למידת מכונה - תקציר פרויקט+ קישור לפרויקט

מגישים:
מעוז גרוסמן
לירון ארד

קישור לגיטהאב:
https://github.com/maoz-grossman/Chord-recognition

השאלה שאנחנו מתכוונים לענות עליה:
בהינתן אקורד חדש, או קטע מוזיקלי כלשהו תוכל התוכנית לזהות את האקורדים המתנגנים.

תיאור של המאגר הנבחר:
המאגר מכיל 2000 אקורדים מפוצלים לעשר מחלקות, בכל מחלקה יש עד 200 דוגמאות לכל סוג אקורד.
הקבצים מאוחסנים בפורמט .WAV 16 bits mono 44100Hz
קישור למאגר המידע:
https://people.montefiore.uliege.be/josmalskyj/research.php

-Preprocess
השלב הראשון (והחלק הקשה יותר) היה למצוא דרך להמיר את קבצי האודיו של הדאטה סט (שהם בפורמט wav) לתצוגה דיגיטלית, כך שיהיה ניתן להשתמש בה כדאטה לאלגוריתמים.
בכדי לעשות זאת היינו צריכים למצוא מתודה שלוקחת גלים ומצליחה להמיר אותם לחישובים מספריים.
למזלנו הצלחנו למצוא פלייליסט באינטרנט ברמה גבוה ומובנת שמסביר כיצד ניתן להשתמש בקבצי קול כדאטה עבור רשתות נוירונים (עם הסבר מעולה על רשתות נוירונים).
קישור: Valerio Velardo - The Sound of AI
לכאורה מה שהוא הציג היה בדיוק מה שחיפשנו- דרך להמיר קבצי אודיו לדאטה ולהשתמש בהם לצורך למידת מכונה, אך לאחר ניסוי וטעייה גילינו שהשיטה שהוא משתמש בה לא יעילה עבור מציאת אקורדים, הוא השתמש במתודה שנקראת MFCC שבאמצעות פונקציות פורייה מצליחה לקחת את ה"עוצמה" של הצליל ולכמת אותה לרצף של מספרים אי-רציונליים.
ניסינו להשתמש במתודה זו ברשתות נוירונים על הדאטה שלנו ,כפי שהוא הציג בסרטונים, אך התוצאות שהצלחנו להפיק היו בנוניות מאוד (דיוק של 60 אחוז בממוצע).
משום ש-MFCC נועדה למציאת עוצמת הצליל היא מעולה כדי לתת קלאסיפיקציה בין ג'אנרים של מוזיקה, או כדי להבדיל בין קול נשי לגברי ,אבל בכדי למצוא מה האקורד נצטרך למצוא פיצרים ספציפים שמתארים את האקורד.
לאחר הרבה חיפוש באינטרנט הצלחנו לגלות שמה שאנחנו מחפשים זה בעצם את הHPCP של האקורדים.
על קצה המזלג- ישנם 12 תווים: C, C#(or Db),D , Eb, E, F, F#, G, G# (or Ab), A, Bb,B.
(או: דו ,דו-דיאז, רה, מי-במול, מי, פה, פה-דיאז, סול, לה-במול, לה ,סי-במול ,סי)
12 התווים האלו חוזרים על עצמם במחלקות המוגדרות לפי אוקטבות , C של אוקטבה אחת יהיה זהה

ל-C של אוקטבה אחרת רק בטונציה גבוהה (או נמוכה) יותר.
לכל אקורד יש כמה תווים שנחשבים דומננטים, למשל באקורד C התווים הדומיננטים הם C, G ,E(דו, סול ו-מי)
מה שאנחנו צריכים למצוא זה אילו פיצרים (תווים) יותר דומיננטים בתוך הקובץ אודיו, וכך נדע את ה PCP שלו (*pitch class profile*).
לאחר חיפושים רבים (מאוד) הצלחנו למצוא קוד בגיטהאב שמוצא את הpitch של הצליל (ועובד) ולא מפריע לקוד שהשתמשנו בו לפני כן, כלומר הוא עוזר לנו למצוא את הpitch של הצליל ואנחנו יכולים להכניס אותו לקוד שהיה לנו לפני כדי לקבל קובץ גייסון.
קישור לגיטהאב שהשתמשנו בו:
https://github.com/amrondonp/Chords.py/blob/master/final_project/preprocessing/pitch_class_profiling.py
הערה: *יש הבדל בין אקורד שמצוין כאות באנגלית לתו שמצוין כאות באנגלית.*
אומנם שנהם משקפים דבר דומה, הצליל של האקורד C משקף את התו דו, בדיוק כמו שהתו C משקף את התו דו, רק שאקורד בנוי מכמה מכמה תווים והדומיננטי שבהם הוא התו דו.

