למידה חישובית ממידע: הרצאה 6 – Linear Classifiers and Logistic Regression

Logistic Regression (LOR)

זהו אלגוריתם קלסיפיקציה, בניגוד לרגרסיה לינארית אשר מניב ערכים רציפים לכל דגימה, אלגוריתם זה מניב סיווג בדיד. כמו בכל מסווג,

הדאטה אימון במקרה הבינארי, מכילה וקטורי פיצ'רים ואת הלייבלים המתאימים (labels) : $\{(\vec{x}^{(i)}, y^{(i)})\}_{i=1..m}$

אנחנו נלמד מודל שהוא יהיה פונקציה h המקבלת וקטור פיצ'רים x (בממד k למשל) כך ש-h מקיימת : $h\colon \mathbb{R}^k \to \mathbb{R}$

מודל logistic regression חשוב לשיטות למידה מודרניות יותר וכן עבור deep learning.

אנחנו נרצה שהפונקציה שנמצא, h, תחזיר עבורנו הסתברויות כך שלמעשה היא תחזיר ערכים מהקטע [0,1] ולאו דווקא מ-R.

ב-**<u>logistic regression אנחנו מניחים את ההנחה המרכזית הבאה:</u>**

נניח כי ההתסתברות P (Y | X) (ההסתברות להיות משוייך ללייבל מסוים Y בהינתן דגימה X) יכולה להיות משוערכת כפונקציית sigmoid המיושמת על קומבינציה לינארית על ה-input features. באופן מתמטי, עבור נקודת דאטה $(\vec{x}, y)$, אנו מניחים ב-LoR כי :

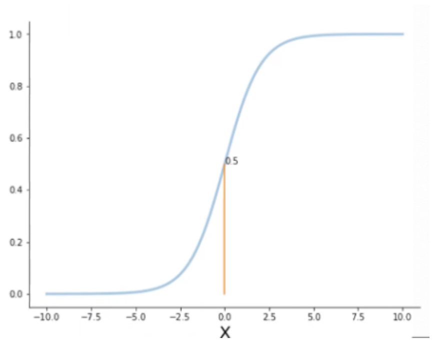
כך ש $P(Y=1 \mid X=\vec{x}) = \sigma(z)$ - $\sigma(z) = \dfrac{1}{1+e^{-z}}$ (וזהי פונקציית ה-sigmoid), ו - $z = \theta_0 + \sum_{j=1}^{k} \theta_j x_j$ (קומבינציה לינארית של הוקטור x המקורי)

לכן z הוא למעשה סקלאר, מספר ממשי, מפני שיש כאן מכפלה פנימית של הוקטור טטה עם הוקטור x).

ההצבה היא להלן, כאשר השוויון השני מתקיים מפני שאנחנו מכפילים מונה ומכנה ב e ^ ((theta transpose) * x)

$$\sigma(\theta^T x) = \frac{1}{1+e^{-\theta^T x}} = \frac{e^{\theta^T x}}{1+e^{\theta^T x}}$$

- הפונקציה הזו מונוטונית עולה ממש
- הטווח של הפונקצייה הוא [0,1], הפונקציה ראשית כל חיובית, המכנה גדול מהמונה ולכן קטנה מ-1.
- הגבולות באינסוף : 0 במינוס אינסוף ו-1 באינסוף.



תכונה נוספת של פונקציית סיגמויד שיכולה לסייע הינה :

Derivative:
$\sigma'(x) = \sigma(x)(1-\sigma(x))$

מכאן, שנוכל להפוך את פונקציית ההיפותזה שלנו / המודל שלנו לצורה הבאה :

$$\boxed{h_\theta(\vec{x}) = P(y=1 \mid \vec{x}) = \sigma(\langle \theta, (1, \vec{x}) \rangle)}$$

מכפלה פנימית של טטה עם הרחבה של הוקטור איקס בתוך הפונקציה של סיגמויד. כאשר השוויון השני נובע מהנחת ה-logistic regression.

כאשר הפרדיקציה / הקלסיפיקציה תערך באופן הבא :

$$\begin{array}{ll} \text{if } h_\theta(\vec{x}) > 0.5 & \text{then } 1 \\ & \text{else } 0 \end{array}$$

עלינו ללמוד את הטטות שמניבות את הניבוי הטוב ביותר – נבצע תהליך למידה עבורן. טטה הוא וקטור פרמטרים באורך k+1 (למען טטה0),

אנחנו נלמד את הערכים שלה תחת מסגרת **maximum likelihood**. לשם כך נשתמש ב-training data set שנסמן אותה ב-D שיש בה m דגימות.

