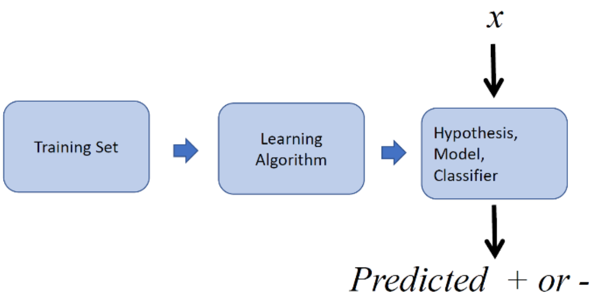
למידה חישובית - שיעור 7

classification

תהליך הקלסיפיקציה הוא התהליך בו האלגוריתם שנוצר מתהליך הלמידה מסווג את המידע שלי (באיור שלנו יסווג את x) לקבל label + או -. תהליך הקלסיפיקציה מתבצע לאחר שהאלגוריתם הלומד, מבצע את תהליך הלמידה על ה-training set ומייצר היפותזה, מודל ומסווג.



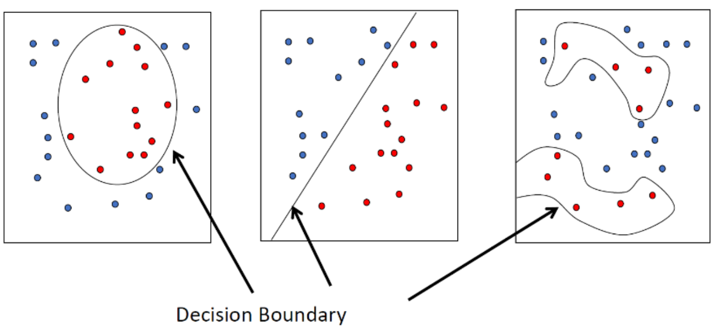
בתום תהליך הקלסיפיקציה נקבל פרדיקציה: האם x שייך ל + או ל -.

Discrete Space (הגדרה פורמלית):

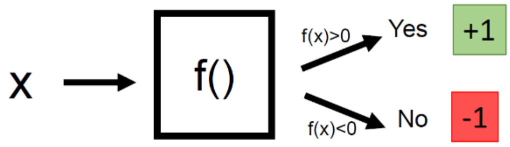
1. X הוא מרחב של instances. $x \in X$.
2. הקונספט (הדיכוטומיה) c שאנו מנסים ללמוד הוא תת מרחב, כך שמתבצעת חלוקה של המרחב ל-2.
3. ה-training data D הוא קבוצה של זוגות $< x, C(x) >$ כך ש$C(x)$ הוא הסיווג המתקבל.
4. אנו מחפשים היפותזה (מודל) לכל נקודה ב-data.

דוגמה לדיכוטומיה (קונספט) ב-$\mathbb{R}^2$:

נחפש פונקציה שתייצר מסווג שיחלק לנו את המרחב ל-2, כך ש-instances שסווגו + יהיו בצד אחד של המסווג ו-instances שסווגו - יהיו בצידו השני של המסווג.
כאשר תגיע data point חדשה, נשאף שהפונקציה תסווג את הנקודה בצורה הנכונה ונוכל לדעת לאיזה class נקודה זו שייכת.



סיווג באמצעות פונקציית דיסקרימיננטה



בהינתן instance x כלשהו, נרצה לבדוק האם $f(x) > 0$ או $f(x) < 0$. לאחר שנדע את התשובה לשאלה נוכל לסווג את הנקודה x ל-class המתאים לה.

*<u>**דיסקרימיננטה**</u> היא שמם המשותף של כמה מדדים מספריים הקשורים לפולינומים ולאובייקטים מורכבים יותר.

תזכורת אלגברה לינארית:

<u>מכפלה פנימית</u> היא פונקציה, הפועלת על זוג איברים מתוך מרחב נתון, ומחזירה סקלר מעל השדה הנתון. אם מכפלה פנימית של 2 וקטורים מניבה 0, הוקטורים אורתוגונליים (ניצבים) זה לזה.

כאשר אנו מזיזים ימינה למעלה את הוקטור שלנו (נעים לכיוון הרביע ה-I) – זה מינוס
כאשר מזיזים שמאלה למטה את הוקטור שלנו (נעים לכיוון הרביע ה-III)– זה פלוס.

