מבוא לבינה מלאכותית – תרגול 7

נושאים:

- **Boosting ו- XGBoost**
- **למידה לא מפוקחת – אַשכוּל:**

אלגוריתמים מבוססי מרחק: Prim, Graph Partitioning, אשכול היררכי.

אלגוריתמים לאשכול ב-$\mathbb{R}^n$: K-means, K-medoids, Fuzzy C-means, GMM.

Boosting:

דמיינו שאתם לומדים למבחן עם הרבה חומר (תנ"ך 5 יחידות לדתיים, אם תרצו, או מבנ"ת). מה עדיף – ללמוד במכה אחת את כל החומר, או ללמוד בחלקים?
אינטואיטיבית, הגישה של Boosting היא ללמוד בחלקים, לדעת שלמדת טוב כל חלק ולהתקדם לחלק הבא.

בפירוט, Boosting משתמש באוסף של מה שמכונה "Weak Learners" לצורך סיווג. בצורה איטרטיבית, כל weak learner מצליח לסווג חלק מתוך הדאטא בוודאות גבוהה. כאשר כבר סיווגנו נקודות מסוימות בוודאות גבוהה, אין מה לחזור ולסווג אותן ולכן ה-weak learner הבא ייתן עדיפות לסיווג של נקודות שעוד לא סווגו בוודאות גבוהה.

<u>אלגוריתם:</u>

בהינתן סט אימון $S = (x_1, y_1), \dots, (x_m, y_m)$, weak learner שבחרנו מראש ומספר סיבובים T.

1. אתחל את הבעיה כאשר לכל הנקודות אותה חשיבות לסיווג: יש לנו וקטור התפלגות $D^{(1)} = (\frac{1}{m}, \dots, \frac{1}{m})$ שמתאר את החשיבות של כל נקודה.
2. לכל $t = 1, \dots, T$:
   - הפעל Weak learner ללמוד על הדאטא.
   - חשב את $\epsilon_t = \sum_{i=1}^{m} D_i^{(t)} \mathbb{I}_{[y_i \neq h_t(x)]}$, פונקציית שגיאה של הלומד, ממושקלת לפי חשיבות הנקודות.
   - חשב משקל מתאים ללומד w_t כתלות בשגיאה ϵ_t. ככל שהשגיאה נמוכה יותר, נרצה שהמסווג יהיה בעל משקל גבוה יותר.
   - עדכן את וקטור החשיבויות $D_i^{(t+1)} = D_i^{(t)} \cdot something(i)$, כך שאם הוודאות של הסיווג של i גבוהה אז החשיבות תקטן ולהפך. ($something$ יכול להיות למשל $softmax$, כך ש- $D_i^{(t+1)} = \frac{D_i^{(t)} e^{-w_t y_i h_t(x)}}{\sum_j D_j^{(t)} e^{-w_t y_i h_t(x)}}$).
3. לבסוף נקבל מסווג $h_{final}(x) = \sum_{t=1}^{T} w_t h_t(x)$, עד כדי נרמול בסכום המשקלים.

דוגמה – XGBoost:

זהו אלגוריתם שעושה Boosting ומשתמש בעצים כ-weak learners.
מבלי להיכנס לפרטים, האלגוריתם מבסס את הניחוש שלו על אוסף של עצי החלטה. פונקציית השגיאה מורכבת משני איברים: loss מבוסס על הפער בין התיוג האמיתי והתיוג שניחשנו, ורגולריזציה שנועדה לשמור על העצים בגודל סביר.
המאמץ להקטנת פונקציית השגיאה (או, בשפת בני אדם, אימון) מתבצע בצורת Boosting, כאשר העץ שמתווסף בכל איטרציה יהיה זה שיקטין את השגיאה ככל האפשר (מעשית, באלגוריתם מבצעים קירוב מסדר שני לשגיאה ובעזרתו מוסיפים את העץ המתאים).
באלגוריתם יש עוד גורמים שמנסים להילחם ב-overfitting מעבר לרגולריזציה: מצמצמים את התרומות שעצים אינדיבידואליים מכניסים (דומה ל-learning rate ב-gradient descent) ומשתמשים ב-subsampling (דגימה של חלק בלבד מכלל הפיצ'רים לבניית העצים).
לא אכנס לשאלה איך המודל בוחר לפצל עלים בעצים.

מוזמנים לקרוא בפירוט את המאמר שמתאר את האלגוריתם ולראות כיצד כמה רעיונות שראינו בקורס משתלבים באלגוריתם.

למידה לא מפוקחת:

ההבדל בין מה שעשינו עד היום (למידה מפוקחת) לבין למידה לא מפוקחת הוא שאנחנו לא יודעים תיוג לדוגמות. המשימות שאנחנו יכולים להעמיד לעצמנו הן לא משימות של סיווג, אלא משימות כמו חלוקה של הדגימות לקבוצות (אשכול), הורדת ממדים (PCA, ICA, Auto-encoding), זיהוי חריגות ועוד.
דוגמה קונקרטית: טלסקופ חלל אוסף נתונים על גרמי שמיים לא ידועים. הטלסקופ יכול לאסוף פיצ'רים של גרמי השמיים (כמו ספקטרום התדרים המגיעים מכל גרם שמיימי ועוד), ומכיוון שהתיוג לא ידוע, אין אלא לנסות ולבדוק אם יש קבוצות של כוכבים שדומים זה לזה.

אשכול:

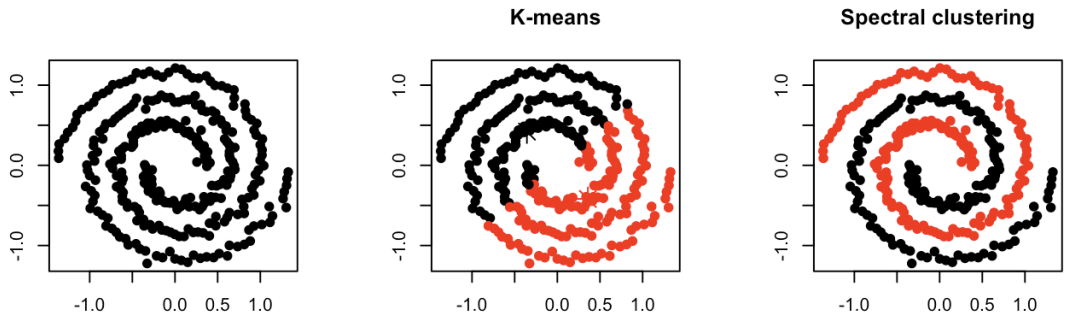
אלגוריתמי אשכול מבוססי מרחק:

לא תמיד יש לנו פיצ'רים לכל קדקוד. לעתים, מה שיש לנו הוא רק מרחקים פנימיים בין הדוגמות, אבל גם זה מספיק לנו לאשכול. נראה כעת 3 אלגוריתמים לאשכול קשיח, כלומר לכל נקודה יהיה סיכוי 1 להשתייך לאחד האשכולות ו-0 לשאר.
כאמור, יש לנו אך ורק מרחקים בין נקודות.

Graph Partitioning (ידוע גם כאשכול ספקטרלי):
בעזרת המרחקים נבנה גרף ממושקל, כאשר משקלי הגרף הופכיים למרחקים ($w_{ij} = 1/d_{ij}$, מה שמכונה דמיון). השיטה הזו מוצאת חלוקה של כלל הגרף לשתי קבוצות A, B זרות כך ש-$\sum_{i\in A, j\in B} w_{ij}$ מינימלי (למרות שאפשר להשתמש בה כדי לחלק ליותר קבוצות, בעזרת שימוש ביותר וקטורים עצמיים).
בקצרה (כי למדתם את זה בהרצאה), מחשבים לפלסיאן של הגרף, מחשבים לו ערך עצמי ווקטור עצמי שני (כי הראשון חסר משמעות עבורנו – ע"ע 0 עם ו"ע קבוע לכל רכיב). כעת, לכל נקודה אפשר לשייך גודל מסוים שניתן לה בהתאמה לרכיב בווקטור. כעת, נפריד את הנקודות לפי ערך סף שכל הנקודות עם רכיבים בגודל גדול ממנו ישתייכו ל-A והשאר ל-B.

