מבוא לבינה מלאכותית - תרגול 2.

רגב יחזקאל אימרה.

10 בנובמבר 2025

1 SGD.

למדנו על אלגוריתם מורד הגרדיאנט, שניתן לתארו באמצעות

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \mathcal{L}(\theta_t)$$

כאשר

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

היתרון הוא שכל צעד שאנחנו מתקדמים הוא מדוייק אל עבר כיוון הירידה הכי טוב.

החסרונות הם:

1. אם יש לי N ממש ממש גדול, לעבור על כל הדגימות יקח נצח.
2. אני עלול להיתקע במינימום מקומי.

לכן אנחנו רוצים לשנות מעט את האלגוריתם כדי שיפתור את שני החסרונות האלו ושעדיין ישמור על היתרון. הדרך לעשות את זה היא באמצעות עדכון הגרדיאנט לפי קבוצה קטנה יותר של דגימות:

$$\bar{\mathcal{L}}(\theta) = \frac{1}{k} \sum_{j=1}^{k} \left(y_{i_j} - \hat{y}_{i_j}\right)^2$$

כאשר $k << N$, ובכך לשנות את כלל העדכון שלנו להיות

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \bar{\mathcal{L}}(\theta_t)$$

יתרונות:

1. הרבה יותר מהיר לחישוב.
2. יכול לברוח ממינימום מקומי כי כל עדכון הוא "רועש".

חסרונות:

1. כל עדכון הוא "רועש" - אנחנו לא הולכים בכיוון הירידה הכי חדה בשביל כל הנתונים אלא בכיוון הירידה הכי חדה בשביל תת הקבוצה של הנתונים שבחרנו.

לכן שיטה יש את היתרונות והחסרונות שלה, לכן אין תשובה חד משמעית למה הכי כדאי להשתמש בו.

2 רגרסיה פולינומיאלית.

נניח עכשיו במקום להניח שיש קשר לינארי בין הנתונים שלנו $\hat{y} = \beta_0 + \beta_1 x$, אנחנו חושדים שהנתונים שלנו לא מגיעים מקשר לינארי, אז אנחנו יכולים להרחיב את מודל הרגרסיה הלינארית להיות

$$\hat{y} = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_d x^d$$

למודל כזה קוראים מודל רגרסיה פולינומיאלית.

אבל, השם הזה מטעה, כיוון שהקשר בין y לכל הפרמטרים של המודל הוא עדיין קשר לינארי.

איך מאמנים? בעיקרון אם בעבר ברגרסיה לינארית היה לנו $x = (x_1, x_2, \ldots, x_d)$, אנחנו יודעים איך לאמן מודל מהצורה

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_d x_d$$

לכן פשוט נחשב $x_poly = \left(x, x^2, \ldots, x^d\right)$ ופשוט יש לנו

$$\hat{y} = \sum_{n=1}^{d} x_poly_n \beta_n + \beta_0$$

לכן נוכל להתייחס לזה כתור מודל רגרסיה רגיל ולהשתמש בכל שיטות האימון שאנחנו כבר מכירים.

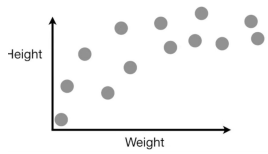
במידה ולכל דגימה x יש יותר מפיצ'ר אחד, כלומר $x = (x_1, \ldots, x_p)$ ואנחנו רוצים את כל הפולינומים מדרגה לכל היותר d, אנחנו יכולים להגדיר

$$\hat{y} = \beta_0 + \sum_{1 \le i \le n} \beta_i x_i + \sum_{1 \le i \le j \le n} \beta_{ij} x_i x_j + \sum_{1 \le i \le j \le k \le n} \beta_{ijk} x_i x_j x_k$$

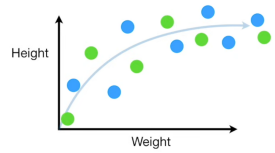
ושוב להתייחס למודל הזה כתור מודל רגרסיה לינארית.

3 $BIAS\ VARIANCE\ TRADEOFF$.

