למידה חישובית – שיעור 8

תזכורת Perceptron

אלגוריתם סיווג המחפש מישור מפריד בין סוגים שונים של דגימות הוא עושה זאת על ידי חיפוש הw. לדוגמה האלגוריתם יאבחן בין נשים לגברים על פי מידת הנעל ואורך השיער.

האלגוריתם מקבל דגימות כקלט ומחזיר כפלט את סיווג הדגימה ,הוא מאבחן בין סוגים שונים של דגימות על ידי ציון שייתן לאותה דגימה אותו הוא יחשב על ידי הכפלת וקטור הדגימה שקיבל בווקטור המשקולות אותו הוא שומר. וקטור המשקולות מאפיין את חשיבותם של משתני הדגימה, לדוגמה אם אורך השיער שולי לעומת מידת הנעל כאשר מבחינים בין גבר לאישה המשקולת שלאורך השיער תהיה 0.2 ושל מידת הנעל תהיה 0.8.

<u>נגדיר:</u>

x_d– נקודת הdata עליה אנו נמצאים.

o_d– הפרדיקציה של נקודת הdata עליה אנו נמצאים.

t_d– התוצאה האמיתית של נקודת הdata עליה אנו נמצאים. ה-label של נקודה d.

במידה וטעינו בפרדיקציה, נוסיף או נוריד את החלק היחסי מהוקטור w_i. במידה וצדקנו אנו לא צריכים לתקן – לכן נוסיף לוקטור w_i 0.

האלגוריתם stochastic

1. ניחוש ערך הw
2. מעבר על כל נקודה x_d של הdata וחישוב: $o_d = sgn(\vec{w} \cdot \vec{x}_d)$
3. עדכון ערך הw בהתאם לתוצאות שהתקבלו בשלב (2) באופן:
$$\Delta w_i = -\eta(o_d - t_d)x_{id}$$
$$w_i = w_i + \Delta w_i$$
4. בדיקה האם הערך של O_d שהתקבל הוא 0 או מספר קטן ממש. אם כן, סיימנו, אם לא:
5. חזרה על שלבים (4)-(2).

אלגוריתם הperceptron משפר את עצמו בכל פעם שהוא טועה באבחון ,השיפור נעשה על ידי הכפלה ב$t_d = -1$ ובעצם שינוי כיוון וקטור המשקולות בכל פעם שהוא טועה באבחון.

עדכון המשקולות:

$$\Delta w_i = -\eta(o_d - t_d)x_{id}$$

רק לדגימות שסווגו לא נכון יש השפעה על עדכון המשקולות (ובעצם על התזוזה שלנו).

אם הפרדיקציה (o_d) שלילית והtarget (t_d) חיובי, נעלה את המשקולת ע"י הוספה של שבר חיובי של הinstance.

אם הפרדיקציה (o_d) חיובית והtarget (t_d)שלילי, נוריד את המשקולות ע"י הוספה של שבר שלילי של הinstance.

כלומר כאשר נעדכן את w, תמיד נוסיף שבר של t_d, x_d (אם סיווגנו לא נכון).

לכן בסיום הריצה נקבל ש: $w = \sum a_d t_d x_d$.

a_d – השבר שהוספתי בשביל התיקון, ו- $a_d = 0$ במקרים בהם מיינתי את הדגימה בצורה טובה בהתחלה.
לכן, ניתן לרשום את פונקציית ההחלטה כך:

$$sgn(w \cdot x) = \vec{w} \cdot \vec{x} = \left(\sum a_d t_d \vec{x_d}\right) \cdot \vec{x} = \sum a_d t_d (\vec{x_d} \cdot \vec{x})$$

* המשקולות הם קומבינציה לינארית של x

לכן, נשמור רק את המידע של כל מי שמקיים $a_d \neq 0$ והם יקראו ה - ***Support Vectors*** כלומר קבוצת כל הוקטורים אותם מיינתי בצורה טובה.

Dual Perceptron

אלגוריתם שבמקום לעדכן משקולות על ה-features, מעדכן את מקדמי ה-instances (את a_d).

