למידת מכונה הרצאה 9

תזכורות

ראינו בשליש הראשון של הקורס את המתמטיקה של למידת מכונה. אם יש חוק פשוט (ממשפחה של מימד-VC נמוך), והוא מתאים למדגם, אז בהסתברות גבוהה הוא מתאים לכל העולם.

בשליש השני ראינו אלגוריתמים שמוצאים חוק פשוט שמתאים למדגם. *Perceptron*, *SVM*, עצי החלטה ויער, שכן קרוב, ועוד. בכל אחד ראינו מתי זה חוק פשוט.

(בלמידה עמוקה, משתמשים ב *dropout* כדי לגרום לרשת להיות פשוטה יותר. דומה לגזימה שראינו בעצי החלטה).

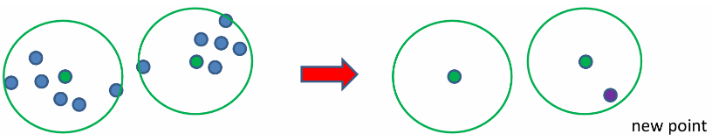
נתחיל את השליש האחרון של הסמסטר. נסתכל בנושאים שונים של למידת מכונה:

מקבוץ – *clustering*

בלמידה מפוקחת (*supervised learning*), לנקודות במדגם יש תיוג. נרצה ללמוד גם במקרים של דאטא לא מתוייגת – למידה לא מפוקחת (*unsupervised learning*). דרך אחת לעשות את זה היא *clustering* – נחלק את הנקודות למקבצים, ולפי זה נלמד.

K-clustering

בהינתן נקודות, נרצה לייצר מהן k קבוצות. ואז כל נקודה חדשה תקבל תיוג לפי קבוצה שנשייך אותה אליה.



למה נרצה לעשות מקבוץ?

- **למידה** – נקודה חדשה תשוייך לאחד המקבצים.
- **דחיסה** – נצטרך לשמור רק את קבוצת המרכזים K, במקום את כל S. זה נותן לנו חסמי הכללה טובים יותר (כי החסם הכללה תלוי גם ב-n).
- **זמן ריצה** – חיפוש שכן קרוב על K במקום S. באופן כללי, מציאת שכן קרוב היא בעיה קשה במימדים גבוהים.

חסמי איחוד אחרי דחיסה

מה המימד-VC של למידה דרך *k-clustering*?

נבחר קבוצת מרכזים K מתוך S: עבור $|S| = n$, יש $\binom{n}{k} \in O(n^k)$ אפשרויות. אם אפשר לנפץ d נקודות, אז:

$$n^k \geq 2^d$$

$$k \log n \geq d$$

$$d \in O(k \log n)$$

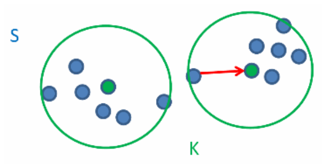
לדוגמה לחוקים של "שכן קרוב אחד", יש מימד-VC אינסופי. אם ניקח רק k נקודות, המימד-VC הוא $k \log n$.

אלגוריתמים של מקבוץ

בהינתן קבוצה S של n נקודות ופרמטר k, נרצה לחלק את S ל-k קבוצות.

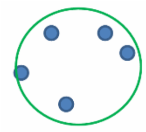
עולה השאלה של **מדד** – מה נחשב מקבוץ טוב? יש מדדים שונים: מרכז, חציון, ממוצע: *k-center, k-median, k-means*.

מרכז – *k-center*



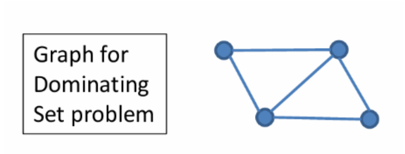
נבחר קבוצת נקודות $K \subset S$ שממזערת את: $\max_{v \in S} d(v, K)$. כלומר, לכל נקודה במדגם נמדוד את המרחק שלה מהנקודה הכי קרובה אליה מתוך K. ניקח את הנקודה שהכי רחוקה מהקבוצה K. נרצה למצוא קבוצה K כך שהמרחק הזה הוא מינימלי.

אפשר להסתכל על זה בתור קבוצה של כדורים.



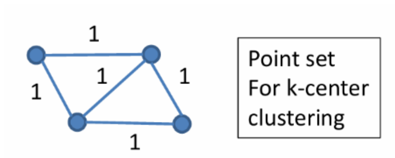
בנתיים, הנחנו ש $K \subset S$. זו הגרסה הבדידה. יש את הגרסה הרציפה – המרכזים יכולים להגיע מהמרחב האוקלידי של הנקודות:

אלגוריתם לגרסה הבדידה



Brute-force ייקח זמן $O(n^k)$. האם נוכל לשפר את זה?