<u>נורמל</u> – וקטור המאונך למשטח כלשהו.

Linear Separability

נניח שמרחב ה-instances הוא $\mathbb{R}^2$ (יש 2 feature-ים), וכן כל instance הוא נקודה במרחב, אם קיים קו כלשהו במרחב המפריד בין 2 ה-class-ים, אז ה-class ייקרא מופרד לינארי (linearly separable).

נרצה לבצע מכפלה פנימית בין **הוקטור של ה-instance** לבין **הוקטור הממיין שלי**. אם נקבל שהמכפלה הפנימית ביניהם היא 0, הרי שהם אורתוגונליים זה לזה. המכפלה הפנימית הזו היא **משטח ההפרדה** שלי.

<u>נגדיר פורמלית:</u>
- פונקציה ב-2D המתארת את הקו המפריד ב-$f(x,y) = 0$ היא: $f(x,y) = w_1x + w_2y + w_0\ (= 0)$
- פונקציית הדיסקרימיננטה במקרה זה: $f(x,y) = w_1x + w_2y + w_0$.

<u>הערה:</u> (x,y) הוא וקטור הפרמטרים שמדדתי. w הם המקדמים שאנו משתמשים לקלסיפיקציה.
- הקלסיפיקציה: אם $f(x,y) > 0$ נסווג +, אחרת נסווג -.

נוכל להרחיב את ההגדרה עבור 3D:

המסווג הלינארי יתקבל מהפונקציה: $w_1x + w_2y + w_3x + w_0 = 0$, כאשר (x,y,z) מיצג את המישור של ה-instance. אם $w_0 = 0$ המשטח עובר בראשית.

נרצה להרחיב את ההגדרה עבור $\mathbb{R}^n$:

כאשר $g(x)$ היא פונקציה לינארית, היא מגדירה את העל-מישור. נוכל לכתוב אותה כצירוף לינארי של x:

$$g(x) = w^Tx + w_0$$

כאשר:
- w^T הוא וקטור המשקולות של ה-attribute
- w_0 הוא ה-bias

הסיווג יהיה:

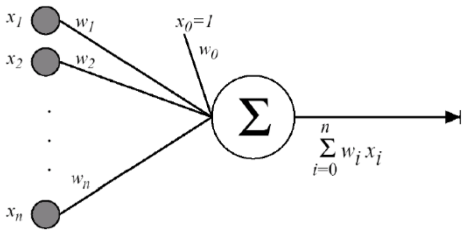
$$\begin{cases} 1, & \text{if } w_0 + w_1x_1 + \cdots + w_nx_n > 0 \\ -1, & \text{otherwise} \end{cases}$$

יחידת חישוב לינארית

יחידת חישוב לינארית היא יחידה שלוקחת כקלט את המספרים $\{x_1, x_2, \dots, x_n\}$, ואת המשתנה המלאכותי $x_0 = 1$ ומכילה את $\{w_0, w_1, \dots, w_n\}$ ומחשבת את הסכום (ללא threshold):

$$o = w_0 + w_1 x_1 + \dots + w_n x_n = \vec{w} \cdot \vec{x}$$

יחידת החישוב הלינארית מוגדרת על ידי המקדמים, אנחנו רוצים ללמוד את המקדמים הללו. האלגוריתם הלמידה מחפש את המקדמים האלה.
היחידה הלינארית הזו תהיה אבן בסיס בהרבה שיטות למידה. הטכניקה הזו נפוצה מאוד ב-deep learning.



זה מאוד מזכיר linear regression.

פונקציית הטעות (של רגרסיה לינארית):

$$E(\vec{w}) = \frac{1}{2}\sum_{d\in D}(o_d - t_d)^2 = \frac{1}{2}\sum_{d\in D}(\vec{w}\cdot\vec{x_d} - t_d)^2$$

- o_d - החישוב של היחידה הלינארית (ה-output) על ה-data point: d.
- t_d- הערך האמיתי של data point: d. הסיווג של הנקודה d.
- D - training data.
- $\vec{w}$- המשקולות שיש ביחידת החישוב הלינארית
- $\vec{x_d}$- ה-features שיש ב-instance: x_d

אנו מחשבים את המכפלה הפנימית של $\vec{w}\cdot\vec{x_d}$ ומחסירים ממנה את ה- t_d.