האלגוריתם:

1. נאתחל את a_i למספר רנדומלי קטן.
2. לכל נקודה x_d בdata, נחשב: $o_d = \sum_{i \in D} a_i t_i (\vec{x_i} \cdot \vec{x_d})$
3. נבדוק האם הקלסיפיקציה שגויה, כלומר האם $t_d o_d < 0$,
   a. אם כן:
      i. $a_d = a_d + \eta$,
   b. אם לא: לא נעשה דבר.
4. נחזור על שלבים (2)-(3) עד אשר השגיאה קטנה ממש או שהגענו למינימום.

נשים לב ש - $\vec{x_i} \cdot \vec{x_d}$ היא מכפלה פנימית, ואין צורך לחשב אותה בכל פעם אלא נוכל לשמור את המכפלות מראש במטריצה סימטרית בגודל $m\ x\ m$.

נעבור לאלגוריתם הלא לינארי ולכן:

$$\sum a_D t_D (\varphi(\vec{x_D}) * \varphi(\vec{x}))$$

מיפוי למימד גבוה יותר

נרצה להמיר את המידע שלנו למימד גבוה יותר וכך נוכל לסווג את הdata שלנו בצורה קלה ומהירה יותר, אך מיפוי מימד הוא פעולה יקרה ולעיתים תארוך זמן כה רב כך שהיא אינה פרקטית הלכה למעשה.

מונום- ביטוי מהצורה ax^n כאשר x הוא משתנה, a הוא המקדם שלו וn הוא מספר טבעי שנקרא הדרגה של המונום. המונום מרכיב את הפולינום.

Rational Variety

הגדרה: full rational variety מדרגה d, הוא כל המונומים עד דרגה d. יהיו $d+1$ איברים.

בפונקציה בעלת דרגה r עם n משתנים שנראית כך - מונום:

$$\varphi_i(x) = 1^{r_0} x_1^{r_1} \dots x_n^{r_n}$$

כאשר $\sum_{j=0}^{n} r_j = r$

לכן, כמות המונומים השונים שלנו היא:

$$\frac{(n+r)!}{n!\,r!} = \binom{n+r}{n}$$

זה מספר עצום בעיקר כשיש לנו דרגה גבוהה ומשתנים רבים, לכן נרצה להשתמש ביתרונות המיפוי למרחב הגבוה מבלי באמת למפות.

kernel

פונקציה $K: \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ תיקרא kernel, אם קיימת עבורה פונקציית מיפוי: $\varphi: \mathbb{R}^n \to \mathbb{R}^N$ כך שתמיד מתקיים:

$$K(x, y) = \varphi(x) \cdot \varphi(y)$$

הkernelים עוזרים לנו להימנע מחיפוש של המרחב ומיפוי למימד גבוה יותר, הם הופכים את הלמידה לפעולה ישירה (במימד נמוך) במימד המקורי.

אנחנו רוצים למפות למימד גבוה את ה-data שלנו ואז להפעיל את ה-data החדש perceptron, כך נמצא מפריד פולינומיאלי למימד הנמוך:

The Kernel Perceptron

אלגוריתם זהה לאלגוריתם הperceptron פרט לכך שבמקום לבצע את החישוב של $(\vec{x_i} \cdot \vec{x_d})$ ולעבור למימד אחר, אנו מחשבים את הKernel כלומר את $K(\vec{x_i} \cdot \vec{x_d})$. באלגוריתם זה אנו לא עוברים למימד גבוה יותר, אלא נמצאים במימד הנוכחי ומבצעים חישובים כאילו היינו במימד גבוה יותר.

<u>**האלגוריתם:**</u>

1. נאתחל את a_i למספר רנדומלי קטן.
2. לכל נקודה x_d בdata, נחשב: $o_d = \sum_{i \in D} a_i t_i K(\vec{x_i} \cdot \vec{x_d})$
3. נבדוק האם הקלסיפיקציה שגויה, כלומר האם $t_d o_d < 0$, אם כן: $a_d = a_i + \eta$, אם לא:
4. נחזור על שלבים (2)-(3) עד אשר השגיאה קטנה ממש או שהגענו למינימום.

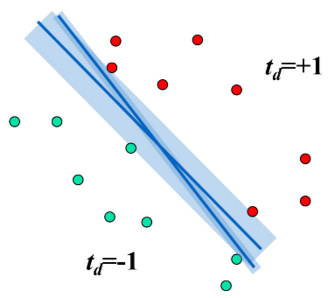
הMargin

בהינתן f(x) כלומר מפריד לינארי, הmargin הוא פונקציה של f ומוגדר:

$$Margin \equiv \min_{d \in D} dist(x_d, f(x) = 0)$$

בהינתן מפריד לינארי, נרצה למצוא margin שהוא בעצם גודל של השוליים (החלק התכלת בציור). אנו נשאף למצוא margin מקסימלי.

