SimHash & MinHash Algorithms

Bar Aloni[redacted]

Leah Orlin[redacted]

תוכן עניינים

תקציר

בעבודתנו נתמקד במימוש אלגוריתמים שמטרתם לענות על שאלת ה-Set Similarity, הערכת הדמיון בין שני סטים של אובייקטים, האלגוריתמים Minhash ו- Simhash.
בתחום אחזור המידע מהאינטרנט יש צורך להתמודד ביעילות עם שאלה זו, כדי למצוא כפילויות בין מסמכים בעת עדכון או יצירה של אינדקס.

בשיטה הנאיבית נשמור את המידע באופן לא מעובד, ונחשב את הדמיון לכל זוג מסמכים קיימים במאגר. כלומר נבצע פתרון זה אינו אפשרי מבחינה מעשית – הן בגלל המקום הרב שיקח כדי לשמור את כל המסמכים בצורתם על המחשב, וכן משום הזמן שיידרש על מנת לעשות את החישוב כל פעם מחדש: $O(n^2)$ פעולות. עבור n גודל מסד הנתונים.

האלגוריתמים שנממש מבקשים להתמודד עם שאלה זו באופן יעיל, כל אחד בדרכו:
אלגוריתם ה-MinHash האינטואיטיבי מאוד להבנה ולמימוש, חסכני יותר מבחינת גודל האינדקס וזמן תהליך יצירתו, אך בעל זמן תשאול גבוה יותר. כמו כן, הוא מאפשר למצוא דמיון גם בין מסמכים שאינם מאוד דומים.
לעומת זאת, יצירת אינדקס בעזרת אלגוריתם ה-SimHash לוקחת זמן רב יותר, ולאחר יצירתו האינדקס הוא בעל נפח רב יותר. מנגד, האינדקס הנוצר יעיל מבחינת זמן התשאול.
בנוסף, אלגוריתם זה מסובך להבנה ולמימוש ומסייע בעיקר במציאת מסמכים בעלי דמיון מאוד גבוה ואפילו זהים.

בהמשך נציג את הניסויים והתוצאות שקיבלנו ונדון במסקנות שלנו והאם הן תואמות את הספרות הקיימת בנושא.

שיטות

MinHashing:

מבוא:

אלגוריתם ה MinHash נכלל במשפחת אלגוריתמי ה-Locality Sensitive Hashing. אלגוריתמים אלו, כאמור, הם חלק מפתרון בעית ה-Set Similarity. טכניקה זו מיועדת למציאת דמיון בין 2 קבוצות בצורה יעילה ומהירה.
טכניקה זו הומצאה בידי Andrei Broder בשנת 1997 ושימשה את מנוע החיפוש AltaVista לזיהוי דפי אינטרנט זהים על מנת למנוע כפילויות בתוצאות החיפוש. טכניקה זו משמשת גם בבעיות קליסתור (Clustering) על מנת למצוא קבוצות מסמכים על סמך הדמיון בין תוכנם המילולי.
טכניקה זו מבוססת על מדד ה Jaccard Index על מנת לקבוע את מידת הדמיון בין שתי הקבוצות.
מדד Jaccard מחושב בצורה הבאה:

$$J = \frac{A \cap B}{A \cup B}$$

כאשר אנו מעוניינים בחישוב מדד זה על פני זוגות רבים של מסמכים אנו מקבלים חישוב שהוא לרוב יקר מדי מבחינה חישובית.
מסיבה זו אלגוריתמים שונים [1-6] מציעים דרך שבה ניתן לייצג כל מסמך בצורה דחוסה, כלומר – לחלץ מכל מסמך מעין "חתימה" ע"י פונקציית גיבוב (Hashing) המכילה בצורה תמציתית את המידע הרלוונטי לחישוב הדמיון וכך לשמור ולחשב את השונות בהינתן רק ה"חתימות" התמציתיות הללו.

שיטה זו מיושמת בשני שלבים:

1. **Shingling**:

ראשית, נמיר כל קובץ לצורת ***w-shingles***, אותם נגדיר בצורה הבאה:

*A **w-shingle** of a document D is a sequence of w adjacent tokens in D*

S(D,w) = all w-shingles of D

נגדיר גם את ***מדד הדמיון*** ***(r_w)*** בין מסמכים על סמך קבוצות ה Shingles בצורה הבאה:

$$r_w(D_1, D_2) = \frac{|S(D_1, w) \cap S(D_2, w)|}{|S(D_1, w) \cup S(D_2, w)|}$$

2. **Sketching**:

כעת, נחשב עבור כל מסמך את ה"חתימה" התמציתית על מנת שנוכל לחשב את הדמיון בצורה יעילה.
בהינתן h פונקציית גיבוב (במקרה שלנו, SHA-256), ובהינתן $t_1,\ldots,t_k$ רצפי אותיות רנדומיות, נחשב לכל מסמך את ה***חתימה*** בצורה הבאה:

```
GetSketch(D,w)
S = {}
for i = 1 to k:
   m = MAX_INT
   for s in S(D,w):
       if h(s%ti) < m
           m = h(s%ti)
   add m to S

return S
```

ע"י פונקציית גיבוב ששומרת על התפלגות אחידה וב"ת אנו נוכל להשתמש בחתימות שיצרנו על מנת ***לחשב את מדד Jaccard***:

$$P(h_j(A) = h_j(B)) = P(h_j(A \cap B) = h_j(A \cup B)) = \frac{|A \cap B|}{|A \cup B|} = J.$$

<u>מימוש:</u>

אחד היתרונות הבולטים של שיטת ה minhash היא הפשוטת שלה, הן ברמה האבסטרקטית והן ברמת המימוש. המימוש של minHash שאנו בחרנו היה פשוט בצורה של מילון, כאשר לכל מסמך יש מספר מזהה ייחודי שהוא ה key וה- value היה ה sketch של המסמך. נציין כי שיטה זו הייתה יעילה, פשוטה והגיונית בהנתן מסד הנתונים שעבדנו איתו. ייתכן כי אם היינו עובדות עם מסד נתונים גדול יותר (למשל מאות אלפי כתבות רויטרס) היינו בוחרות בשמירה חסכנית ויעילה יותר.

SimHashing

מבוא:

שיטת MinHash פותחת על ידי משה צ'ריקר ב-2006, ומאז היא בשימוש, בין היתר, ע"י זחלן האינטרנט של גוגל.