הערה חשובה:

כאשר אנו מציגים שאלת קלסיפיקציה על ידי רגרסיה, ייתכן מצב בו ה-instance סווג לצד הנכון, אך הוא רחוק מהמפריד אותו אנו רוצים ללמוד ולכן ה-instance יסומן כטעות גדולה יחסית.

נרצה למצוא את היחידה הלינארית שמביאה את פונקציית הטעות למינימום. נזכור שהיחידה הלינארית מוגדרת על ידי w, לכן נגזור לפי w:

$$\frac{\partial \mathrm{E}}{\partial \mathrm{w_i}} = \frac{\partial}{\partial \mathrm{w_i}}\frac{1}{2}\sum_{d\in D}(o_d - t_d)^2 = \sum_{d\in D}(o_d - t_d)\, x_{id}$$

כעת, לפי חוק הtraining: $\Delta\vec{w} = -\eta\nabla E[\vec{w}]$ ולכן:

$$\Delta\vec{w} = -\eta\nabla E[\vec{w}] = -\eta\sum_{d\in D}(o_d - t_d)\, x_{id}$$

- η זה ה-learning rate שלנו.

LMS – Least Mean Square

אלגוריתם למידה המבוסס על אלגוריתם Gradient decent. האלגוריתם יגדיר ערך רנדומלי כלשהו עבור המשקולות (לרוב יבחר להגדיר 0) ובכל איטרציה ישתמש בנוסחת העדכון (שמצאנו כעת) ויעדכן את המשקולות בהתאם.
לאחר כל איטרציה נחשב את הטעות, אם היא 0 או קטנה מסף כלשהו שנגדיר, נסיים את ריצת האלגוריתם.

<u>האלגוריתם:</u>

1. נעבור על כל הנקודות ב-data, ונחשב עבורן:
$$o_d = \vec{w} \cdot \vec{x_d}$$
2. עבור כל משקולת w נחשב:
$$\Delta w_i = -\eta \sum_{d \in D} (o_d - t_d)\, x_{id}$$
$$w_i = w_i + \Delta w_i$$
3. נחזור על התהליך עד שהטעות 0 או קטנה מספיק (לפי תנאי עצירה שנגדיר)

האלגוריתם מזכיר רגרסיה לינארית
ההבדל הוא שברגרסיה לינארית
הlabel יכול היה לקבל כל ערך בדיד
או רציף (לדוגמה מחיר הבית שיכול
להיות כל מספר בעולם), ובLMS ניתן
לקבל ערך בינארי 1+, 1-.

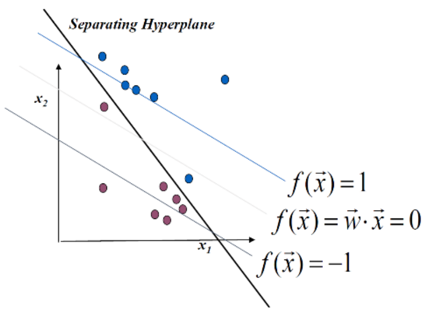
פונקציית הטעות (כפי שהגדנו בתרגול 6):

$$E[\vec{\theta}] = \frac{1}{2}\sum_{d \in D}\left(o^{(d)} - t^{(d)}\right)^2 = \frac{1}{2}\left[\sum_{d \in D^+}\left(o^{(d)} - 1\right)^2 + \sum_{d \in D^-}\left(o^{(d)} + 1\right)^2\right]$$

Minimize the distance between the positive instances and the +1 iso-line of the function

Minimize the distance between the negative instances and the -1 iso-line of the function

נשים לב:
אלגוריתם ה-LMS עשוי לא להתכנס למפריד הלינארי המושלם, כיוון שפונקציית הטעות של האלגוריתם מוצאת את מינימום המרחק בין הנקודות לבין המפריד הלינארי **<u>ולא</u>** את סיווג הנקודה. פונקציית הטעות מביאה למינימום את המרחק בין המכפלה הפנימית $\vec{w} \cdot \vec{x_d}$ לבין הנקודה t_d.



פונקציית הטעות **<u>לא</u>** סופרת כמה פעמים טעיתי בקלסיפיקציה.

אלגוריתם ה-LMS כפי שהגדרנו אותו מחשב את o_d לכל אחת מהנקודות ולאחר מכן מעדכן את המשקולות על ידי נוסחת העדכון. האלגוריתם המתואר הינו אלגוריתם מסוג Batch.

