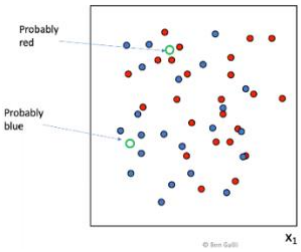
למידה חישובית ממדעי: תרגול 6 – KNN - K Nearest Neighbors

האלגוריתם הוא ממשפחה שנקראת instance based learning. נניח כי נרצה להשכיר דירה, נתבונן בדירות דומות ונראה שהמחיר שלהן הוא פחות או יותר כמו הדירה אותה אנו רוצים להשכיר. מבחינה יותר פרקטית, נניח שלפנינו הדאטה הכחול והאדום ונרצה לסווג את הנקודה החדשה הירוקה, מכיוון שמסביבה רק נקודות כחולות נרצה לסווג אותה ככחולה. זהו מאפיין ייחודי של **אלגוריתמים שהם מבוססי דגימות**.

לעומת זאת באלגוריתמים **שמבוססי-מודל**, כמו רגרסיה למשל, מסתכלים על הדאטה ומנסים למדל אותה לפונקציה או מישור שממדלים הכי טוב את הדאטה הקיים ובסוף הלמידה **נותר רק המודל** ולא הדאטה הקיים.

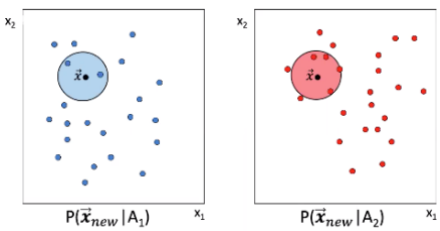


- משפחת האלגוריתמים שהם instance-based **אינם בונים מודלים לדאטה** (כמו עץ החלטה), במקום הם משווים אינסטנס חדש לאינסטנסים הקיימים ב-training data.
- ה**סיבוכיות** של אלגוריתמים מסוג זה: הלמידה היא מהירה מפני שבפועל אין באמת למידה, אבל קיים פוטנציאל לסיווג/חיזוי/קלסיפיקציה איטיים (ב-O(n)), מכיוון שיתכן מצב בו נצטרך לעבור על כל הדאטה שלנו, ולכן גם ה-**space complexity** גדול, והוא עבור כל הדגימות ולכן הוא O(n).
- ניתן להשתמש באלגוריתמים מסוג זה הן לקלסיפיקציה והן לרגרסיה.

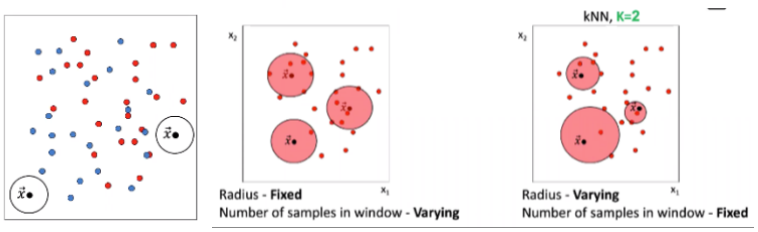
האלגוריתם הבסיסי שנלמד נקרא Parzen window אשר מבצע קלסיפיקציה

זהו אלגוריתם שמייצר **רדיוס חיפוש קבוע**, וכאשר מגיעה דגימה חדשה הוא מחפש את כל הדגימות הקיימות שנכנסות לתוך "הכדור" הרב ממדי שנוצר על פי הרדיוס הקבוע וקובע את הסיווג על פי ההסתברות. ועושים זאת על פי הנוסחה הבאה:

$$p(\vec{x}_{new}|A_i) = \frac{1}{n_i}\sum_{\vec{x}\in A_i}\frac{1}{h^d}K\left(\frac{\vec{x}_{new}-\vec{x}}{h}\right)$$



אבל אלגוריתם זה אינו יודע לטפל במצבים בהם הרדיוס לא תופס אף דגימה הקיימת בדאטה. אלגוריתם דומה שעזר לנו לפתור היה תיקון לפלס. אז נשנה את האלגוריתם – במקום שהרדיוס יהיה קבוע, נקבע את מספר הדגימות. אם קודם המעגלים היו בגודל קבוע, אז כעת נשנה את גודל הרדיוס כך שיכנסו לתוכו המספר הרצוי של הדגימות. **זהו האלגוריתם של KNN. נשים לב שה-k אינו תלוי בקלאס.**



אלגוריתם KNN

עכשיו השאלה שעולה, כיצד מגדילים את הרדיוס? בפועל, אנחנו לא באמת "מגדילים רדיוס", אלא אנחנו עוברים על כל הדאטה ומחשבים את מרחק כל הדגימות הקיימות מהדגימה החדשה, ו-k הדגימות בעלות המרחק הכי קצר הן אלו ש"יכנסו לרדיוס" = כלומר על פיהן נחשב הסתברות ונבצע סיווג/קלסיפיקציה או נמצע וכך נבצע פרדיקציה לרגרסיה.

