עצי החלטה

עצי החלטה משתייכים למשפחה של אלגוריתמים מונחים/מפוקחים, אחד המאפיינים שמייחדים אותם זה שאפשר להשתמש בהם גם לבעיות רגרסיה וגם לבעיות קלסיפיקציה.

המטרה בשימוש בעצי החלטה זה לאמן מודל שבאמצעותו נוכל לקחת משתנה-מטרה ולחזות את סוג המחלקה שלה הוא שייך או את ערכו הנומרי וזה באמצעות שימוש בחוקי החלטה פשוטים (לדוגמה: כן/לא, קטן/גדול וכן הלאה) על שאלות שאנחנו שואלים על הדאטה סט שיש בידינו.

סוגי עצי החלטה:

1. עץ החלטה קטגורי - משתנה המטרה של עץ ההחלטה הוא קטגורי
2. עץ החלטה רציף - משתנה המטרה של עץ ההחלטה הוא מספר רציף

דוגמא לעץ החלטה :
אנחנו מחפשים עבודה וקיבלנו הצעה מחברה מסוימת לבוא לעבוד בה.
אנחנו מאוד מתלבטים אז חבר שלמד קורס בסיס בלמידת מכונה אומר שיש לו פתרון פשוט לבעיה שלנו והוא לבנות עץ החלטה.
החבר בונה לנו עץ החלטה ואנחנו מקווים שהוא יענה לנו על השאלה אם לקבל את ההצעה או לדחות אותה.

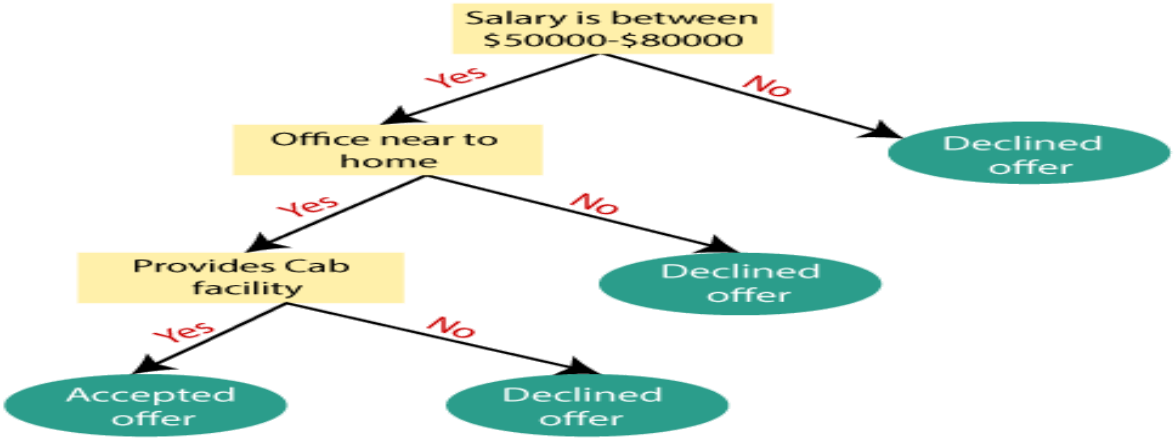
- השאלה הראשונה : האם המשכורת היא בין 50,000-80,000 דולר? אם התשובה היא כן אנחנו פונים שמאלה בעץ אחרת אנחנו פונים ימינה ודוחים את ההצעה.

לשמחתנו התשובה הייתה כן ופנינו שמאלה

- השאלה השנייה : האם העבודה קרובה לבית?, אם התשובה היא כן אנחנו פונים שמאלה בעץ אחרת אנחנו פונים ימינה ודוחים את ההצעה.

למזלנו התשובה היא כן ואנחנו פונים שמאלה

- השאלה השלישית (האחרונה): האם החברה מספקת שירותי הסעות לעבודה אם התשובה היא כן אנחנו פונים שמאלה ומקבלים את ההצעה אחרת אנחנו פונים שמאלה ודוחים אותה.