היתרון באלגוריתם זה מגיע מהגדרת הבעיה. הוא מסוגל לפתור בעיית אשכול שמבוססת רק על המרחק של הנקודות אלה מאלה, ולא על איך הן מוצבות במרחק. אלגוריתמים שנראה בהמשך מניחים שלאשכולות יש מרכזים שסביבם מפוזרות (בצורה כלשהי) הנקודות, אבל זה לא תמיד נכון.

למשל:



יש לאלגוריתם גם חסרון עיקרי – חישוב ערכים עצמיים בהינתן סט אימון גדול הוא משימה שלוקחת הרבה זמן לחישוב.

Prim:

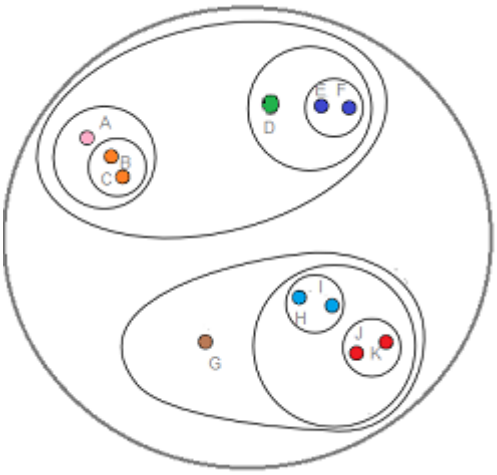
אלגוריתם Prim הוא אלגוריתם שמוצא עץ פורש מינימלי בגרף ממושקל חיובי. הפעם, מקבוצת הנקודות שלנו נבנה גרף ממושקל עם משקלים שהם המרחקים עצמם.
הפעלת האלגוריתם על גרף (אם הכל אידיאלי) משאירה עץ שבו שני סוגי קשתות: הרבה קשתות במשקלים נמוכים במיוחד (בין דוגמות שנמצאות באותו אשכול) ומעט קשתות במשקלים גבוהים (בין דוגמאות מאשכולות שונים). נותר להסיר את המשקלים הגבוהים ויישארו לנו אשכולות נפרדים.

הבעיה העיקרית באלגוריתם זה – אין לנו דרך ממשית לדעת כמה טוב הצלחנו, זה אלגוריתם היוריסטי. למרות זאת, זה אלגוריתם פשוט מאוד שבעבר (לפני שרשתות נוירונים הפכו למוצלחות מאוד) השתמשו בו ל-image segmentation.

אשכול היררכי:

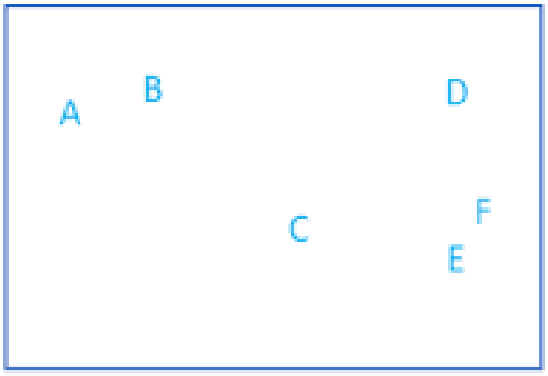
בגישה שנלמד לשיטה זו (bottom up), האלגוריתם מתחיל מאשכולות המורכבים מנקודות בודדות ובכל איטרציה שלו מחברים בין שני אשכולות שהמרחק ביניהם הוא הקטן ביותר. אפשר להשתמש בפונקציות שונות לחישוב מרחקים בין אשכולות (כמו מרחק מינימלי בין נקודות מאשכולות שונים, מרחק ממוצע ועוד).

