פרוייקט גמר כיתה י"ב של תכנית מדע חישובי. הפרוייקט הינו שיטות במציאת מקורות  
של טקסטים, עם דגש על מדד  
ג'קארד לדמיון, ועל מציאת ז'אנרים לשירים.

פרויקט מ"ח

מגישים: איתי חי, ג'יי טננבאום, ברק ניב

שנה: 2013

תמצית

הפרויקט מאפשר לזהות לאיזה קורפוס מתוך המאגר שיש לנו הכי מתאים הקלט אותו הכניס המשתמש. נוסף על- כך, יש אפשרות מיוחדת לזהות ז'אנרים של שירים לפי המילים שלהם. בתוך הפרוייקט השתמשנו במספר סוגי פונקציות שונות לזיהוי קרבה בין טקסטים, ביניהם מרחק ג'קארד, ומרחק שיצרנו בעזרת קידוד האפמן. מרחק ג'קארד מתבסס על קרבה בין טקסטים במובן של מילים שלמות. עבור מציאת מדד ג'קארד בין שני טקסטים, יש לקחת את קבוצת המילים של כל טקסט, והמדד יהיה גודל חיתוך קבוצות אלו, חלקי גודל איחודן. עבור מרחק האפמן, ננסה לכווץ עם מפתח האפמן שהוצאנו ממחרוזת הקלט כל אחד מהטקסטים במאגר. ככל שהכיווץ שנעשה בעזרת המפתח שקיבלנו מהקלט יותר טוב, אזי הקלט יותר דומה לקורפוס, ומכאן נוכל למצוא איזה קורפוס הכי מתאים לקלט. השתמשנו גם בפונקציית מרחק שדומה לג'קארד, אך במקום לקחת מילים שלמות לקחנו את אוסף הזוגות של תווים עוקבים בטקסט, והסתכלנו על גודל החיתוך לעומת האיחוד. לבסוף לקחנו מרחק שנקבע לפי אורך המילה הממוצעת, שציפינו שהוא לא יצליח לזהות את מקור הטקסט, אלא כפונקציית בקרה.

תוכן עניינים

**מבוא .............................................................................. עמוד 3**

**ניבויים ............................................................................ עמוד 4**

**הסבר הממשק ................................................................. עמוד 4**

**שיטות השוואה .................................................................. עמוד 6**

**הסבר הפונקציות ............................................................... עמוד 10**

**שלב הניסוי וניתוח התוצאות .................................................... עמוד 14**

**דיון מסכם .............................................................................. עמוד 17**

**ביבליוגרפיה .......................................................................... עמוד 18**

מבוא

פעמים רבות במהלך חיינו נדרשים לזהות טקסטים דומים אחד לשני, ואף לנסות ולמצוא סוג, מקור או היבט מסוים של טקסט. גם המשטרה יכולה להשתמש בחיפוש דמיון בין טקסטים כדי לקשר בין מהלכים כלשהם בין עבריינים, לזהות קבצים ולסווג למי הם שייכים, ולהיסטוריונים, בזיהוי המקור או הזמן של הטקסט בו הוא נכתב כדי להעשיר את הידע של העולם המודרני אודות העבר.  
בפרויקט שלנו המטרה הייתה לבחון בין שיטות השוואה שונות של טקסטים, וכך לנסות לזהות את הקשרים של טקסטי הקלט הניתנים לתוכנה כ-STRING. השוואה היא אמצעי אמנותי ואף טכניקה להבנת טקסט:   
"השוואה היא בדיקת קווי דמיון וקווי שוני בין שני טקסטים, והסקת מסקנות - בעיקר מן ההבדלים.  
כלומר, ההשוואה שואלת מה דומה בין שני הטקסטים, מה שונה, ומה משמעות ההבדל. מה מוסיף טקסט אחד על חברו? ההשוואה בודקת רעיונות, מבנה, וגם סגנון וניסוח: ביטויים חוזרים, מילים מנחות, שורשים חוזרים וכדומה". ציטוט זה לקוח מהאינציקלופדיה מקרא גשר, ומתאר את ההשוואה בין טקסטים שונים. באופן ידני ע"י בני אדם אפשר לנתח את הרעיונות הדומים בין הטקסטים, המילים המנחות, להבין את הטקסט באופן עמוק ואת רעיונותיו, אך באופן ממוחשב לא ניתן לעשות זאת ברמה גבוהה, ולכן נותר רק להתייחס אל תוכנו של הטקסט, כגון מילים חוזרות, ביטויים חוזרים וכדומה.  
יתר על כן, אדם מאדם הבוחן את הטקסטים כדי לקשר ביניהם נדרשות מיומנויות שונות, והוא צריך להבין באופן עמוק את הטקסטים, ולא תמיד ניתן לעבור על כמות טקסטים גדולה ונרחבת, באופן שהמחשב יכול. לכן, בהנחה שהפרויקט עובד כראוי, נוכל לזהות מקורות של טקסטים באופן מהיר, ולסרוק מבנה נתונים גדול כדי למצוא את מקורו של הטקסט. בתוך העבודה שמנו דגש מיוחד למציאת הז'אנר של מילים לשירים באנגלית, זאת כיוון שיכולנו לקחת מדגם קטן יחסית ולאפיין אותו, ולא היינו נדרשים לשלם כסף כדי לקחת ממאגרי מידע רשמיים. כמו- כן, יכולנו לקחת מספר ז'אנרים מצומצם, וכך לבדוק האם באמת התכנית מצליחה להבדיל בין ז'אנרים שנחשבים בעינינו כיחסית דומים כגון ראפ ופופ, ולראות את היחס של ההצלחה היחסית בהבדלה בין ראפ לשירים נוצריים, שכמובן יש ביניהם "מרחק" גדול יותר, וכך היינו מצפים לאחוזי דיוק גדולים יותר בזיהוי. לכן בניתוח התוצאות נתמקד בזיהוי ז'אנרים של שירים מסויימים. בפרויקט יש לנו פונקציה, אשר מוסברת תחת הסעיף של הסברים לפונקציות, שמקבלת STRING ומנסה לזהות עם כל פונקציות ההשוואה השונות בין הטקסטים, וכך מוצאת את מקור הטקסט, וגם כותבת ל-OUTPUT את התוצאה לפי כל סוג השוואה. כך יכולנו לבחון את ההצלחות לפי סוגי ההשוואה השונים, ואת ההצלחה הכוללת של הניסוי. חשוב להדגיש כי לא עשינו שימוש בעבודה ברשת נוירונים, כיוון שרצינו לקבל רשימה ארוכה של הטקסטים הכי סבירים, כי ידענו מראש שאי - אפשר לצפות תמיד לקבל את התשובה הנכונה מתוך מאגר כה גדול. עם זאת, רשת הנוירונים הוא תחליף לא רע לפרוייקט, בהנחה שהיינו רק רוצים לקבל תשובה אחת.