מה שעשינו עכשיו זה שימוש בעץ החלטה ונתונים (שלוש תשובות כן) על מנת לקבל פרדיקציה האם אנחנו מקבלים את הצעת העבודה או לא.

אנחנו מתלהבים מהרעיון של חבר שלנו שנשתמש בעץ החלטה, אבל אנחנו שואלים אותו איך בכלל בנית אותו?

אלגוריתם לבניית עץ החלטה ID3:

ה ID3 הוא אלגוריתם חמדן שבוחר תמיד את החלוקה שכרגע היא הכי טובה מבחינת entropy ולעולם לא מתחרט על הבחירה שלו.

אנחנו רוצים לדעת איך לבצע את החלוקה ואיזה מאפיין נבחר ראשון, בשביל לעשות זאת עלינו להכיר כמה מושגים חדשים.

- אחת מהמשימות שיש בתורת האינפורמציה זה העברת מסרים בין תחנות מידע בצורה הכי יעילה ע"י שימוש במספר ביטים קטן ככל הניתן, מספר הביטים שבהם אנחנו צריכים להשתמש הוא log(2^uncertainty).

מספר הביטים תלוי במרחב ההסתברות שבו אנו נמצאים המושג אנטרופיה זוהי מדידה שבודקת עד כמה יש חוסר ודאות לגבי אירוע מסוים ובמילים אחרות זה הממוצע של כמות הביטים שאנחנו צריכים להעביר המשוואה של אנטרופיה מוגדרת כך :

$$\mathrm{H}(X) = -\sum_{i=1}^{n} \mathrm{P}(x_i) \log \mathrm{P}(x_i)$$

כאשר אנחנו משתמשים באנטרופיה אנחנו רוצים למצוא את המאפיין בעל ה entropy הנמוך ביותר,

לעומת זאת אם נשתמש ב information gain אנו נרצה לקבל את המאפיין בעל הערך הגבוה ביותר.

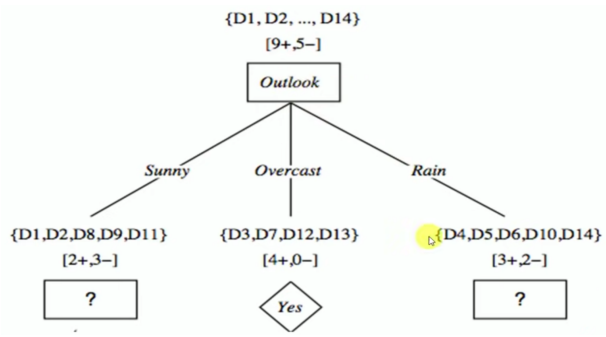
הנוסחה ל information gain היא:

Information Gain = entropy(parent) – [average entropy(children)]

$$\text{Gain}(S,A) = \text{entropy}(s) - \sum_{v \varepsilon value(A)} \frac{|S(v)|}{|s|} * entropy(s(v))$$

אנחנו מחשבים את ה entropy של האבא פחות ממוצע ה entropy של הילדים.

1. עבור על הדאטה סט וחשב לכל תכונה את ה entropy או ה information gain שלה ניקח את ה entopy הכי קטן או את ה information gain הכי גדול והוא יהיה השורש שלנו(בדוגמא שלנו זה המאפיין outlook).

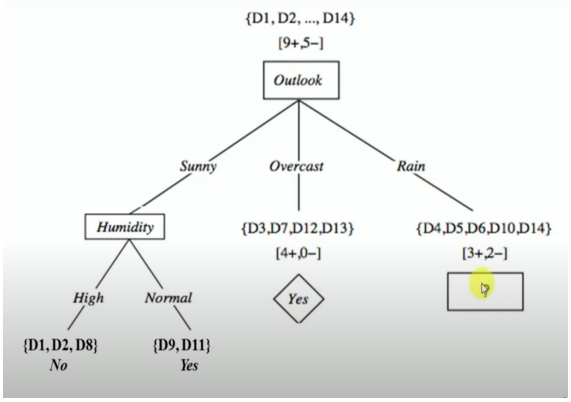


Day	Outlook	Temp	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

2. לשורש יש מספר תכונות (בדוגמה שלנו sunny,outlook,wind), ניקח כל תכונה בנפרד ונבנה דאטה סט חדש.