- אם אנחנו **ברגרסיה נעשה פרדיקציה על פי הממוצע** $\hat{f}(x) = \frac{1}{k}\sum_{i=1}^{k} f(x^{(i)})$
- ואם **אנחנו בקלסיפיקציה נעשה פרדיקציה על פי ה-majority vote** $\hat{f}(x) = MAJ_i(\{f(x^{(i)})\})$

היתרונות והחסרונות של האלגוריתם

יתרונות

- זמן מהיר ללמידה / אין training
- אנחנו לא מאבדים מידע בתהליך המידול מפני שאנחנו שומרים את כל הדאטה
- ניתן ללמוד פונקציות קונספט / מטרה מאוד מורכבות

חסרונות

- זמן ריצה בפרדיקציה יכול להגיע ל-O(n) ובאופן כללי לא יהיה מהיר
- דורש אחסון זיכרון מרובה
- מושפע בקלות מדגימות / אטריביוטים פחות רלוונטיים

השאלות שעולות מהאלגוריתם הן:

(1) איך נגדיר "קרוב"?

(2) איך מתמודדים עם שאילתה איטית ועם מקום האחסון הגדול שעלינו להחזיק?

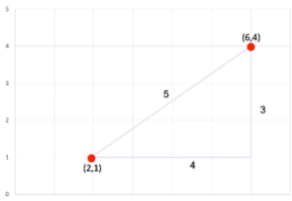
(3) ואיך נבחר את k?

1- איך נגדיר "קרוב"? יש המון סוגי מרחקים.

- מרחק עבור פיצ'רים נומריים נחשב על ידי מרחק L-P : כאשר l הוא האינדקס של ממד הוקטור ו-d הוא הממד של הוקטור.

$$Lp(x^{(i)}, x^{(j)}) = \sqrt[p]{\sum_{l=1}^{d} \left|x_l^{(i)} - x_l^{(j)}\right|^p}$$

- **אם p=2:** פונקציית **מרחק אוקלידי** משתמשת בשורש ריבועי ומעלה ריבועית (כמו בעולם שלנו). מרחק אווירי.
- **אם p=1:** פונקצייה זו נקראת **מרחק מנהטן**. סכום הצלעות.



L1 distance = 3+4=7

L2 distance = sqrt(16+9)=5

L_inf distance = max(3,4) = 4

להלן דוגמה בה משמאל מצאנו מרחק אוקלידי שיצא 5,

ומרחק מנהטן שיצא L1 distance = 3+4=7

שניהם מודדים את אותו המרחק בדרכים שונות (ודרך נוספת מחשבת 4).

Distance For Numeric Features

- When $p = 2$ the Lp distance is called the Euclidean distance
- When $p = 1$ the Lp distance is called the Manhattan distance
- When $p = \infty$ we define this function as follow:

$$L\infty(x^{(i)}, x^{(j)}) = MAX_l |x_l^{(i)} - x_l^{(j)}|$$

דוגמות חישוב 3:

- $x^{(1)} = (1,2,4),\ x^{(2)} = (4,0,3)$
- When $p = 2$:

$$L2(x^{(1)}, x^{(2)}) = \sqrt[2]{\sum_{l=1}^{3} \left(x_l^{(1)} - x_l^{(2)}\right)^2} = \sqrt[2]{(-3)^2 + (2)^2 + (1)^2} = \sqrt[2]{14}$$

- When $p = 1$:

$$L1(x^{(1)}, x^{(2)}) = \sum_{l=1}^{3} \left|x_l^{(1)} - x_l^{(2)}\right| = 3 + 2 + 1 = 6$$

- When $p = \infty$:

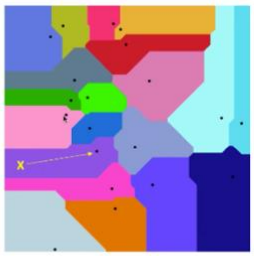
$$L\infty(x^{(1)}, x^{(2)}) = MAX(|-3|, |2|, |1|) = 3$$

Example:
$x^{(1)} = (7,2),\quad x^{(2)} = (5,5)$
Which point is closer to origin?
Depend on the distance method:
Euclidean - $x^{(2)}$
Manhattan - $x^{(1)}$

דוגמה נוספת מבחינת הסיבוכיות וההבדל בין מרחק אוקלידי ומרחק מנהטן - דיאגרמת vornoi:

כל נקודה בתוך צבע מסויים, הנקודה השחורה שקרובה אליה היא הנקודה שבתוך הצבע לפי שתי השיטות – תיאורטית KNN יכול ללמוד מכך פונקציית מאוד מורכבות.