ניבויים

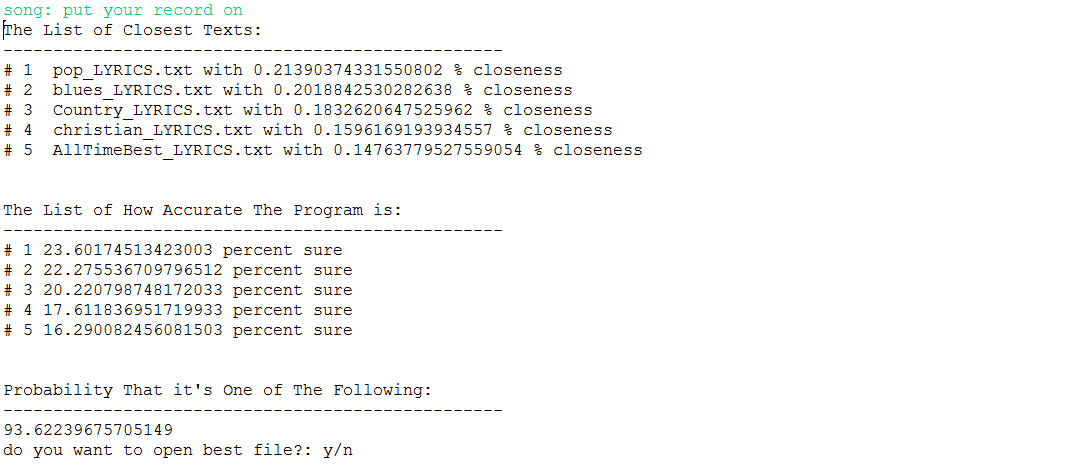
לדעתנו, התכנית אמורה להצליח לזהות את מקורם של הטקסטים באחוזים מפתיעים וגבוהים למדי. אנחנו כבני אדם מתקשים לראות בהבדלים קטנים בתוך הטקסטים, ואילו המחשב יכול לזהות הבדלים אלו, ולבחור מבין אוסף גדול של טקסטים דומים, בעזרת פונקציות המרחק, את הטקסט שהכי דומה, גם אם זה באחוז או שניים יותר טוב מכל השאר. לרוב, זה אמור להיות המצב, כיוון שרוב הטקסטים באנגלית, בכל זאת, מכילים מילים דומות ונפוצות. בעזרת התוכנה אנו אמורים להצליח לזהות את הז'אנר של שיר לפי המילים שלו. אנו מצפים לאחוזי הצלחה גבוהים מאוד, בעיקר עם פונקציית המרחק לפי ג'קארד, כיוון שזה לוקח בחשבון מילים חוזרות, וידוע שבתוך ז'אנרים ישנן לרוב אותן מילים פחות או יותר. רק לקחנו לתוך אוסף הז'אנרים 5 ז'אנרים שונים, ולכן אנו באמת מצפים שמתוך 5 ז'אנרים, התכנית תצליח לנחש את התשובה הנכונה. עם זאת, ידוע לנו שבזיהוי טקסטים אין ודאות לגבי ההצלחה בזיהוי המקור של טקסט, ובגלל הגבולות הדקים בין הז'אנרים, כגון פופ וראפ, אפשר לצפות שמידי פעם התכנית תתבלבל ביניהם.

הסבר הממשק

כאשר מריצים את הקובץ עולה הדבר הבא:



כלומר, אם רוצים לחפש סתם מחרוזת כלשהי ולנסות למצוא הקשר שלה למאגר שלנו, פשוט כותבים אותה. אם לא רוצים לחפש כלום, מקישים n ולוחצים אנטר. (יש לולאת while על החיפוש, שמופסק כאשר הוא מקבל את התו n)אם רוצים לחפש ז'אנר לפי מילים לשיר, תחילה כותבים  
"song: ", ואז מקלידים את המילים לשיר. לאחר החיפוש תוחזר הודעה מהצורה הבאה:



תחילה מודפסות התוצאות הכי טובות, לפי דירוג של 1 עד 5. (אפשר לשנות את מספר התשובות שמוחזרות, אך ה-default הוא 5)- ראה שורה 12 בתכנית "find\_source\_for\_string(string, 5)", אפשר פשוט לשנות את המספר למה שרוצים, או להפעיל את הפונקציה בעזרת ה-SHELL.   
גם מודפס המדד דמיון של ג'קארד, לדוג' עבור pop יש 0.21 אחוזי התאמה.

לאחר מכן מודפס הביטחון היחסי של התוכנה בכל אחת מהתוצאות, כאן לדוגמה הוא 23% בטוח בפופ מבין החמישה. הצלחנו להסיק את הביטחון היחסי בעזרת מניפולציה מתמטית פשוטה על אחוזי הדמיון של טקסט הקלט לז'אנרים השונים.

לבסוף מודפס הסיכוי המשוער שהקלט אכן מתאים לאחת התוצאות המודפסות, לדוגמה כאן התוכנה בטוחה ב-93% שהז'אנר של השיר הוא פופ, בלוז, קאנטרי, נוצרי או להיט גדול.

כעת התוכנה מציעה למשתמש לפתוח את קובץ ה-txt שמתאים ביותר לקלט. זה שימושי לדוגמה כאשר מחפשים מידע על דברים שונים, ורוצים לא רק לקבל את המקור של הקלט, אלא גם לקרוא עליו. בשביל לפתוח את הקובץ לוחצים על y ואז אנטר, ואחרת n ואז אנטר, כדי להתחיל חיפוש מחדש.