בדאטה סטים הללו הורדנו את הרשומות שבהן התכונה שנלקחה לא מופיעה.

נבצע על הסאב סט הזה חלוקה ע"י מציאת המאפיין בעל ה entropy הקטן ביותר או ה information gain הגדול ביותר.



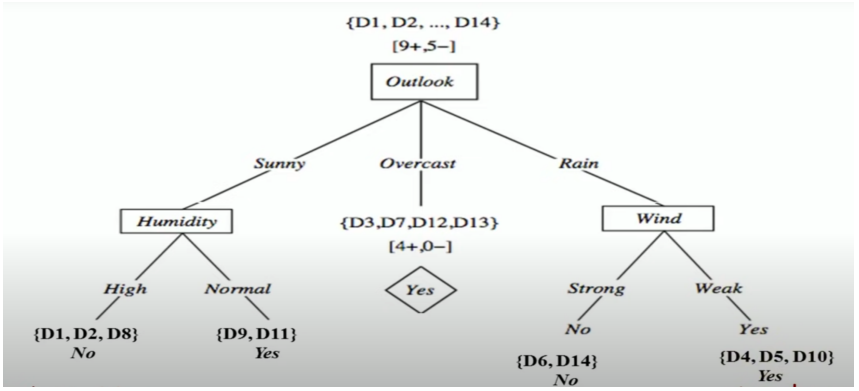
Day	Temp	Humidity	Wind	Play Tennis
D1	Hot	High	Weak	No
D2	Hot	High	Strong	No
D8	Mild	High	Weak	No
D9	Cool	Normal	Weak	Yes
D11	Mild	Normal	Strong	Yes

3. נבצע את התהליך של החלוקה עד שנתקל באחד מתנאי העצירה.

תנאי העצירה הם:

- אם ה subset הוא pure או שאין כבר רשומות אז נעצור.

התוצאה הסופית היא:



חסרונות:

פתרונות:

מצרף קישור למימוש שכתבתי לעץ החלטה (רגרסיה) כמובן נעזרתי במימושים מרחבי האינטרנט, תהנו.

https://github.com/joshkatz1/ml/blob/b0a4efb42c59959a2a6439c10dea639cffe30e3f/decision_tree_from_scratch_regression.py

ensembles

אנסמבל זוהי פרדיגמה מובילה בעולם למידת המכונה, אנו מחברים בין מספר מודלים על מנת לקבל יותר דיוק ויותר רובסטיות, אנסמבלר עוזר לנו להילחם בבעיית ה bias-variance trade off. זה מתבטא ביצירת מודל עם הטיה נמוכה שמקבל תוצאות מדויקות על סט האימון וגם שונות נמוכה שתעזור למודל שלנו לעבוד טוב גם במקרה של דאטה סט חדש ולא מוכר.

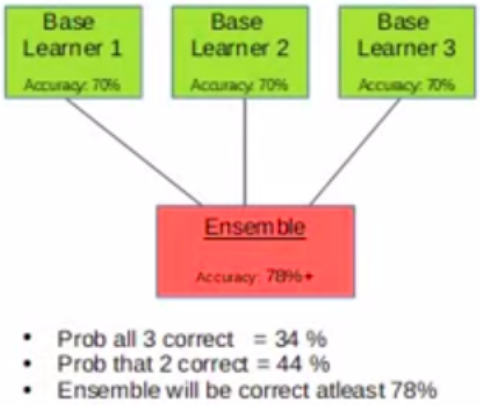
זה נשמע אחלה פרדיגמה אבל איך אני בונה את המודלים הללו?.
אז בואו נציע מספר הצעות:

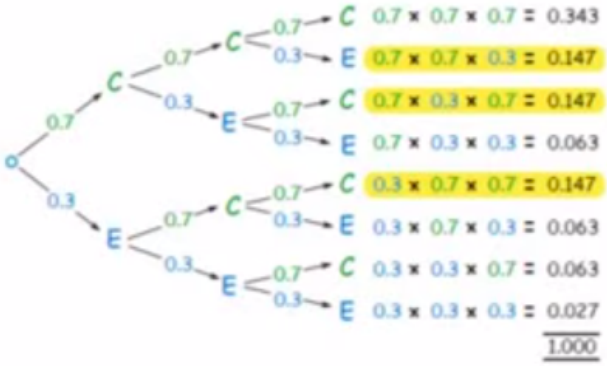
- שימוש בהיפר פרמטרים שונים של אותו אלגוריתם
- אלגוריתמים שונים לחלוטין
- אותו אלגוריתם אבל אימון על דאטה סט שונה

חשוב להבהיר שעל מנת שהפרדיגמה תעבוד המודלים צריכים להיות בעלי קורלציה נמוכה אחד מהשני.

השאלה הגדולה למה זה עובד?

התשובה היא מאוד פשוטה, ניקח שלושה מודלים שכל אחד הוא בעל 70% דיוק, אם נחבר את שלושתם נקבל דיוק של 78%.
מכיוון שההסתברות שכל המודלים צדקו זה 0.7x0.7x0.7 = 0.343.
ההסתברות ששניים מתוך השלוש צדקו זה 0.7x0.7x0.3 x3 = 0.441.
ולכן קיבלנו יותר מ 78 אחוזי דיוק שזה שיפור משמעותי של 8%.





ואם נכליל את האינטואיציה שראינו בדוגמה הזאת לשפה מתמטית פורמלית נקבל התפלגות בינומית.

$$\Pr(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

for $k = 0, 1, 2, \ldots, n$, where

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

שתי הגישות הפופולאריות הן **bagging** ו **boosting**.

- הגישה של bagging היא לקחת מודלים לא יציבים עם הטיה נמוכה לשלב ביניהם לאנסמבל, מה שאנחנו נשיג בזה זה הורדה של השונות ושמירה על הטיה נמוכה.
- הגישה של boosting זה לבנות מודל שמורכב מהרבה מודלים ולעשות זאת בצורה סדרתית בשביל בעיות קשות, ומה שאנחנו מרווחים זה הורדה של ההטיה והשונות.

bagging

המטרה היא לחלק את הדאטה סט ל T חלוקות, וכל אחד מהמודלים מקבל סט אחד מהחלוקה שעשינו ועליו אנחנו בונים ומאמנים אותו.

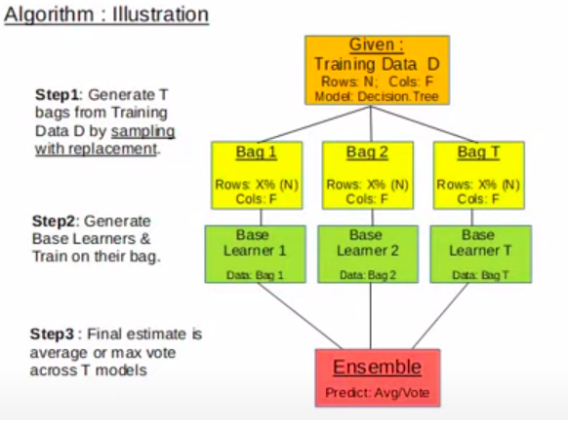
מספר מאפיינים של מודל ה bagging הם:

- שימוש בבוטסטראפ על מנת לייצר את החלוקה החדשה.
- יכול להיות שהמודלים יהיו יותר מדי מתאימים לדאטה הספציפי שהם אומנו עליו מה שמוביל low bias ו high variance
- אנחנו מאפשרים חפיפה בין הדאטה שבהם משתמשים המודלים השונים.

