

מבוא לבינה מלאכותית – תרגול 2

נושאים:

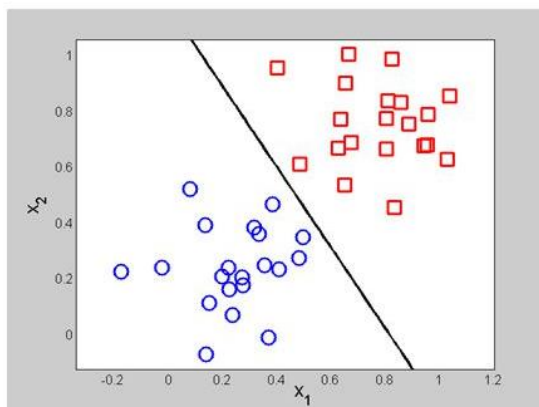
SVM – Hard Margin, Soft Margin, Online, Kernel Methods -

SVM (Support Vector Machine)

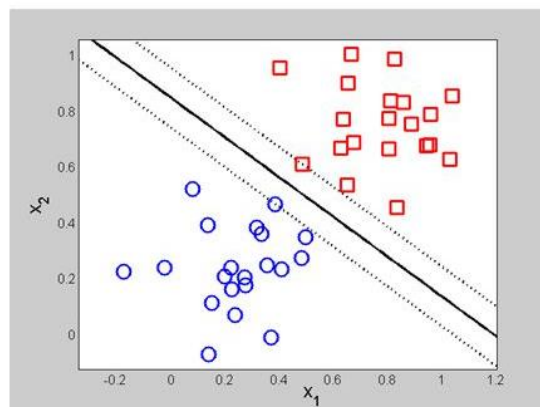
כזכור, בתרגול הקודם התחלנו לדבר על איך עושים סיווג באמצעות מפרידים לינאריים. כזכור, אנחנו מחפשים את (\vec{w}, b) (או לפי סימון אחר, \vec{w} בלבד כאשר b "נבלע" בתוך הווקטור), כך שלכל דגימה מתוך הדאטא \vec{x} המתויגת y , יתקיים $y(\vec{w} \cdot \vec{x} + b) > 0$. ראינו אלגוריתם שנקרא perceptron המוצא בדרך אחת משקולות (\vec{w}, b) שעונות על הדרישה.

SVM הוא אלגוריתם אחר למציאת מסווג לינארי לדאטא. באלגוריתם זה, מחפשים מישור מפריד כזה שיהיה רחוק ככל האפשר מהנקודות בדאטא, כלומר ישאיר שוליים (margin) רחבים. שלא כמו פרספטרון, שבו כל פתרון שבו המסווג מפריד בין הדגימות החיוביות לשליליות בדאטא לגיטימי עבורנו, ב-SVM דואגים גם לכך שיהיה בעל יכולת הכללה גבוהה ככל האפשר. כלומר, המפריד ב-SVM נלמד ככה שנוכל להיות הכי בטוחים שאפשר בכך שאם נדגום נקודה חדשה מהדאטא היא תסווג נכון.

Perceptron:



SVM:



שימו לב שלמעשה הדרישה ל-margin גבוהה היא בעצם הכנסה של prior למודל שלנו, ואכן נראה בהמשך שניתן לבטא את ההנחה הזאת להנחה הזאת בתור רגולריזציה. מה שנעשה בתרגול הזה הוא לראות כל מיני וריאציות של השיטה.

SVM – Hard Margin

הסיבה לשם "hard margin" היא שדורשים שלא רק שהמסווג יפריד בין כל הדגימות מכל סוג, אלא גם שהדגימות יהיו במרחק מהמפריד, כלומר שה-margin אכן יישאר. בעיית חיפוש הפרמטרים היא בעיית האופטימיזציה הבאה (נסמן את מספר הדגימות בסט האימון ב- N ואת מימד הקלט ב- d):

$$(w^*, b^*) = \arg \max_{(w,b): \|w\|_2=1} \min_{i \in \{1, \dots, N\}} |w \cdot x_i + b|$$