כל דגימה תכיל וקטור של פיצ'רים x ממימד k וערך בינארי y שנקרא label.

Maximum Likelihood

ראשית, נשים לב כי תחת ההנחה המרכזית שאנו מניחים ב-LoR, מתקיים עבור נקודת דאטה יחידה :

$$P(y \mid x\,;\theta) = \left(h_\theta(x)\right)^y \cdot \left(1-h_\theta(x)\right)^{1-y}$$ או : $$P(x, y;\theta) = \left(h_\theta(x)\right)^y \cdot \left(1-h_\theta(x)\right)^{1-y} P(x)$$

אבל זה מתקיים עבור נקודת דאטה אחת, לכן כעת עבור הדאטה אימון D עם m דגימות שנניח כי הן בלתי תלויות, נרצה למצוא את המודל שממקסם את ה-likelihood. עבור כל טטה, הלייקליהוד שלה בהינתן הדאטה, P (D | theta), היא ביחס ל-

נרצה למצוא את הטטה שמביאה זאת למקסימום.

$$\prod_{d=1}^{m} P\left(y^{(d)} \mid x^{(d)}\,;\theta\right) =$$

$$\prod_{d=1}^{m} \left(h_\theta\left(x^{(d)}\right)\right)^{y^{(d)}} \cdot \left(1-h_\theta\left(x^{(d)}\right)\right)^{1-y^{(d)}}$$

לכן נגזור ונשווה ל-0. השורה הראשונה מועתקת מהעמודה הקודם. בשורה השניה אנחנו רוצים לבטל מכפלות כדי שיהיה קל יותר לכפול ולכן אנו מפעילים לן (מונוטונית עולה ולכן איננה משנה את המקסימון). בשורה השלישית המכפלה הפכה לסכום והופכל לן נוסף על שתי החזקות כדי שנוכל להוריד את y(d) מהחזקה – שורה רביעית.
ובשורה החמישית והאחרונה אנחנו הופכים סימן למינוס כדי למצוא את ה-minimum.
כי אז ניתן לחשוב על זאת במונחי cost.

$$\underset{\theta}{argmax}\prod_{d=1}^{m}\left(h_\theta(x^{(d)})\right)^{y^{(d)}}\cdot\left(1-h_\theta(x^{(d)})\right)^{1-y^{(d)}}=$$

$$\underset{\theta}{argmax}\ln\left(\prod_{d=1}^{m}\left(h_\theta(x^{(d)})\right)^{y^{(d)}}\cdot\left(1-h_\theta(x^{(d)})\right)^{1-y^{(d)}}\right)=$$

$$\underset{\theta}{argmax}\sum_{d=1}^{m}\ln\left(\left(h_\theta(x^{(d)})\right)^{y^{(d)}}\right)+\ln\left(\left(1-h_\theta(x^{(d)})\right)^{1-y^{(d)}}\right)=$$

$$\underset{\theta}{argmax}\sum_{d=1}^{m}y^{(d)}\cdot\ln\left(h_\theta(x^{(d)})\right)+(1-y^{(d)})\cdot\ln\left(1-h_\theta(x^{(d)})\right)=$$

$$\underset{\theta}{argmin}\sum_{d=1}^{m}-y^{(d)}\cdot\ln\left(h_\theta(x^{(d)})\right)-(1-y^{(d)})\cdot\ln\left(1-h_\theta(x^{(d)})\right)$$

לכן נרצה להביא למינימום את: (למדא של טטה)

$$\Lambda(\vec{\theta})=-\sum_{d=1}^{m}y^{(d)}\ln\left(\sigma(\theta,\vec{x}^{(d)})\right)+(1-y^{(d)})\ln\left(1-\sigma(\vec{\theta},\vec{x}^{(d)})\right)$$

נציין כי כאן לא עובד לחשב גרדיאנט ולהשוות ל-0 (חישוב אספליסיטי לא יעבוד כאן).
לכן נשתמש **בגרדיאנט דיסנט**.
<u>נוכל להראות כי: זהו הגרדיאנט (וקטור בגודל k+1)</u>

$$\frac{\partial}{\partial\theta_j}\Lambda(\vec{\theta})=\sum_{d=1}^{m}\left(\sigma(\vec{\theta},\vec{x}^{(d)})-y^{(d)}\right)x_j^{(d)}$$

הנגזרת הכיוונית לפי טטהj המחושבת בנקודה טטה שהיא וקטור k+1 ממדי.