הבעיה של מציאת k מרכזים אופטימליים היא *NP-hard*. אפילו קשה למצוא קירוב לרדיוס בתוך סדר גודל של $2 - \epsilon$ (קצת פחות מפי 2).



נוכיח על ידי רדוקציה מבעיית *minimum dominating set*:

בהינתן גרף $G = (V, E)$, נרצה למצוא תת קבוצה $V' \subset V$ בגודל מינימלי כך שכל קודקוד $v \in V \setminus V'$ הוא שכן של קודקוד כלשהו ב-V.

ניתן משקל 1 לכל צלע בגרף. אם נפתור את בעיית *k-center* על הגרף, זה פותר את בעיית הקבוצה השלטת. וגם הפוך. זה גם מראה למה קשה לקירוב עבור רדיוס קטן מ-2: כי אי אפשר למצוא בגרף מרחק בין 1 ל-2, זה רק מספרים שלמים.

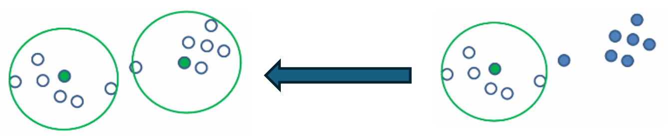
יש שני אלגוריתמי קירוב אפשריים בזמן פולינומיאלי למקרה הבדיד:

1. רדיוס אופטימלי, עם $k \ln n$ מרכזים.
2. k מרכזים, עם רדיוס גדול יותר (לפחות פי 2 הרדיוס האופטימלי).

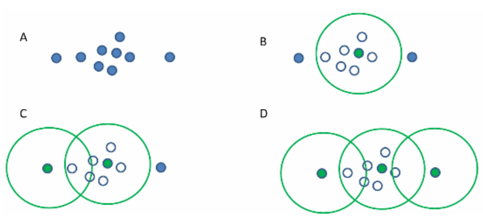
האלגוריתם הראשון

רדיוס אופטימלי, עם $k \ln n$ מרכזים. אלגוריתם חמדן:

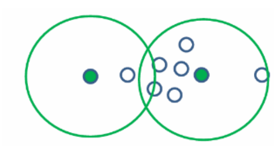
1. נניח שהרדיוס האופטימלי ידוע.[1]
2. ניקח מרכז שמכסה את רוב הנקודות.
3. נתעלם מהנקודות המכוסות, ונחזור לשלב 2.



האלגוריתם החמדן לא בהכרח אופטימלי:



האופטימלי:



[1] נניח שאנחנו יודעים, כי אפשר לנסות את כל ה- n^2 אפשרויות.

ניתוח *standard greedy analysis*:

K מרכזים מכסים את S, וגם כל תת קבוצה של S. מרכז כלשהו מכסה $1/k$ של S או תת קבוצה שלה.

בכל שלב, $1/k$ נקודות מכוסות. אחרי i חזרות, כמה נקודות נשארו?

$$n\left(1-\frac{1}{k}\right)^i \le n\cdot\exp\left(-\frac{1}{k}\right)$$

לכל היותר, $i = k\ln n$ חזרות. אז $k\ln n$ מרכזים במקום k.

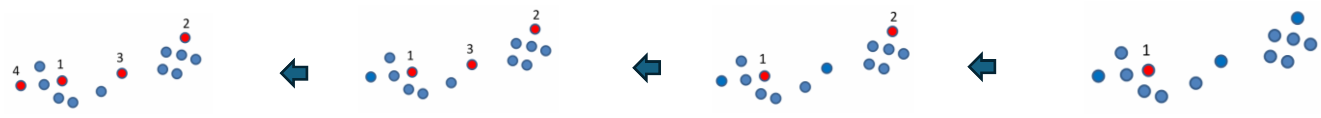
האלגוריתם השני

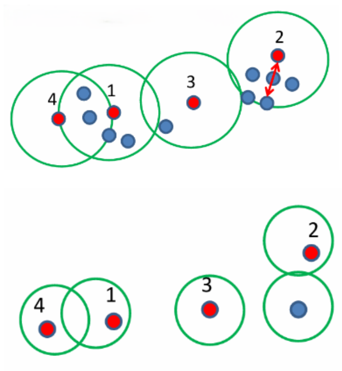
k מרכזים, עם רדיוס גדול יותר. אלגוריתם חמדן:

אנחנו לא נצליח למצוא קירוב עם רדיוס פחות מפי 2 האופטימלי (כי זה *NP-hard*) אבל נצליח למצוא בדיוק פי 2.

בכל שלב, נבחר את הנקודה הכי רחוקה. (הבחירה הראשונה שרירותית).

דוגמה עבור $k = 4$:





ונבחר את הרדיוס המינימלי כך שכל הנקודות מכוסות.