הערה: ממש בתחתית הקובץ project, שורה אחת לפני האחרונה הוצאה כ-comment. אם המשתמש רוצה קודם כל לחפש ז'אנר של שיר בעזרת סוגי מדדים שונים לדמיון בין טקסטים, הוא יכול להחזיר את השורה לתכנית, ולהחליף את הארגומנט שבפנים למילים לשיר כרצונו. (אפשר פשוט להריץ את הפונ"ק**testLyricsWithDifferentFunctions**(string) דרך ה-SHELL).  
  
**מקור הקורפוסים**- מרבית הקורפוסים שלנו לקוחים מתוך ערך הויקיפדיה שלהם. עבור הז'אנרים השונים, לקחנו ממאגר של מילים לשירים באינטרנט שירים רבים מאותו הז'אנר, ואת כולם העתקנו לקובץ אחד מהצורה <GENRE>LYRICS. כך כל קובץ TXT מייצג ז'אנר, בהכלתו הרבה שירים שונים מאותו הז'אנר.

שיטות ההשוואה

**שיטת ג'קארד למציאת קשר בין טקסטים:**

בשיטה זו מחפשים את הקשר בין שני טקסטים על- ידי מציאת מספר המילים המשותפות שלהם חלקי מספר המילים הכולל בין שני טקסטים. כך, ככל ששני טקסטים יהיו יותר דומים אחד לשני, החיתוך של קבוצות המילים יחסית לסך המילים הכולל גדול יותר, וכך המדד לקשר ג'קארד יגדל לכיוון ה-1. ברור כי יש חסם למדד שהוא קטן שווה ל-1, כיוון שחיתוך שתי קבוצות תמיד קטן שווה בגודלו לאיחוד. לכן אם נקבל קובץ קלט, ונמצא את המרחק ג'קארד שלו עם כל קבצי הקורפוס, אזי הקובץ שהכי מתאים לקלט הוא זה עם מדד ג'קארד הכי גדול עם הקלט.  
יש לשים לב שהשתמשנו גם כן בשיטה שטוענים שהיא יותר טובה משיטת ג'קארד, בו השתמשנו במקדם סורנסון- דיס. לכן נבדוק את התוצאות, ונראה האם כך הדבר.

 J(A,B) = {{|A \cap B|}\over{|A \cup B|}}.

**שיטת אורך המילה הממוצעת למציאת קשר בין טקסטים:**

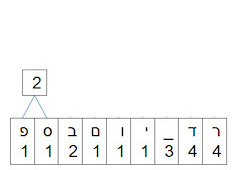
זוהי שיטה שלקחנו, שיטת בקרה. לא באמת ציפינו שהיא תביא לתוצאות נכונות, אלא זאת בעיקר כדי להשוות עם השיטות השונות למציאת קשר בין הטקסטים. כל מה שעשינו בשיטה זו היא לקחת את אוסף המילים, לעבור עליה ולמצוא את אורך המילה הממוצעת. לזה מצאנו את השגיאה היחסית בין שני הטקסטים. ברור כי בשגאה יחסית אנו מקבלים מספרים בין 0 ל-1, וככל שהקשר יותר גבוה בין הטקסטים, כך השגיאה יותר קטנה, ולכן אחד פחות זה יביא למספר שיותר קרוב לאחד, כמבוקש ממדד לדמיון בין טקסטים.

**שיטת כיווץ האפמן למציאת קשר בין טקסטים:**

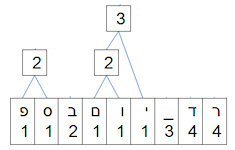
באופן כללי, כל תו טקסט נשמר במחשב, לרוב, בבייט יחיד (8 ביטים).  
בייט יכול לשמור כל אחד מ-256 תווים שונים.

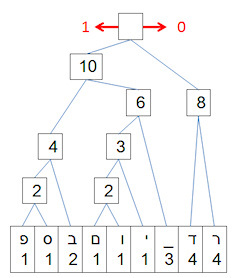
אין באמת צורך בכזו כמות:  
לדוגמה בעברית יש 22 אותיות + 5 סופיות. אליהן נוסיף עשר ספרות ועוד כמה סימני פיסוק. גם אם נתאמץ מאד לא נצליח, לרוב, למצוא יותר מ-64 תווים. כדי לשמור תו אחד מתוך 64 מספיקים 6 ביטים, כלומר ששניים מתוך שמונת הביטים של הבייט ה"רגיל" מיותרים. כבר הצלחנו לכווץ 25%.

בטקסטים רגילים וארוכים מספיק, האותיות השונות מופיעות בשכיחות שונה. לדוג': אותיות אהו"י נפוצות מאד בעברית, ואילו אותיות כמו ט' או צ' נדירות יחסית.  
.MITהאלגוריתם הומצא ב-1952 ע"י האפמן באוניברסיטה   
בשלב הראשון, האלגוריתם של האפמן יוצר טבלת הסתברויות של האותיות השונות. הסתברות של אות מחושבת כמספר הפעמים שהיא מופיעה חלקי מספר האותיות הכולל בטקסט.

נסביר בעזרת דוגמה כיצד עובדת השיטה:  
ברד ירד בדרום ספרד  
נספור כמה פעמים מופיע כל תו:  
ב: 2  
ר: 4  
ד: 4  
\_: 3  
י, ו, ם, ס, פ: 1  
  
עוברים על הרשימה, בוחרים את זוג האותיות שסכום ההסתברויות שלהן הוא הנמוך ביותר, ויוצרים צומת מעליהן, כפי שמודגם בתמונה משמאל.

 כעת, חוזרים על התהליך – רק שהפעם מתייחסים לצומת שנוצר בתור יחידה חדשה עם הסתברות משלה (סכום ההסתברויות של האותיות שמתחתיו), ומתעלמים מהאותיות עצמן. הביטו, כפי שאפשר לראות בתמונה משמאל.



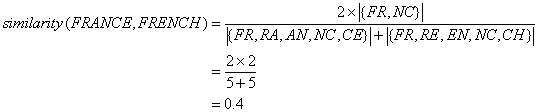
ממשיכים באותו אופן שוב ושוב, עד שכל היחידות מסודרות בתוך מין "עץ". נגדיר גם ש-0 הוא כל ענף ימני (ביחס לצומת ממנו הוא יוצא) ו-1 הוא כל ענף שמאלי.  
  