מכאן, נוכל להתחיל להריץ גרדיאנט דיסנט מפני שכל מה שכתוב בגרדיאנט נמצא ברשותנו (בהנחה שיש לי ניחוש התחלתי לוקטור טטה).

<u>אלגוריתם גרדיאנט דיסנט עבור Logistic Regression</u>

- נאתחל את טטה להיות ערך התחלתי קטן / ניחוש
- נחזור על התהליך הבא עד אשר נגיע להתכנסות:

נבצע עידכון של הווקטור טטה באופן הבא:

$$\theta_j^{New}=\theta_j^{Old}-\alpha\frac{\partial\Lambda}{\partial\theta_j}(\theta^{Old},X,y)$$

או באופן יותר אקספליסיטי, אם נציב את הגרדיאנט שהראינו לעיל:

$$\theta_j^{New}=\theta_j^{Old}-\alpha\cdot\sum_{d=1}^{m}\left(\sigma(\vec{\theta},\vec{x}^{(d)})-y^{(d)}\right)x_j^{(d)}$$

- כאשר אלפא הוא ה-learning rate

חישוב הגרדיאנט של למדא (שכבר ראינו לעיל) – הוספת סיגמויד (שמאל) ולאחר מכן גזירה לפי טטהj (ימין):

$$\boxed{\Lambda(\vec{\theta})=-\sum_{d=1}^{m}y^{(d)}\ln\left(\sigma(\theta,\vec{x}^{(d)})\right)+(1-y^{(d)})\ln\left(1-\sigma(\vec{\theta},\vec{x}^{(d)})\right)}$$

$$\ln(\sigma(\vec{\theta},\vec{x}^{(d)}))=\ln\left(\frac{1}{1+e^{-\theta^Tx^{(d)}}}\right)=-\ln\left(1+e^{-\theta^Tx^{(d)}}\right)$$

$$\begin{aligned}\ln(1-\sigma(\vec{\theta},\vec{x}^{(d)}))&=\ln\left(1-\frac{1}{1+e^{-\theta^Tx^{(d)}}}\right)\\&=\ln\left(\frac{1+e^{-\theta^Tx^{(d)}}}{1+e^{-\theta^Tx^{(d)}}}-\frac{1}{1+e^{-\theta^Tx^{(d)}}}\right)\\&=\ln\left(\frac{e^{-\theta^Tx^{(d)}}}{1+e^{-\theta^Tx^{(d)}}}\right)\\&=-\theta^Tx^{(d)}-\ln\left(1+e^{-\theta^Tx^{(d)}}\right)\end{aligned}$$

IDC HERZLIYA

$$\frac{\partial}{\partial\theta_j}\left(-\sum_{d=1}^{m}y^{(d)}\theta^Tx^{(d)}+\ln\left(1+e^{-\theta^Tx^{(d)}}\right)\right)$$

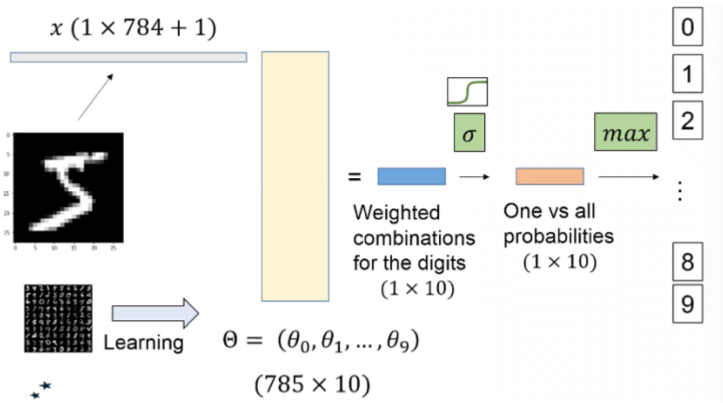
$$=-\sum_{d=1}^{m}y^{(d)}x_j^{(d)}+\sum_{d=1}^{m}\left(\frac{1}{1+e^{-\theta^Tx^{(d)}}}\right)x_j^{(d)}$$

$$\boxed{\frac{\partial}{\partial\theta_j}\Lambda(\vec{\theta})=\sum_{d=1}^{m}\left(\sigma(\vec{\theta},\vec{x}^{(d)})-y^{(d)}\right)x_j^{(d)}}$$