שיטה זו מתבססת על 2 רעיונות מרכזיים:

1. **ביטוי המסמך על ידי חתימה:**

נרצה להצמיד לכל מסמך חתימה תמציתית שמבטאת את מאפייניו החשובים ביותר, ולשמור אותה באינדקס במקום את המסמך עצמו. פעולה זו מאפשרת לנו חסכון הן במקום והן בזמן, מכיוון שאנו שומרים ביטוי מקוצר יותר של המסמך, ומולו גם נשווה מרחק, כפי שנסביר בהמשך.

חתימת SimHash של מסמך מחושבת באופן הבא:

(1) מבטאים את המסמך כאוסף של ***features:***

A set of features such that:

- changing a small fraction of the features will make little change to the sequence
- changing a large fraction of the features will cause the sequence to look very different

כלומר, קבוצה מתומצתת של המאפיינים הקריטיים שמגדירים את המסמך. ה-Features של מסמך יכולים להיות מילים, n-grams, shingles וכדומה. בפרויקט זה בחרנו להגדיר את ה-features כ-shingles.

נשים לב ששמירת המאפיינים תופסת מקום רב בזכרון, בעיה שתיפתר בשלבים (2), (3).

(2) מחשבים לכל feature ערך hash, ואותו שומרים במקום את המאפיין עצמו. בחרנו להמשיך ולהשתמש בפונקצית הגיבוב SHA-256 .

(3) מחשבים את החתימה: החתימה תהיה ה-bit-wise average של ערכי ה-hash שחישבנו לכל feature. (ניתן לבצע גם מיצוע משוקלל, אנו בחרנו למצע).

2. **מדידה יעילה של דמיון בין מסמכים:**

ראשית נגדיר את פונקציית המרחק בין חתימות SimHash: ***Hamming Distance***:

The **Hamming Distance** between 2 bit sequences is the number of places in which they differ

כאשר מרחק Hamming בין 2 חתימות הוא קטן, אז Jaccard coefficient בין המסמכים הוא גבוה, כלומר יש תאימות גדולה ביניהם. לפיכך מספיק למדוד את מרחק ה-Hamming בין מסמכים.

איך נחשב מרחקים (באופן יעיל)?

1) **השיטה הנאיבית-**
חישוב מרחק Hamming בין כל 2 חתימות.

נבחין שמימוש זה <u>אינו יעיל בזמן</u>, ולכן נציג:

2) **אינדקסי פרמוטציות-**
אחרי שקבענו את גודל חתימת ההאש להיות 256, נבחר את מספר הביטים ביניהם נרצה לאפשר שונות ונסמן מספר זה ב-***k.*** מספר זה מבטא את אחוזי הדמיון בין המסמכים.
נניח ש-2 חתימות נבדלות זו מזו ב-k ה-least significant bits בלבד. אזי הם חולקים רישא באורך של $256-k$ ביטים. ולכן אם נכניסן למערך ממויין עם חתימות נוספות, סביר להניח שהן יהיו קרובות זו לזו, ולכן במקום להשוות כל 2 חתימות באופן נאיבי, מספיק להשוות שכנים קרובים בעלי רישא משותפת.
איך נוכל להבטיח הנחה מפשטת זו? נחשב את כל הפרמוטציות של כל החתימות של מסמכי הקורפוס לקבלת חתימות "חדשות" למסמכים. נבחין שהחתימות ה"חדשות" הן בעלות אותו מרחק Hamming כמו החתימות המקוריות, וכי כל פרמוטציה מציגה least significant bits שונים כפי שרצינו. נשמור אינדקס חדש וממויין לכל פרמוטציה, בו החתימות יהיו החתימות המקוריות שעליהן הופעלה אותו הפרמוטציה. בבואנו לתשאל אותו נפעיל את אותה הפרמוטציה על חתימת השאילתא, וסיימנו.

נשים לב שאנו שומרים לכל מסמך $256!$ אינדקסים, <u>תהליך יקר במקום ובזמן</u>.
ננסה לאזן בין 2 הגישות:

3) **שיטת הבלוקים-**
במקום ליצור את כל הפרמוטציות האפשריות, ומפני שאנחנו רוצים לאפשר שונות בעד כ-k ביטים, נוכל לחלק את החתימת ל-***B*** בלוקים ($B > k$), ולחשב פרמוטציות על *בלוקים של ביטים* במקום על *כל הביטים*.

כך נפחית מספר האינדקסים ל- $B\ choose\ k$ (ובכך נחסוך במקום ובזמן אינדוקס בהשוואה לשיטה 2), ובמקביל נקטין את מרחב החיפוש ונחסוך בזמן תשאול (בהשוואה לשיטה 1).

השאלה שנשארת פתוחה היא מהם ערכי הפרמטרים B, k האופטימליים עבור Dataset מגודל מסויים, וחתימה מגודל מסויים.
בשאלות אלו, בין היתר, התמקדנו בניסויינו כפי שיוצג בהמשך.

מימוש:

חישוב ערך הגיבוב נעשה בעזרת מימוש ה - sha256 של פייתון. חישוב חתימת המסמך, סכימת החתימות וחישוב ה-bitwise average, נעשה על תיאור המחרוזות של ערכי הגיבוב הללו. לאחר שהושג הערך, החתימה הומרה לייצוג כ- int בבסיס 2, על מנת להקל בחישוב פעולות על ביטים שייעשו בהמשך.

לאחר מכן, כתלות ב-threshold ובערך בלוקי הרישא אותם רצינו למדוד, חישבנו את החלוקה של גודל החתימה איתו עבדנו לבלוקים, ואת כל הפרמוטציות האפשריות על מספר בלוקים שכזה. לאחר מכן חילקנו כל חתימה במאגר לבלוקים, ועליהם החלנו את כל הפרמוטציות האפשריות לפי נוסחאות הפרמוטציה אותן חשבנו בשלב הקודם.

תהליך זה נעשה בשלבים: חישבנו את ההשפעה של כל נוסחת פרמוטציה (רשימה של int) כל החתימות במאגר, ושמרנו את המיפוי (doc_id, permutated_singature) במילון שמייצג את אינדקס הפרמוטציה הנ"ל. את המילון מיינו לפי ערך החתימה ושמרנו בקובץ עצמאי.
כדי שנהיה מסוגלים לחפש בהמשך באינדקס הפרמוטציות, נצטרך לדעת איזו פרמוטציה הופעלה בייצור האינדקס הנתון. לפיכך שמרנו קובץ נוסף שממפה את שם המסמך לנוסחת הפרמוטציה.
בחרנו לשמור את האינדקס כמילון, לאחר ששקלנו מימושים אחרים דוגמת עץ בינארי, רשימה.
עץ בינארי אמנם תומך בעדכון מהיר יותר של האינדקס מכיוון שמיונו נעשה אוטומטית, אך הוא תופס מקום רב יותר. רשימה חסכנית יותר במקום, אך משום שנצטרך לשמור את המספר המזהה של המסמך (כדי לטפל במקרה הנדיר ביותר של חתימות זהות) ברשימה נוספת, הוחלט לממש כמילון בשל הנוחות שהוא מספק.