כדי להגיע לאות מסוימת בעץ שנוצר, מתחילים בצומת העליון ועוברים ימינה או שמאלה לפי הצורך, תוך צבירה של 0 ו-1 לאורך הנתיב. למשל, כדי להגיע לאות י', צריך ללכת שמאלה (1), ימינה (0), שמאלה (1) ושוב ימינה (0). כלומר הקוד של י' הוא 1010. הקוד של ד' הנפוצה, לעומת זאת, קצר הרבה יותר: ימינה ואז שמאלה, כלומר 01.

לסיום, נתרגם את המחרוזת כולה לקודי הופמן המתאימים:

Article_Paragraph_Thumb_1817671  
המחרוזת החדשה היא בת 53 תווים, אך אלה הם רק אפסים ואחדים, ואפשר להמיר אותם לבייטים – ואז המחרוזת הזאת תתפוס 7 בייטים בסך הכל, לעומת 18 במחרוזת המקורית. כמובן שצריך לאחסן איפשהו גם את המבנה של העץ עצמו, כך שבסופו של דבר הקוד אולי לא יהיה חסכוני עבור המחרוזת הספציפית הזאת, אבל כשעוברים לטקסטים ארוכים יותר הכיווץ יכול להיות משמעותי מאד.

איך השיטה קשורה למציאת טקסטים?  
התוכנה מנסה לכווץ את הטקסט בהינתן מחרוזת הקלט.  
לפי השיטה שהוסברה לעיל, נשים לב שככל שהמפתח לכיווץ טקסט מסויים יותר מתאים לטקסט עצמו, אז מתקבלים אחוזי כיווץ טובים יותר. למעשה, אחוזי הכיווץ הכי טובים מתקבלים ממפתח שמיוצר ע"י הטקסט עצמו. בתוכנה, כדי למצוא איזה קורפוס מתאים לקלט, עברנו על כל אחד מהם, וניסינו לכווץ אותו בעזרת מפתח שמתקבל ממחרוזת הקלט. לכן, כשהקורפוס ומחרוזת הקלט דומים, ואותיות יחזרו על עצמן יותר, נקבל ממחרוזת הקלט מפתח שהוא יותר דומה למפתח שהיינו מקבלים מהקורפוס עצמו, ולכן אמורים לקבל אחוזי כיווץ גבוהים יותר. לדוגמא, אם ננסה לכווץ את טקסט הערך בויקיפדיה "צרפת" לפי טקסט הערך בויקיפדיה "מהפכה" סביר להניח שיהיו יותר אותיות ומילים דומות מאשר אם נכווץ את הערך "ספגטי" לפיו. לכן נקבל אחוזי כיווץ שונים בשני המקרים, והקובץ שמכווץ "יותר טוב" (שוקל פחות) הוא הדומה יותר לטקסט הקלט. כלומר, בדוגמא לעיל נקבל את התוצאה צרפת ולא ספגטי. כיווץ האפמן נראה לנו רעיון טוב מאוד להשוואה בין טקסטים, כיוון שהוא מתחשב בטקסט כולו, וביחס של כל האותיות שמופיעות בו. יש לשיטת השוואה זו פוטנציאל גבוה מאוד בזיהוי שפות שונות, מעצם לקיחתו בחשבון חזרות ומופעים של אותיות שונות בטקסט. למען הגילוי הנאות, רצינו לקחת רק את שיטת השוואה זו לצורך הפרוייקט, וחשבנו בהתחלה שהיא תעבוד, מתוך השימוש העמוק שלה בקשר האמיתי בין שני טקסטים, אך נוכחנו לראות שאחוזי הגילוי שלה נמוכים מעט משציפינו, וניסינו לראות כיצד לשפר את הזיהוי בדרכי השוואה שונים.

שימוש בשיטת סורנסון - דיס לצורך מציאת קשר בין טקסטים:

השיטה משווה בין צמדי אותיות של שני הStrings הנתונים. בצורה דיי פשוטה, השיטה מציגה את הקירבה של שני Strings באחוזים ע"פ האלגוריתם הבא (המיוצג בהשוואה בין 'FRANCE' ו 'FRENCH'):  
  


בפרוייקט שלנו השתמשו השיטה הנ"ל לצורך השוואה בין שני טקסטים. לקחנו את כל צמדי המילים משני הטקסטים וע"י חיתוך המידע מצאנו את הצמדים הדומים בשניהם, ולפיהם מצאנו את הקירבה בין הטקסטים.

מדוע אנו יוצאים מנקודת הנחה שדמיון בין טקסטים מצביע על דמיון בין ז'אנרים?

ובכן, אם נבחן את מגוון המילים המצויות לדוגמה במוסיקה השחורה, נמצא שבאחוז גבוה מאוד מהשירים, בייחוד המודרניים, ישנן מילים המצביעות על אורח חייו של האמן (בעיקר 'סלנג'):  
כל המילים המצביעות על ביתו של האמן (לדוגמה crib), על עושרו של האמן (לדוגמה dough), על בת זוגו של האמן (נמנע מלהדגים מטעמי צנזורה) וכן הלאה. כאשר אם נבחן את מגוון המילים המצויות במוזיקה קאונטרי, לא נופתע לגלות שאין כמעט בכלל מילים מהשדה הסמנטי של "I'm cool", אין קללות, וכו'.

השיטה אמורה להיות יותר מדוייקת משתי סיבות עיקריות:  
- היא לא מבחינה בין מילים, אלא בין מה שמרכיבים את המילים. לכן, אין חשיבות לאות המתחילה את המילה או לשינוי סדר המילים.

- השיטה משתמשת בהשוואה בין צמדי אותיות ולא בין מילים, ולכן היא תמצא מילים כמו French Republic ו Republic of Cuba דומה פחות מל- Republic of France מאשר בשיטות האחרות שימצאו אותן שוות באותה מידה בגלל המילה היחידה שמשותפת, ולא יתייחסו לצמדי האותיות המשותפות לFrance ולFrence.

בפרוייקט השתמשנו במספר וריאציות כמעט שקולות לשיטת סורנסון- דיס, בו במקום זוגות לקחנו שלשות ורביעיות.

הסבר הפונקציות ומבנה התוכנה

**הסבר על הקבצים**

הקובץ corpusManagement נועד לטיפול בקבצי הקורפוס. יש בו מתודה לקבל כל הקורפוסים, או רק את הקורפוסים עבור ז'אנרים של מוזיקה. יש בו גם מתודה שבודקת את תקינות קבצי הקורפוס.  
הקובץ DistanceFuncs מכיל את כל פונקציות המרחק בין שני טקסטים, דוגמת ג'קארד.  
הקובץ project זהו הקובץ הראשי של הפרוייקט. הוא מכיל את ה-"main", כלומר הרצת התוכנה מתבצעת דרכו.  
הקובץ Huffman\_Base מכיל את המתודות השונות עבור כיווץ האפמן, וקבלת מדד לדמיון בין טקסטים לפי כיווץ האפמן.

**הפונקציות הראשיות**  
main

הפונקציה הראשית של הפרויקט. קודם מתקנת את כל קבצי הקורפוס למקרה שהם פגומים, ולאחר מכן מריצה בלולאת WHILE את החיפושים ואת הקלטים ע"י המשתמש, המוסברים בפונ"ק find\_source\_for\_string. כל פעם מקבלים קלט מהמשתמש. אם הוא 'n' אז עוצרים בתכנית, ואחרת ממשיכים עם החיפוש. הקוד די STRAIGHT FORWARD.

find\_source\_for\_string

מקבלת את string שמכיל בתוכו את המחרוזת לחיפוש ואת מספר התוצאות שבאופן דיפולטי הוא 5.

הפונקציה נעזרת בשתי הפונקציות getSearchType וב- getRelevantCorpusFiles כדי לקבל את סוג החיפוש ואת קבצי הקורפוסים המתאימים, בהתאמה.

לאחר מכן הפונקציה משתמשת בפונקציה analyizeSimilarities כדי לקבל את התוצאות המתאימות.

לבסוף, אם אין תוצאה מתאימה היא תחזיר הודעה בהתאם ואם יש תוצאה מתאימה אז בעזרת הפונקציה printResults היא תדפיס את התשובות שקיבלנו מהחיפוש. לבסוף, מציעים למשתמש לפתוח את הקובץ שנמצא הכי דומה לקלט. גם כאן אין קוד מסובך, והכל יחסית פשוט. רק נשים לב ש- closeness\_list הוא רשימה שתכיל זוגות סדורים של שם הקורפוס ושל הדמיון בין הקלט לקורפוס, וש- best\_results יכיל רק את התת- רשימה עם התוצאות הכי טובות. הערה: closeness זה מדד לקרבה בין שני הטקסטים.

**אופן הפעולה:**

getSearchType

מקבלת את הסטרינג עם מילות החיפוש. התוכנה תומכת בחיפוש כללי או בחיפוש ז'אנר של שיר ולכן היא מתאימה את עצמה לצורך חיפוש כזה. כפי שניתן לראות בחלק ההסבר של המנשק, כאשר מתחילים את החיפוש עם "song: ", אז נרצה לקבל תוצאות לז'אנרים של מוסיקה. לכן פשוט סורקים את התוים הראשונים, ובודקים אם הם המחרוזת לעיל. עם כן, מחזירים "song", אחרת מחזירים "".

top\_n\_results

פונקציה שמחזירה את n התוצאות הטובות ביותר מבחינת דמיון לקלט. הפונ"ק מקבלת רשימה, ואמורה להחזיר את ה-n תוצאות הטובות, כלומר את n הרשימות באורך 2, שהאיבר באינדקס 1 שלהם הכי גדול (זהו אינדקס הדמיון), לכן מסדרים את הרשימה בעזרת SORT עם KEY שהגדרנו אותו להיות מינוס האינדקס הראשון, כדי לקבל סידור מהגדול לקטן. כעת פשוט מחזירים את n האיברים הראשונים של הרשימה.

getRelevantCorpusFiles

פונקציה המקבלת את סוג החיפוש ומחזירה את הקורפוסים המתאימים לאותו סוג חיפוש. זאת בעזרת הקובץ של corpusManagement, שמכיל שתי פונקציות שאחת רק מחזירה את הז'אנרים של שירים, והשנייה את כל קבצי הקורפוס.

promptFileOpening

פונקציה שמטרתה לפתוח את התוצאה הטובה ביותר בתוכנת הnotepad. פונקציה זו שואלת את המשתמש אם ברצונו לפתוח את הקובץ ומבצעת את הפעולה על פי רצון המשתמש.

printResults

היא פונקציה המקבלת את מספר התוצאות, את הרשימה של הדומים לקלט ואת הרשימה של התוצאות הטובות ביותר. הפונקציה נעזרת בשלוש הפונקציות הבאות כדי להדפיס את המידע הרלוונטי:

printClosestTexts, printProgramAccuracy, printTopResultsTotalProbability.

printClosestTexts

פונקציה המקבלת את הרשימה של התוצאות הטובות ביותר ומדפיסה אותן יחד עם הקירבה שלהן למילות החיפוש באחוזים. קודם מדפיסים כותרת, ואז את התוצאות עצמן בעזרת לולאת FOR על best\_results, שממוינת מדמיון גבוה לנמוך. בכל איבר, מדפיסים עם פונקציית ה-format של מחרוזת, את הדירוג, אז את השם של הקובץ (זהו הרכיב האחרון במסלול המלא לקובץ, ולכן נעזרנו בספליט), ולבסוף את הדמיון כפול 100, כדי להחזיר את התשובה באחוזים.

printProgram/accuracy

פונקציה שלוקחת את מספר התוצאות הרצויות ורשימה של התוצאות הטובות ביותר ומדפיסה באחוזים מה ההסתברות שכל אחת מהתוצאות נכונה. נעזרים ב-confindence\_percentage, שמוסבר מלרע, כדי לקבל את ההסתברויות הנ"ל.

printTopResultsTotalProbability

פונקציה המקבלת את מספר התוצאות הרצויות ורשימה של הדמומים, ומדפיסה את ההסתברות שקיימת תוצאה נכונה בפלט. ההסתברות היא כפי שחישבנו, סכומי הדמיון הכולל של התוצאות הגבוהות, חלקי הדמיון הכולל של כל הקורפוסים, גם אלו של מצאנו מתאימים. בסוף מכפילים פי 100, בשביל תוצאה באחוזים.