<u>חזרה על עקרונות ה-Logistic Regressions שלדמנו עד כה:</u>

- פונקציית ההיפותזה או המודל, על פי הנחת LoR הינה: $h_\theta(\vec{x})=P(y=1|\vec{x})=\sigma(\theta^T\vec{x})$
- פונקציית המחיר / השגיאה, שנובעת מחישוב ההסתברות P (D | theta) הינה: $\Lambda(\theta)=\frac{1}{m}\sum_{d=1}^{m}-y^{(d)}\cdot\ln\left(h_\theta(x^{(d)})\right)-(1-y^{(d)})\cdot\ln\left(1-h_\theta(x^{(d)})\right)$
- נשתמש בגרדיאנט דיסנט במטרה למצוא את: $\underset{\theta}{\text{argmin}}\ \Lambda(\theta)$

דוגמה : דאטה שנקרא MNIST Digits ושמשמעותו ספרות שכותבים בני אדם. כל ספרה מוצגת בגודל 28x28 פיקסלים ונרצה למצוא מודל שמזהה את הספרה הכתובה. אנחנו ניקח כל פיקסל ונשטח אותו לוקטור אחד, **אינפוט וקטור x,** (28x28 = 784) , שהוא שורה אחת ו-1+784 תאים כלומר 785. לכל ספרה למדנו מודל = טטה,ששואל את השאלה, האם זה 0 או לא? האם זה 1 או לא?... כלומר כל אחת מהטטות שואלת שאלה בינארית האם הספרה שמופיעה בתמונה היא i לכל i בין 0 ל-9 או שלא. לכן קיבלנו 10 טטות. התוצאה היא **מטריצה (הצהובה)** שגודלה 785 שורות ו-10 עמודות, 785 שורות מפני שכל טטה מקבלת אינפוט שהוא 785. נבצע מכפלה פנימית בין הוקטור x של התמונה ה"משוטחת" לבין המטריצה הזו ונקבל **וקטור בעל 10 ערכים (1x10)** ונכניס אותו לתוך פונקציית סיגמויד (המעבר בין הכחול לכתום) ונקבל וקטור כתום שהוא גם כן בגודל 1x10 אבל הערכים בו הם בין 0 ל-1, אלה הן ההסתברויות : כמה המודל חישב שהסיכויים שמדובר במספר 0, הסיכויים שמדובר ב-1, הסיכויים שמדובר ב-2 וכו... ומתוך הוקטור הזה ניקח את המיקום בו ההסתברות היא מקסימלית ולפיכך נסווג את התמונה לקלאס מ-0 עד 9. הוקטור הכתום הוא למעשה כבר ה-posterior, האלגוריתם LoR מגיע ישירות ל-posterior. גישה זו נקראית one vs all. ומספר הטטות שנלמד הוא כמספר הקלאסים (אם עלינו ללמוד אותיות אנגליות קטנות היינו לומדים 26 טטות)



לסיכום,

- LoR היא גישה לקלסיפיקציה בעלת שם מטעה (כי ברגרסיה לינארית לא ביצענו סיווג).
- האלגוריתם המבצע (או המודל) מסווג בהתבסס על וקטור בעל פיצ'רים נומריים.
- הגישה מבוססת על הנחת ה-LoR שהינה $P(y=1|\vec{x}) = \sigma(\theta^T \vec{x})$
- תהליך הלמידה משתמש בגרדיאנט דיסנט כדי למצוא טטה המניבה מקסימום לייקליהוד / מינימום טעות בהינתן ה-observed data, תחת הנחת ה-LoR.
- אין אפשרות להשתמש ב-pseudo inverse ב-LoR.
- ניתן להשתמש באלגוריתם זה עבור קלאסים רבים (כפי שראינו בדוגמה לעיל one vs all).

קלסיפיקציה לינארית – Linear Classifiers

הערה : מסווג בייסיאני יכול לסווג גם משתנים קטגוריאליים וגם רציפים, LoR לעומת זאת מסווגת לפי פיצ'רים נומריים בלבד.

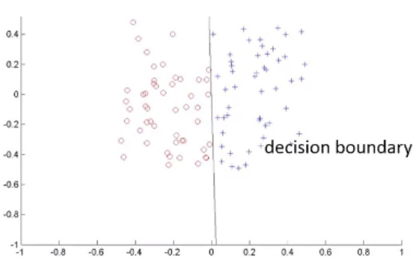
מפרידים לינאריים – נניח כי מרחב הדגימות שלנו הוא R^2 (כלומר 2 פיצ'רים). כל דגימה היא נקודה. אם יש קו המפריד את שני הקלאבים (או קו המפריד בין קונספטים) אזי שני הקלאסים הם **ניתנים להפרדה לינארית** (או : הקונספט ניתן להפרדה לינארית).