$$s. t. \quad y_j(w \cdot x_j + b) > 0 \quad \forall j \in \{1, \dots, N\}$$

שימו לב שכאשר $\|w\|_2 = 1$, הביטוי $|w \cdot x_i + b|$ משמעותו המרחק בין הנקודה x_i למישור המפריד, ולכן מה שהבעיה מנסה למצוא הוא את המשקלים שעבורם הנקודה הכי קרובה מסט האימון תהיה רחוקה ככל האפשר, תחת האילוץ שלא תהיה טעות בסיווג סט האימון. ניתן לנסח את הבעיה הזו בצורה שקולה כך:

$$(w^*, b^*) = \arg \min_{w,b} \|w\|_2^2$$

$$s. t. \quad y_j(w \cdot x_j + b) \geq 1 \quad \forall j \in \{1, \dots, N\}$$

וזו בעיית אופטימיזציה ריבועית שניתן לפתור (כלומר, אחת הדרכים לפתרון בעיית סיווג עם SVM היא פתרון בעיית אופטימיזציה בצורה offline – על כל הדאטא יחד, בשונה ממה שראינו ב-perceptron).

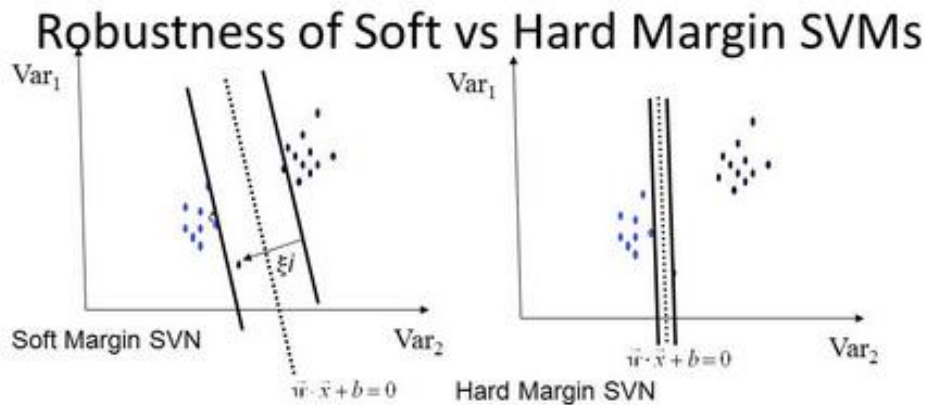
את בעיית האופטימיזציה הזו ניתן לפתור ישירות, או ע"י פתרון בעיה שקולה המתאימה לבעיה הזו, שנקראת הבעיה הדואלית לבעיה הזו. כאן, הבעיה הדואלית שפותרים היא:

$$\max_{\lambda = (\lambda_1, \dots, \lambda_N) \in \mathbb{R}^N} \sum_{i=1}^N \lambda_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \lambda_i \lambda_j y_i y_j \langle x_i, x_j \rangle$$

$$s. t. \quad \lambda_i \geq 0 \quad \forall i, \quad \sum_{i=1}^N \lambda_i y_i = 0$$

SVM – Soft Margin

הדרישה בגרסת ה-hard margin היא דרישה די נוקשה (למה שיהיה פתרון מושלם?). לעתים, היא אפילו לא נראית כמו הפתרון החכם ביותר, אלא כמו דרישה שעלולה להביא ל-overfitting. בתמונה מטה אפשר לראות מצב שבו נראה הרבה יותר הגיוני לטעות בסיווג של נקודה אחת חריגה במחיר של הכללה טובה הרבה יותר, מאשר לדייק לחלוטין על סט האימון.



לכן, ישנה גרסת soft margin של SVM: משנים מעט את בעיית האופטימיזציה באמצעות הכנסה של משתנים $\xi_i \geq 0$, ומקבלים את הניסוח הבא:

$$(w^*, b^*, \xi^*) = \arg \min_{w, b, \xi} \|w\|_2^2 + C \sum_{i=1}^N \xi_i$$

$$s. t. \quad \gamma_j (w \cdot x_j + b) \geq 1 - \xi_j, \quad \xi_j \geq 0 \quad \forall j \in \{1, \dots, N\}$$