עבור מפריד לינארי שונה, נקבל margin שונה.



המוטיבציה ביצירת margin גדול ככל שניתן, היא כדי להתמודד בצורה מיטבית עם רעש בניסוי, או הוספה של נקודה חדשה ל-data שלנו, כך שהיא עדיין תסווג כראוי. ככל שנאפשר יותר margin אנחנו מאפשרים לנקודה החדשה סיווג נכון, גם במקרה של רעש בניסוי.

SVM - Support Vector Machines

אלגוריתם לסיווג. האלגוריתם מקבל בשלב האימון דוגמאות חיוביות ושליליות ובאמצעותן הוא מוצא מפריד נכון ככל האפשר. לאחר הלמידה, האלגוריתם יקבל נקודה חדשה ויזהה היכן היא ממקומות ביחס למפריד.

SVM יכול לבצע גם סיווג לא לינארי על ידי הוספת kernel ומיפוי הקלט למרחב במימד גבוה יותר.

אלגוריתם ה-SVM מבוסס על 3 רעיונות:

1. kernel trick. מיפוי ה-data למימד גבוה, כדי למצוא מפריד לינארי בצורה קלה יותר.
2. Max margin. מציאת השוליים הרחבים ביותר שנוכל עבור בעיות מופרדות לינארית.
3. soft margin and regularization. הרחבה של ההגדרה ב-(2) עבור בעיות שאינן מופרדות לינארית, כך שיתמודד עם misclassification.

<u>האלגוריתם:</u>

(1) ניקח חלק מהנקודות ב-training , נקרא להן: support vectors
(2) ניקח את קבוצת המשקולות α לוקטורים שבחרנו ב(1).
(3) נמצא פונקציית kernel לא לינארית הממפה למימד גבוה יותר - יעשה לעיתים קרובות ע"י ניסוי וטעיה.
(4) מסווגים כך:

$$class(\vec{x}) = sign\left(\sum_{d\in SV} a_d t_d K(\vec{x_d}, \vec{x})\right)$$

כאשר:

- x_d - ה-support vector.
- t_d- הערך האמיתי של נקודה d. יהיה -1 או +1.
- α_d- המשקולת של נקודה d.
- x- ה-instance אותו אנו רוצים לסווג.

מתקיים instance based learning, כי אנחנו משתמשים ב-instances קיימים (שהם ה- support vectors) ועל סמך ה-instances האלה מסווגים את ה-instance החדש.

עולה השאלה: כיצד נבחר את ה-support vectors?

על ידי למידה. נעשה בעיית אופטימיזציה, ובעיה זו באופן עקיף תגדיר לי את המשקולות (את ה-a_d) ואלה יגידו לי שכל מי ששונה מ-0 הוא ה-support vector שלי.

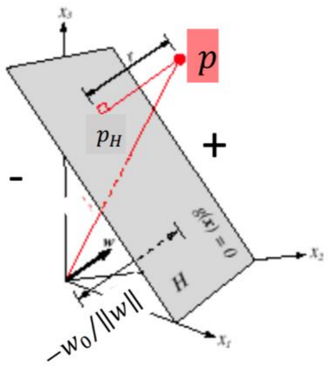
Linear Decision Boundary

בהינתן וקטור נורמלי למישור (מאונך למישור, אורתוגונלי), w, אני תמיד יכולה לבחור את w_0 כך שהייצוג למישור יהיה:

$$w \cdot x + \frac{w_0}{||w||} = 0$$

מרחק הנקודה מהמישור יהיה:

$$r = \frac{w \cdot p}{||w||} + \frac{w_0}{||w||}$$



אם יש לי נקודה כללית p ואני רוצה לדעת מה המרחק של הנקודה p למישור המפריד.

המרחק של p למישור יסומן: r, יחושב:

נטיל את p על w , מהאורך של הוקטור w שהתקבל מההטלה נחסיר את האורך של הוקטור w עד למישור המפריד.

החישוב שתיארנו מביא אותנו לנוסחה הבאה:

$$\text{dist(x,boundary)} = \frac{1}{||w||}\left(\sum_{i=1}^{n} w_i x_i + w_0\right)$$

w הוא הנורמל למישור המפריד (לא כולל w_0).