ב-R^2 נייצג קו על ידי משוואת קו ישר : $f(x,y) = w_1x+w_2y+w_0 \quad (=0)$

(דיכוטומיה = דאטה עם עיגולים ופלוסים כפי שרואים באיור)

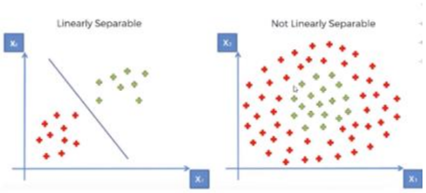
דיכוטומיה ב-R^2 ניתנת להפרדה ע"י קו אם :

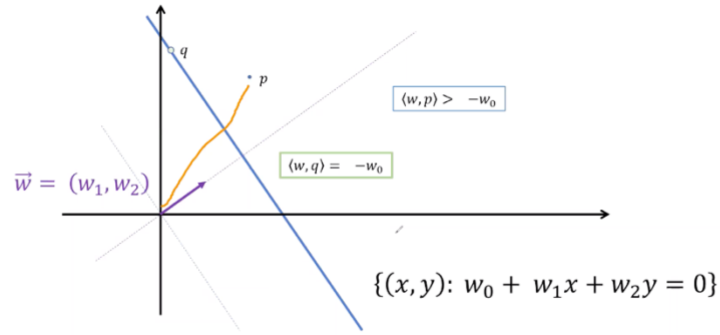
$$f(x,y)>0 \Rightarrow +1,$$
$$f(x,y)<0 \Rightarrow -1$$



2D Dichotomy Example

המשמעות הגיאומטרית עבור המשוואה שבה בחרנו לייצג קו ישר :





Linear Discriminant Functions

בהכללה כללית יותר, ב-R^n, כאשר g(x) היא לינארית היא מגדיאה היפר-מישור ע״י ההשוואתה ל-0 : $g(x) = w^Tx + w_0 = 0$

w יקרא וקטור המשקולות ו-w0 הינו ה-bias. נשים לב כי נוכל לחשוב על כך כמכפלה פנימית של (w0,w) עם (1,x). והסיווג יתבצע באופן הבא :

$$\begin{cases} 1 & \text{if } w_0 + w_1x_1 + \ldots + w_nx_n > 0 \\ -1 & \text{otherwise} \end{cases}$$

בהינתן שקיים דאטה כזה שהוא ניתן להפרדה לינארית, נרצה ללמוד את w שיקנה עבורנו מסווג.

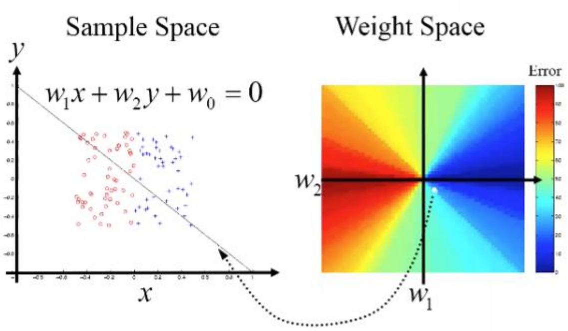
לכן עלינו למזער איזשהי פונקציית טעות ולמצוא את ה-w-ים אשר ממזערים אותה.

ננסה ללמוד את ה-w-ים אשר מביאים למינימום את השגיאה הריבועית על כל דגימות הדאטה (D הוא הדאטה סט) : $E(\vec{w}) = \frac{1}{2}\sum_{d\in D}(\vec{w}\cdot\vec{x}_d - t_d)^2$

פונקציית הטעות מקבלת וקטור w ומה שהיא עושה היא לסכום את : המכפלה הפנימית של הוקטור w עם וקטור הפיצ׳רים xd פחות הלייבל td של הנקודת דאטה הזו (זה למעשה yd וסתם סומן באופן שונה) בריבוע. זו פונקצייה המזכירה סכום ריבועים פחותים. זהו למעשה בדיוק MSE לרגרסיה לינארית! לקחנו את שאלת הקלסיפיקציה שלי והפכתי אותה לשאלת רגרסיה לינארית. ולכן גם נשתמש כאן בגרדיאנט דיסנט. ומכאן נקבל את האלגוריתם – LMS Classification Algorithm.

LMS Classification Algorithm

סימונים : אטה (נראית כמו n) תסמן את ה-learning rate שלנו (למשל 0.05), ו-D יסמן את הסט של דגימות ה-training. כל דגימת אימון הינה זוג מהצורה <x,t> כאשר x הוא וקטור הפיצ׳רים ו-t הוא ה-label או ה-target output value.