analyzeSimilarities

פונקציה שמחזירה את הרשימה של כל הזוגות של קורפוס ודמיון, וגם את התוצאות הטובות ביותר. מה שעושים, זה לוקחים את הקלט, ויוצרים רשימה words של כל המילים בקלט. כעת עוברים על כל קבצי הקורפוס, קוראים את תוכנם למחרוזת, גם אותה מפרקים למילים, ואז מוסיפים לרשימה של הדמיונות את שם הקורפוס, ואת המרחק ביניהם עם פונקציית ההשוואה המתאימה. בסוף לוקחים את התוצאות הטובות ביותר, בעזרת הפונקציה top\_n\_results, ומחזירים את שתי הרשימות.

find\_source\_for\_file

מחזירה את המקור עבור קובץ. למעשה היא פשוט פותחת אותו, קוראת את תוכנו, ומריצה בעזרת התוכן את הפונ"ק find\_source\_for\_string. פונ"ק פשוטה, אך חוסכת הרבה זמן בניתוח ובניסוי, ומאפשרת שימוש יותר נרחב בתוכנה.

confidence\_percentage

מקבלת את רשימת התוצאות, שממוינת בסדר יורד לפי הדמיון, ומחזירה רשימה שמכילה את ההסברויות היחסיות של כל אחד מהקבצים. למעשה, כל מה שעושים זה סוכמים את כל המדדים לדמיון, ובודקים את החלק היחסי של כל אחד מהדמיון של כל קובץ קורפוס לסכום הכולל של הדמיונות. לבסוף, מכפילים פי 100 כדי לקבל את התשובה באחוזים.

isThereAMatch

מחזירה האם יש התאמה. עוברים על רשימת הצמדים של קורפוס ודמיון, ובודקים האם יש דמיון חיובי של הקלט לקורפוס כלשהו מהמאגר. אם אין, אז מחזירים false, ואחרת מחזירים true.

return\_lyrics\_genre  
מחזירה את הז'אנר של שיר כלשהו בהינתן קלט של STRING, ופונקציית השוואה, כאשר הדיפולטי הוא JACCARD, שראינו בבדיקות שהוא עבד הכי טוב. זה פחות או יותר הפונ"ק find\_source\_for\_string, אלא במקום להחזיר את הקובץ של הז'אנר המתאים, מהצורה pop\_LYRICS, אז פשוט החזרנו את השם של הז'אנר עצמו, בלי התוספת של \_LYRICS, בעזרת פונ"ק ספליט של מחרוזת.

testLyricsWithDifferentFunctions

מבצע חיפוש של ז'אנר למילים של שיר שהוא מקבל כקלט. כל פעם מדפיסים עם איזו פונקציית השוואה בדקנו את הדמיון לכל קורפוס, ואז מדפיסים את הז'אנר הכי מתאים שקיבלנו בעזרת אותה פונקציית השוואה, וזאת בעזרת הפונ"ק return\_lyrics\_genre, שגם מקבלת פונקציית השוואה. זו הפונקציה בה השתמשנו כדי לקבל תוצאות ולעבד אותן עבור שלב הניסוי והבדיקות.

**פונקציות ההשוואה**

נשים לב תחילה שהקלט הוא שני LISTS של מילים, כאשר כל LIST מקובץ אחר.

findJaccardDistance(wordList1,wordList2)

צריך להחזיר את גודל החיתוך של המילים המשותפות חלקי גודל האיחוד של המילים המשותפות. לשם כך מבצעים המרה של הליסטים לקבוצות SETS, ומבצעים עליהן פעולות &,| ומחזירים את הגודל.

findJaccardWithRepetitionsDistance(wordList1,wordList2)

כאן משתמשים בשיטת ג'קארד, אלא שבמקום קבוצות משתמשים במולטי- קבוצות, כלומר לוקחים בחשבון גם את הכפילויות של המילים, ולא רק אם הן מופיעות במשותף בטקסטים. שיטה זו טכנית אמורה לעבוד מעט יותר טוב מהקודמת מהסיבות הברורות לעיל.

לשם כך עושים בדיוק את מה שעשינו לעיל, אלא משתמשים במבנה הנתונים COUNTER במקום SET, שמאפשר כפילויות.

findPairsDistance(wordList1, wordList2)

הפונ"ק מחזירה את כמות הרצפים באורך 2 המשותפים לטקסטים, חלקי הכמות הכוללת. לשם כך נעבר מהליסטים לסטרינגים בעזרת פונ"ק string.join(list). כעת, בונים ליסטים חדשים בשיטת LIST COMPREHENSION ע"י ריצה על הסטרינגים ולקיחת זוגות של תוים. מכאן ממשיכים באופן דומה לפונ"ק findJaccardDistance...

findTripletsDistance(wordList1, wordList2)

באופן דומה לזוגות, אלא עם שינוי קטן באינדקסים ופונקציות כדי לקבל שלשות במקום זוגות. כך גם findQuadrupletsDistance(wordList1, wordList2).

findHuffmanDistance(wordList1, wordList2)

מוצא את מרחק ההאפמן שהגדרנו בהנתן שני הליסטים. גם כאן נעבור לסטרינג של שני הקלטים, וניעזר בפונ"ק compression\_ratio של קובץ ההאפמן, שמחזיר כמה מהווה הסטרינג המקווץ בעזרת הסטרינג השני, יחסית למה שהיה קודם. את זה בדיוק נחזיר.

findAverageWordLengthDistance(wordList1, wordList2)

מחזיר אחד פחות השגיאה היחסית בין האורך הממוצע. לשם כך לוקחים את כל המילים, משרשרים אותם בלי רווח, ובודקים את האורך הכולל חלקי האורך של ליסט המילים. כך מקבלים את האורך הכולל חלקי אורך המילים, כלומר אורך המילה הממוצעת. לאחר מכן פשוט מחזירים את אחד פחות השגיאה היחסית.