נניח ואנחנו מודדים משקל וגובה של עכברים ומשרטטים את הדגימות שלנו בגרף



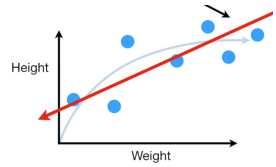
עכברים קלים הם נמוכים, עכברים כבדים הם גבוהים אבל בשלב מסויים עכבר נהיה שמן ולא גבוה יותר. אנחנו רוצים לחזות את גובה העכבר בהינתן המשקל שלו.



נניח ומשקל העכברים מגיע מכזה קשר, נרצה ללמוד אותו.

נחלק את הדאטה לקבוצת אימון וקבוצת מבחן, כאן הנקודות הכחולות יהיו האימון והירוקות יהיו המבחן.

אם ננסה להתאים קו לינארי בין הנקודות שלנו, ניתקל בבעיה, אין לו את "הגמישות" לתאר את היחס האמיתי בין המשקל לגובה (ה"קשת" שנוצרת בדאטה)



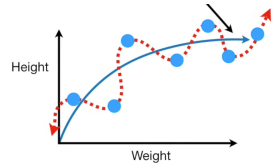
לא משנה איך נעביר את הקו הישר הוא בחיים לא יתעקם, כלומר הקו הישר בחיים לא ישחזר את היחס בין המשקל והגובה, לא משנה כמה טוב הוא על הקבוצת אימון שלנו.

חוסר היכולת של המודל שלנו לתפוס את היחס בין האמיתי של הפרמטרים שלנו נקרא $bias$.

בגלל שהקו הישר לא יכול להתעקם, יש לו הרבה $bias$ (מאנגלית - הטיה), כלומר הוא בקושי מתאר טוב את הדגימות.

לכן נאמר שהקו הישר עושה $underfit$ - לא תופס את הצורה של הדאטה שלנו.

מנגד, אנחנו יכולים להתאים פולינום ממעלה מאוד גדולה שיתאים בדיוק לכל הנקודות בקבוצת האימון שלנו.



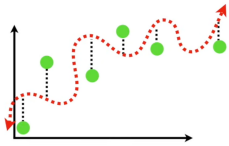
הפולינום הזה מתעקם באופן מעולה כמו שקבוצת האימון מתעקמת וממש מתאים את עצמו לקבוצת האימון.

בגלל שהפולינום הזה יכול להתעקם בצורה ממש טובה, יש לו $bias$ נמוך.

מבחינת MSE על ה-$train\ set$, הפולינום ינצח, כי יש לו $MSE = 0$.

עכשיו נעבור לקבוצת המבחן:

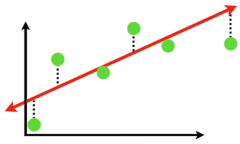
למרות שהפולינום עשה עבודה טובה בללמוד את קבוצת האימון, הוא ממש רחוק מלחזות את קבוצת המבחן



לכן יש לו שונות גבוהה מאודת שכן ההפרש בין הערך שנחזה לאיברים בקבוצת המבחן שונה ממש מהערך האמיתי שלהם.

במילים אחרות, קשה לחזות כמה הפולינום שלנו באמת למד. לפעמים הוא יהיה טוב, ולפעמים ממש גרוע.

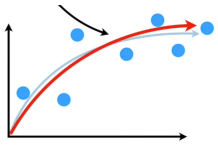
מנגד, לקו הישר היה $bias$ מאוד גבוה, אבל יש לו פחות שונות כי הוא התאים את עצמו לצורה הכללית של הדאטה, ולא ספציפית לצורה של קבוצת האימון.



במילים אחרות, הקו הישר יביא תוצאות סבירות (ולא מעולות) ,אבל יהיה עקבי בתחזיות שלו.

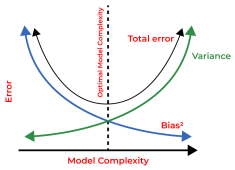
בגלל שהפולינום מתאים את עצמו לקבוצת האימון אבל לא לקבוצת המבחן, נאמר שהוא עושה $overfit$.

אנחנו רוצים שהמודל האידיאלי שלנו יהיה בעל $bias$ נמוך כלומר שיוכל לחקות את הקשר בין המשתנים שלנו בצורה סבירה, ושתהיה לו שונות נמוכה כלומר שיפיק תוצאות עקביות על דגימות שונות.