הערות –

- נשים לב לתפקיד של המשתנים ξ_i :
 - אם בפתרון המתקבל $\xi_i = 0$ אז הנקודה x_i מקבלת פרדיקציה בצורה הכי טובה שאפשר (בצד הנכון, וגם תוך שמירה על ה-margin).
 - אם בפתרון המתקבל $0 < \xi_i < 1$ אז הנקודה x_i מקבלת פרדיקציה נכונה (בצד הנכון של הישר המסווג), אבל המסווג המתקבל קרוב אליה, כך שהיא תימצא בתוך השוליים. אחרת, הפרדיקציה לתיג של הנקודה שגויה.
- הקבוע C הוא קבוע שרירותי שאנחנו יכולים לקבוע את גודלו מראש. הוא נקרא גם "Box Constraint". נשים לב שככל שהוא גדול יותר, המשקל שניתן לטעות גדול יותר ולכן נעדיף לדייק בחיזוי על סט האימון (ערך גבוה מדי עלול להביא ל-overfitting).

מאידך, אם C קטן מדי, אז מה שישנה מאוד במשימת האופטימיזציה הוא ה- $margin$, אפילו במחיר של טעויות בחיזוי (ערך נמוך מדי עלול להביא ל- $underfitting$).
 למעשה, C משחק תפקיד כמו של קבוע רגולריזציה, ואת הפונקציה $\sum_i \xi_i$ אפשר להחליף בפונקציות אחרות.

- גם זו בעיית אופטימיזציה ריבועית שניתן לפתור, ובדרך כלל פותרים אותה על ידי מעבר לבעיה הדואלית. הבעיה הדואלית לבעיה זו היא:

$$\max_{\lambda=(\lambda_1, \dots, \lambda_m) \in \mathbb{R}^m} \sum_{i=1}^m \lambda_i - \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \lambda_i \lambda_j y_i y_j \langle x_i, x_j \rangle$$

$$s. t. \quad 0 \leq \lambda_i \leq C \quad \forall i, \quad \sum_{i=1}^m \lambda_i y_i$$

אפשר להבין מכאן למה C נקרא אילוץ קופסה – הוא למעשה חוסם את המשתנים λ (כופלי לגראנז' המתאימים לאילוצים מהבעיה המקורית) בתוך קופסה.

Online SVM

שתי הגרסאות הקודמות הן גרסאות offline, כלומר גרסאות כאלה שדגימות האימון שבהן האלגוריתם משתמש מוכנסות כולן יחד ללא אפשרות להוספה או עדכון בדגימות חדשות. ישנה גרסה אחרת של SVM שבה זה לא המצב. גרסת ה-online הזו מתבססת על פונקציית ה-loss הבאה (כאן נכתוב את המשקולות בעזרת w בלבד במקום (\vec{w}, b)):

$$L = \frac{1}{N} \sum_{i=1}^N \max \{0, 1 - y_i \vec{w} \cdot \vec{x}_i\} + \frac{\lambda}{2} \|\vec{w}\|_2^2$$

פונקציית השגיאה לכל דגימה נקראת hinge loss. שימו לב לדמיון בינה לבין האילוץ במקרה הקודם, $y_i \vec{w} \cdot \vec{x}_i \geq 1$. באמצעות שיטת האופטימיזציה SGD שנלמד בהמשך הקורס, מקבלים את האלגוריתם בצורתו האיטרטיבית וה-online:

קלט: סט האימון, $(\vec{x}_1, y_1), \dots, (\vec{x}_n, y_n)$, קבוע למידה η וקבוע רגולריזציה λ .
נאתחל את המשקולות \vec{w}^1 .
לכל איטרציה t :

נעבור על סט האימון (בין אם בסדר הנתון או לא). לכל דגימה (x_i, y_i) :

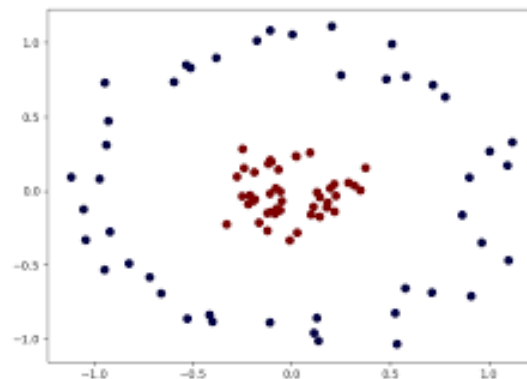
אם $y_i \vec{w} \cdot \vec{x}_i \geq 1$, עדכן את המשקולות כך: $\vec{w}^{t+1} = (1 - \lambda\eta)\vec{w}^t$.