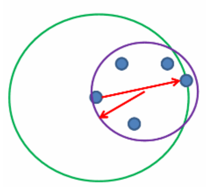
טענה: הרדיוס הסופי r הוא לכל היותר פעמיים הרדיוס האופטימלי r^*.

הוכחה: הרדיוס נקבע לפי $k+1$ נקודות: הנקודות שבחרנו למרכזים, והנקודה שהכי רחוקה מהמרכז הכי קרוב אליה.

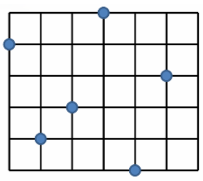
צריך לפחות $k+1$ כדורים עם רדיוס לכל היותר $r/2$ כדי לכסות את כל הנקודות. אז:

$$r/2 \le r^* \le r$$

במקרה הרציף

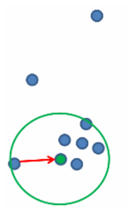


אפשר פשוט לפתור את המקרה הבדיד, ולאבד סדר גודל פי 2 מהרדיוס:



אפשר גם להשתמש ברשת:

כל ריבוע ברשת הוא $\epsilon/\sqrt{d}$. יש $\left(\sqrt{d}/\epsilon\right)^d$ נקודות שיכולות להיות שיכולות להיות מרכז.



בעיה שיש ב *k-center*, היא שהוא לא עמיד לחריגים:

אם יש נקודה שהרבה יותר רחוקה מכל השאר,
היא משפיעה יותר מידי.

יש מדדים אחרים יותר פופולריים:

k-median: נבחר K שממזערת את $\frac{1}{|S|}\sum_{v\in S} d(v,K)$.

k-means: נבחר K שממזערת את $\frac{1}{|S|}\sum_{v\in S} d^2(v,K)$. (הכי פופולרי).

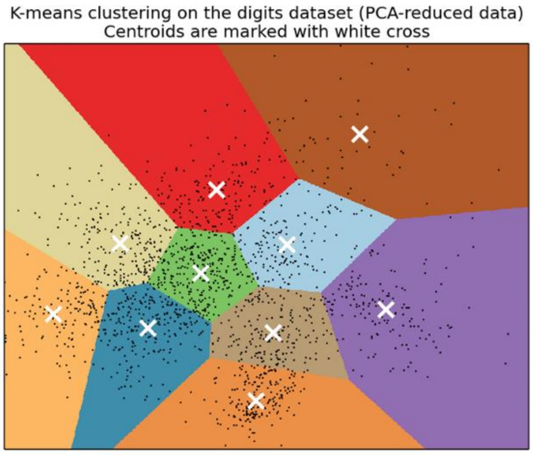
סוגי מדדים, והיחס למרחקי-ℓ_p שראינו:

במרחק ℓ_∞, הקורדינטה הכי גדולה קובעת את המרחק. זה דומה ל *k-center* בכך שהנקודה הכי גרועה קובעת את המדד עבור כולם.

במרחק ℓ_1, המרחק הוא סכום המרחקים בכל קורדינטה. ב *k-median*, כל נקודה תורמת את המרחק שלה.

במרחק ℓ_2, כל קורדינטה תורמת את ההפרש, ואז סכום הריבועים. דומה ל-*k-means*.

דיאגרמת וורונוי (*Voronoi*) ל *k-means*:



ממוצע – *k-median*

הצעת אלגוריתם:

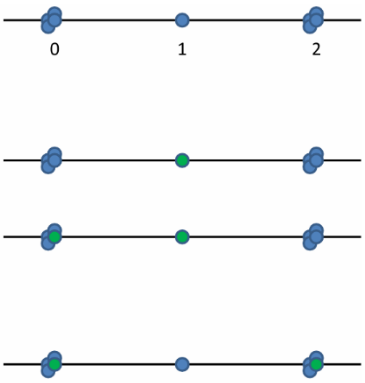
1. נגדיר: $K \leftarrow \emptyset$
2. כל עוד $|K| < k$:
   a. נמצא $p \in s$ שעבורו הקנס של $K \cup \{p\}$ הוא מינימלי.
   b. $K \leftarrow K \cup \{p\}$.

האלגוריתם החמדן לא אופטימלי:

בכל "גוש" יש $(n-1)/2$ נקודות. עבור $k = 2$:

החמדן ייתן קנס כולל $(n-1)/2$:

הקנס האופטימלי הוא 1:

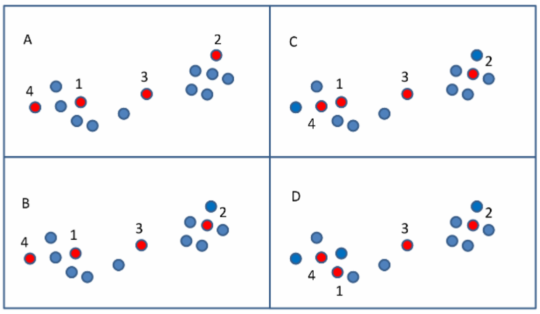


ננסה אלגוריתם *reverse-greedy*: (הוצע ע"י עמוס פיאט. *Chroback*, *Kenyon*, ו-*Young* הוכיחו שהוא נותן קירוב $\log n$).