לדוגמה

אנחנו רוצים למצוא מודל שהוא בין $bias$ נמוך לבין שונות נמוכה.



4 BIC.

כעת אנחנו נתקלים בבעיה: אנחנו מניחים שהדאטה שלנו לא לינארי, כלומר אנחנו צריכים לבחור איזשהו דרגה d שאנחנו מאמינים שתתאים לדאטה שלנו הכי טוב. הבעיה היא שדרגה נמוכה מדי תביא ל־$underfitting$ ויותר מדי דרגה תביא ל־$overfitting$. אנחנו צריכים דרך לאזן בין איכות המודל לבין כמות הפרמטרים שבמודל.

לכן בדיוק יש לנו כלי בשביל למדוד איכות של מודל: נגדיר שבשביל מודל רגרסיה עם n דגימות ו־k פרמטרים, נגדיר

$$BIC = n\ln(MSE) + k\ln n$$

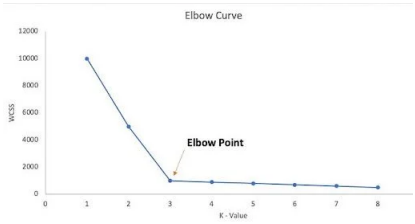
אנחנו רוצים BIC כמה שיותר נמוך, כלומר ההתאמה המירבית בין איכות המודל לגודל המודל.

אינטואיציה: $n\ln(MSE)$ מתגמל התאמה טובה של הדאטה (MSE נמוך $\Leftarrow$ BIC נמוך). $k\ln n$ נותן עונש למודלים מסובכים יותר (יותר פרמטרים $\Leftarrow$ BIC נמוך).

אם נעלה את הדרגה של הפולינום שאנחנו מתאימים, אולי נמצא התאמה יותר טובה אבל זה יעלה לנו את ה־BIC, אז אולי לא שווה להגדיל את המודל.

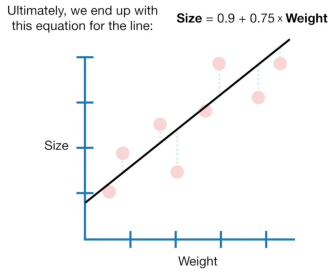
לכן, כדי למצוא את הפולינום הכי טוב, נאמן מודלים עם דרגות $d = 1, 2, 3, \ldots, D$, נחשב BIC לכל מודל ונבחר את המודל עם ה־BIC הכי נמוך.

הערה: אם ככל שאנחנו מגדילים את המודל ה־BIC לא עולה/ משתנה טיפה, נשתמש ב־$elbow - method$ לבחור את d המתאים.



5 .$RIDGE$

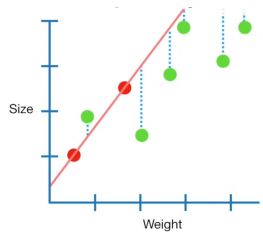
נניח ושוב אספנו מלא דאטה על משקל וגודל של עכברים



ואנחנו רוצים למדל את הקשר ביניהם. נאמן מודל לינארי ונגיע למשוואה

$$Size = 0.9 + 0.75 \cdot Weight$$

כשיש לנו הרבה דגימות אנחנו יודעים שהישר שנקבל מתאר את היחס בין גודל ומשקל. אבל מה אם היו לנו רק שתי דגימות ב־$train\ set$?



פה לדוגמה אם נאמן רק על שתי הנקודות האדומות נקבל את המשוואה $Size = 0.4 + 1.3 \cdot Weight$.

הרעיון מאחורי $ridge$ הוא למצוא קו ישר שלא עושה את ההתאמה הכי טובה על ה־$train\ set$, אבל בתמורה יהיה לו הרבה פחות $variance$ על ה־$test\ set$. במילים אחרות, אנחנו מעלים בכוונה את ה־$bias$ של המודל בקצת כדי לקבל ירידה חדה ב־$variance$.