Euclidean distance



Manhattan distance

מה לגבי דאטה שאיננו נומרי? למשל כחול, ירוק, אדום... נהפוך לדאטה נומרי ולאחר מכן נמדוד מרחקים (יש כאן קאץ' – אם נהפוך אותם ל-1,2,3 זה יאמר שלמשל ירוק קרוב יותר לכחול וכדומה ולכן יש להיזהר עם זה). או שנשתמש בדרכים כמו Hamming, Value Difference Measure וכו'... שאלו הן שיטות שמחשבות מרחקים על ערכים שאינם נומריים.

- "roses" and "toned" is 3
- "karolin" and "kerstin" is 3
- 1011101 and 1001001 is 2
- 2143896 and 2233796 is 3

מרחק Hamming = מרחק ההמינג בין שני סטרינגים בעלי אורך זהה הוא מספר המקומות שבהם ה"אותיות" בסטרינגים שונות. להלן דוגמות חישוב.

אפשרות נוספת היא ליצור וקטור מקודד כך שעבור כל צבע ניצור וקטור שכל slot בו מייצג איזה צבע אתה ולכן זהו וקטור יחידה (1,0,0), (0,1,0), (0,0,1), אבל קידוד זה יכול ליצור בעיה כאשר יש המון פיצ'רים ויש לבצע סוג של איזון.

אלגוריתם KNN ממושקל = מה קורה כאשר יש לי שכן שהוא "ממש יותר קרוב" אליי? נרצה להעניק לו משקל גבוהה יותר בהכרעה.

לכן הנוסחאות עבור KNN ממושקל יעניקו למרחק קצר יותר משקל גבוה יותר:

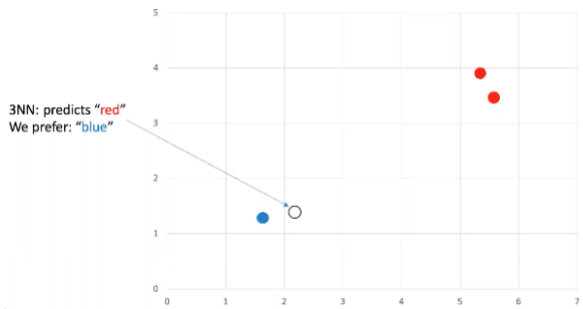
- עבור רגרסיה/ערכים רציפים נשתמש בנוסחה (ממוצע ממושקל):

$$\hat{f}(x) = \frac{\sum_{i=1}^{k} w_i f(x^{(i)})}{\sum_{i=1}^{k} w_i}$$

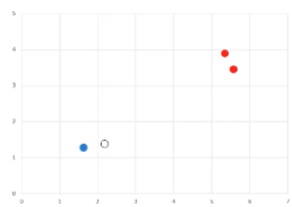
$$\text{Where } w_i = \frac{1}{distance(x^{(i)}, x)}$$

- ועבור קלסיפיקציה נשתמש ב- 1 חלקי המשקל. להלן דוגמה.

Weighted kNN - motivation



- K=3
- The 3 nearest neighbors:
  - X1 distance = 5, class = No
  - X2 distance = 2, class = Yes
  - X3 distance = 5, class = No
- Regular kNN will output 'No'
- The weighted:
  - $MAJ\left(\frac{No}{5}, \frac{Yes}{2}, \frac{No}{5}\right) =$ 'Yes'



2- שיפור היעילות של האלגוריתם מבחינת זמן ריצה ומבחינת מקום בזיכרון:

אנחנו מחשבים זמן ריצה כרגע באופן הבא: $T_{predict\ sample} = N_{samples} * T_{compute\ distance}$

נרצה לצמצם את זמן הריצה בעת שאילתה ואת המקום בזיכרון בו אנו משתמשים. לשם כך, למשל נצמצם את N הדגימות על ידי סינון דגימות.

מה שיעזור לנו לצמצם את זמן החיפוש בנוסף הוא שימוש במבנה נתונים מתאים למשל K-D Tree.

כדי לצמצם את זמן החישוב של המרחק – T, נבצע interrupt calculation, או שנצמצם את מספר הפיצ'רים.