1. נגדיר: $K_n \leftarrow S$
2. עבור $i = n$ עד $i = k + 1$:
   a. נמצא $p \in K_i$ שעבורו הקנס של $K_i \setminus \{p\}$ הוא מינימלי.
   b. $K_{i-1} \leftarrow K_i \setminus \{p\}$.

(ההוכחה לא בחומר)

עוד אלגוריתם: *local search*



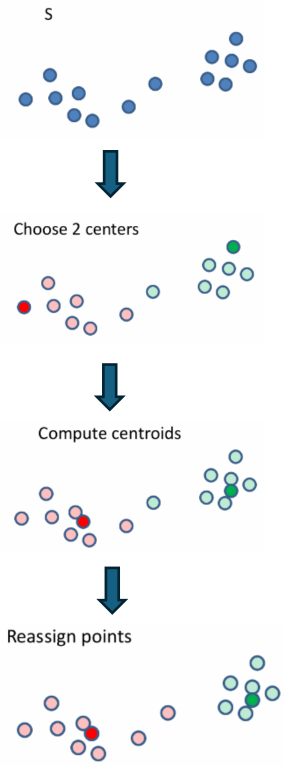
במקום לפתור את הבעיה באופן גלובלי, נבצע שינויים מקומיים קטנים יותר:

1. נתחיל עם קבוצת קודקודים שרירותית K.
2. נחליף קודקוד $p \in K$ עם קודקוד אחר $s \in S \setminus K$,
שממקסם את הורדת הקנס.
3. נחזור על 2 עד שיש איטרציה בלי שינוי.

אפשר להוכיח שהוא נותן קירוב של בערך פי 5 מהאופטימלי. (הרבה יותר טוב מ $\log n$).

מה זמן הריצה? יש מצבים שיכול להיות שזה משתפר רק קצת כל פעם, וזמן הריצה אקספוננציאלי.

אפשר להוסיף מנגנון שאם השיפור בין האיטרציות לא עובר רף מסויים, עוצרים.

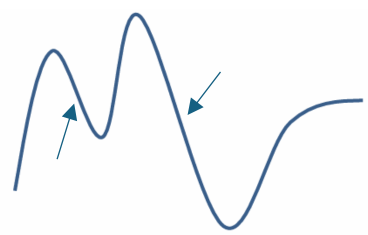


ממוצע בריבוע – *k-means*

אלגוריתם עבור המקרה הרציף (הכי פופולרי, **ה**-אלגוריתם):

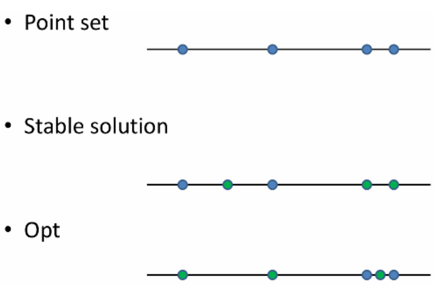
1. נבחר k קודקודים. (אפשר שרירותית, יש הרבה שיטות לבחירת קבוצה טובה. נקרא *seeding*).
2. כל נקודה משוייכת למרכז הכי קרוב.
3. לכל קבוצה, נחשב את נקודת האמצע (*centroid*) של הקבוצה:
אם כל נקודה היא וקטור, אז ה *centroid* הוא ממוצע של הווקטורים:
$$\frac{1}{|C|}\sum_{x\in C} x$$
4. ה-*centroids* הם המרכזים החדשים. נבצע שיוך מחדש של הנקודות.
נשים לב שהמרכזים החדשים הם לא בהכרח נקודות מהמדגם.
5. נחזור על שלבים 2,3,4 עד שאין שינוי.

גם פה, יש מקרים שזמן הריצה הוא $2^{\sqrt{n}}$. אבל בפועל, הזמן הממוצע הוא הרבה יותר טוב.



מה איכות הפיתרון? לאורך הריצה, הקנס יורד. ברגע שהוא מתחיל לעלות שוב, עוצרים. אבל אולי היה יכול להיות שיפור בעתיד:

לכן הבחירה הראשונה חשובה – זה ה *seeding*. נרצה להתחיל בחץ הימני יותר.



האלגוריתם הזה לא תמיד אופטימלי –

האלגוריתם ייעצר בפיתרון היציב (*stable*) אבל יש פיתרון יותר טוב: