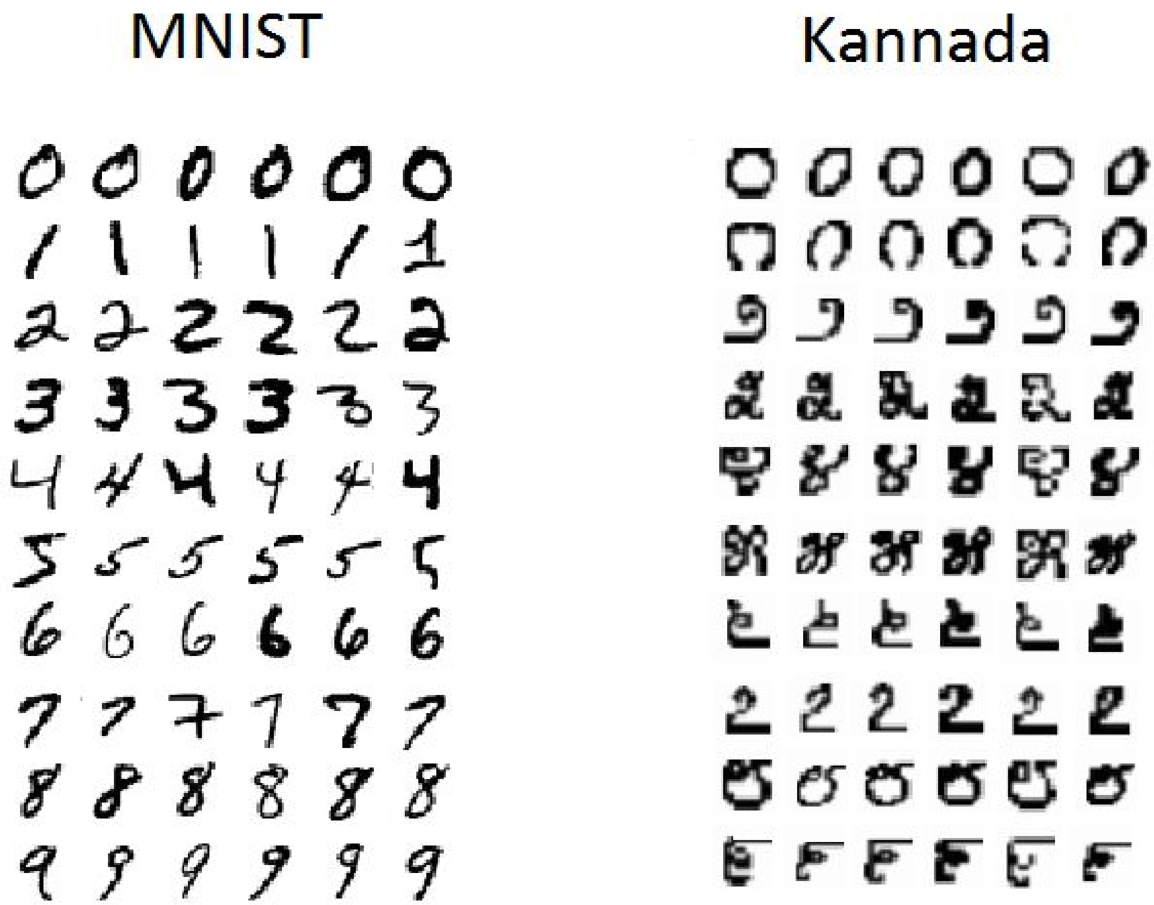
אבחנה 1: הסבר אפשרי הינו שככל שהמדגם יותר גדול כך גם כל ספרה בו מופיעה במספר גדול יותר של וריאציות ועם ניואנסים שונים, ולכן העץ צריך יותר צמתים פנימיים (= שאלות מתוך קבוצת התנאים) על מנת לסווג אותן באופן טוב. כך בעצם קיבלנו שגודל העץ גדל ככל שהמדגם גדל. באופן כללי, עבור כל מדגם וגרסה נסיק כי מדגם למידה על מדגם גדול יותר תחזיר עץ גדול יותר.

אבחנה 2: בגרסה השניה של האלגוריתם ישנו מגוון רחב יותר של שאלות שמתייחסות למספר פיקסלים בכל שאלה (שאלות על מלבנים ועל שורות ועמודות), ומאפשרות לסווג את הספרות בצורה טובה יותר. לעומת זאת, בגרסה הראשונה יש סוג אחד של שאלות שמתייחס כל פעם לפיקסל בודד בדוגמה. אי לכך, בלי תלות בסוג הדטה סט (MNIST או Kannada), האלגוריתם של הגרסה הראשונה יצטרך יותר צמתים פנימיים (= יותר שאלות) מאשר האלגוריתם של הגרסה השניה בשביל להתקרב לאותה רמת סיווג כמו בגרסה השניה (אך עדיין בעל שגיאה יותר גדולה).