תהליך החיפוש באינדקס נעשה על ידי חיפוש בינארי של קצוות התחום בו מצויות החתימות בעלות הרישא הזהה לרישא של השאילתא הנבדקת. לאחר שצומצמו חיפושינו לטווח בו מצויים להיות החשודים ממרחק hamming נדרש, נוסיפם לסט החשודים ונעבור לאינדקס הפרמוטציה הבא.

כאשר סיימנו לעבור על כל האינדקסים, נעבור על החשודים אותם שמרנו ונחשב מרחק hamming מלא ביניהם לבין השאילתא הנבדקת, ונחזיר את אלו בעלי מרחק נמוך מזה שביקשנו למצוא.

נוסיף שאת האינדקסים השמורים היינו רוצים לדחוס לפי אחת הטכניקות שהוצגו לנו במהלך הקורס, אך לא עשינו זאת מאחר ורצינו להתעמק בהבנה של האלגוריתמים החדשים אותם בדקנו.

התנסות והסקת מסקנות

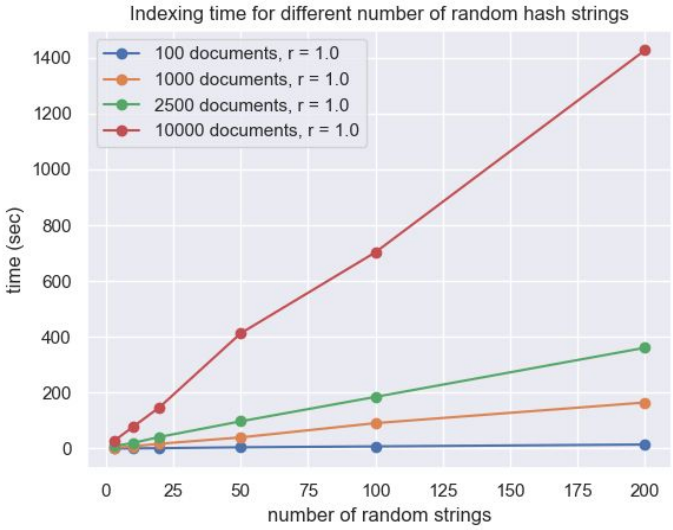
1. ייעול MinHash:

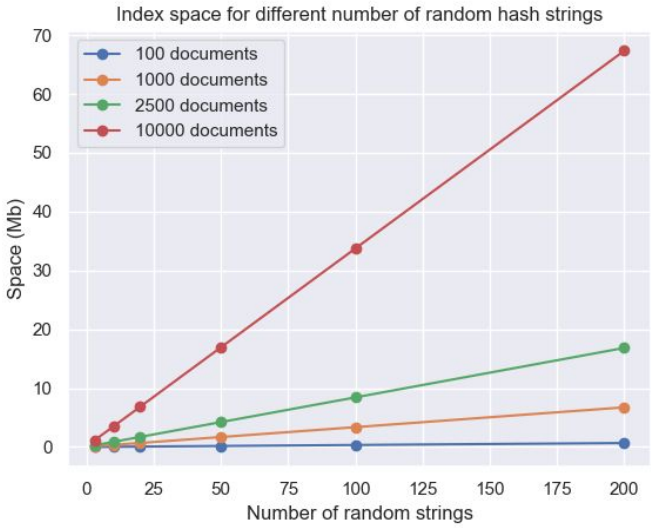
(1) השפעת כמות המילים הרנדומליות

כחלק מיצירת האינדקס באלגוריתם minhash, מייצרים sketch לכל מסמך ע"י שרשור מילים רנדומיות לסוף כל shingle, חישוב פונק' hash ולבסוף בחירת הערך המינימלי מכל התוצאות ה hash. רצינו למדוד את השפעת מספר המילים הרנדומיות המשורשרות על יצירת האינדקס - הן מבחינת מקום והן מבחינת זמן. נצפה למצוא כי מספר שרשורים גדול יותר יהיה יקר מבחינת מקום וזמן, אך יתן דיוק גבוה יותר בהשוואה למספר שרשורים נמוך יותר.

i. השפעה על יצירת האינדקס

על מנת לבדוק את השפעת מספר השרשורים על יצירת האינדקס, בדקנו את זמן יצירת האינדקס וכן את המקום שהוא תפס על הדיסק עבור 3, 10, 20, 50, 100 ו- 200 מילים רנדומיות.





בגרפים ניתן לראות בבירור כי שימוש במספר גדול יותר של מילים רנדומיות גובה מחיר גם בזמן וגם במקום. תוצאה זו מתאימה לציפיות שלנו. לפיכך עולה השאלה, האם אכן יש יתרון בשימוש במספר מילים רנדומיות רב. נבדוק זאת בניסויי הבא על מידת הדיוק שכל שיטה כזו מציעה.

ii. על תשאול האינדקס:

כעת רצינו לבדוק האם יצירת אינדקס תוך שימוש במספר גדול יותר של מילים רנדומיות יביא לתוצאות טובות יותר במידת הדיוק של השיטה. בנוסף לכך רצינו לבדוק האם השיפור יהיה תלוי במידת הדמיון (ה threshold) אותו נחפש. הציפייה שלנו הייתה שאכן שימוש במספר גדול יותר