תהליך הלמידה-

בתכנון הראשוני שלנו רצינו להשתמש באלגוריתמי למידת מכונה שלמדנו בשיעור (כמו עצי החלטה או AdaBoost) לאחר שמצאנו את הסרטונים ביוטיוב ניסינו לממש את התכונית כפי שהסרטונים למדו עם רשתות נוירונים, לצערנו התוצאות היו לא מספיק טובות.
לאחר שהחלפנו שיטה לPCP החלטנו לחזור לתכנון המקורי שלנו ולהשתמש באלגוריתמים הקלאסיים של למידת מכונה מכמה סיבות:
א. לאחר ששינינו את השיטה של הpreprocess הגייסון שקיבלנו לא היה מתאים לשמימוש ברשתות נוירונים של tensorflow , כנראה כי השתמשנו ב Librosa שנותן מימד נוסף לגייסון וכך ניתן להשתמש בו ברשתות הנוירונים.
ב. Tensorflow לא עבדה כמו שצריך על המחשב לינוקס, והשתמשנו בו הרבה ,כנראה בגלל שהוא מבקש גישה לgpu במקום בcpu של המחשב, ובמידה ואין הוא עובד עם הcpu , ובלינוקס זה עושה בעיות.
ג. רשתות נוירונים הן הרבה יותר איטיות.
ד.הרבה יותר קל להשתמש בספרייה sklearn והיא גם לא כלכך כבדה כמו הtensorflow, ומה שכי טוב בה זה שהיא חוזרת על עצמה, דבר שמקל עלינו לבדוק את המודל אח"כ על שיר קיים בלי יותר מידי שינויים בקוד.

אלגוריתמים שהשתמשנו בהם. (שנתנו תוצאות טובות)

- AdaBoost
- Support vector machine
- k-nearest neighbors algorithm
- Decision tree

אלגוריתמים שניסינו להשתמש בהם (אבל לא נתנו תוצאות טובות):

- Multilayer perceptron
- Convolutional neural network
- Recurrent neural network

ניתן למצוא את הקודים שלא הצליחו לנו בתיקייה "failed attempts…"

תוצאות –

בכל אלגוריתם חילקנו את הדאטהסט ל75 אחוז training ו-25 אחוז testing.

באלגוריתמים Knn הרצנו כמה גירסאות- אחד של שכן אחד, שלושה, חמישה ושבעה.,
הרצנו את האלגוריתם כמה הרצות של 100 פעמים פעם, כל פעם שינינו את הדאטה של הtest והtrain,
ובדקנו איזה מספר שכנים נותן את התוצאה הטובה ביותר.
התוצאות לא היו שונות מידי, כל הווריאציות נתנו תוצאות בסביבות ה95~96 אחוזי דיוק,
במקום הראשון (תמיד) היה כאשר מספר השכנים הוא שלוש, במקום השני לפעמים כאשר היו חמישה שכנים ולפעמים שכן אחד, ובמקום האחרון בפער קטן (מאוד) עבור שבעה שכנים.
תוצאות:

```
~~~Errors Comparison: ~~~
Number of neighbor=  1  ,number of errors:  20.892 , accuracy:  0.9582160000000025
Number of neighbor=  3  ,number of errors:  20.227 , accuracy:  0.9595460000000028
Number of neighbor=  5  ,number of errors:  20.655 , accuracy:  0.9586900000000026
Number of neighbor=  7  ,number of errors:  20.94 , accuracy:  0.9581200000000005
```

(מתוך הקובץ KNN_AVG.py בתיקיה Test).

גם את האלגוריתם SVM הרצנו בכמה גירסאות- הרצה אחת של פונקציה לינארית, והשניה של פונקציה רדיאלית (rbf- radial basis function).
כאן היו פערים יותר גדולים בין האלגוריתמים, מתוך הרצה של 1000 פעמים
יצא שה-rbs מדויק בערך בשלושה אחוז יותר מהלינארי.