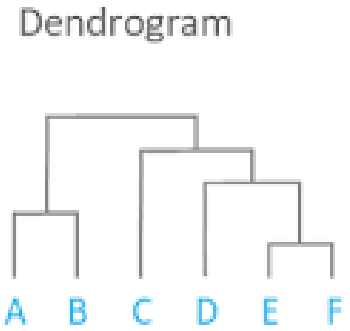
מה שנוצר הוא דנדרוגרם (dendrogram):



נחליט על רמה מסוימת בעץ, שממנה נסתכל (כלומר, אין לנו שום אינטרס להשתמש בשלב הסופי של האלגוריתם, שם הכל מחובר לאשכול אחד ענק. נצטרך "לחתוך" ברמה מסוימת כך שנקבל מספר אשכולות).

אלגוריתם זה בעייתי גם הוא – שוב אין לנו פונקציית מטרה שרוצים למזער. מעבר לכך, האלגוריתם יכול פשוט שלא לעבוד כמו שצריך כשהדנדרוגרם עשוי להיות כמו שרואים למטה – מכיל אשכול אחד גדול שבמרבית האיטרציות פשוט מתווספות אליו עוד ועוד נקודות.





אלגוריתמי אשכול ב-$\mathbb{R}^n$:

כעת, בשונה מקודם, לכל דגימה משויכת נקודה במרחב ממש. נלמד אלגוריתמים שמשתמשים במיקומים האלה במרחב כדי לחלק את הנקודות לאשכולות.

K-means:

עוד אלגוריתם לאשכול קשיח. האלגוריתם הזה משתמש ב-K מרכזי אשכולות שמתעדכנים באיטרציות ובהתאם שיוך הנקודות לאשכולות משתנה.

<u>אלגוריתם:</u>

1. אתחול אקראי של מרכזים $\mu_1, \dots, \mu_k$.
2. לכל איטרציה:

- שייך כל נקודה i לאשכול $j = \arg\min_{\mathrm{n}} \left|\left|x_i - \mu_n\right|\right|^2$ (כאשר המרחק הוא אוקלידי).

- עדכן את המרכזים: $\mu_n^{new} = \frac{1}{m_n}\sum_{j \in Cluster_n} x_j$ כאשר m_n מספר הנקודות באשכול ה-n.

יש לציין: האלגוריתם יתכנס למינימום כלשהו על השגיאה $L = \sum_{n=1}^{k}\sum_{j \in Cluster_n}\left\|x_j - \mu_n\right\|^2$ (ובגלל זה לוקחים מטריקה אוקלידית ולא כל מטריקה אחרת), אבל הוא לא מוכרח להיות מינימום גלובלי, וזה נכון גם לאלגוריתמים אחרים שמופיעים כאן.

K-medoids:

הפעם, משתמשים במרחב רק במה שיש. כלומר, לא יוצרים נקודות חדשות, אלא למרכזי האשכולות לוקחים נקודות קיימות.

<u>אלגוריתם:</u>

1. אתחול אקראי של k נקודות שמשמשות מרכזים.
2. שייך כל נקודה למרכז הקרוב אליה, לפי מטריקה שנבחרה מראש.
3. כל עוד פונקציית העלות $\sum_{C_i}\sum_{P_i \in Cluster_i}\|P_i - C_i\|$ (לפי המטריקה שבחרנו) יורדת:
   לכל מרכז ולכל נקודה שאינה המרכז, החלף בין הזוגות וחשב מחדש את העלות.
   אם היא ירדה, השאר בתוקף את ההחלפה בין המרכזים,
   אחרת החזר למצב שהיה.