איך עושים? בעבר אם יש לי $\hat{y} = b + m_1x_1 + \cdots + m_dx_d$ (עבור x עם d פיצ'רים), אז מצאנו $\arg\min\limits_{b,m_1,\ldots,m_d} \sum\limits_{i=1}^{N} (y_i - \hat{y}_i)^2$, אבל עכשיו ננסה למצוא

$$\arg\min_{b,m_1,\ldots,m_d} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{d} m_j^2$$

כלומר אני מוסיף עונש למודל שלי ככל שיש לי יותר שיפוע, ו־λ קובע לי כמה חמור העונש הזה. מה הרעיון? ככל שיש לי שיפוע יותר גדול, ככה המודל שלי יותר רגיש לשינויים קטנים בדאטה, לכן כשאני ממזער גם את $\sum\limits_{j=1}^{d} m_j^2$ אני מנסה למצוא את הפרמטרים לקו הישר שימדל את הדאטה הכי טוב ויהיה כמה שפחות רגיש לשינויים. במקרה הזה נקבל את הישר $Size = 0.9 + 0.8 \cdot Weight$

אכן, עבור הישר שעושה $overfit$ ועבור $\lambda = 1$ לדוגמה נקבל

$$\sum_{i=1}^{N} (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{d} m_j^2 = 0 + 1 \cdot 1.3^2 = 1.69$$

והישר השני שקיבלנו יקבל

$$\sum_{i=1}^{N} (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{d} m_j^2 = 0.3^2 + 0.1^2 + 1 \cdot 0.8^2 = 0.74$$

כלומר באמת קיבלנו התאמה טובה יותר.

הערה 1.5. λ יכול להיות כל ערך מ־0 ל־∞ (לא כולל, כמובן). עבור $\lambda = 0$ נקבל פשוט MSE רגיל ועבור $\lambda \to \infty$ נקבל ישר שמקביל לציר ה־x. איך נחליט איזה ערך λ אנחנו רוצים? פשוט ננסה הרבה ערכים $\lambda_1, \lambda_2, ..., \lambda_k$ ונשתמש ב־$k - fold\ cross\ validation$ (נדבר בהמשך) כדי לקבוע איזה מהם הכי טוב לנו

הערה 2.5. הרבה פעמים $RIDGE$ נקרא גם L^2 כי אם נסמן $m = (m_1, \ldots, m_d)$ אז העונש שהוספנו $\lambda \sum\limits_{j=1}^{d} m_j^2$ הוא $\lambda\|m\|_2^2$.

הערה 3.5. λ הוא היפרפרמטר, כלומר אני קובע לו ערך לפני שאני מאמן את המודל, וכאן אני לא מעדכן אותו תוך כדי האימון (בניגוד ל־lr שאני יכול להגדיל ולהקטין תוך כדי האימון).

6 .LASSO

אותו קונספט כמו $RIDGE$, אבל במקום להעניש את המודל על ידי הוספה של $\lambda\|m\|_2^2$, נעניש אותו על ידי הוספת $\lambda\|m\|_1 = \sum_{j=1}^{d} |m_j|$.

מה ששונה פה זה שעבור λ ממש גדול אנחנו כן יכולים לקבל ישר שמקביל לציר ה־x, כלומר בניגוד ל־L^2 שיכול רק להקטין את הפרמטרים, L^1 יכול ממש לאפס אותם.

מתי זה שימושי? אם עכשיו אני רוצה לחזות גודל של עכברים כתלות במשקל שלהם, בגיל שלהם ובמה הטמפרטורה בחוץ. כפי שאפשר לנחש לטמפרטורה אין שום השפעה על גודל העכברים כי הם לא מתכווצים כשקר וגדלים כשחם, לכן היינו רוצים שבמהלך האימון המשקל שהמודל ישים על הטמפרטורה יהיה 0 ובכך יעלים את הפרמטר הטיפשי הזה. לכן יש לנו

$$Size = b + m_1 \cdot Weight + m_2 \cdot Age + m_3 \cdot Temp$$

ב־$LASSO$ נפתור

$$\underset{b,m_1,m_2,m_3}{\arg\min} \sum_{i=1}^{N} (y_i - Size_i)^2 + \lambda\left(|m_1| + |m_2| + |m_3|\right)$$

ואז נגלה כי $m_3 = 0$ בעוד ש־$RIDGE$ אף פעם לא יאפס אותו.