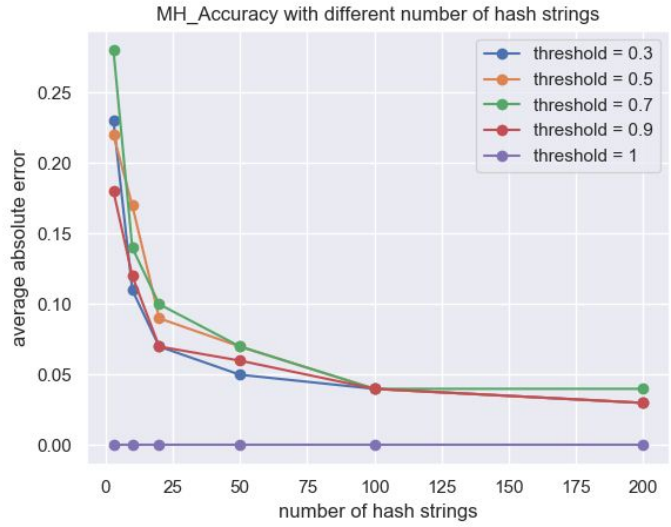
של מילים רנדומיות יביא לשיפור באיכות. על מנת לבדוק את איכות ההתשובה בחרנו במדד הבא:

1. בחרנו מסמך אקראי ויצרנו לו "מסמך תאום" ע"י שימוש ב jaccard coefficient תוך שאנו משנים את המסמך כל פעם קצת עד שאנו מגיעים לרמת דמיון שהיא מעט מעל ה threshold שקבענו. לעת עתה השארנו את המסמכים כמות שהם, בצורת טקסט.
2. כעת בדקנו את המרחק בין המסמכים השונים ע"י מדד ה jaccard .
3. לקחנו את שני המסמכים (המקורי והדומה) והפכנו אותם ל sketches בעזרת המודל שבנינו.
4. מדדנו את המרחק בין ה sketches ע" מדד jaccard.
5. חישבנו את הטעות של המודל לפי המדד הבא:

$$absolute\ error\ =\ |\ actual\ distance\ -\ sketch\ distance\ |$$

כלומר, ההפרש במדד Jaccard בין ההצגות השונות של מסמכי האינדקס: כקלט לא מעובד וכקלט שעובד על ידי אלגוריתם הMinhash.

ניתן לראות את התוצאות בגרף הבא:



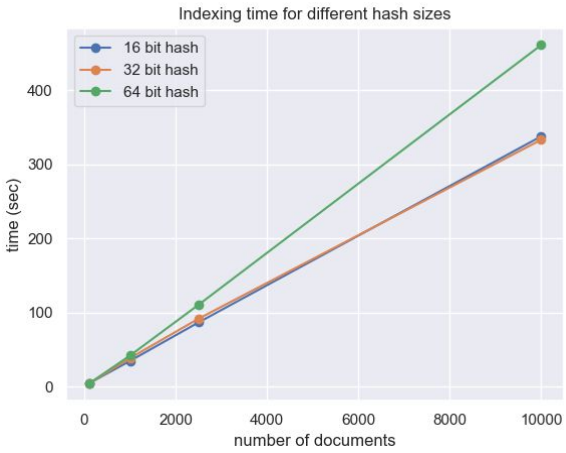
בגרף ניתן לראות כי ככל שמספר המילים הרנדומיות עולה - כך הטעות שהאלגוריתם מציג פוחת. כלומר - ככל שיש יותר מילים יש יותר דיוק בקביעת המרחק בין המסמכים. תוצאה זו אכן מתאימה להשערות שלנו. ניתן להסיק מכך כי אכן מספר המילים הרנדומיות יקר יותר במקום ובזמן אך הוא נותן בתמורה מודל מדויק יותר. כשנרצה לבחור את הפרמטרים הסופיים נרצה להביא את כל הגורמים האלו בחשבון ולהחליט לפי המדדים שחשובים לנו יותר.

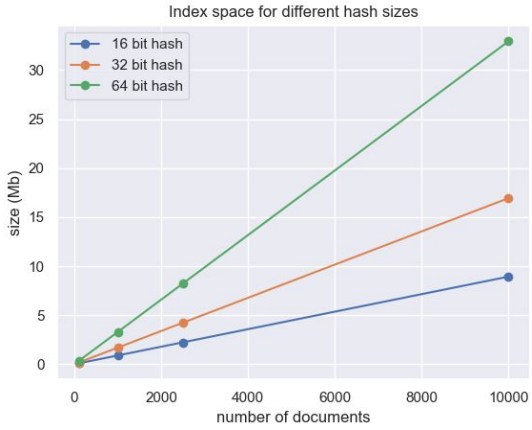
(2) השפעת גודל ההאש:

אלגוריתם ה Minhash עושה שימוש בפונקציית גיבוב אותה אנו מפעילים על ה shingles. רצינו לבדוק מה ההשפעה של גודל ה hash הנבחר על יעילות האלגוריתם, הן מבחינת זמן ומקום והן מבחינת יעילות. ההנחה שלנו הייתה כי שימוש בפונק' hash שנותנת ערך גדול יותר תייקר את הביצועים של האלגוריתם מבחינת זמן ומקום - אך תספק לנו רמת דיוק גבוהה יותר. על מנת לבדוק פרמטר זה בחרנו 3 פונקציות גיבוב שונות (של 16, 32 ו- 64 בתים), בנינו אינדקס עם הפונק' השונות ובדקנו את ביצועי השיטה על שלושת הפרמטרים הנ"ל (זמן, מקום ודיוק). את התוצאות ניתן לראות בגרפים הבאים:

i. השפעת גודל ההאש על יצירת האינדקס:

כאמור, רצינו לבחון את ההשפעה של גודל ה hash על יעילותו של האינדקס מבחינת זמן ומקום. לכן יצרנו אינדקסים שונים עבור כל פונק' hash ומדדנו כמה זמן לקח ליצור את האינדקס וכמה מקום הוא תפס על הדיסק. את התוצאות ניתן לראות בגרפים הבאים:





ניתן לראות כי הגרפים יצאו כפי המצופה, כאשר hash גדול יותר גורם לאינדקס לתפוס מקום גדול יותר על הדיסק. כמו כן רואים השפעה, אמנם נמוכה יותר, על הזמן שלוקח ליצור את האינדקס. נוכל להבין מכאן שאכן יצירה של hash גדול יותר לוקח יותר זמן חישוב. נוכל גם להסיק מכאן שכדאי להוריד את גודל ה hash למינימום ההכרחי כך שעדיין האלגוריתם יוציא תשובות מדויקות לשאילתא.

2. יעול SimHash:

(1) השפעת הפרמטרים k, B

כפי שהצגנו בהסבר על השיטה, ננסה להבין מהם ערכי הפרמטרים B, k האופטימליים עבור Dataset מגודל מסויים, וחתימה מגודל מסויים, ובפרט מהי ההשפעה מבחינת ה-memory/processing trade-off.