אלגוריתם :

האלגוריתם הוא פשוט ומורכב משני שלבים:

- אנחנו מייצרים שק המכיל דוגמאות אימון מהדאטה סט שיש לנו, אנחנו יכולים לקחת כמה אחוזים שאנחנו רוצים מהשורות, מהעמודות אנחנו צריכים לקחת את כולם, אנחנו יכולים לעשות החלפה ואפילו לקחת דוגמא מסוימת מספר פעמים.
- אנחנו מייצרים מודל ומאמנים אותו על הדאטה סט שבנינו.
- אנחנו חוזרים על הפעולה הזאת T פעמים.
- הפרדיקציה הסופית בבעיות קלסיפיקציה תינתן על פי הצבעת הרוב, ובבעיות רגרסיה על פי הממוצע.



עוד שאלה שנרצה לשאול זה למה למרות שלכל מודל יש שונות גבוהה לאחר שיצרנו אנסמבל השונות קטנה?.

התשובה היא בגלל שאנחנו מניחים שהמודלים הם בלתי תלויים והקורולוציה ביניהם מאוד נמוכה, אז מבחינה מתמטית יצרנו משתנה מקרי X חדש שמכיל הרבה דוגמאות, הממוצע שלו עדיין ישאר μ אבל השונות תקטן ותהיה $\frac{\sigma^2}{T}$.

המגבלות של bagging:

1. אנחנו יודעים שיש סיכוי של 63% שרשומה מהדאטה סט נמצאת בתוך אחד מתיקי החלוקה, מה שאומר שההסתברות שרשומה לא נמצאת באחד מהתיקים היא 37%.
2. עצים שהם מאוד מוכווני דאטה סט ספציפי(overfitting) מובילים לתנודתיות מוזרה.
3. הקורולציה בין העצים היא מאוד גבוה כי הם משתמשים באותו דאטה ואז היכולת לייצר שונות נמוכה נתקלת בקשיים.

בצורה פורמלית זה נראה כך:

$$\sigma^2 \rightarrow \sigma^2 x \rho + \frac{(1-\rho)}{T} x \sigma^2$$

Random Forest

הפתרונות שהאלגוריתם הנ"ל מספק נוגע בשלוש הבעיות המצוינות לעייל:

- בנוסף לזה שהאלגוריתם בוחר דאטה עכשיו הוא גם בוחר פיצ'רים.
- האלגורתים מבצע cross validation על out of bags error במהלך האימון.
- אופציה לכוון את ההיפר פרמטרים כך שנוכל להשתמש ב רגולריזציה כדי להילחם בבעיית ה overfitting.

אלגוריתם

- אנחנו מייצרים T שקים של דוגמאות אימון שנלקחו מהדאטה סט שלנו, ניקח אחוז מסוים מהדאטה, בנוסף נבחר מספר פיצ'רים מתוך כלל הפיצ'רים (כלל אצבע כדי לדעת כמה פיצ'רים לבחור?, זה לקחת את השורש של סך הפיצ'רים).
- נבנה T עצי החלטה כל אחד נבנה ומאומן מהשק של הדאטה שהוא קיבל, נבדוק ואיכות (validation) על שאר הדוגמאות בדאטה סט שנשארו מחוץ לשק הספציפי של אותו עץ החלטה.
- הפרדיקציה הסופית בבעיות קלסיפיקציה תינתן על פי הצבעת הרוב, ובבעיות רגרסיה על פי הממוצע.

שימושים נוספים

1. דאטה עם ערכים חסרים
   - ראשית נשים בערכים חסרים את הממוצע אם זה ערך נומרי או בחירת הרוב עם זה ערך קטגורי.
   - לאחר מכן נריץ את הדאטה סט על כל העצים ונבנה טבלה שבה לכל דוגמה יש שורה ולכל דוגמה יש עמודה (ככה שאם יש n דוגמאות נקבל טבלה של nxn), כאשר שתי דוגמאות מביאות את אותה תוצאה(מסיימות באותו עלה בעץ) נוסיף 1 לטבלה.
   - ניקח את הטבלה ונחלק את הערכים במספר העצים שבנינו.
   - לסיום נחזור לערכים החסרים בטבלה, נשתמש בטבלה שבנינו למצוא ערכים יותר מדויקים מהניחוש הראשוני.
     ערכים קטגוריאלים - נעבור על הטבלה ונחשב מה ההסתברות למופע של כל קטגוריה ואותה נכפול במשקל שיש לדוגמאות הללו בטבלה
     ערכים נומריים - ניקח את הערכים המופיעים בעמודה עם הנתון החסר, נכפיל כל ערך במשקל שלו שמופיע בטבלה, מחיבור כל הערכים נקבל את הערך של הנתון החסר.
   - נעשה את התהליך הזה מספר פעמים עד שנראה שהערכים שאנחנו מקבלים מתכנסים.
2. דוגמה עם ערכים חסרים בתוכה ובנוסף ללא תיוג לערך שאנחנו רוצים לבצע עליו פרדיקציה.
   - הרעיון הוא מאוד פשוט ראשית נכפול את הדאטה במספר הקלאסים שיש לנו בקלאס המטרה
   - לכל העתק של הדאטה נמצא את הערכים החסרים על ידי השיטה שהצגנו לעייל.