שלב הניסוי וניתוח התוצאות

>>> testLyricsWithDifferentFunctions(open("c:/songs/**Christian**1.txt").read())

findJaccardDistance returns: christian

findJaccardWithRepetitionsDistance returns: christian

findPairsDistance returns: christian

findTripletsDistance returns: christian

findQuadrupletsDistance returns: Country

findHuffmanDistance returns: AllTimeBest

findAverageWordLengthDistance returns: blues

>>> testLyricsWithDifferentFunctions(open("c:/songs/**Christian**2.txt").read())

findJaccardDistance returns: christian

findJaccardWithRepetitionsDistance returns: christian

findPairsDistance returns: Country

findTripletsDistance returns: Country

findQuadrupletsDistance returns: Country

findHuffmanDistance returns: AllTimeBest

findAverageWordLengthDistance returns: blues

>>> testLyricsWithDifferentFunctions(open("c:/songs/**Christian**3.txt").read())

findJaccardDistance returns: christian

findJaccardWithRepetitionsDistance returns: christian

findPairsDistance returns: rap

findTripletsDistance returns: rap

findQuadrupletsDistance returns: rap

findHuffmanDistance returns: blues

findAverageWordLengthDistance returns: blues

>>> testLyricsWithDifferentFunctions(open("c:/songs/**Country**1.txt").read())

findJaccardDistance returns: Country

findJaccardWithRepetitionsDistance returns: rap

findPairsDistance returns: Country

findTripletsDistance returns: Country

findQuadrupletsDistance returns: Country

findHuffmanDistance returns: Country

findAverageWordLengthDistance returns: blues

>>> testLyricsWithDifferentFunctions(open("c:/songs/**Country**2.txt").read())

findJaccardDistance returns: Country

findJaccardWithRepetitionsDistance returns: Country

findPairsDistance returns: Country

findTripletsDistance returns: Country

findQuadrupletsDistance returns: Country

findHuffmanDistance returns: AllTimeBest

findAverageWordLengthDistance returns: blues

>>> testLyricsWithDifferentFunctions(open("c:/songs/**Country**3.txt").read())

findJaccardDistance returns: Country

findJaccardWithRepetitionsDistance returns: Country

findPairsDistance returns: Country

findTripletsDistance returns: Country

findQuadrupletsDistance returns: Country

findHuffmanDistance returns: christian

findAverageWordLengthDistance returns: blues

>>> testLyricsWithDifferentFunctions(open("c:/songs/**Rap**1.txt").read())

findJaccardDistance returns: blues

findJaccardWithRepetitionsDistance returns: rap

findPairsDistance returns: rap

findTripletsDistance returns: rap

findQuadrupletsDistance returns: rap

findHuffmanDistance returns: christian

findAverageWordLengthDistance returns: blues

>>> testLyricsWithDifferentFunctions(open("c:/songs/**Rap**2.txt").read())

findJaccardDistance returns: blues

findJaccardWithRepetitionsDistance returns: rap

findPairsDistance returns: rap

findTripletsDistance returns: Country

findQuadrupletsDistance returns: Country

findHuffmanDistance returns: rap

findAverageWordLengthDistance returns: blues

>>> testLyricsWithDifferentFunctions(open("c:/songs/**Rap**3.txt").read())

findJaccardDistance returns: Country

findJaccardWithRepetitionsDistance returns: rap

findPairsDistance returns: Country

findTripletsDistance returns: Country

findQuadrupletsDistance returns: Country

findHuffmanDistance returns: AllTimeBest

findAverageWordLengthDistance returns: blues

>>> testLyricsWithDifferentFunctions(open("c:/songs/**Rock**1.txt").read())

findJaccardDistance returns: pop

findJaccardWithRepetitionsDistance returns: rap

findPairsDistance returns: pop

findTripletsDistance returns: pop

findQuadrupletsDistance returns: pop

findHuffmanDistance returns: Country

findAverageWordLengthDistance returns: blues

>>> testLyricsWithDifferentFunctions(open("c:/songs/**Rock**2.txt").read())

findJaccardDistance returns: christian

findJaccardWithRepetitionsDistance returns: rap

findPairsDistance returns: christian

findTripletsDistance returns: christian

findQuadrupletsDistance returns: christian

findHuffmanDistance returns: AllTimeBest

findAverageWordLengthDistance returns: blues

>>> testLyricsWithDifferentFunctions(open("c:/songs/**Rock**3.txt").read())

findJaccardDistance returns: christian

findJaccardWithRepetitionsDistance returns: rap

findPairsDistance returns: Country

findTripletsDistance returns: Country

findQuadrupletsDistance returns: Country

findHuffmanDistance returns: AllTimeBest

findAverageWordLengthDistance returns: blues

**ניתוח תוצאות:**

כפי שניתן לראות, הפונ"ק findAverageWordLengthDistance לא עוזרת בכלל. היא תמיד מחזירה blues, לא משנה מהו סוג הקלט. זאת כנראה כיוון שבקבצי המדגם שלנו של ז'אנרים אחרים היו מספר שירים עם מילים הרבה יותר מידי ארוכות או קצרות, והדבר שינה את אורך המילה הממוצעת, מה שמשאיר אותנו עם הכי קרוב לאמיתי- blues, ולכן כנראה תמיד מוחזר blues. הערה: איכסנו את הקורפוסים לז'אנרים השונים בתוך קובץ txt אחד, ולכן ייתכן מאוד שכאשר אנו עושים parse בעזרת הפונקציה של פייתון, אז הוא עושה את ה-parse, ומחזיר גם שורות ריקות שהשארנו בין שיר אחד לשני, כדי להבדיל ביניהם, ולכן אורך המילה בשורות אלו יהיה 0- "". כנראה שבקובץ של blues השארנו הכי פחות רווחים כאלו, ולכן ממוצע אורכי המילים עוות הכי קצת בקובץ זה. בכל מקרה, הפונקציה לא נועדה לעזור מראש לזיהוי, אלא פשוט כפונקצית בקרה.

הפונ"ק findHuddmanDistance מחזירה לנו כמעט תמיד AllTimeBest, כלומר אוסף של השירים הכי טובים מכמה העשורים האחרונים. זה אינו מפתיע, כיוון ששם יש לנו מאגר מכל הז'אנרים השונים, והוא באמת עושה הערכה טובה לשפה האנגלית, בניגוד אולי לראפ, בו לפי הבדיקות שלנו, בממוצע המילים יותר קצרות, ופחות משקפות את השפה האנגלית. למעשה, קידוד האפמן ידוע בתור המפתח האוניברסלי לשפה האנגלית, ולכן דרך AllTimeBest הצלחנו לקבל פחות או יותר את המפתח הידוע, וזה עזר לנו לכווץ כל טקסט, בלי קשר לז'אנר של השיר.

