עדי מלץ - 318438223
מיתר גולדברגר - 207130345

Evolutionary AI

For WINE Types

by: Adi Malz, Meitar Goldberger

https://github.com/Meitargo/evolutionAlgorithnWine.git

תוכן עניינים

מבוא

הפרויקט שלנו עוסק בפיתוח אלגוריתם אבולוציוני בעזרת למידת מכונה על סוגי יין.

אנחנו עובדות עם dataset של 178 יינות. נתונים אלו הם תוצאות של ניתוח כימי של יינות שגדלו באותו אזור באיטליה אך נגזרו משלושה זנים שונים. הניתוח קבע את הכמויות של 13 מרכיבים שנמצאו בכל אחד משלושת סוגי היינות.

נרצה לפתח אלגוריתם אבולוציוני שפותר בעיית קלסיפיקציה - בהינתן יין ומרכיביו נרצה להסיק לאיזה סוג משתייך היין.

הקוד הסופי נמצא בתיקיית הRepository בGitHub הבא:

https://github.com/Meitargo/evolutionAlgorithnWine

תיאור הבעיה

הבעיה : בהינתן יין ומאפייניו, נרצה להסיק לאיזה סוג משתייך היין מבין שלושת הסוגים – 0, 1 או 2.

מאפייני היין :

1) Alcohol
2) Malic acid
3) Ash
4) Alcalinity of ash
5) Magnesium
6) Total phenols
7) Flavanoids
8) Nonflavanoid phenols
9) Proanthocyanins
10) Color intensity
11) Hue
12) OD280/OD315 of diluted wines
13) Proline

המטרה : בהינתן מאפיינים של יין נרצה לבנות מערכת שתצליח לנבא באחוזי הצלחה גבוהים ככל האפשר את סוג היין.

תיאור הפתרון

מימשנו אלגוריתם אבולוציוני שפותר בעיית קלסיפיקציה - בהינתן יין ומרכיביו נרצה להסיק לאיזה סוג משתייך היין.

אלגוריתם אבולוציוני

אלגוריתם אבולוציוני הוא סוג של אלגוריתם המשתמש בעקרונות האבולוציה הטבעית כדי למצוא פתרונות לבעיות. האלגוריתם הוא אלגוריתם חיפוש, מידול ואופטימיזציה המשתמש בקבוצה של פעולות מתמטיות בהשראת האבולוציה הטבעית כגון בחירה, הצלבה ומוטציה כדי ליצור פתרונות חדשים לבעיה.

המטרה של אלגוריתם אבולוציוני היא למצוא את הפתרון הטוב ביותר לבעיה מתוך קבוצה של פתרונות פוטנציאליים. זה מתבצע ע״י יצירת אוכלוסיית פתרונות ושיפורים באופן איטרטיבי באמצעות שימוש בפעולות לעיל.

למידת מכונה

נעזרנו בספריית machine learning בפייתון שנקראת sklearn. בעזרת ספרייה זו יכלנו לחלק את המידע ל- מידע עבור בדיקה (test) ומידע עבור אימון (train). אימנו את המודל ע״י ה- training data . השתמשנו בtest data על מנת להעריך את ביצועי המודל. לאחר מכן, השתמשנו במודל על מנת לבצע תחזיות על מידע חדש.