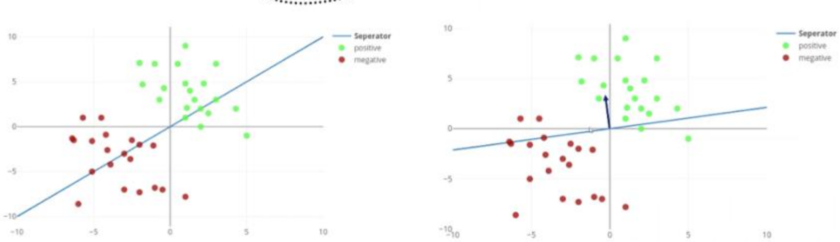


- נאתחל ערכים עבור וקטור המשקולות w
- נחזור עד התכנסות על הצעד הבא :

-- לכל וקטור פיצ׳רים xd בדאטה-סט D נחשב : $o_d = \vec{w}\cdot\vec{x}_d$

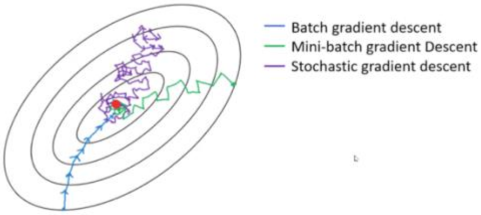
$$\Delta w_i = -\eta \sum_{d\in D}(o_d - t_d)x_{id}$$

-- ועבור כל יחידת משקל לינארית wi, נבצע : $w_i = w_i + \Delta w_i$

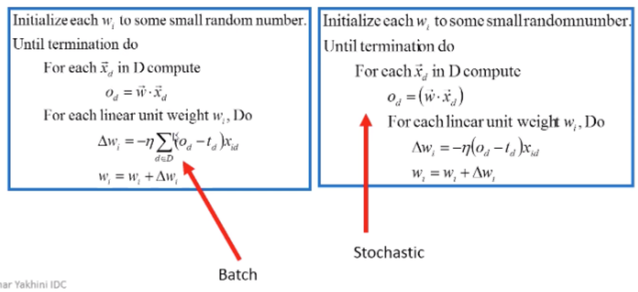


דוגמה לתהליך הלמידה: משמאל הדאטה שיהיה עלינו ללמוד (הירוקים מול האדומים) ואינו כיצד תהליך הלמידה משנה את כיוון המפריד, כך שהחץ המאונך בדוגמה מימין (שלב מסויים מתהליך הלמידה) מייצג את וקטור המשקלים הלינאריים w.

Stochastic Vs. Standard Gradient Dsecent



בגרדיאנט דיסנט סטנדרטי טעויות נסכמות על כל הדגימות לפני העידכונים : $E[\vec{w}] \equiv \frac{1}{2}\sum_{d\in D}(o_d - t_d)^2$.
לפעמים ה-training set הוא מאוד גדול ואנחנו לא נרצה לחכות לעדכון עד שנריץ לולאה על כל הדגימות. בגרסה הסטוכסטית של האלגוריתם, משקלים מתעדכנים על פי **דגימה אחת** רנדומלית או על **פי קבוצה קטנה של דגימות mini-batch**. המשמעות היא שדרושים פחות חישובים לפני כל עידכון, אבל גם דרושים צעדים קטנים יותר מכיוון שלא עושים שימוש בגרדיאנט האמיתי. הגישות הסטוכסטיות יכולות להתמודד יותר טוב עם הבריחה ממינימום לוקאלי וכן מאפשרות מיקבול.
מה שראינו עד עכשיו היא גישת ה-Batch (חישוב הטעות על כל הדאטה).



<u>הערה נוספת</u>: גישת ה-psuedo inverse (או pinve) גם כן עובדת עבור LMS

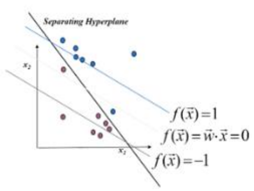
$$\mathbf{X}\cdot\vec{\mathbf{w}} = \vec{\mathbf{t}}$$

$$\begin{pmatrix} x_{10} & x_{11} & \cdots & x_{1n} \\ x_{20} & x_{21} & \cdots & x_{2n} \\ \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ x_{m0} & x_{m1} & \cdots & x_{mn} \end{pmatrix} \begin{pmatrix} w_0 \\ \vdots \\ w_n \end{pmatrix} = \begin{pmatrix} t_1 \\ t_2 \\ \\ \\ t_m \end{pmatrix}$$

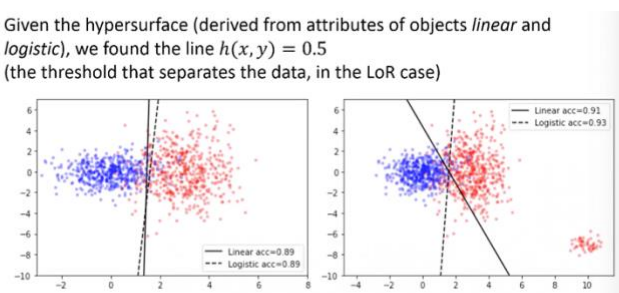
ar Yakhini IDC

$$E[\vec{w}] \equiv \frac{1}{2}\sum_{d\in D}(t_d - o_d)^2 = \frac{1}{2}\left[\sum_{d\in D^-}(\vec{w}\cdot\vec{x}_d - 1)^2 + \sum_{d\in D^+}(\vec{w}\cdot\vec{x}_d + 1)^2\right]$$

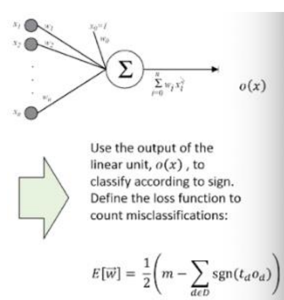
<u>מהן הבעיות באלגוריתם LMS? ננסה למנות אותן באמצעות הדוגמה הבאה:</u>
נפעיל את האלגוריתם על הדאטה הכחול-סגול שלנו ונקבל את הקו האפור הבהיר. ולמעשה הוא איננו מפריד מושלם! יש טעויות, ישנה נקודה סגולה בין כחולים ונקודה כחולה בין סגולים! אבל הדאטה אכן ניתן להפרדה לינארית – להלן הקו השחור! אבל למה ההפרדה איננה מושלמת? זאת מכיוון שהגדרנו פונקציית טעות שאיננה סופרת מיס-קלסיפיקציות! אלא רק מגיעה לסוג של איזון, רגרסיה, מרחקים "טובים" מספיק בין הנקודות להפרדה.



<u>נשווה בין LMS לבין LoR:</u>
נפעיל את שני האלגוריתמים על שני דאטה-סטס, כך שהקלסיפיקציה עבור שני קבוצות הדאטה היא כחול-אדום, ונחשב את הדיוק של כל אחד מהאלגוריתמים (בכמה אחוזים צדקנו). בדאטה השמאלי שנוצר משני גאוסיאנים אחוז הדיוק זהה 89% עבור שני האלגוריתמים. לעומת זאת בדאטה הימני אשר נוצר מגאוסיאן כחול אחד ושני גאוסיאנים אדומים, אחוז הדיוק של LoR גובר על אחוז הדיוק של המסווג הלינארי LMS. הפונקציה הלינארית רואה את הענן האדום הקטן כ-1 או 1-, לכן היא לא "מסתובבת" יותר מידי ולכן מתווספות אליה טעויות.

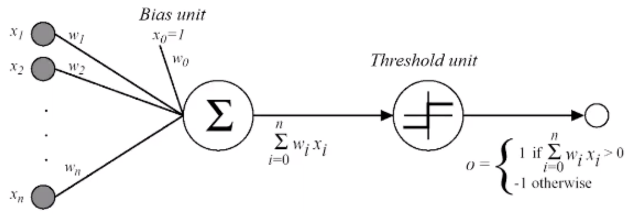


לכן, נרצה לשפר את LMS ולכפות עליו למזער טעויות: נרצה למצוא פונקציית טעות שאכן סופרת טעויות.
הפונקציה מימין: m הוא הגודל של D, ונרצה לספור את כמות הטעויות. הסכום על פונקציית הסימן אשר מבצעת מכפלה בין הלייבל ובין הפרדיקציה שלנו (כאשר אנו מסווגים בין 1 ל-1-), תסכום ל-m אם אין טעויות מפני שמינוס מינוס יניב פלוס, ופלוס פלוס יניב פלוס, וכל מיס קלסיפיקציה תיספר כמינוס. לכן ברגע שנבצע m פחות הסכום הנ"ל, ככל שהסכום יותר קרוב ל-m, יש פחות טעויות, וככל שנתרחק מ-m, האלגוריתם שלנו יבצע יותר מיס-קלסיפיקציות. השאלה היא איך ניתן לעבוד ולמזער את הפונקציה הזו (שמחזירה ערכים בין 0 טעויות ל-m טעויות) שכן לגזור את פונקציית הסימן לא תניב תוצאה יפה. מה שמביא אותנו ל-***Perceptron***.