<u>סעיף 5</u>

הרצנו את שני הניסויים עם הפרמטרים $L = 10,\ P = 15\%$, וקבוצת התנאים השניה (מסעיף 1).

נתבונן בספרות מתוך שני הדטה סטים:



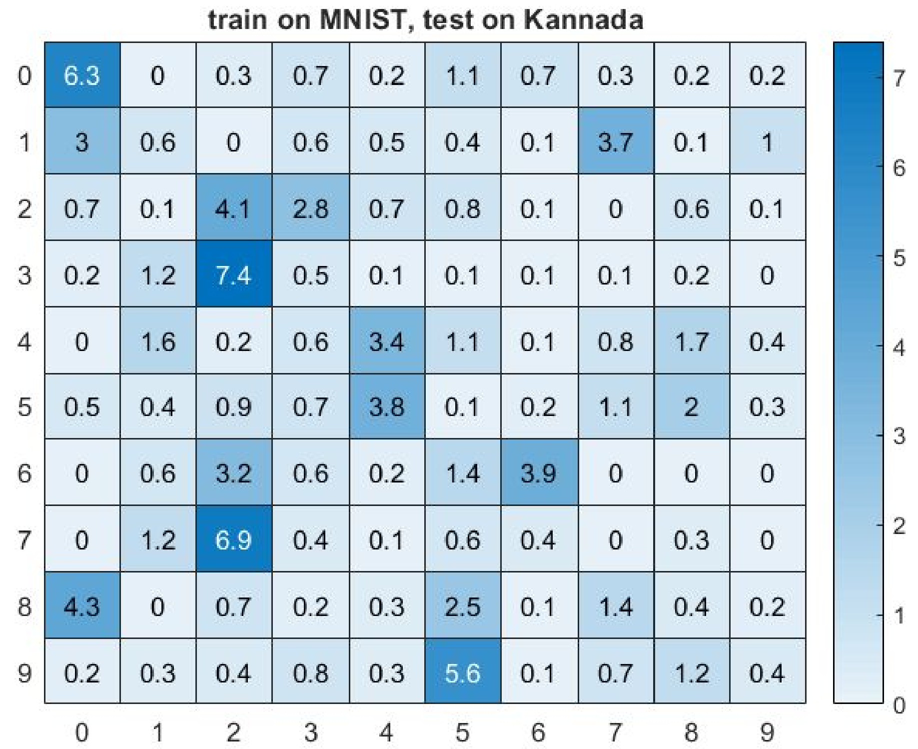
ראשית נשים לב כי רוב הספרות כתובות בצורה שונה לחלוטין בכל דטה סט, ולכן נצפה לאחוזי שגיאה גבוהים מאוד. הספרה היחידה שכתובה באותו אופן בשני המאגרים היא 0 , ושאר הספרות שונות לחלוטין. עם זאת, יש ספרות בדטה סט אחד שדומות לספרות בדטה סט אחר, אך בעלות תווית שונה:

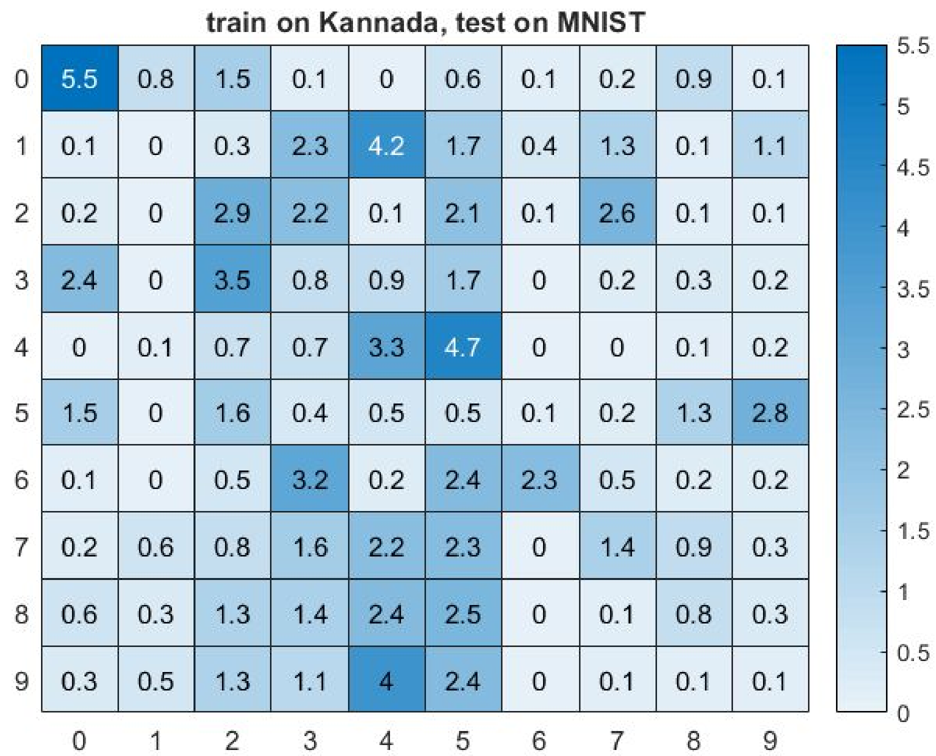
- כאשר האימון הוא על MNIST והבדיקה על Kannada:
  - הספרה 2 (ב-Kannada) דומה ל- 9 (ב-MNIST), לכן 2 עשויה להיות מסווגת ל- 9 .
  - הספרה 3 דומה ל- 2 , לכן 3 עשויה להיות מסווגת ל- 2 .
  - הספרה 4 דומה ל- 8 , לכן 4 עשויה להיות מסווגת ל- 8 .
  - גם הספרה 5 דומה ל- 8 , לכן גם 5 עשויה להיות מסווגת ל- 8 .
  - הספרה 7 דומה ל- 2 , לכן 7 עשויה להיות מסווגת ל- 2 .
  - הספרה 9 דומה ל- 5 בחלקה העליון והמרכזי, לכן 9 עשויה להיות מסווגת ל- 5 (בסיכוי נמוך יותר, יחסית).

- כאשר האימון הוא על Kannada והבדיקה על MNIST, עבורנו נראה כאילו התוצאות יהיו זהות מכיוון שבשבילנו יחס "הדמיון" הוא סימטרי, אולם לא מובטח שסימטריה זו תעלה גם מהתחזיות של העצים מכיוון שהם אינם עובדים כמו מוח של בני אדם.
  - הספרה 9 (ב-MNIST) דומה ל- 2 (ב-Kannada), לכן 9 עשויה להיות מסווגת ל- 2 .
  - הספרה 2 דומה ל- 3 , לכן 2 עשויה להיות מסווגת ל- 3 .
  - הספרה 8 דומה ל- 4 , לכן 8 עשויה להיות מסווגת ל- 4 .
  - הספרה 8 דומה ל- 5 , לכן 8 עשויה להיות מסווגת ל- 5 .
  - הספרה 2 דומה ל- 7 , לכן 2 עשויה להיות מסווגת ל- 7 .
  - הספרה 5 דומה ל- 9 בחלקה העליון והמרכזי, לכן 5 עשויה להיות מסווגת ל- 9 (בסיכוי נמוך יותר, יחסית).

נראה את התחזיות באמצעות מטריצות בלבול בעמוד הבא.

להלן מטריצות בלבול לכל ניסוי (שמוצגות באמצעות מפות חום), כך שבכל כניסה בשורה i ובעמודה j מופיע אחוז הדוגמאות שהתווית האמיתית שלהן היא i, והתווית החזויה היא j.





התוצאות שקיבלנו מתאימות למה שציפינו, אולם קיימות מספר דוגמאות שלא חזינו.
למשל:

- הספרה 9 ב-MNIST סווגה ל- 4 ברוב המקרים, כלומר האלגוריתם למד את Kannada וסיווג את הספרה 9 ב-MNIST ל- 4 . לא צפינו זאת מכיוון שלא זיהינו את הדמיון מבחינה ויזואלית.
- הספרה 5 ב-Kannada סווגה לרוב לספרה 4 (כשלמדנו את MNIST), אפילו יותר פעמים מלספרה 8 כפי שציפינו.
- הספרה 1 ב-Kannada סווגה לרוב לספרה 7 (כשלמדנו את MNIST), בשונה ממה שציפינו.

הסבר אפשרי לשונויות הללו הוא שהעץ עובד אחרת מהמוח של בני האדם. כלומר, העץ לא מסווג ספרות על פי "איך שהן נראות", אלא על פי התנאים שהצבנו לו (ובגישה חמדנית). העץ מזהה דפוסים מסוימים שאנו לאו דווקא זיהינו.
אי לכך, העץ חיפש תבניות מאוד ספציפיות שהגיע אליהם בסידור השאלות שנבחר על ידי האלגוריתם, ובסיווג מחפש את הספרה עם התבניות הכי קרובות.