**דוח פרוייקט – מערכת המלצה לשירים**

**תקציר:**

תקציר – מערכת ההמלצה שלנו הינה מערכת המלצה לשירים. בהרבה מקומות (מסעדות, פאבים, ארועים, מועדונים) נדרשת מוזיקה רציפה ברקע שתנוגן באופן אוטומטי. המערכת שלנו תציע פתרון לבעיה הזו בכך שתמליץ על השיר הבא שכדאי לנגן בהנתן N השירים האחרונים שנוגנו. אנו נסתמך על תכונות שונות של שירים כגון – קצב השיר (טמפו), סגנון השיר, מפתח השיר וכו'

דוגמא למקרה – בסט מוזיקה המתנגן בפאב מנוגנים כבר 10 שירים בסגנון ישראלי. המערכת שלנו תזהה את היסטוריית השירים והסגנון שלהם, תחליט על השיר הבא באופן שמתחשב בשירים האחרונים כדי לא לצאת מאיזון ברמה המוזיקלית, אך עם זאת לא לשעמם את הקהל. למשל במקרה הנ"ל המערכת תציע שיר נוסף בסגנון ישראלי אך בזאנר טיפה שונה, או לחילופין מוזיקה דומה לסגנון הישראלי אך לא בעברית (לצורך גיוון).

**תאור האלגוריתם**

האלגוריתם מקבל כקלט מאגר שירים גדול (מעל 10000 שירים). כל שיר מכיל תכונות כגון שם השיר, שם אמן, קצב, ז'אנר, סולם וכו'. המאגר הינו בפורמט של XML, ואנחנו מבצעים פרסור לכל שיר, שלאחר מכן השיר נשמר באובייקט Song. חלק מהשירים יורדים במהלך הפרסור עקב חוסר בחלק מהפרטים ההכרחיים שלהם, אותם אנו חייבים לצורך החישובים.

לאחר הטעינה, אנו יוצרים רשימת השמעה (Playlist) ריקה, אותה האלגוריתם ימלא בשירים שיומלצו על ידיו. מאחר ובהמלצה הראשונה של האלגוריתם רשימת ההשמעה עדיין ריקה, אנחנו ניבחר שהשיר הראשון שיומלץ יהיה רנדומלי, תוך התחשבות בכמות הפעמים שהשיר נוגן בעבר.(זוהי תכונה של שיר בשם Play Count שנשמרת באובייקט Song).

לאחר ההמלצה הראשונה, האלגוריתם ממליץ על שירים ע"י חישוב דמיון בין שיר מסויים לרשימת ההשמעה הנוכחית. השיר שייבדק אינו שייך לקבוצת השירים ברשימת ההשמעה. עבור כל שיר שנבדק, האלגוריתם מחשב כאמור את הדמיון בינו לבין רשימת ההשמעה. הדמיון מחושב באופן הבא: (טווח הערכים לכל מרכיב הינו בין 0 ל 1)

דומות בקצב השירים – חישוב ההפרש בין הקצבים, ככל שההפרש קטן יותר הדומות גדלה.

דומות בשם האמן – שירים של אותו אמן יקבלו דומות של חצי. אחרת 0. האלגוריתם לא נותן דומות 1 על מנת ליצור גוון בהמלצות ולא להמליץ לעיתים קרובות מדי על שירים של אותו אמן.

דומות בסולמות השירים – שירים שנמצאים על אותו סולם יזכו לדומות גדולה. הבדיקה נעשית ע"י לקיחת כל תו (char) מהשיר הראשון, ובדיקה אם תו זה מופיע בשיר השני.

דומות בסגנון השירים – שירים באותו סגנון (היפ הופ, אלקטורוני וכדומה) יקבלו דומות גדולה.

עבור כל מרכיב – הדומות מנורמלת. לדוג' – הדומות בקצב השירים מנורמלת בטווח המקסימלי האפשרי בין קצבי שירים. הדומות בסגנון השירים מנורמלת בסה"כ הסגנונות שיש לשירים הנוכחיים שנבדקים. (כל שיר יכול להכיל מספר סגנונות).

לאחר שקיבלנו ניקוד עבור כל מרכיב, האלגוריתם משקלל את התוצאות לתוצאה סופית בצורה של ממוצע משוקלל – ככל שמתקדמים ברשימת ההשמעה לכוון השיר האחרון שנוגן, המשקל עולה. אנו יוצאים מנקודת הנחה שדמיון לשירים שנוגנו לאחרונה ברשימת ההשמעה צריכים לקבל משקל רב יותר מאשר דמיון לשירים שנוגנו מזמן.

לבסוף, האלגוריתם מוצא את השיר שקיבל את הניקוד הסופי המקסימלי, ואותו האלגוריתם מחזיר כשיר הבא שינוגן.

פסאודו קוד המתאר את חישוב הדומות:

//This function gets the current playlist, and returns the next song that should be played

public Song Recommend(List<Song> playList)

{

if (DataSet == null || DataSet.Count == 0)

return null;

Random rnd = new Random();

int limit = 1000;

List<Song> randomData = DataSet.Where(x => x.playCount > 2).ToList(); //taking only the songs which were played more then twice

if (randomData.Count < limit)

limit = randomData.Count;

randomData = randomData.OrderByDescending(x => (x.playCount \* rnd.Next())).Take(limit).Where(x=> !playList.Contains(x)).ToList(); //dont take the songs that were already played in the current playlist

Song ans = randomData.First();

if (playList == null || playList.Count == 0)

{

return ans; //choose random song to begin the playlist with

}

double maxScore = calcScore(ans,playList);

randomData.ForEach(song =>

{

double score = calcScore(song, playList);

if(score > maxScore)

{

maxScore = score;

ans = song;

}

});

return ans;

}

//we assume that the last song that was played is in the last index in the playlist

private double calcScore(Song song, List<Song> playList)

{

double score = 0;

int index = 0;

playList.ForEach(psong =>

{

index++;

double sim = calcSimilarity(song, psong);

score += (index \* sim); //give more weight to songs that was recenlty played

});

score = score / playList.Count;

return score;

}

//Calculate the similarity between songs, based on BPM, Genre, Key and Title.

private double calcSimilarity(Song song1, Song song2)

{

//BPM Similarity (Maximun value - 1 )

double bpmSim = 1 - (Math.Abs(song1.bpm - song2.bpm)/ 60 ) ;

if (bpmSim < 0)

bpmSim = 0;

//Key Similarity (Maximum value - 1)

double keySim = 0;

if (song1.key.Equals(song2.key))

keySim = 1;

else

{

int countSimKey = 0;

song1.key.ToCharArray().ToList().ForEach(c =>

{

if (song2.key.Contains(c))

countSimKey ++;

});

int maxLength = Math.Max(song1.key.Length, song2.key.Length);

keySim = countSimKey / maxLength;

}

//Artist Similarity (Maximun value - 0.5)

double artistSim = 0;

if (song1.artist.Equals(song2.artist))

artistSim = 0.5;

//Genre Similarity (Maximum value - 1

int countSimGenre = 0;

if(song1.genre.Count !=0 && song2.genre.Count != 0)

{

song1.genre.ForEach(g =>

{

if (song2.genre.Contains(g))

countSimGenre++;

});

}

double genreSim = 0;

if (countSimGenre != 0)

{

genreSim = countSimGenre / (song1.genre.Count + song2.genre.Count - countSimGenre);

}

return genreSim + artistSim + bpmSim + keySim;

}