אלגוריתם זה דומה מאוד ל-K-means, אבל K-medoids הוא קצת יותר יציב מ-K-means, מכיוון שהוא ממזער פונקציית שגיאה שמבוססת על מרחקים כלליים בין זוגות הנקודות ולא בהכרח על סכום מרחקים אוקלידיים. לעומת זאת, האלגוריתם הזה פחות יעיל חישובית מ-K-means ואתחולים שונים שלו עלולים להביא לפתרונות שונים לגמרי.

Gaussian Mixture Model:

זהו אלגוריתם אשכול שמכונה אשכול רך. הסיבה – כל נקודה לא משויכת לאשכול יחיד אלא יש לה סיכויים להשתייך לכל אשכול, והאשכול שנבחר להתאים לכל נקודה תהיה זה שהסיכוי של הנקודה להשתייך לשם יהיה הכי גבוה.

הפעם, נניח שקיימים K אשכולות, לכל אשכול יש מרכז $\overrightarrow{\mu_j}$, מטריצת שונות covariance Σ_j שמתארת את השונות של כל הנקודות שבאשכול, וסיכוי אפריורי (כלומר, בלי קשר למרחקים וכו') של נקודה בדאטא להשתייך לאשכול π_j. לכל נקודה יש משתנה סמוי המסמן לאיזה מרכז הוא משתייך z_{ij} (דגימה i נלקחה מאשכול j).

אפשר לכתוב פונקציית מטרה המעריכה את ערכי המשתנים $\overrightarrow{\mu_j}, \Sigma_j, \pi_j$:

$$\sum_z P(z_{ij}) \ln(P(\{x\} | \overrightarrow{\mu_j}, \Sigma_j, \pi_j, z_{ij}))$$

באמצעות אלגוריתם Expectation maximization אפשר להתכנס למקסימום (לוקלי) של הפונקציה ולמצוא את הפרמטרים המתאימים, ומהם לקבל את הסיכוי של כל נקודה להשתייך לגאוסיאן.

הערה – בדרך כלל, חישוב המטריצות Σ_i סובל מאוד מרעש (בגלל ריבוי פרמטרים להערכה בעזרת לא מספיק נקודות). לכן, לעתים מניחים הנחות מקלות על מבנה המטריצה (למשל: אלכסונית, זהה עבור כל האשכולות, כדורית = שונות שווה בכל כיוון ועוד).

Fuzzy C-means:

זהו אלגוריתם אשכול רך שמשלב אלמנטים מ-K-means ומ-GMM.

הפעם, מגדירים את פונקציית המטרה להיות $L = \sum_{i \in Points, j \in Centers} u_{ij}^m \left\| \overrightarrow{x_i} - \overrightarrow{\mu_j} \right\|^2$. נדרוש גם שלכל נקודה i, $\sum_j u_{ij} = 1, u_{ij} \geq 0$

(u_{ij} מסמל את השיוך של נקודה לאשכול, זה כמו $P(z_{ij})$ ב-GMM). חיפוש מקסימום על הפונקציה נותן כלל עדכון חשוב ל-μ_j:

$$\mu_j = \frac{\sum_i u_{ij}^m \overrightarrow{x_i}}{\sum_i u_{ij}^m}$$

הכלל אומר לנו: אם $m \to 0$, מקבלים ממוצע של כל הנקודות הנתונות. אם $m \to \infty$, רק נקודות מעטות שקרובות למרכזים תורמות, המרכזים יהיו באזורים צפופים מאוד.

אם $m = 1$, אנחנו במקרה פרטי של GMM כדורי. בסך הכל, m הוא פרמטר של המודל (יותר מדויק יהיה לכתוב היפר-פרמטר) שקובע האם המודל מחפש לוקאליות (m גדול) או גלובאליות (m קטן) של הדאטא.