מילה בנושא Curse of Dimensionality (נחזור לזה בהמשך)

מדוע לא נרצה להתחשב בכל הפיצ'רים באלגוריתם שלנו:

- מספר הדגימות הדרושות גדל באופן אקספוננציאלי עם מספר המשתנים.
- המידע הרלוונטי שמור רק במס' מועט של פיצ'רים מתוך כל הפיצ'רים הנתונים לנו.
- בממדים גבוהים כל הדגימות רחוקות אחת מהשניה – והדבר לא טוב עבור kNN.

בפועל, מעבר לנקודה מסוימת, הוספה של פיצ'רים נוספים מובילה לביצועים לא טובים.

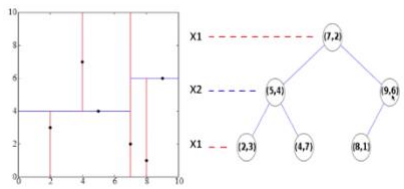
שיפור היעילות על ידי שימוש ב-K-D Tree:

במקום לחפש את השכן הקרוב ביותר על כל ה-training data נבנה מבנה נתונים שיעיל לחיפוש. נחלק את הדאטה ל-partitions, בכל פעם בממד שונה. נחפש אחר שכנים, ראשית נמצא את החלוקה הרלוונטית ואז נבצע חיפוש על החלוקה הזו בלבד.

הרעיון הכללי נכון אבל יש ניואנסים קטנים שאכן מובילים לאופטימיזציה, לא ניכנס אליהם.

דוגמה:

Points set: (2,3), (5,4), (9,6), (4,7), (8,1), (7,2)

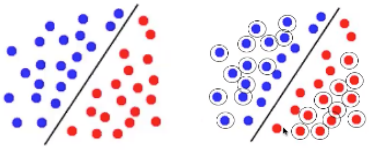


כיצד נצמצם דגימות מה-data למען שיפור היעילות?

המטרה שלנו היא: להוריד נקודות שלא משפיעות על גבולות ההכרעה – ככל שהנקודה רחוקה יותר מהגבול הסיכוי שהיא תשפיע על ההכרעה קטן.

ישנן שתי שיטות גרידיות לעשות זאת – מפני שהן תלויות סדר.

- השיטה הישירה: נכניס ל-training set נקודות אחת אחרי השנייה אבל נשמור רק את אלו שאינן מסווגות נכון.
- השיטה ההפוכה: נקבל את כל הנקודות לתוך ה-training set ונעבור על הנקודות ונסיר את אלו שמסווגות נכון על פי שכני ה-KNN שלהן.

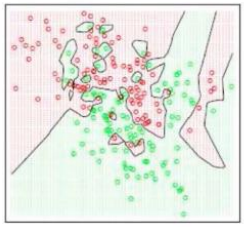


- **Backward KNN**(S)
  - T = S
  - For each instance x in T
    - if x is classified correctly by T-{x}
      - remove x from T
  - Return T
- **Forward KNN**(S)
  - T = ∅
  - For each instance x in S
    - if x is **not** classified correctly by T
      - add x to T
  - Return T

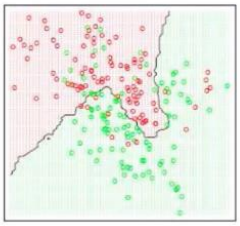
Overfitting מתקבל ב-kNN עבור k-ים קטנים מידי:

לכן יהיה עלינו להגדיל את ה-k עד לנקודה מסוימת. מפני שכל נקודה יוצרת מסביבה מרחב של כל הנקודות שקרובות אליה, לכן כאשר ה-k קטן מאוד ייווצרו איים כמו שניתן לראות בדוגמה עבור k=1, הדבר משפיע מאוד על הגבולות וזהו מצב של overfit. בנוסף, עבור training data אחר יתקבלו "נקודות רעש" שונות ולכן גם שם עשוי להתקבל overfitting.