הערה 1.6. בעיה שמיד צורמת לנו זה ש־$|m|$ לא גזיר, אבל פה אנחנו יכולים להסכים להגדיר את הנגזרת של זה להיות

$$\frac{d}{dm}|m| = \text{sign}(m) = \begin{cases} 1 & m > 0 \\ -1 & m < 0 \\ 0 & m = 0 \end{cases}$$

הערה 2.6. גם כאן λ הוא היפרפרמטר, ולא תמיד הוא יהיה זהה לזה של $ridge$.

7 .ELASTIC NET

אז L^2 עובד הכי טוב כשיש לי הרבה פרמטרים שימושיים בעוד ש־L^1 עובד הכי טוב כשיש לנו הרבה פרמטרים שאנחנו רוצים לאפס את ההשפעה שלהם בתחזית שלנו.

לכן אם אנחנו יודעים מה המודל שלנו מנסה לחזות ומה הפרמטרים שיש לנו אנחנו יודעים מה שימושי ומה לא, אבל מה אם יש לנו אוסף של פרמטרים גנריים שאנחנו לא יודעים עליהם כלום? איך נבחר L^1 או L^2?

אנחנו לא חייבים לבחור, אנחנו יכולים להשתמש ב־$Elastic\ net$! מה זה עושה? כמו $lasso$ ו־$ridge$, פשוט נפתור

$$\sum_{i=1}^{N} (y_i - \hat{y}_i)^2 + \lambda_1\|m\|_1 + \lambda_2\|m\|_2^2$$

כדי למצוא את השילוב הכי טוב של (λ_1, λ_2) נשתמש לרוב ב־$cross\ validation$.

זה טוב בעיקר כשיש לנו קורולציה בין הפיצ'רים: אם לדוגמה x_1 ו־x_2 בעלי קורולציה חיובית, L^1 יבחר לאפס את אחד מהם (כי בעיניו להחזיק 2 עותקים של אותו פיצ'ר (בערך) זה לא יעיל) בעוד ש־L^2 פשוט יקטין את המשקל שלהם לבערך אותו דבר, ככה הם לא יתאפסו וכל אחד יוכל להשפיע בדרכו שלו על תחזיות המודל.

כשנשלב בין L^1 ו־L^2, לא נאפס שום פרמטר, פשוט נקטין פרמטרים לא שימושיים.

8 .EARLY STOPPING

נניח והפעם יש לנו המוני דגימות ואנחנו יכולים לאמן את המודל שלנו כמה שנרצה.

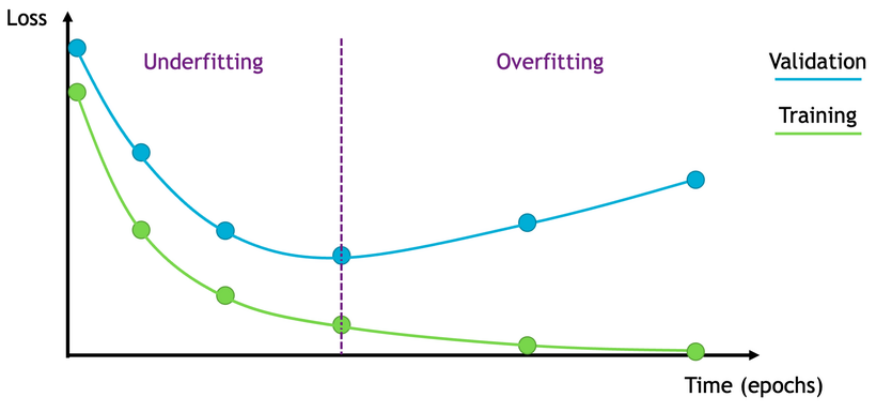
מצד אחד, אם נאמן את המודל יותר מדי, יש סכנה שנעשה $overfitting$ כי נאמן אותו כל כך הרבה זמן שהוא יתחיל "לשנן" את ה־$train\ set$. מצד שני, אם אנחנו רוצים להימנע מזה ונאמן את המודל קצת מדי זמן, אנחנו מסתכנים ב־$underfitting$, כלומר המודל לא הספיק ללמוד את הקשר בין הפרמטרים שלנו וחוזה ג'יבריש.

איך נדע מתי הזמן האופטימלי לעצור את תהליך האימון של המודל?