בניסויינו בחרנו לבדוק מסדי נתונים מגדלים: $[100,\ 1000,\ 2500,\ 10000]$,

חתימות בגדלים: $[64,\ 256]$ ביטים.

וערכי הפרמטרים:

- $Thresholds \in [0.9,\ 0.8,\ 0.7]$

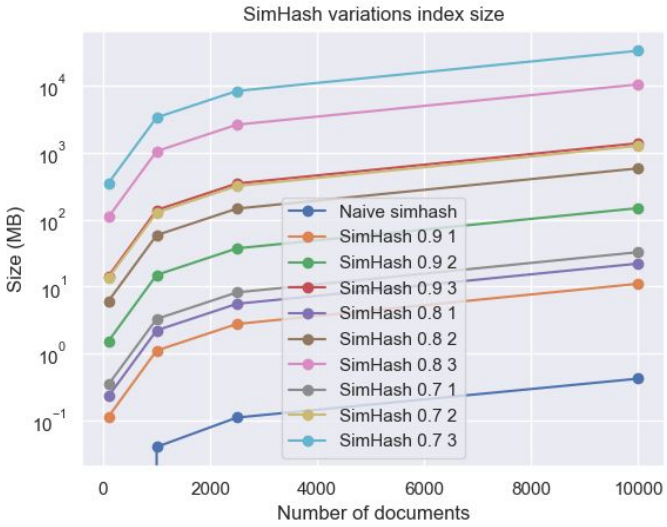
(שמורה על k באופן ישיר: $k = floor((1 - Threshold) * |Hash|)$)

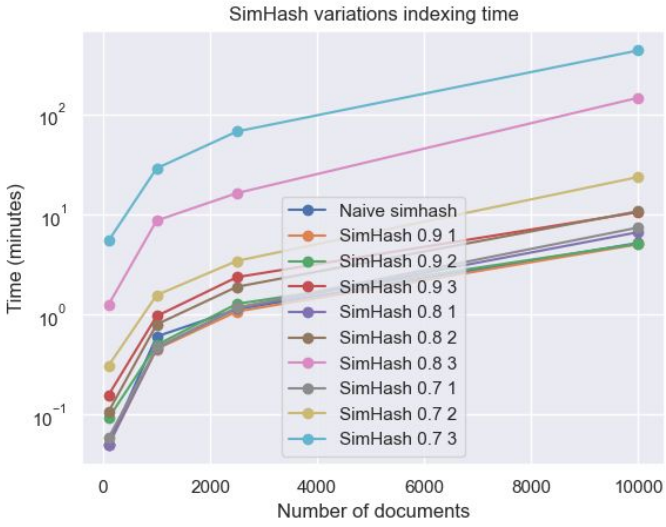
- $Blocks \in [k+1,\ k+2,\ k+3]$

נציג כעת את ניסויי השפעת הפרמטרים:

i. על תהליך יצירת האינדקס:

בניסוי זה בחנו את השפעת הפרמטרים השונים על גודל האינדקס וזמן יצירתו.
יצרנו גרסת Simhash שונה לכל קומבינציה של פרמטרים $Thresholds \times Blocks$, 9 במספר, וגרסא נוספת של Naive Simhash, כלומר אינדקס לא ממויין ולא מחולק לבלוקים של חתימות simhash.





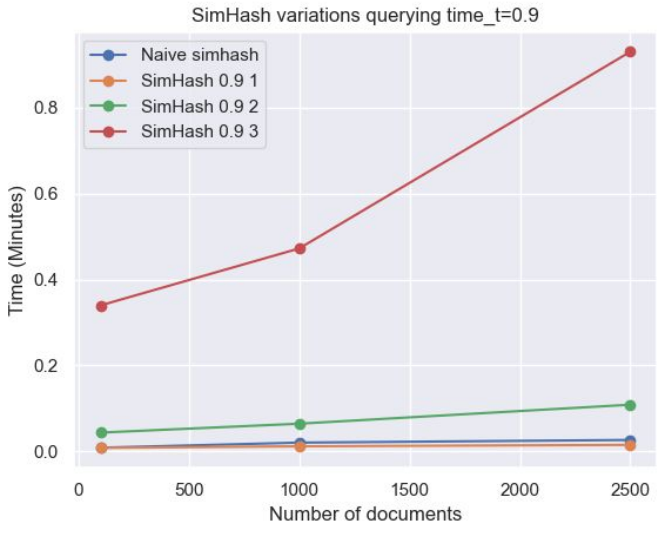
ניתן להבחין בבירור שמתקיים:

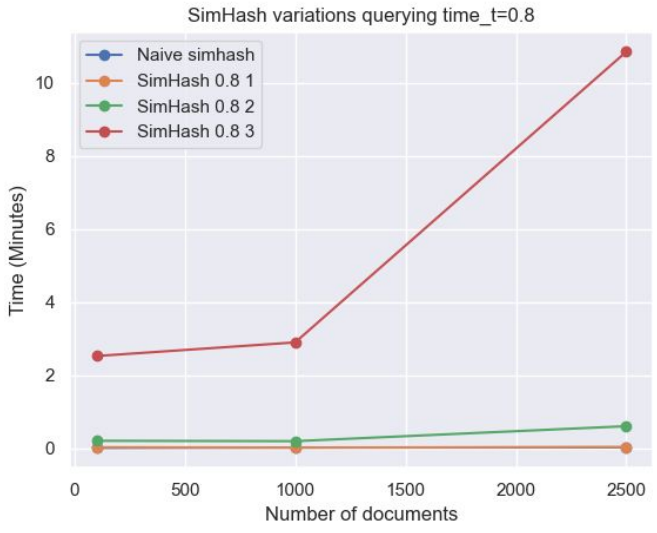
- הגרסאות המוסיפות 3 בלוקי רישא הן הכי פחות יעילות הן במקום והן בזמן.
- ככל שסף הדמיון יורד, כך תהליך האינדוקס לוקח יותר זמן ומקום.
- בין כל הקומבינציות, למעט $(0.7,\ 3),\ (0.8,\ 3)$, אינן הבדל משמעותי מבחינת זמני יצירת האינדקס והמקום הנדרש לאחזקתו.
- זמן האינדוקס וגודל האינדקס תלויים באופן ישיר בגודל מסד הנתונים (ולכן יש ביניהם יש קורלציה גבוהה).
- יש יחסים קבועים בין השיטות השונות, כל הגרפים בעלי אותה הצורה.
- אבחנה מעניינת היא שתהליך האינדוקס של הקומבינציות $(0.7, 1),\ (0.8,\ 1),\ (0.8, 2),\ (0.9,\ 1),\ (0.9,\ 2)$ נעשה בסדר גודל של זמן ושל מקום כשל תהליך האינדוקס הנאיבי (שמירה של טבלה יחידה לא ממויינת).

