השוואה בין שני אלגוריתמים: Gradient Descent

במהלך החקירה של ההסבר המתמטי לאלגוריתם Gradient Descent, נתקלתי בשאלה: **למה אנחנו עוברים בלולאה על כל אחד מערכי הוקטור x ומחשבים את ערך הטעות עבורו, במקום לחשב את ערך הנגזרת החלקית של Cost לפי m ו b עבור הוקטור השלם?**

האלגוריתם השלדי של הפתרון המקורי:

1. בצע epochs פעמים:
   1. עבור כל i מ-0 עד מספר הנתונים בסדרות x ו y:
      1. מצא את
      2. בצע
      3. בצע

האלגוריתם השלדי של הפתרון שלי:

1. בצע epochs פעמים:
   1. מצא ווקטור
   2. בצע
   3. בצע

הסבר לאלגוריתם השלדי של הפתרון שלי

ארשה לעצמי להשתמש בסימונים מתמטיים פורמליים כאן.

כך נראית הפונקציה Cost:

נגזור עבור m:

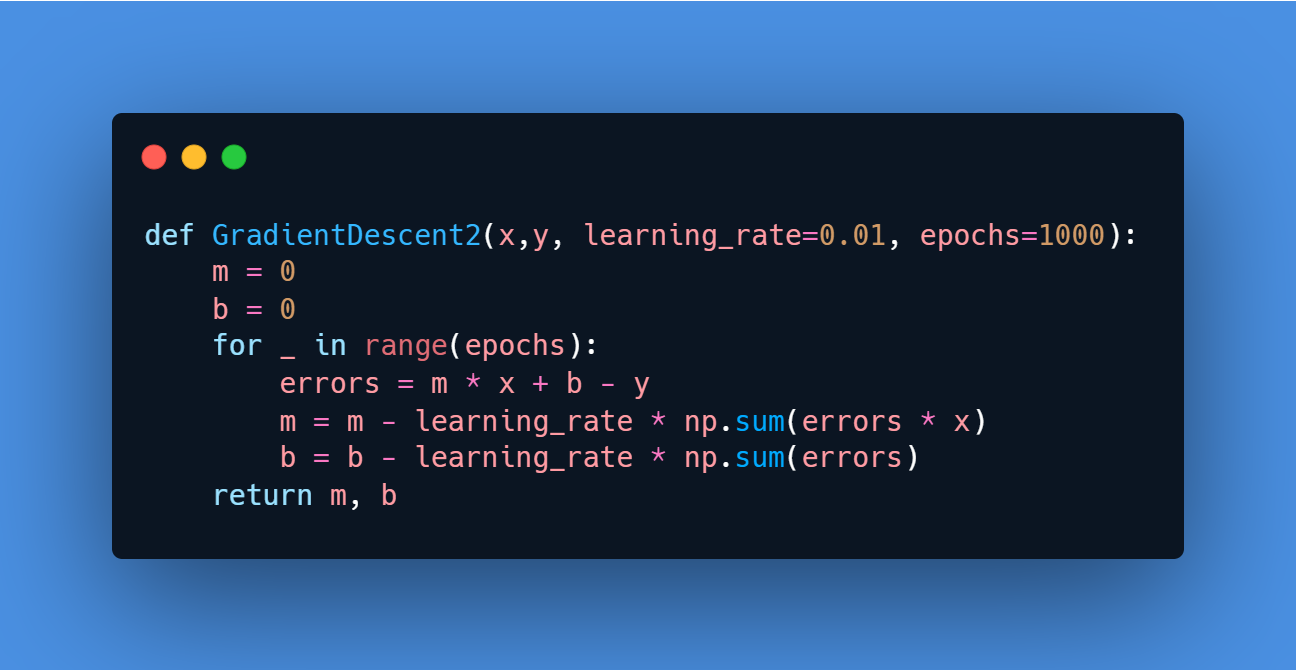
נגזור עבור b:

בדומה לפתרון המקורי, אנחנו כופלים בקבוע ולכן הכפל ב-2 זניח.

קיבלנו כי:

מימוש האלגוריתם בפייתון

כך מימשתי את האלגוריתם:



(מקור: carbon.now.sh – אחלה של אתר ליצירת תמונות של קוד ☺)

השוואה בין יעילות האלגוריתמים

|  |  |  |
| --- | --- | --- |
| קריטריון | הפתרון המקורי | הפתרון שלי |
| יעילות |  |  |
| הסבר | לולאה המתבצעת epochs פעמים, ובתוכה לולאה המתבצעת n פעמים. | למרות שיש רק לולאה אחרת, פעולת הסכימה היא פעולה שמחירה n. |

אין הבדל ביעילות. אז מדוע הפתרון המקורי מפרק את החישובים?

השוואה בין דיוק האלגוריתמים

בחרתי להשתמש בקובץ של הקשר בין שנות לימוד לתוחלת החיים. לשם בקרה, אשווה את שני האלגוריתמים לאלגוריתם של חישוב רגרסיה לינארית בדרך ה "ישנה", שלמדנו לפני.

הקוד של ההשוואה: <https://github.com/itays123/DataScience/blob/main/Z1%20Grad-Descent-Math/comparison.py>

תוצאות ההשוואה:

לאחר שחילקתי בקבוע מתאים ושיחקתי קצת עם ו בפתרון הGradient Descent המקורי, הצלחתי להגיע לCost דומה מאוד בין הפתרון הישן ופתרון הGradient Descent המקורי. מתברר שעבור כמות גדולה של נתונים, רצוי להקטין את לקבלת דיוק מירבי.

אבל לא משנה באיזה קבוע חילקתי, מתברר שהפתרון שלי פשוט לא מסוגל לעבד כמויות כאלה של מידע, ואני מקבל את השגיאה שקיבלתי אתמול – אין מספיק מקום ב double על מנת לשמור את גודל המספרים.

|  |  |  |  |
| --- | --- | --- | --- |
| האלגוריתם | הפתרון הישן | הפתרון המקורי | הפתרון שלי |
| m | 2.103453104925404 | 2.0839924180753933 | Nan |
| b | 0.4410888913291284 | 0.44718173587719917 | Nan |
| Cost | 1053.5528322402017 | 1057.5785456060084 | Nan |

זהו Cost יפה מאוד עבור כמות כזאת גדולה של נתונים – כמעט 3000.

המסקנה: הסיבה למימוש המקורי היא בעיקר טכנית, על מנת להקל על התוכנה.