פשוט. נחלק את הדאטה שלנו ל־$train\ set$ ו־$validation\ set$. נתחיל לאמן את המודל שלנו פעם אחרי פעם אחרי פעם על ה־$train\ set$ שלנו, כל פעם שאנחנו מאמנים כל ה־$train\ set$ נקרא $epoch$. אחרי כל $epoch$ נמדוד את הביצועים של המודל על ה־$validation\ set$. בהתחלה ה־$error$ שם ירד כי ככל שאנחנו מאמנים יותר זמן המודל לומד יותר טוב את הדאטה, אבל מתישהו ה־$error$ יתחיל לעלות - שם אנחנו עושים $overfit$. היינו רוצים לעצור ברגע שאנחנו רואים שהמודל מתחיל לשנן אבל בשביל לדעת שהוא באמת עושה את זה אנחנו צריכים לחכות קצת לראות שאולי לא בטעות השגיאה עלתה אבל מיד אחרי זה ב־$epoch$ הבא חזרה לרדת, לכן כשנעצור את המודל הוא יהיה לא טוב כי הוא כבר עשה שינון, לכן אחרי כל $epoch$ נשמור את הפרמטרים של המודל כל עוד ה־$error$ שלו הכי קטן, ואז כשאנחנו עוצרים את הלמידה נחזיר את סט הפרמטרים האחרון ששמרנו. בפסאודו קוד זה נראה ככה:

```
for epoch in range(epochs) do
    if loss < best_loss then
        Save model parameters;
        best_loss ← loss;
    end
    if model is overfitting then
        return best saved parameters;
    end
end
```

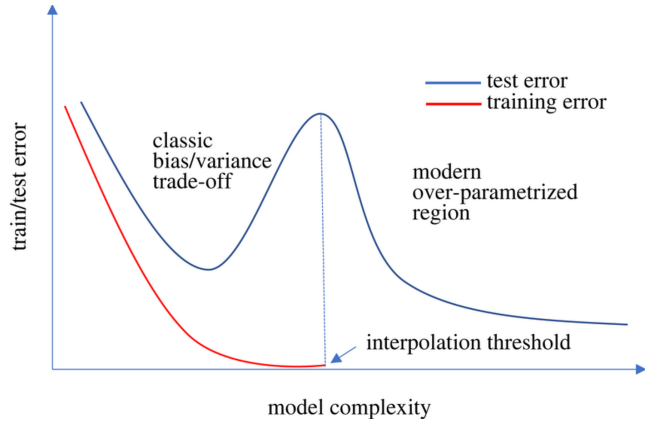
ובזמן האימון ה־$error$ יראה כך:



כמו שאנחנו מצפים, ה־$error$ של האימון תמיד יורד בעוד שה־$error$ על הדאטה החדש יורד עד לשלב מסויים, ואז מתחיל לעלות. לכן ניקח את הפרמטרים שיצאו לנו ב־$epoch$ הרביעי.

הערה 1.8. ב־2019 יצא מאמר בשם

"DEEP DOUBLE DESCENT: WHERE BIGGER MODELS AND MORE DATA HURT" שמראה כי למודלים מסויימים, אם ממשיכים לאמן אותם, אז למעשה ה־$error$ יכול לרדת אחרי שהוא עולה, לפעמים מביא אפילו לשגיאה נמוכה יותר מאשר איפה שבעבר היינו מרימים ידיים ומכריזים שהמודל עושה $overfit$ ומפסיקים את האימון. המאמר אמר שיש 3 מקרים בהם זה יכול לקרות:

1. כשמגדילים את גודל המודל.
2. כשממשיכים לאמן את המודל כשהוא עושה $overfit$.
3. כשמגדילים את כמות הדאטה שנכנס למודל

בעיקרון מה שקורה זה שככל שאנחנו מגדילים כל אחד משלושת הגדלים האלו, המודל מתחיל בללמוד את הנתונים עד שהוא כבר מתאים את עצמו בדיוק לנתונים שנתנו לו, כלומר עושה $overfit$, אבל אם אני ממשיך להגדיל אז בשלב מסויים המודל יכול להתחיל ללמוד קשרים יותר "עמוקים" בדאטה, שמביאים לתוצאות יותר טובות.