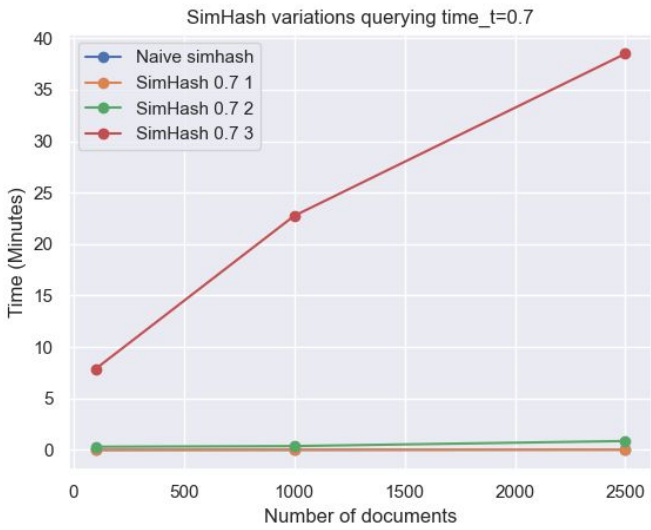
מן ההבחנות נקבע שאלא אם יש הבדל משמעותי מבחינת יכולת הדיוק של השיטות שבהן $Block = [k+3]$, נעדיף שלא להשתמש בהן מכיוון שהן דורשות באופן משמעותי יותר משאבים. השיטות שאינן משתמשות בגודל בלוק זה, צורכות משאבים כשל הנאיבי, ולכן נפנה למדוד זמני תשאול אינדקס ודיוק התוצאות מול השיטה הנאיבית (והפשוטה למימוש).

ii. **על תהליך תשאול האינדקס:**

בניסוי זה בחנו את השפעת הפרמטרים השונים על זמן תשאול האינדקס ועל הדיוק של התוצאות שתהליך זה מחזיר.
אנחנו ממשיכים להשוות בין עשרת הגרסאות שיצרנו.







בניסוי זה מדדנו את הזמן שלוקח לתשאל אינדקס של שיטה מסויימת בכעשר שאילתות רנדומליות. פיצלנו את תוצאות הניסוי לשלושה גרפים שונם, כאשר בכל בגרף מוצגים זמני תשאול שאילתות שמתייחסות לאותו threshold.

ניתן לראות כי בקשה לקבלת דמיון משיטה שמממשת 3 בלוקי רישא, יקרה מאוד בהשוואה לשיטות האחרות, ובאופן מובהק כשאר מבקשים למצוא דמיון של כ- 70%.
מעבר לכך, תשאול שאר האינדקסים, ביניהם הנאיבי, הינו באותו סדר גודל של זמן.
תוצאות אלו מחזקות את המסקנות שהסקנו בחלק הקודם של הניסוי, לפיהן נעדיף את השיטות המשתמשות ב-1, או 2 בלוקי רישא.

3. השוואת MinHash ו-SimHash והמימוש הנאיבי:

1. השוואת MinHash, SimHash והמימוש הנאיבי:

(1) תהליך יצירת האינדקס:

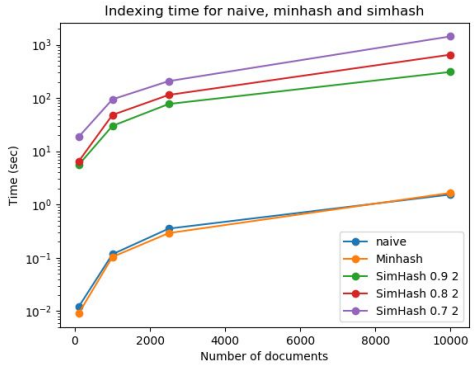
המטרה העיקרית של הפרויקט שלנו היה לבחון את יעילותם, חוזקתם וחולשתם של האלגוריתמים minHash ו- SimHash ולקבל תחושה של היתרונות והחסרונות שלהם אחד ביחס לאחר. הפרמטרים העיקריים שרצינו להשוות את האלגוריתמים לאורם היו:

- זמן יצירת האינדקס

- המקום שהאינדקס תופס בזכרון
- יעילותם של האלגוריתמים, מידת הדיוק שלהם
- זמן התשאול של האלגוריתמים.

לשם מטרה זו, מימשנו את שני האלגוריתמים ויצרנו אינדקסים בעזרתם ובנוסף לכך שמרנו את מסד הנתונים כפי שהוא אחרי עיבוד מקדים בסיסי, בצורה טקסטואלית, ללא כל שינוי. את הפרמטרים השונים מדדנו על שלושת האינדקסים שנוצרו ואת התוצאות נמחיש בעזרתם הגרפים הבאים:

<u>זמן:</u>



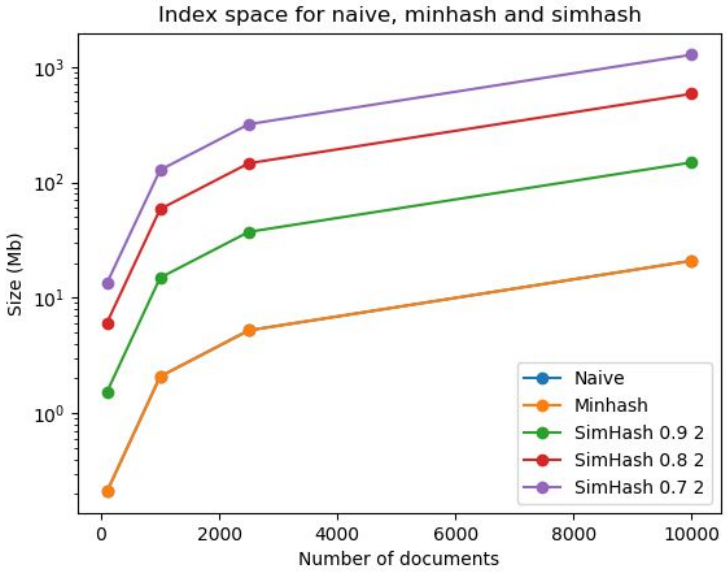
בגרף זה ניתן לראות את הזמן שלקח לעבד את הדאטא וליצור את האינדקסים עבור כל שלושת השיטות. נשים לב שעבור מינהאש ישנם שלושה מדדים, בהתאם לזמנים הטובים ביותר שקבלנו על ה thresholds השונים בהם נרצה להשתמש מאוחר יותר לתשאול המאגר. את הזמנים הראנו בסקאלה לוגריתמית כדי שניתן יהיה לראות גם את הזמנים של הנאיבי והמינהאש וגם את הזמנים של סימהאש, למרות הפערים הניכרים ביניהם. בציר ה - x אנו רואים את הזמן שלקח ליצור את האינדקס עבור כל גודל של מאגר מסמכים ובציר ה - y את הזמן שלקח לאנדקס אותו בשניות.