אחרת, עדכן את המשקולות כך: $\vec{w}^{t+1} = (1 - \lambda\eta)\vec{w}^t + \eta y_i \vec{x}_i$.

פלט: וקטור המשקולות הסופי \vec{w}^{final} .

:Kernel SVM

מסווגים לינאריים מוגבלים מאוד ביכולת שלהם בגלל הפשטות שלהם. בתרגיל 1 הופיעה הדוגמה של הפונקציה XOR, אותה לא ניתן ללמוד באמצעות מסווג לינארי. עוד צורה די מוכרת שאין מסווג לינארי יעיל להתמודדות איתה היא הצורה הבאה:



יש שתי דרכים עיקריות שנלמד כדי להפוך את המסווגים שלנו לכאלה שמסוגלים ללמוד יותר צורות: הראשונה היא ה-kernel trick ואותה נלמד בתרגול הזה, השנייה היא רגרסיה לוגיסטית (שהיא הבסיס לרשתות נוירונים) ואותה נלמד בתרגול הבא.

השימוש ב-kernel משמעותו בעצם להעתיק את הדאטא למרחב אחר, שבו ניתן לסווג בקלות. כלומר, נחפש פונקציה $\psi: \mathcal{X} \rightarrow \mathcal{F}$ כך שבמרחב \mathcal{F} ניתן להפריד את הדאטא $\{(\psi(\vec{x}_i), y_i)\}_{i=1}^N$ באמצעות מסווג לינארי.

איך זה מתבצע טכנית? זהו ה-kernel trick: משתמשים בפונקציית גרעין, שהיא בעצם מטריצה $K_{ij} := \langle \psi(\vec{x}_i), \psi(\vec{x}_j) \rangle$ (הערת אגב - המטריצה K היא מטריצה חיובית, כלומר כל הערכים העצמיים שלה אי שלילים. משפט מאנליזה פונקציונלית קובע שכל מטריצה כזו מייצגת מכפלה פנימית בין הווקטורים \vec{x}_i באיזשהו מרחב הילברט). בבעיות הדואליות שראינו למעלה הייצוג היחיד של דגימות הדאטא הוא בתור מכפלות פנימיות, ולכן החלפה של המכפלה הפנימית ב- K_{ij} המתאים גורם לביצוע של ההעתקה למרחב החדש.

המהות של הטריק היא שגם מבלי לדעת מה הפונקציה ψ ורק בידיעה של K אפשר לבצע העתקה כזאת, כלומר ניתן להעתיק למרחב גם מבלי לדעת איך לייצג מפורשות את התוצאה של הייצוג במרחב הזה.

דוגמות שונות ל-kernels:

1. גרעין פולינומיאלי – להעתקה מהמרחב המקורי למרחב שמהווה פולינום ממעלה k ,

$$K(\vec{x}_i, \vec{x}_j) = (1 + \langle \vec{x}_i, \vec{x}_j \rangle)^k$$

2. גרעין גאוסיאני (ידוע גם כ-RBF – Radial Basis Function) - $K(\vec{x}_i, \vec{x}_j) = e^{-\frac{\|\vec{x}_i - \vec{x}_j\|_2^2}{2\sigma^2}}$ בגרעין

כזה, זוג נקודות קרובות זו לזו גורמות לערך המטריצה המתאים להיות קרוב ל-1, וזוג נקודות רחוקות זו מזו גורמות לערך המטריצה המתאים להיות קרוב ל-0. למעשה, ההעתקה ψ שמתאימה ל-RBF היא למרחב הילברט מממד אינסופי (אפשר לראות את זה בגלל פיתוח לטור טיילור של אקספוננט).

למעשה, כל העתקה של הדאטא למרחב שבו ההפרדה נעשית בקלות היא העתקה לגיטימית, ולא ידוע על הסבר כללי מתי להשתמש ב-kernel מסוים. למשל, לדוגמת הדאטא שהופיעה לעיל, ישנה העתקה (ψ מפורשת שאפשר למצוא) שמתאימה לציור המופיע מטה ובעזרתו קל להפריד. נסו לחשוב מהי (יופיע גם בתרגיל 2).