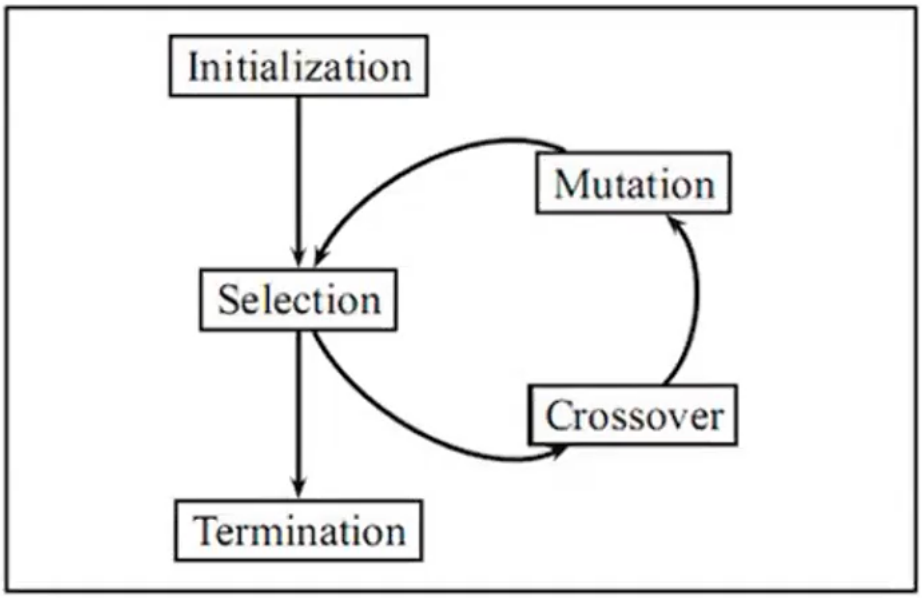
מימוש אלגוריתם האבולוציוני

מבנה הפרט – הפרט מיוצג כעץ המעריך את מרכיבי היין, שצמתיו הן פונקציות החישוב ועליו הם תכונות היין.

מבנה האוכלוסייה – האוכלוסייה נבנית מ- RampedHalfAndHalf, כדי ליצור אוכלוסייה מגוונת ככל הניתן. האוכלוסייה מורכבת מרשימה של אינדיבידואלים וכל אינדיבידואל מיוצג ע״י עץ.

<u>האבולוציה</u>:

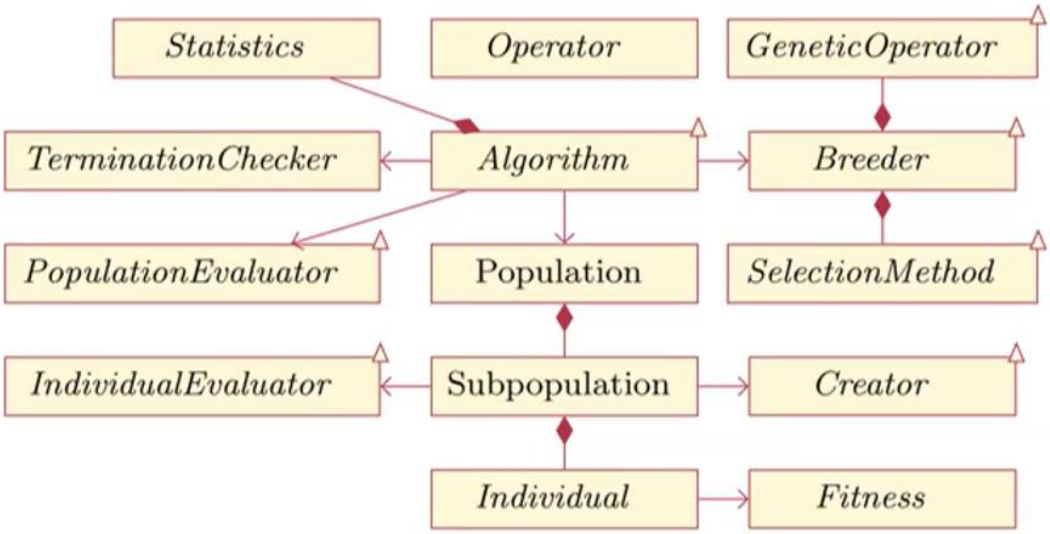
1. Fitness – הevaluator שלנו נבנה מClassificationEvaluator שהוא מחשב בכל תחילת דור, את הפיטנסים של כל האינדיבידואלים.
2. Elitism - הגדרנו משתנה elitism = k , בכל תחילת דור הוספנו את k העצים בעלי הפיטנס הגבוה ביותר ללא שינוי. כך אנחנו יכולים לשמר את העצים הטובים וליצור אוכלוסייה טובה יותר עבור שאר הדורות.
3. Selection – ביצענו selection tournament, ניקח בכל דור n אינדיבידואלים ונחשב את הfitness הגבוה ביותר מבניהם ואותו מעבירות לדור הבא.
4. Mutation – בחרנו עץ מאחד האינדיבידואלים ובהסתברות של p נחליף תת-עץ שלו בעץ רנדומלי אחר.
5. Crossover – בחירה רנדומלית של שני צמתים משני עצים שונים והחלפתם בהסתברות p.