בהתאם למה שציפינו, אנו מוצאים בגרף הזה כי השיטה היעילה ביותר מבחינת זמנים היא השיטה הנאיבית. מדובר בשיטה שעושה עיבוד מינימלי בלבד לנתונים ושומרת אותם כמו שהם, ללא שינוי.

כמו כן, רואים בבירור כי שיטת הסימהאש היא האיטית ביותר. תוצאה זו נראית הגיונית בגלל העיבוד הרב שהאלגוריתם מבצע על הנתונים מראש על מנת לאפשר זמן תגובה מהיר לשאילתות בשלב מאוחר יותר. בנוסף לכך, יש לקחת בחשבון שהרצנו את האלגוריתמים על כמות קטנה יחסית של דאטא (עד 10,000 מסמכים באורך כ 260 מילים בממוצע)

מקום:

כעת באנו לבדוק את היעילות של האלגוריתמים השונים מבחינת המיקום שהם תופסים על הדיסק. בדקנו את האינדקסים שיצרנו וניתן לראות את התוצאה בגרף הבא:



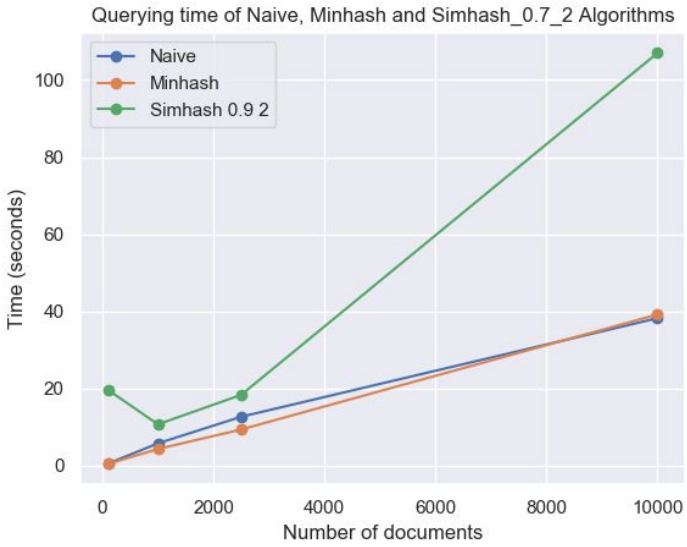
אנו רואים כי האלגוריתמים הנאיבי ומינהאש הם החסכניים ביותר במקום. שוב - בגלל שמדובר בסט דאטא קטן יחסית וגם בגלל שסימהאש לוקח יותר מקום כדי לבצע תשאול יותר מהיר אחר כך. אנו רואים כי גם באלגוריתם של סימהאש ככל שסף הדמיון בתשאול יותר נמוך כך הוא לוקח יותר מקום וזה מתאים לציפיותנו, משום שזה מסתדר עם ההגיון של האלגוריתם לשמור יותר אינדקסים ולבדוק עוד אופציות כדי לא לשלול מילים יותר רחוקים.

(2) תהליך תשאול האינדקס:

כעת נרצה להשוות את שלושת השיטות לפי הביצועים שלהם בתשאול (query). נרצה לבדוק את יעילותם של האלגוריתמים הן בזמנים והן ברמת הדיוק.

<u>זמן:</u>

כדי לבדוק את יעילות האלגוריתמים יצרנו טסט סט דומה והרצנו queries על האינדקס הנאיבי והן על אינדקסים של שני האלגוריתמים. ציפינו למצוא כי הסימהאש יתן תוצאות מהירות יותר, בגלל המחיר הרב ששילמנו במקום ובזמן כדי לבנות אינדקס שיהיה נח לתשאול. בנוסף לכך ציפינו שהמינהאש יהיה בקצב של הנאיבי, משום שמדובר בפעולה דומה מבחינת החיפוש, אלא רק השוואות בין סקצ'ים במקום בין מילים. את התוצאות ניתן לראות בגרף הבא:

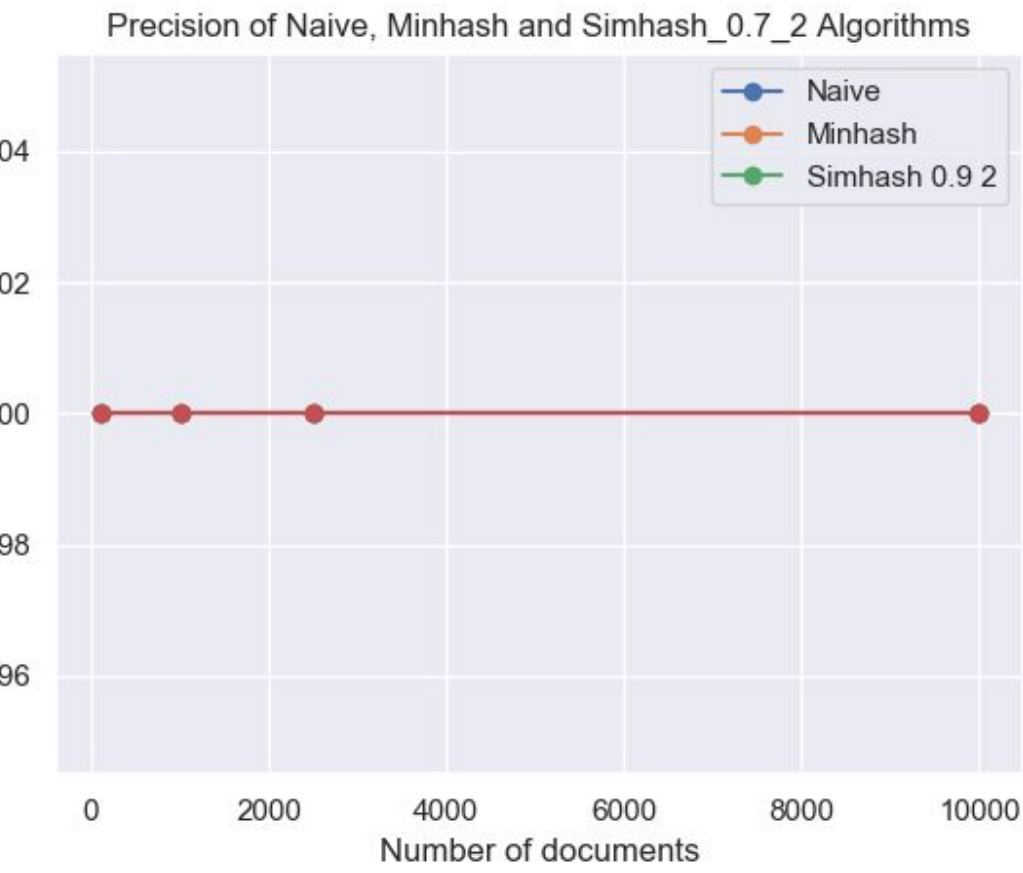


בגרף הזה ניתן לראות כי בניגוד לציפיותנו דווקא אלגוריתם ה simHash היה היקר ביותר מבחינת זמנים. לאחר סקירת ספרות מעמיקה יותר הגענו למסקנה שייתכן ותוצאה זו קרתה בשל ה"היפר-פרמטרים" שנקבעו לאלגוריתם. ערך-הסף של הדמיון, מספר הפרמוטציות וגורמים נוספים יכולים להשפיע מאוד על הביצועים של האלגוריתם ובחירת פרמטרים לא אופטימליים יכולים להסביר את התוצאה שאנחנו רואות כאן. בנוסף לכך ניתן לראות ש מינהאש ונאיבי מאוד קרובים. שניהם יעילים מאוד וכן תהליך התשאול בשניהם זהה (חוץ מהפעולה המתמטית עצמה של חישוב jaccard coefficient שמתבצעת על מספרים במינהאש ועל טקסט בנאיבי)

אנחנו מסיקות מהנתונים הנ"ל שאכן אלגוריתם הסימהאש עלול להיות יעיל יותר, אך חייבים להיות זהירים בבחירות הפרמטרים כדי לקבל את הביצועים האופטימליים.

<u>דיוק:</u>

בכדי לבדוק את דיוק המודלים, הרצנו שוב את שלושתם על אותו סט דוגמאות דומה ומדדנו את ה precission של המודלים בהשוואה לתוצאות של הנאיבי. התוצאות להלן:


Precision of Naive, Minhash and Simhash_0.7_2 Algorithms
Naive
Minhash
Simhash 0.9 2
04
02
00
98
96
0
2000
4000
6000
8000
10000
Number of documents

ראינו ששלושת השיטות יעילות ומביאות לרמת דיוק מצויינת.

סיכום

בעבודתנו מימשנו שני אלגוריתמים שונים לבדיקת דמיון יעילה בין מסמך לבים סט קיים של מסמכים.
מניסויינו על מימוש ה-minhash למדנו שלפעמים אלגוריתמים שהם פשוטים להבנה ולמימוש הם גם יעילים ביותר.

בחינת ערכי ה-threshold ובלוקי הרישא במימוש Simhash תרמו לנו בהבנה של מהו הטווח הישים לבחירת פרמטרים אלו. למדנו כי ככל שאנו מקטינים את ערך ה-threshold, אנחנו משלמים בזמן ובמקום הנחוצים לשמירת האינדקס, וכן בזמן התשאול שלו. כמו כן אותה ההבחנה חלה גם ככל שמגדילים את מספר בלוקי הרישא. הדבר מתיישב עם ההבנה שלנו את השיטה: אחז דמיון קטן יותר משמעו מספר בלוקים שונים רב יותר.
בשיתוף עם מספר בלוקי רישא רב יותר אנחנו מגדילים את ערך הביטוי B choose k שמיתרגם למספר טבלאות רב יותר אותן נצטרך לחשב, לשמור, ולתשאל.
לפיכך, ובהתאם לגודל פונקציית הגיבוב שבחרנו, 256, גודלם ואופיים של מסדי הנתונים אותם אינדקסנו, בחרנו לצמצם את טווח ה-threshold ל- 0.75 ואת מספר בלוקי הרישא לעד כ-2, אך פרמטרים אלו יכולים להיקבע לפי גדלי מסדי הנתונים, ולא להקבע באופן עיוור.

[redacted]

עבודה נוספת

SimHash

כפי שנאמר בפרק המימוש של אלגוריתם Minhash, נרצה לבחון את:

1) השפעת הדחיסה של האינדקסים של אלגוריתם ה- SimHash בשיטת הבלוקים, על זמן האינדוקס וגודל האינדקס.
2) מימוש טבלאות הפרמוטציה:
שמירת טבלאות הפרמוטציות בעץ מספקות לנו מימוש קל להרחבה.
שמירתן ברשימה ממוינת חסכונית יותר בזמן ובמקום, ומייתרת את המעבר על כל העץ בשלב התשאול.
בסנוף, את הרשימה המויינת ניתן לדחוס אף יותר על ידי ויתור ציון רישא של חתימה, הזהה לרישא של החתימה הקודמת, כמו שעשינו.
3) בחירה ומשקול הפיצ'רים בחתימה: לבחון עבור מאגר הנתונים אותו אינדקסנו האם ישנם פיצ'רים יעילים יותר מאשר shingles לייצג בעזרתם את התכונות החשובות ביותר של המסמך. כמו כן, האם ישנם פיצ'רים חשובים יותר מאשר אחרים.

ביבליוגרפיה

http://www2007.org/papers/paper215.pdf
http://benwhitmore.altervista.org/simhash-and-solving-the-hamming-distance-problem-explained/
http://matpalm.com/resemblance/simhash/
https://www.geeksforgeeks.org/interpolation-search/

---

iini[1] *Broder, Andrei Z.* (1997), "On the resemblance and containment of documents", *Compression and Complexity of Sequences: Proceedings, Positano, Amalfitan Coast, Salerno, Italy, June 11-13, 1997 (PDF)*, *IEEE*, pp. 21–29, *CiteSeerX 10.1.1.24.779*, *doi:10.1109/SEQUEN.1997.666900*, *ISBN 978-0-8186-8132-5*, archived from *the original (PDF)* on 2015-01-31, *retrieved 2014-01-18*.