```
~~~Errors Comparison: ~~~
Type of SVM: Linear , number of errors:  28.378 accuracy:  0.9432440000000021
Type of SVM: radial basis function , number of errors:  14.769 accuracy:  0.9704620000000019
```

(מתוך הקובץ SVM_AVG.py בתיקיה Test).

את שאר האלגוריתם (עצי החלטה וadaboost) הרצנו פעם אחת בלבד.

השוואה בין כל האלגוריתמים:
כמו כן עשינו השוואה בין כל האלגוריתמים השונים, כדי לראות איזה אלגוריתם הוא המדויק ביותר.
הרצנו את כל האלגוריתמים כמאה , כאשר ב-knn בחרנו שמספר השכנים יהיה 3 וב-svm פונקציית הבסיס רדיאלית.
במקום הראשון עם ממוצע גבוה של בערך 97 אחוזים- SVM radial basis function .
במקום השני עם ממוצע לא רחוק ממנו של בערך 96 אחוזים- KNN, with 3 neighbors

במקום השלילי והמכובד- decision tree, עם ממוצע של בין 94 ל- 95 אחוז דיוק.
ובמקום האחרון והחביב- Adaboost עם ממוצע בין 93 ל-94 אחוזי דיוק.

```
knn average accuracy :  0.9605199999999994
SVM average accuracy:  0.9713799999999995
Decision Tree average accuracy:  0.9478799999999998
Adaboost average accuracy:  0.9359999999999999
```

(מתוך הקובץ bestClass.py בתיקייה Test).

<u>טסט על שיר קיים-</u>
רצינו לבחון עד כמה המודל שלנו טוב גם על שירים קיימים.
בגיטהאב שממנו לקחנו את הקוד של ה-PCP היה גם טסט שהוא עשה על שיר קיים:
הוא לקח את החצי דקה הראשונה של שיר *about a girl* של נירוונה ופירק אותו למלא קטעים של חצי שניה ובדק מה המודל אומר שהאקורדים של השיר, הוא השתמש ברשתות נוירונים והגיע לתוצאה נכונה, אנחנו השתמשנו באלגוריתמי למידת מכונה קלאסיים ולהלן התוצאה שלנו:

```
KNN:
em g em g em em em g em em g g em g g g em g g g em em g em em g em bm em g em
Adaboost:
em em em g em em em g em em g g em em g em em em g em em em g em em g g em em em em
Decision tree:
em g g g g g bm g em g g g g g g g g g g g g g g em em g g bm em g g
SVM rbf:
em g em g em em em g em em g g em g g g em g g g em em g em em g g bm em g em
SVM linear:
em g em g em em em g em em g g em g g g em g g g em em g em em g g em em g g
Real chords:
em g em g em em em g em em g g em g g g em g g g em em g em em g g em em g g
```

הקובץ test.py בתיקייה Test

בגלל הפערים הגדולים בין rbf ללינארי, החלטנו לראות מה יותר מדויק מבין שניהם, להפתעתנו דווקא הלינארי צדק יותר מרדיאלי, ככל הנראה כי החיתוך של בקבצים לא היה מספיק מדויק, היו מקטעים של חצי שניה שהכילו שני סוגים של אקורדים, וכנראה זה מה שגרם לתוצאה להיות שונה בין הווריאציות השונות.
אם הייתה לנו היכולת לפרק שיר למקטעים לפי האקורדים של השיר אנחנו מאמינים שהיינו מגיעים לדיוק של 100 אחוז.