Graph: Evolutionary Process

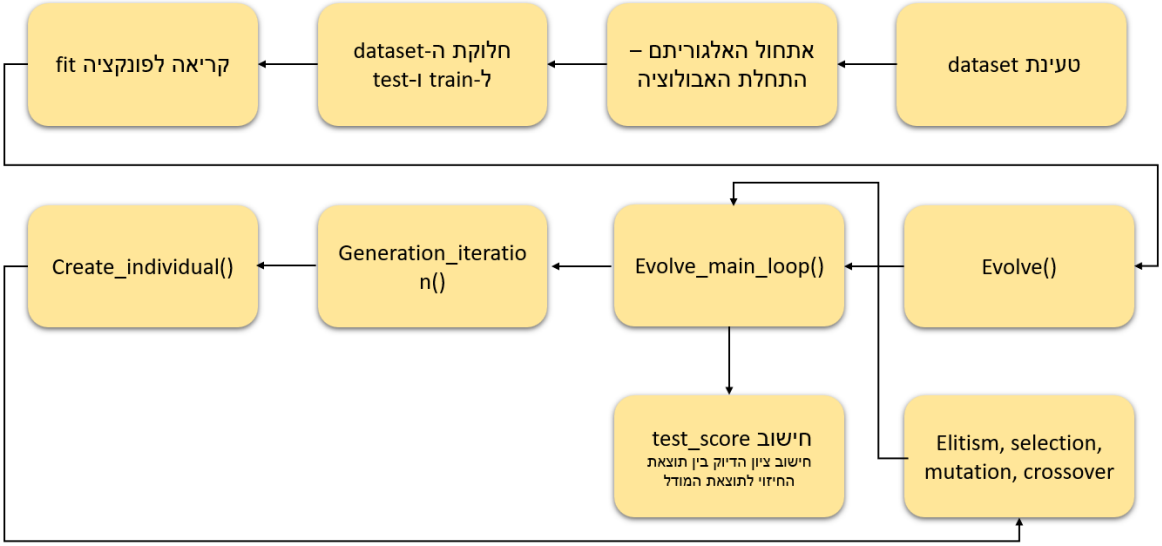
מבט על של התוכנה

דיאגרמת המחלקות של ה-EC-Kity:



<u>מתודות עיקריות</u>:

- Load_wine() – טוען את ה-dataset מ-sklearn ל-X ו-Y. X ישמור את התוכן של ה-dataset עצמו. וב- Y יהיה מערך של הסיווגים (0,1,2) בהתאמה.
- SKClassifier() – הפונקציה עוטפת את האלגוריתם האבולוציוני בקלסיפיקציה של sklearn על מנת להשתמש בפונקציות של הספריה sklearn.
- fit() – פונקציה המריצה את האלגוריתם האבולוציוני ע"י skleran. פונקציה זו מאמנת את המודל ע"י ה- training data שסופקו המורכב מ-X ו-Y. המודל לומד דפוסים בנתונים המאפשרים לו לבצע תחזיות לגבי נתונים חדשים.
- evolve() – הפונקציה נמצאת במחלקה Algorithm והיא מאתחלת את ריצת האבולוציה ויוצרת אוכלוסייה.
- evolve_main_loop() - עוברת בלולאה על מספר הדורות, אם הדור לא "מספיק" ($|fitness - 1| \leq threshold$) טוב אז יוצרת דור חדש.
- generation_iteration() - יוצרת דור חדש וקוראת לbreeder.
- create_individual() - יוצרת את העצים של האינדיבידואלים.
- apply_breed() – מיישם את ה-elitism וה-selection.