האלגוריתם היה יכול באותה המידע לחשב את o_d עבור נקודה אחת בלבד, לעדכן את המשקולות עבור נקודה זו ולהמשיך הלאה לנקודה הבאה. זהו אלגוריתם מסוג

Batch – מעבר על כל ה-instances. הסדר בו נעבור על ה-instances לא משנה.
Mini-Batch – מעבר על תת קבוצה מה-data בכל פעם.
Stochastic – (סטוכסטי פירושו מקרי). מעבר על instance אחד בכל פעם. הסדר בו אנו עוברים על ה-instances מקרי. נשים לב שבגישה זו הסדר בו אנו עוברים על ה-instances משנה. שימוש בגישה זו עוזר לנו להימנע ממינימום לוקאלי.

Stochastic LMS

1. נאתחל את המשקולות בערכים.
2. כל עוד לא הגענו להתכנסות (או לטעות קטנה דייה):
   a. לכל instance $\overrightarrow{x_d} \in D$, נחשב:
      i. $o_d = \vec{w} \cdot \overrightarrow{x_d}$
   b. לכל $1 \le i \le n$
      i. $\Delta w_i = -\eta(o_d - t_d)x_{id}$
      ii. $w_i = w_i + \Delta w_i$

נציג כעת פונקציית טעות חדשה שמטרתה לפתור את הבעיה שהצגנו עם פונקציית הטעות של LMS. פונקציית הטעות תמצא את המסווג בעל הטעות האמיתית, כך שאם אין בכלל טעויות, כלומר היה סיווג מושלם- הפונקציה תחזיר 0.
לשם כך, נרצה לבדוק כי t_d, o_d בעלי אותו סימן, כך שהכפל ביניהם יניב 1.
הפונקציה:

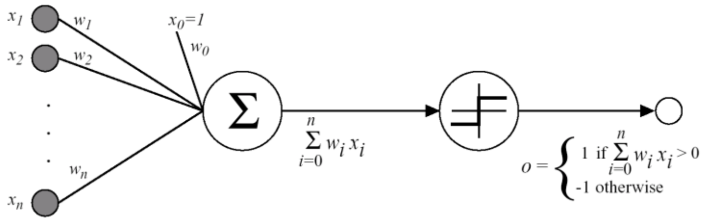
$$E(\vec{w}) = \frac{1}{2}\left(m - \sum_{d \in D} sign(t_d o_d)\right)$$

כאשר
- m- מספר הנקודות עבורן $d \in D$

הוספנו למסווג הלינארי שלנו יחידה שתקבע האם התוצאה שקיבלנו היא חיובית (או שלילית). כעת, בעקבות השינוי שיצרנו o_d יחזיר ערך 1- או 1+.

היחידה הלינארית החדשה שהוספנו נקראת perceptron.

Perceptron



אלגוריתם ה-perceptron הוא אלגוריתם stochastic שמייצר מפריד לינארי. האלגוריתם עושה תהליך איטרטיבי המשפר את w בכל איטרציה.

האלגוריתם

1. נאתחל את המשקולות בערכים רנדומליים.
2. כל עוד לא הגענו להתכנסות (או לטעות קטנה דייה):
   a. לכל instance $\overrightarrow{x_d} \in D$, נחשב:
      i. $o_d = sign(\vec{w} \cdot \overrightarrow{x_d})$
   b. לכל $1 \leq i \leq n$
      i. $\Delta w_i = -\eta(o_d - t_d)x_{id}$
      ii. $w_i = w_i + \Delta w_i$

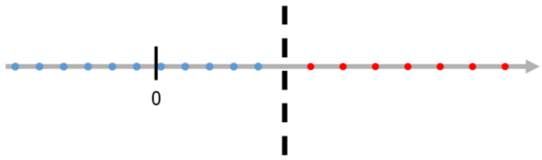
ההבדל בין האלגוריתמים הוא אופן חישוב ה-o_d (o_d מחזיר לי את ה-class שאני מנבאת לנקודה d).
יחידת החישוב החדשה מבצעת מכפלה פנימית עבור $\vec{w} \cdot \overrightarrow{x_d}$ ומסתכלת על הסימן של המכפלה הפנימית שהתקבלה. לפי הסימן שהתקבל מחליטה אם זה +1 או -1.

כל שנותר לנו לעשות הוא למצוא את ה-w שמביאים למינימום את הטעות. נעשה זאת על ידי עדכון w בכל איטרציה. אם צדקתי, הביטוי: $-\eta(o_d - t_d)x_{id}$ יהיה שווה ל-0 ואז Δw_i יקבל ערך 0 ובשלב התיקון של w_i, ה-w_i לא ישתנה.

<u>הערה לגבי עדכון w:</u> אם x הוא חיובי, נתקן את הוקטור הנורמלי w כך שאנו מוסיפים ל-w משהו חיובי בכיוון x. אם x הוא שלילי, נוסיף ל-w משהו שלילי בכיוון x.

האלגוריתם מבטיח שאם קיים מפריד לינארי במרחב, הוא ימצא אותו, אבל האלגוריתם לא יעבוד עבור מקרים בהם לא קיים מפריד לינארי.

נביט בנקודות הבאות, הן ניתנות להפרדה בקלות על ידי מפריד לינארי:



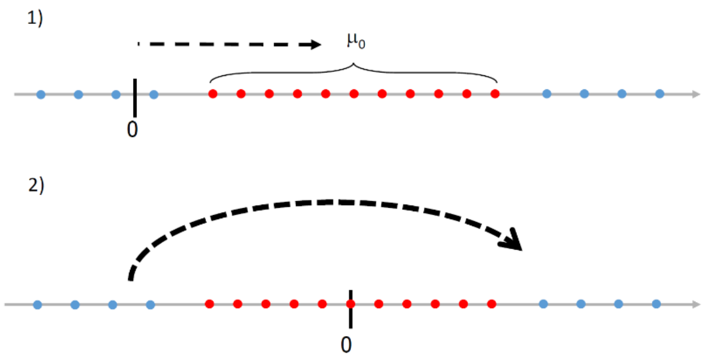
אם נרצה לייצר מפריד שמקיים שעבור נקודה x מתקיים: $w_1x + w_0 > 0$ אז הנקודה אדומה, ונניח כי הנקודה הכחולה השמאלית ביותר היא בנקודה (5), נוכל לבחור את המשקולות: $1, -5$, כיוון שמתקיים:

$$1 \cdot x - 5 > 0 \Rightarrow x > 5$$

כלומר נקודות הגדולות מ-5 יסווגו באדום, כפי שרצינו.

אולם, מה לגבי הדוגמה הבאה:

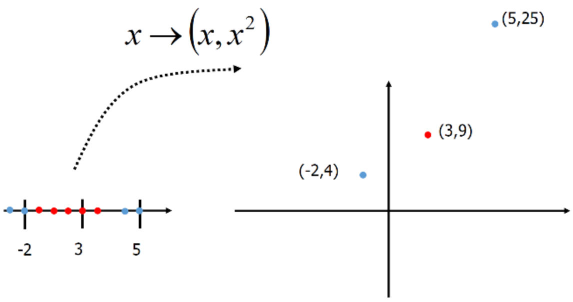
לדוגמה הזו אין מפריד לינארי פשוט, אך נוכל למפות את x ל- $y = (x - \mu_0)$ כך ש:



כלומר המרנו את כל הנקודות הכחולות שנמצאת בצד שמאל כך שיעברו לצד ימין.

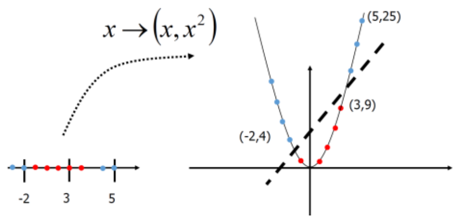
הפונקציה שהצגנו אינה לינארית כדי למפות את הנקודות החדשות, ושם אנו מפרידים את המידע עם מפריד לינארי.

דרך נוספת למפות את המידע הינה העברת המידע למימד גבוה יותר.



נעשה זאת על ידי מיפוי נקודה x לזוג נקודות (x, x^2), כך שקורדינטת ה-x שלהם היא x, וקורדינטת ה-y שלהם היא x^2.

לאחר העברה למימד גבוה נוכל למצוא מפריד ליניארי בקלות:



באופן כללי נוכל להגדיר:

$$F(x) = w_0 + w_1\varphi_1(x) + w_2\varphi_2(x) + \cdots + +w_n\varphi_n(x)$$

נרצה למצוא φ שימפה את האיברים ממימד נמוך למימד גבוה כך שניתן יהיה להפריד את המידע לינארית.

נאמר כי המידע הוא *Seperatable* φ

אם קיימת פונקציית φ וקיים וקטור w במימד גבוה כך שהנקודות החדשות ניתנות להפרדה על ידי וקטור w:

$$w^T\varphi(\mathrm{x}) > 0 \quad x \in C_0$$
$$w^T\varphi(\mathrm{x}) < 0 \quad x \in C_1$$

אנו מחשבים את המכפלה הפנימית של $w^T\varphi(\mathrm{x})$ ולפי סימן המכפלה הפנימית מחליטים לאיזה קונספט הנקודה x שייכת.

Cover's Function Counting Theorem

תאוריה שאומרת שאם נעלה את ה- dataשלנו למימד גבוה יותר, אנו נמצא מפריד ליניארי בהסתברות גבוהה יותר. התאוריה מוסיפה ואומרת כי מציאת המפריד הליניארי הינה אירוע רנדומלי ולא בהכרח אומרת משהו על ה-data שלנו. כלומר העלאת ה-data למימד גבוה יותר, הינה בסך הכל overfitting.

הגדרה: דיכוטומיה של קבוצה S היא חלוקה של S ל-2 קבוצות זרות.

נניח שיש לנו K דוגמאות בקבוצה של ה-instance שלנו S, אז יש לנו 2^{k-1} דיכוטומיות אפשריות מעל ה-instances הללו. כל דיכוטומיה מגדירה קלסיפיקציה.

Cover's Counting Theorem- מספר הדיכוטומיות המאפשרות הפרדה לינארית על קבוצת נקודות בגודל K היא:

$$2\sum_{i=0}^{N}\binom{k-1}{i}$$

לכן ההסתברות לקבל דיכוטומיה המאפשרת הפרדה לינארית הינה:

$$P(K,N) = \frac{\sum_{i=0}^{N}\binom{k-1}{i}}{2^{K-2}}$$

כלומר, ההסתברות לקבל data שניתן להפרדה לינארית במימד גבוה יותר- גבוהה יותר, ללא תלות ב-data.

Kernel

פונקציה תיקרא פונקציית kernel: $K: \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ אם קיימת פונקציית מיפוי $\varphi: \mathbb{R}^n \to \mathbb{R}^N$ כך שמתקיים לכל x, y:

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

כלומר הkernel מייצג מכפלה פנימית במרחב הגבוה.

כלומר יש לנו דרך עקיפה לחשב את חישוב המכפלה הפנימית במימד הגבוה, מבלי באמת לבצע את המעבר למימד הגבוה.

דוגמה למציאת kernel:

$$\text{For } x = (x_1, x_2) \in R^2 \quad \text{let} \quad \varphi(\vec{x}) = \left(x_1^2, \sqrt{2}x_1x_2, x_2^2\right)$$

$$\text{Given } \; x = (x_1, x_2) \quad y = (y_1, y_2) \quad \text{we then get}$$

$$\varphi(x)\varphi(y) = \left(x_1^2, \sqrt{2}x_1x_2, x_2^2\right)\begin{pmatrix} y_1^2 \\ \sqrt{2}y_1y_2 \\ y_2^2 \end{pmatrix}$$

$$= x_1^2y_1^2 + 2x_1x_2y_1y_2 + x_2^2y_2^2$$

$$= \left((x_1, x_2)\begin{pmatrix} y_1 \\ y_2 \end{pmatrix}\right)^2 = (x \cdot y)^2 = k(x,y)$$

$$\boxed{\text{Hence, } k(x,y) = (x \cdot y)^2 \text{ is a kernel for this } \varphi}$$

<u>הערה</u>: קיימות המון פונקציות kernel בעולם שאנשים טובים מצאו לפנינו, ולרוב אנחנו נדע להתאים פונקציית kernel לבעיה שלנו ולא באמת נצטרך למצוא את הפונקציה בעצמנו.

<u>הערה</u>: האלגוריתמים של ההפרדה הלינארית ישתמשו בפונקציות kernel, ולא לעבור למימד גבוה יותר (שכן מעבר למימד גבוה יותר עולה לנו הרבה, ואפילו לא תמיד אפשרי). נרחיב על כך בהמשך.