<u>נרצה למצוא את המרחק האבסולוטי לכן נכפיל ב-t_d:</u>

$$d = \frac{t_d\left(w_0 + \sum_{i=1}^{n} w_i p_i\right)}{||\vec{w}||} \geq b$$

אנו מעוניינים למצוא את ה-w, w_0 שיתנו לי את ה-b המקסימלי (b הוא הmargin) שיקיים:

$$\max_{w\ w_0} b \ \ subject\ to\ \frac{t_d\left(w_0 + \sum_{i=1}^{n} w_i p_i\right)}{||\vec{w}||} \geq b$$

כדי למצוא פתרון ייחודי, נניח ש-$\frac{1}{b} = ||\vec{w}||$:

$$t_d\left(w_0 + \sum_{i=1}^{n} w_i p_i\right) - 1 \geq 0$$

כלומר הmargin שקיבלנו הינו $b = \frac{1}{||w||}$.

האלגוריתם ישאף למקסם את השוליים, לכן לאחר ביצוע הזזות והגדרות נרצה למקסם את b. נזכור כי b הוא מרחק קדימה ואחורה מהמפריד הלינארי, לכן בעצם נמקסם את 2b שאנו יודעים שהוא שווה $\frac{2}{||w||}$

לכן נשאף למצוא את המינימום של $||w||$, שזה אותו דבר כמו למצוא מינימום ל:

$$\frac{1}{2}||w||^2$$

המקיים את המשוואה:

$$t_d\left(\sum_{i=1}^{n} w_i x_i + w_0\right) - 1 \geq 0, \ \ \forall d$$

נרצה למצוא את ה-w עם הנורמה הקטנה ביותר, המקיימת את 3 המשוואות הבאות:

$$w \cdot x^{(d)} + w_0 = 1$$
$$w \cdot x^{(d)} + w_0 = 0$$
$$w \cdot x^{(d)} + w_0 = -1$$

בשביל שהמשוואות האלו יתקיימו, צריך שהמישור יהיה מפריד. כלומר ברגע שהמשוואות מתקיימות- יש מפריד. מבין כל ה-w שמקיימים את זה, נרצה את ה-w עם הנורמה הקטנה ביותר.

נזכור כי ה-w עם הנורמה הקטנה ביותר, מייצג את הmargin הגדול ביותר.

על מנת להביא את w למינימום לא נוכל להשתמש באלגוריתם כמו Gradient decent, כי אנחנו מחפשים נקודה בתחום אילוץ כלשהו, בשונה ממה שעשינו עד כה (ב-GD יכלנו לבחור כל נקודה שהיא). כדי לפתור את בעיית האופטימיזציה הזו, נשתמש בכופלי לגרנז'.

Lagrange Multipliers

שיטה למצוא מינימום / מקסימום לפונקציה תחת אילוצים. נרצה למקסם / לצמצם את הפונקציה $f(x,y)$ בהינתן שהאילוץ: $g(x,y)$ מתקיים. נגדיר משתנה חדש λ שנקרא כופל לגרנז'.

נגדיר את פונקציית לגרנז':

$$L(x,y) = f(x,y) + \lambda g(x,y)$$

נרצה שכל הנגזרות החלקיות של $L(x,y)$ יהיו שוות ל-0, כלומר שהגרדיאנט של הפונקציה יהיה שווה ל-0.

דוגמה ב2D:

- $L(x,y) = x^2 + y^2 + \lambda(x + y - 2)$

- $\frac{\partial}{\partial x}L(x,y) = 2x + \lambda = 0$
- $\frac{\partial}{\partial y}L(x,y) = 2y + \lambda = 0$
- $\frac{\partial}{\partial \lambda}L(x,y) = x + y - 2 = 0$

- Subtracting the two first eqns we get $x = y$ and with the third we get a unique solution $x = 1, y = 1, \lambda = -2$

יש הבדל בין פתרון עם כופלי לגרנז' לבין המשוואות אותן נרצה לקיים ב-SVM, כיוון שכופלי לגרנז' פותרים משוואות וב-SVM הגענו לאילוץ באי שוויון.

ניתן לפתור את זה ע"י משפט KKT ממנו נקבל w שמביא את המפריד עם margin הגדולים ביותר.