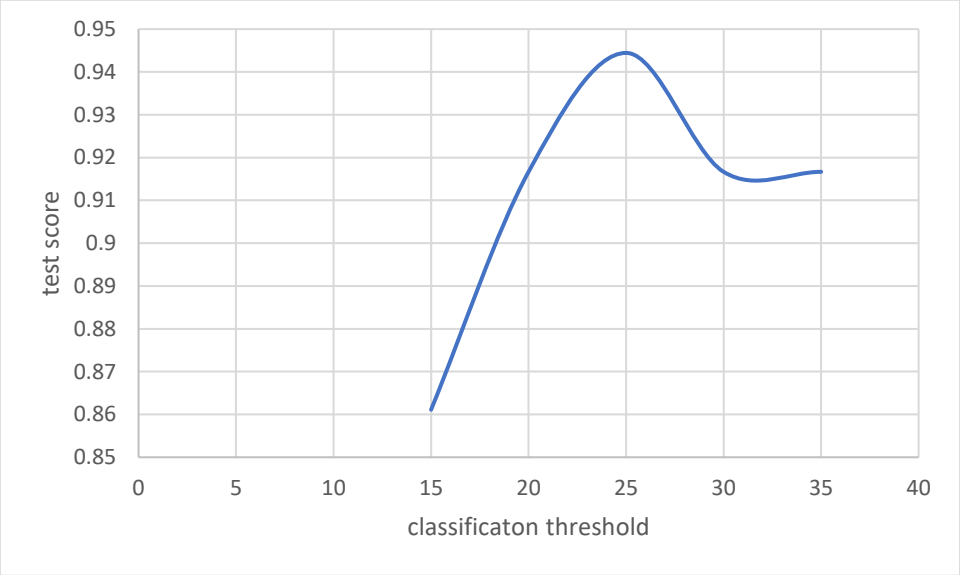


Graph: Program's process

תוצאות, הדגמות ריצה, גרפים

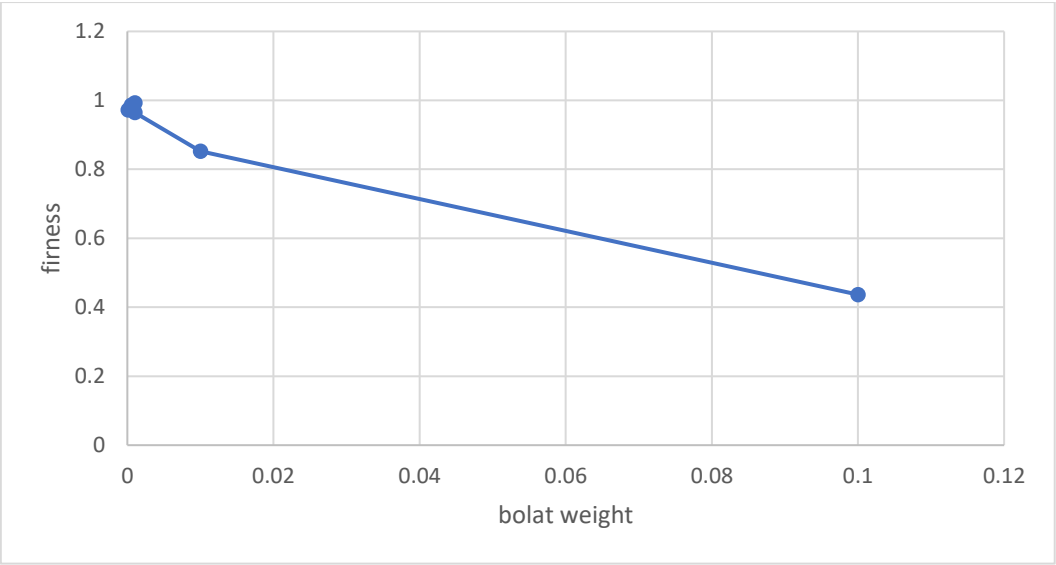
על מנת לקבל את התוצאות הטובות ביותר ביצענו מספר הרצות שונות של האלגוריתם עם נתונים שונים.

תחילה, בדקנו עבור איזה Classification threshold אנו מקבלים את התוצאות הטובות ביותר, יחד עם נתונים רנדומליים קבועים עבור שאר הפרמטרים. בכל ההרצות קיבלנו fitness דומה, ולכן התייחסנו בריצות לפרמטר של הtest_score. הרצנו מספר פעמים עם מספר threshold שונה וכפי שניתן לראות בגרף (1) כאשר ה- Classification threshold = 25 אנו מקבלות את התוצאות הטובות ביותר עבור הtest_score.