הפונ"ק findPairs/Triplets/QuadrupletsDistance פחות או יותר הביאו לאותן תוצאות בכל המדידות, עם נטייה גדולה מאוד ל-Country.

כעת לפונקציות הכי מוצלחות, אלו שגם השתמשנו בהן בסופו של דבר למוצא המקורות שלנו הן אלו של ג'קארד. ברוב המוחץ של הפעמים השגנו לפחות מאחד מהם את הז'אנר הנכון, לרוב שניהם הסכימו, והסכימו על התשובה הנכונה. גם כאשר "הטעינו" את התוכנה לכאורה, עם ז'אנר שהוא אינו מכיר, Rock, אז קיבלנו משתי הפונקציות תמונה יפה למדי שמשקפת את הרוק- נוצרי, ראפ, פופ. שאכן, דומה מאוד לשלושה, וכמצופה, הוא אינו יותר מידי דומה למוזיקת קאנטרי. זאת עם מעט הסתייגות למוזיקה נוצרית, אם כי ניתן להסביר זאת בכך שאחד מקבצי הקלט שלנו היה בעל מילים בעלות הקשר דתי שחזרו רבות במהלך השיר, כגון "father, angels, god". לסיכום, אלו הפונקציות הכי מוצלחות שלנו, שעובדות כמו שצריך, ומבצעות את העבודה. הן פחות או יותר אותו דבר, רק שהפונקציה התחתונה יותר גם לוקחת בחשבון חזרות של המילים, ולא רק אם הן נמצאות בטקסט, ע"י שימוש במבנה נתונים שמממש מולטי- קבוצה. הופתענו מאוד מהפשטות הגדולה של מדד הג'קארד, ומאחוזי ההצלחה הגדולים שלו. מדד זה היווה ללא ספק, ההצלחה הגדולה שאיפשרה לפרוייקט לעבוד כמו שצריך, ולזהות כמעט תמיד את הז'אנר הנכון.

דיון מסכם

הפרוייקט שלנו סבב סביב מציאת קשר בין שני טקסטים. לצורך ביצוע פעולה זו השתמשנו בשלוש שיטות עיקריות:  
- שיטת האופמן.  
- שיטת ג'קארד.  
- שיטת סורנסון - דיס.  
  
ניתן להווכח ששיטת ג'קארד היא השיטה שבה הצלחנו להגיע לתוצאות הטובות ביותר, ולא שיטת סורנסון - דיס כפי שחשבנו לפני הפרוייקט.  
שיטת סורנסון - דיס טובה כאשר מדובר בהשוואה בין מילים בודדות, אך כאשר מדובר בשימוש מול טקסט רב מילים היא עלולה להחזיר תוצאות שגויות בשל העובדה שההסתברות לשוויון בין צמדי אותיות גדולה יותר ככל שמדובר בטקסט גדול יותר שעליו מתבצעת ההשוואה, אבל לאו דווקא יהיה מדובר במילה דומה לקלט לתוכנה.

**שימושים בעולם האמיתי**

המטרה בפרוייקטים של מ"ח בכללי היא גם מציאת שימוש בעולם האמיתי, וזו גם אחת המטרות שהצבנו לפנינו כאשר התחלנו את הפרוייקט.

הפרוייקט שלנו סבב סביב השוואה בין טקסטים ומציאת טקסט מתאים למילות חיפוש, ואנו סבורים שיש לו גם שימושים רבים בעולם האמיתי.  
  
ראשית, מנועי חיפוש, כדוגמת Google, משתמשים באלגוריתם המבוסס על שיטות ההשוואה שבהן השתמשנו בפרוייקט. הפרוייקט שלנו מדמה מנוע חיפוש בסיסי, ומאפשר דברים מעט יותר מורכבים כגון זיהוי ז'אנרים.  
  
שנית, שימוש בהשוואת טקסטים יכול לשמש הסטוריונים המחפשים את שנת הכתיבה של טקסטים מסויימים, ואת מקורו של הטקסט. ניתן להצליב מידע מהפורייקט שלנו עם מאגר מידע של שימוש במילים בטקסטים שנכתבו במשך ההסטוריה.  
  
לבסוף, השוואת טקסטים וחיפוש לפי מילות חיפוש יכול לשמש מוסדות המעוניינים לקטלג את המידע שלהם, כמו למשל ספריות, בתי ספר, בתי חולים, ועוד... וזאת באופן דומה למה שעשינו עם מציאת ז'אנרים לשירים. כל שיש לעשות הוא לבחור מספר דוגמאות התחלתיות ולקטלג אותן, ולאחר מכן להכניס אחד אחד את המוצרים לתוך הקטלוג המשוער שלהם, ותוך כדי כך גם להגדיל את גודל מאגר הנתונים לכל ז'אנר, ולשפר את הסיכוי למיון מוצלח.

אנו רואים פוטנציאל גדול מאוד בפרוייקט, בין היתר כיוון שלא הצלחנו למצוא באינטרנט אתר שיוכל לזהות ז'אנר של שיר מתוך מילותיו או בכל דרך זיהוי אחרת, אלא רק בחיפוש השיר במאגר הנתונים העצום שלו. עם זאת, אנו מודעים לכך שההצלחה במיון היא הסתברותית, ושלא תמיד אמורה לעבוד. לדוג' עבור רימיקסים של שירי קאנטרי או רוק משנות ה-90, אין אנו אמורים לזהות שזה רימיקס, כיוון השיר עצמו מבוצע בז'אנר אחר עם אותן מילים.

ביבליוגרפיה

אתר הקורס מבוא מורחב למדעי המחשב באוניברסיטת ת"א:

<http://tau-cs1001-py.wikidot.com/lecture-presentations>

שיטת סורנסון- דיס:

<http://www.catalysoft.com/articles/StrikeAMatch.html>

מאמר על מרחק ג'קארד:

<http://www.iaeng.org/publication/IMECS2013/IMECS2013_pp380-384.pdf>

קורפוסים רבים עבור ערכים בויקיפדיה:

<http://www.wikipedia.org/>

מאגר מילים לשירים מז'אנרים שונים:

<http://www.songlyrics.com/musicgenres.php>