- כעת כשיש לנו העתקים של הדאטה והערכים אינם חסרים, נריץ את הדאטה על כל העצים ונראה לאיזה ערך קטגוריאלי יש הכי הרבה הצבעות ואותו ניקח כערך הקטגוריאלי של הדוגמה.

boosting

השיטה הפופולרית הנוספת היא בוסטינג נסביר בהתחלה את הרעיון הכללי ואז נתעמק באלגוריתמים השכיחים בנושא.

הרעיון הכללי:

- אנחנו בונים מודל לחיזוי המבוסס על הדאטה שיש בידינו, לאחר מכן אנחנו בוחנים את המודל על הדאטה סט שלנו, המודל שלנו כנראה נכשל בחלק מהמקרים לחזות את התוצאה האמיתית, לכן נבנה מודל חדש שנותן דגש על הטעויות שהיו למודל הנוכחי,גם את המודל החדש נבחן על הדאטה, נבחן את הטעויות שהיו לו ונבנה מודל חדש שיתמודד ספציפית אם הטעויות שהיו למודל האחרון וכן הלאה, בסופו של דבר אנחנו לוקחים את כל המודלים לכל אחד אנחנו נותנים משקל ואז מחברים את התוצאות של כל המודלים וזה יתן לנו את הפרדיקציה הרצויה.

בצורה פורמלית:

- ניצור מודל f_k
- נחשב את השאריות $\varepsilon_i = \frac{\partial l_i}{\partial f(x_i)}$ כאשר פונקציית ההפסד היא mean squared error

$$\frac{1}{2n} x \sum_{i=1}^{n} (y_i - f(x_i)^2$$

- נבנה מודל חדש $g_k(x_i) \approx \varepsilon_i$
- נגדיר מודל חדש $f_{k+1}(x) = f_k(x) + \gamma_k x\ g_k(x)$

Adaboost

הרעיון הכללי: מה שמיוחד באלגוריתם הזה שהוא לוקח מודלים מאוד מאוד חלשים מחבר ביניהם ואנחנו מקבלים מודל עם תוצאות טובות מאוד, כל מודל מורכב משורש ועוד דרגה אחת (decision stump) וברור שזה אחד מהאלגוריתמים היותר חלשים שאפשר לבנות, אבל בגלל שהאלגוריתם כל הזמן מסתכל על הטעויות שעשה המודל הקודם ומשנה את המשקל שהוא מייחס לכל דוגמה, אז המודל הבא שהוא בונה יודע להתמודד עוד יותר טוב עם הבעיה, וכך הלאה עד שאנחנו מגיעים לתוצאות יפות.

האלגוריתם

1. נאתחל את המשקולות כך ש $w_i = \frac{1}{n}$ לכל i
2. אנחנו נרוץ בלולאה מ T=0 עד (#trees) T=n

- אנחנו מייצרים דאטה סט חדש, המורכב מדגימה של דוגמאות מהדאטה סט שיש לנו, מספר הפעמים שדוגמה תופיע היא ביחס ישר למשקל שלה.
- נבנה מודל נבחר את ה השורש ע"י שימוש ב information gain או ב gini impurity, נריץ את המודל על הדאטה סט שלנו ונקבל פרדיקציות, נחשב את הטעות בין הפרדיקציה לתוצאת האמת.
- נחשב את $\lambda_t = \frac{1}{2} ln(\frac{1-e_t}{e_t})$, $e_t = \frac{\sum_{i=1}^{n} (e_i x w_i)}{\sum_{i=1}^{n} w_i}$, הלמבדה תעזור לנו לקבוע מה רמת המשקל שניתן לכל מודל.

- נחשב את השינוי במשקל של כל דוגמה, עבור דוגמאות שהמודל טעה בהם

$$w_i = w_i \, x \, e^{\lambda_t \, x \, prediction \, x \, y}$$

- לנרמל את wi ל 1
- המודל החדש יהיה $f_t = f_{t-1} + \lambda_t x \, g_t$

3. המודל הסופי יהיה $f_t(x) = sgn(\sum_{t=0}^{t=T-1} \lambda_t \, x \, g_t)$

קישור למימוש מאפס
https://github.com/joshkatz1/ml/blob/ab32b63e1ab6bc292eeedebfe06b156245294fde/ml_supervision_algo/adaboost_from_scratch.py

gradient boosting

אלגוריתם:

1. נאתחל מודל עם ערך קבוע $F_0(x) = argmin \sum_{i=1}^{n} L(y_i, \gamma)$, פונקציית ה loss שנשתמש בה היא הפרשי הריבועים $(\frac{1}{2n} x \sum_{i=1}^{n} (y_i - f(x_i)^2)$ ולכן הערך הקבוע שנקבל זה ממוצע הערכים.
2. נרוץ בלולאה מ m=1 עד m=M (כמות העצים במודל)

- נחשב את ההפרש בין הפרדיקציה לתוצאת האמת, $r_{im} = \frac{\partial L((y_i - F(x_i))}{\partial L(F(x_i)} F(X) = F_{M-1}(X)$
- נבנה עץ החלטה שבמקום לתת פרדיקציה על ערך המטרה הוא נותן פרדיקציה על ההפרשים שחישבת בסעיף הקודם, נגדיר את הנקודות הסופיות(העלים) כ R_{jm} m מוגדר מספר העץ ו j מספר העלה .
- בחלק הזה נעבור על כל העלים בעץ ונגדיר מה הערך הסופי שלהם, מבחינה מתמטית

for j = 1…j_m $\gamma_{jm} = argmin(\sum_{x_i \epsilon R_{ij}} L(y_i, F_{m-1}(x_i) + \gamma))$

כאשר y_i שווה לערך האמת, $F_{m-1}(x_i)$ זה הפרדיקציה של המודל האחרון, ואנחנו רוצים למצוא γ שיתן מינימום לביטוי, לכן נשים את הערכים שיש בידינו בתוך פונקציית ההפסד שהגדרנו לעייל, נגזור ונשווה ל 0 ונקבל את ה γ הרצויה.

- אנחנו נבצע פרדיקציה חדשה לכל דוגמא שיש לנו בדאטה סט, הפרדיקציה תהיה מורכבת מ הפרדיקציה של המודל הקודם ועוד קצב הלמידה כפול הערך שקיבלנו מהעלה בעץ החדש.

בכתב פורמלי זה נראה כך: $F_m(x) = F_{m-1}(x) + \nu \sum_{j=1}^{j_m} \gamma_{jm} I(x \epsilon R_{jm})$

https://github.com/joshkatz1/ml/blob/ec7ef91d06668ea62fbe700292871aa3e051b073/ml_supervision_algo/gradient_boosting.py

xgboost

אלגוריתם:

1. השלב הראשון באלגוריתם זה לתת חיזוי ראשוני לסט האימון שלנו, ניקח את הסיווגים/תוצאת האמת ונבצע עליהם ממוצע, הערך שנקבל יהיה החיזוי הראשוני שאנחנו נותנים לסט שלנו.
2. נרוץ בלולאה t=0 עד t=T (מספר העצים)

פונקציות ההפסד הנמצאות בשימוש נרחב הם:

רגרסיה - הפרשי הריבועים $(\frac{1}{2n} x \sum_{i=1}^{n} (y_i - f(x_i)^2)$

קלסיפיקציה- $y_i \, x \, log(p_i) \; + \; (1 - y_i) \, x \, log(1 - p_i)$

- נחשב את הנגזרת של פונקציית ההפסד שלנו ונציב בה את ערכי האמת ואת הפרדיקציה שלנו.
- נחשב את הנגזרת השנייה של פונקציית ההפסד שלנו ונציב בה את ערכי האמת ואת הפרדיקציה שלנו.
- נבנה עץ החלטה, פה זה החלק המרכזי,
- ראשית אנחנו רוצים לבצע כמה מהלכים שיעזרו לנו להילחם בהתאמת יתר, לכן המהלך הראשון בבנייה של העץ יהיה לקחת רק חלק מ הפיצ'רים.
- בתחילתו של כל צומת נחשב מה הערך האופטימלי שהצומת אמורה לקבל אם היא תהיה עלה, זוהי המשוואה לחישוב פונקציית ההפסד,

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n} [g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i)] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} w_j^2$$
$$= \sum_{j=1}^{T} [(\sum_{i \in I_j} g_i) w_j + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) w_j^2] + \gamma T$$

כדי לקבל את ערך הפלט המינימלי נגזור ונשווה ל 0 ונקבל את המשוואה הזאת,

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda},$$

נציב את הערכים ונקבל את הערך המינימלי.

- בשלב הבא אנחנו נעשה סריקה על כל עמודה ונבדוק מה הוא הערך הטוב ביותר לבצע בו את החלוקה, אנחנו נשתמש באלגוריתם חמדן למציאת הערכים הבאים: העמודה , הערך בתוך העמודה, והתוצאה שלנו כאשר בחרנו בערך הספציפי הזה, אחרי שנעבור על כל העמודות נקבל בחזרה את הערכים הללו.
נחשב לכל ספליט את התוצאה שלו ע"י הנוסחה הבאה:

$$G_L \leftarrow G_L + g_j, \; H_L \leftarrow H_L + h_j$$
$$G_R \leftarrow G - G_L, \; H_R \leftarrow H - H_L$$
$$score \leftarrow \max(score, \frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{G^2}{H + \lambda})$$

ואנו ניקח את הערך בעל התוצאה הגבוהה ביותר.
- תנאי העצירה שלנו יהיה אם הצומת שקיבלנו היא עלה
- אם הצומת היא לא עלה, נבצע את החלוקה וכך נקבל את האינדקסים של הערכים שהולכים לצומת שמאל(הערך שלהם קטן מהערך של האיבר שמבצע את החלוקה), ואינדקסים של ערכים שהולכים לצומת ימין.
- נבצע רקורסיה ע"י קריאה לפונקציה שמייצרת לנו צומת, והיא תקבל כפרמטר את האינדקסים לצומת השמאלי שחישבנו בשלב הקודם
- נבצע רקורסיה ע"י קריאה לפונקציה שמייצרת לנו צומת, והיא תקבל כפרמטר את האינדקסים לצומת הימני שחישבנו בשלב הקודם
- בסוף הרקורסיה נקבל עץ החלטה

- נעדכן את הפרדיקציה הראשונית, ניקח את העץ שבנינו ונשתמש בו לחיזוי התוצאות על הדאטה סט שלנו, את החיזויים שקיבלנו נכפיל בקצב הלימוד שלנו, ואת התוצאות נחבר לפרדיקציה הקודמת.

```
base_pred += self.learning_rate * xgboost_tree.predict(self.X)
```

- נוסיף את עץ ההחלטה לרשימת העצים

3. נעבור על כל העצים שברשימה שבנינו ונשתמש בהם לחיזוי התוצאות על סט המבחן כל חיזוי יוכפל בקצב הלמידה שלו, נחבר בין התוצאות של כולם ונוסיף להם את הפרדיקציה הראשונית, וכך אנחנו מקבלים את הפרדיקציה הסופית.