The Perceptron

נרצה לקחת את האינפוטים x1,…,xn ולבצע מכפלה פנימית עם וקטור המשקלים w, לבדוק אם הוא מניב תוצאה חיובית או שלילית ולסווג בהתאמה לקלאס 1, -1. עלינו ללמוד w-ים שיביאו למינימום את הפונקציה הזו.
להלן האלגוריתם הלומד:



$$o(x_1,\ldots,x_n) = \begin{cases} 1 & \text{if } w_0 + w_1x_1 + \ldots + w_nx_n > 0 \\ -1 & \text{otherwise} \end{cases}$$

Or in vector notations: $$o(\vec{x}) = \begin{cases} 1 & \text{if } \vec{w}\cdot\vec{x} > 0 \\ -1 & \text{otherwise} \end{cases}$$

The Perceptron Algorithm

אנחנו נניח כי הלייבל, ה-target value עבוא הקלסיפיקציה הינו 1 או -1.
ואלגוריתם זה הינו דומה מאוד ל-LMS – אך מטרתו היא למצוא את המפריד הליניארי ללא טעויות! וכן פרספטרון הוא תמיד סטוכסטי, אין פרספטרון אשר משתמש בדגימה אחת או קבוצה קטנה של דגימות (אין batch). האם הדבר תמיד מתאפשר?

- נאתחל את ערכי וקטור המשקלים w
- נבצע עד התכנסות:

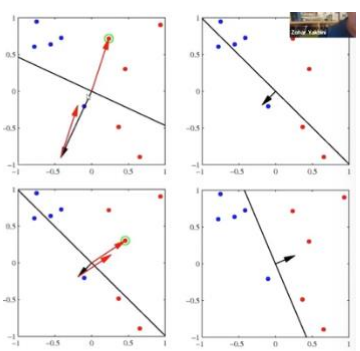
-- לכל וקטור פיצ׳רים xd ב-D נחשב: $o_d = \text{sgn}(\vec{w}\cdot\vec{x}_d)$

-- ולכל יחידת משקל לינארית wi נבצע: $\Delta w_i = -\eta(o_d - t_d)x_{id}$, $w_i = w_i + \Delta w_i$

For any misclassified (at the present iteration) training instance, x, with $C(x) = +1$ we update the weights as:
$w = w + \eta x$
41

<u>הערה:</u> כאשר בחישוב הדלטא w נשים לב כי od-td שווה ל-0 רק כאשר אין טעות! ולכן אין לנו למה לבצע שינוי / לתקן. למעשה העידכון הזה 0 כאשר אנחנו צודקים, כאשר אנחנו טועים ולמשל התקבל 2, אזי w כולו כאשר אנחנו טועים אנחנו מוסיפים ״חתיכה״ של xd. אז איך בעצם פונקציית הטעות משתפרת בכל איטרציה?

נדגים כיצד פרספטרון משפר את פונקציית הטעות:
נשים לב ששלבי השיפור הם: השלב הראשון הוא הגרף השמאלי העליון, השלב השני הוא הגרף הימני העליון, השלב השלישי הוא הגרף השמאלי התחתון והשלב הרביעי בו הגענו למסווג מושלם הוא הימני התחתון. כאשר הוקטור w והשינוי שלו מיוצג על ידי החץ השחור. והחצים האדומים מייצגים את הוקטורים xd אשר מצביעים לנקודת דאטה, ו״חלק״ מהם על פי הטעות מתווספים ל-w.



Red dots are positive ($t = +1$)
The marked one is initially mis-classified
Recall that we then update by:
$$w^{(j+1)} = w^{(j)} + \eta x$$
We therefore get, for a misclassified x with $C(x) = +1$:
$$w^{(j+1)}\cdot x > w^{(j)}\cdot x$$

<u>משפט רוזנבלאט לאלגוריתם הפרספטרון:</u> אלגוריתם הלמידה פרספטרון מתכנס למסווג מושלם (ללא טעויות על ה-training data) אם ורק אם ה-training data, D, ניתנת להפרדה לינארית.

- Note: we also need to control η to really guarantee convergence
  (if its too big we may overshoot the perfect classifier)
- Some results on the rate of convergence were proven and can be useful in the context of ANNs
- The Perceptron itself is not a practical learning approach but is an important component of many modern learning approaches.