Graph 1: best classification threshold

במהלך הריצות, קיבלנו עצים עם גבהים אסטרונומים (גבהים בין 20-30) בכל אחת מהריצות, כדי למנוע bloat, נרצה להגביל את גובה העץ ע״י שינוי הפרמטר bloat weight.

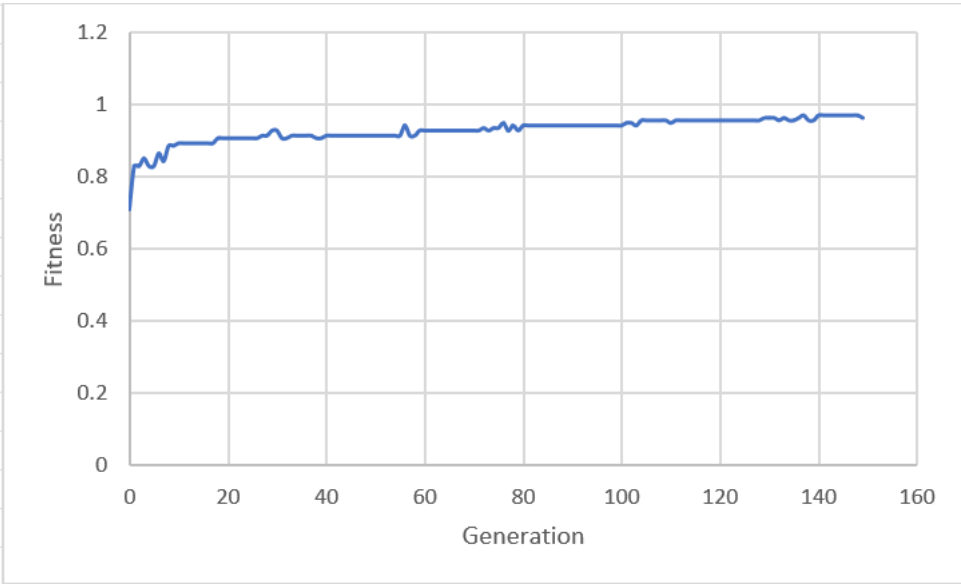


Graph 2: bloat weight influence on fitness

בגרף לעיל, ככל שהגדלנו את הbloat weight - גובה העץ קטן. ראינו כי החל מ-
bloat_weight = 0.01 הגובה אמנם קטן, אך הfitness גם כן קטן משמעותית.
עד bloat_weight = 0.01 הגובה קטן אך הfitness לא השתנה משמעותית.
לכן החלטנו שעבורנו ה- bloat weight האופטימלי הוא 0.01 על מנת לא לנפח את העץ משמעותית אך עדיין לקבל fitness טוב.

לאחר הרצת האבולוציה עם השינויים המתוארים מעלה, ביצענו מספר ניסויים ושינינו את הפרמטרים הכתובים מטה, התוצאות האופטימליות התקבלו עבורנו עבור האבולוציה עם הפרמטרים הבאים:

elitism rate = 0.3, crossover = 0.9, mutation = 0.2, tournament size =0.6



Graph 3: best fitness for generation, in the best evolution

סיכום ומסקנות

האלגוריתם האבולוציוני שבנינו הצליח להתפתח וככל שהגדלנו את מספר הדורות הוא התפתח עוד. כשהגדלנו את מספר הדורות ל-1000 אף סיים לפני מס' הדורות שהוקצה לו.

תחילה, קיבלנו עצים אסטרונומיים, ואמנם הפיטנס היה טוב. אך העדפנו עץ בגודל סביר על מנת שהחישוב יהיה סביר גם כן. לכן הגבלנו את גובה העץ ע"י פרמטר ה-bloat_weight. ראינו כי הגבלה זו לא שינתה את הפיטנס בצורה משמעותית.

בעקבות הגבלת גובה העץ, לאלגוריתם לקח יותר זמן לרוץ אך לא באופן משמעותי.

הגענו ל test_score גבוה מאוד שמעיד על כך שהאלגוריתם מצליח בסבירות טובה לחזות את הסיווג על הדאטא שהוא לא התאמן עליו.

```
best pure fitness over training set: 0.9859154929577465
test score: 0.9722222222222222
Total runtime: 888.1820743083954 seconds.
```

Graph 4: output of 1000 generation