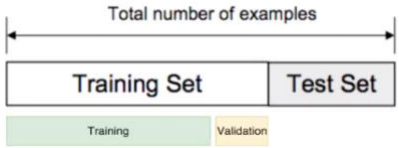
K=1



K=15

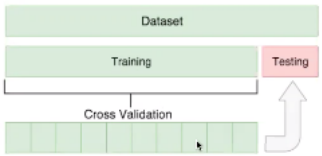


3- פיצול הדאטה ו-Cross Validation:



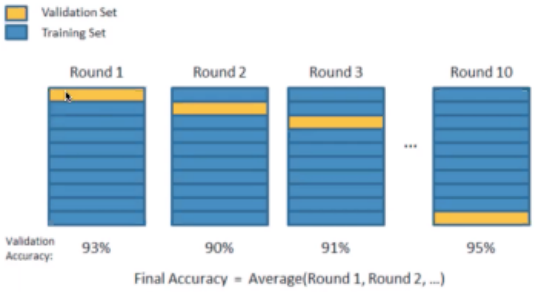
כאשר אנו משתמשים במודל סטטיסטי (כמו רגרסיה לינארית, למשל), אנחנו מתאימים את המודל על ה-training set במטרה לבצע פרדיקציה על דאטה חדשה. במטרה לעשות זאת אנחנו צריכים לפצל את הדאטה לקבוצות-training וtest. נשים לב שאנחנו לא יכולים להשתמש ב-test set במטרה לבחור את ההיפר-פרמטרים שלנו, ואנחנו חייבים בנוסף ל-test set, ליצור קבוצה נוספת של validation.

אבל, לפיצול הדאטה פעם אחת בלבד יש מספר חסרונות: אם הדאטה שלנו אינו מקיף מספיק אז אנחנו נהיה חייבים להימנע מפיצול כדי לא לאבד מידע, ואם נקבל פיצול שאינו מייצג אז שיטה זו לא תעבוד (נוכל לצמצם זאת על ידי הפעלת "שאפל" על הדאטה).



נשתמש ב-**Cross Validation** כאשר הדאטה שלנו איננו מספיק גדול לכדי פיצול ל-3 קבוצות אלה או כאשר אנחנו מעדיפים לא להסתמך על חלוקה ספציפית של 3 קבוצות. למעשה קרוס-ולידיישן עובד באופן דומה לפיצול val/split, מלבד העובדה שהוא מופעל על יותר תתי-קבוצות. כלומר, אנחנו מפצלים את הדאטה ל-k קבוצות, מתאמנים על k-1 קבוצות מתוך ה-k שחילקנו, ואת הקבוצה האחרונה נשמור ל-test. אנחנו יכולים לעשות זאת עבור כל אחת מתתי הקבוצות.

k-folds Cross Validation:



- ב-k-folds cross validation אנחנו מפצלים את הדאטה ל-k תתי קבוצות (או folds).
- אנחנו משתמשים ב-k-1 תתי קבוצות כדי לאמן את הדאטה ומשאירים את הפולד האחרון להיות ה-test set.
- לאחר מכן אנחנו עושים ממוצע על המודל כנגד כל אחד מה-folds ולאחר מכן אנחנו מסיימים למדל ובודקים נגד ה-test set

Cross Validation מניב תוצאה סטטיסטית יותר חזקה מאשר חלוקה רנדומית ל-3 קבוצות וגם מתאפשר להפעלה על גבי datasets קטנים.

Leave One Out Cross Validation (LOOCV):

- בסוג זה של ה-CV, מספר ה-folds (תתי הקבוצות) שווה למספר הדגימות בדאטה-סט ובכל פעם אנחנו משתמשים בכל הדגימות מלבד דגימה אחת שמשמשת טסט.
- לאחר מכן אנחנו ממצעים את כל ה-folds הללו.
- מכיוון שנקבל מספר גדול מאוד של training sets (שווה למספר הדגימות), שיטה זו מאוד יקרה מבחינה חישובית ועדיף להשתמש בה עבור datasets קטנים.
- אם ה-dataset הינו גדול, סביר כי יהיה עדיף להשתמש בשיטה אחרת, כגון k-folds.
- היתרון הגדול בשיטה זו היא רמת הדיוק. החיסרון הוא חוסר הפרקטיות עבור datasets גדולים.

אז מתי נשתמש באיזו שיטה?

- ככל שיש לנו יותר folds: נקטין את השגיאה על ה-bias אבל נגדיל את השגיאה על השונות, המחיר החישובי יגדל, ומן הסתם – ככל שיש לנו יותר folds ייקח יותר זמן לחשב עבור כולם וכן ידרש שימוש גדול יותר בזיכרון.
- ככל שיש לנו פחות folds: אנחנו מקטינים את הטעות על השונות, אבל הטעות על ה-bais תהיה גדולה יותר. מבחינה חישובית, זול יותר.
- לכן, ב-datasets גדולים מומלץ לקבוע k=10.
- ב- datasets קטנים, כפי שכבר צוין, עדיף להשתמש ב-LOOCV.

- For kNN – need to choose
  - K=?
  - P=?
- Cross validation – a method for hyper parameter optimization

parameters	Mean error
K=1, p=2	0.78
K=5, p=2	0.25
K=3, p=1	0.48