

מבוא לבינה מלאכותית – תרגול 7

נושאים:

- (סיום אלגוריתמי האשכול - Fuzzy C-means , GMM)
 - זיהוי קהילות (Community Detection) – אשכול היררכי, Girvan-Newman, Louvain
-

זיהוי קהילות (Community Detection):

למעשה זה המשך די ישיר לנושא של אשכול. הפעם, נתון לנו גרף כלשהו, אפשר עם משקלים על הקשתות שמסמלים את מידת הדמיון שבין קודקודים שכנים אך אפשר גם ללא משקלים כאלה, והמשימה שלנו היא לחלק את קודקודי הגרף לקהילות. האלגוריתמים השונים מחליטים על סמך מה בדיוק לבצע את החלוקה לקהילות (למה קודקוד אחד יהיה באותה קהילה עם קודקוד אחר, ולא יהיה בקהילה עם קודקוד שלישי), אך הרעיון המרכזי של האלגוריתמים הוא ששני קודקודים שקרובים ומקושרים חזק אחד לשני אמורים להיות דומים יותר זה לזה מאשר שני קודקודים שהם רחוקים זה מזה או שהקשר שביניהם נעשה על ידי קשתות במשקלים נמוכים, ולכן נעדיף לשייך את הזוג הראשון לאותה קהילה על פני שיוך של הזוג האחרון.

Community Detection באמצעות אשכול היררכי:

כמו בשיעור שעבר, בהינתן מדד דמיון בין כל זוג קודקודים, אפשר לבצע חלוקה לקהילות באמצעות אשכול היררכי.

ספציפית לתרגול הזה, אתן דוגמה למדד הדמיון הבא:

בהינתן גרף (לא ממושקל) עם מטריצת שכנויות A , כמות המסלולים שבין קודקוד i לקודקוד j שאורכם k היא $(A^k)_{ij}$. לכן, כמות המסלולים הכוללת שבין i ו- j בכל אורך היא $\sum_{k=1}^{\infty} (A^k)_{ij}$. הכמות הזו מתבדרת, ולכן במקום זה ניקח α קטן מספיק ונחשב $\sum_{k=0}^{\infty} \alpha^k (A^k)_{ij}$, כך שהפעם זה יתכנס. זה בעצם טור הנדסי של מטריצות, שמתכנס למטריצה הבאה $D = (I - \alpha A)^{-1}$. המטריצה הזו תהיה מטריצת הדמיון שבין כל זוג קודקודים, ובאמצעותה ניתן לבצע אשכול היררכי ולחלק את הגרף.

אלגוריתם Girvan-Newman:

האלגוריתם הזה מתמקד בהסרה איטרטיבית של קשתות שמהוות "גשרים" שבין קהילות. שימו לב שהוא, כמו זה שלמעלה, מקבל גרף שאינו בהכרח ממושקל. מחשבים לכל קשת מדד שקובע כמה היא "מרכזית", ובצורה איטרטיבית הקשתות ה"מרכזיות" ביותר יורדות מהגרף עד שלא נשארות כאלה. את התהליך אפשר לתאר בתור בנייה מלמעלה למטה של dendrogram (שראינו בשיעור שעבר בנושא של אשכול היררכי, אבל שם הבנייה הייתה bottom up), ואת הבחירה הסופית של החלוקה לקהילות אפשר לקחת בתור שלב מסוים בעץ, כמו באשכול היררכי.

בצורה פורמלית יותר:

אלגוריתם:

כל עוד יש קשתות בגרף:

1. לכל קשת בגרף, מחשבים edge centrality –

$$c(e) = \sum_{u,v} \frac{\sigma(u, v|e)}{\sigma(u, v)}$$

כאשר e קשת, u, v זוג קודקודים, $\sigma(u, v)$ מספר המסלולים המינימליים (ביחס לאורך, ללא משקלים) שבין u -ל- v בגרף, $\sigma(u, v|e)$ מספר המסלולים המינימליים שבין u -ל- v בגרף, שעוברים דרך e .

2. מורידים את הקשת בעלת ה- edge centrality הגבוה ביותר. רכיבי הקשירות שנותרו הם ה-

communities יחסית לשלב הזה, ומהם בונים את ה-dendrogram.

3. חוזרים ל- 1 עבור הגרף הנותר.

חסרון בולט של שני האלגוריתמים שראינו עד כה הוא שאין בהם פונקציית מטרה מוגדרת. האלגוריתם השני טוב יותר מהראשון, כי הוא לא צפוי להיתקל בבעיה שהאשכול ההיררכי bottom up נוטה להיתקל בה (בניית אשכול גדול שבכל פעם מצטרפת אליו נקודה), אך מצד שני הוא יקר יחסית – הסיבוכיות שלו היא בסדר גודל של מספר הקשתות (מס' הורדות הקשתות) כפול מספר הקודקודים בריבוע (עלות חישוב ה- edge centrality לכל קשת).

אלגוריתם Louvain:

סתם לידע כללי, אלגוריתם זה הומצא ע"י Blondel ושותפים שלו, והוא נקרא על שם האוניברסיטה בה למדו (בבלגיה).

זהו אלגוריתם חמדן שמחפש מינימום לפונקציה מוגדרת, שנקראת modularity ומוגדרת כך:

$$L = -\frac{1}{2m} \sum_{i,j} \left[A_{ij} - \frac{k_i k_j}{2m} \right] C(i,j)$$

כאשר:

- m סכום כל הקשתות בגרף (כולל משקל אם יש).
- k_i הדרגה (הממושקלת) של קודקוד i .
- $C(i,j)$ היא למעשה הדלתא של Kronecker לגבי האם הקודקודים i, j שייכים לאותה קהילה (1 אם כן, 0 אחרת).

אינטואיציה: נגיד שיש לנו שני קודקודים שאינם באותו רכיב קשירות. אם נבחר לשים אותם באותה קהילה ($C(i,j)$ יהיה 1), נוסיף סתם $\frac{1}{4m^2} k_i k_j$ לערך הפונקציה, וזה מרחיק אותנו מחיפוש המינימום. אם i, j מחוברים זה לזה בקשת עם משקל A_{ij} גדול מאוד, אז כדאי לנו ששניהם יהיו באותה קהילה, כי אז זה יקטין את ערך הפונקציה, ולכן נעדיף שבמקרה כזה $C(i,j)$ יהיה שווה 1.

הערה – לאלגוריתם קיימת גרסה עם קבוע שרירותי γ שכופל את $\frac{k_i k_j}{2m}$. התפקיד שלו:

אם γ קטן, אז נעדיף לקבץ הרבה קודקודים יחד ולקבל מעט קהילות גדולות יחסית, כי $\left(A_{ij} - \gamma \frac{k_i k_j}{2m} \right)$ יהיה חיובי וזה יקטין את ערך הפונקציה. אם γ גדול, בדיוק ההפך – נקבל מספר קהילות גדול ומעט קודקודים בכל קהילה.

איך מוצאים את חלוקה שמהווה מינימום לפונקציית ה-modularity?

אלגוריתם:

1. כל קודקוד מאותחל בתור שייך לקהילה משל עצמו.
2. בכל שלב, נעבור על כל הקהילות הקיימות. לקהילה ה- i , נבדוק עבור כל קהילה אחרת j מה ההפרש בפונקציית ה-modularity כאשר "נוציא" את הקהילה i ו"נכניס" את הקודקודים שלה לקהילה j . עבור הקהילה שמתקבל בה הרווח הגדול ביותר עבורנו (הפונקציה תרד בהכי הרבה), אם אכן קיים רווח כזה, נאחד את הקהילות ונמשיך בתהליך עבור הקהילה הבאה. אם אין איחוד רווחי, לא מאחדים.
3. כאשר אין עוד צעד איחוד שמקטין את ה-modularity, מסיימים.

אלגוריתם זה מהיר יותר מהקודמים, עלות החישוב שלו עבור גרף עם n קודקודים היא $O(n \log^2 n)$.
חוץ מזה, כאן באמת יש פונקציית מטרה שמשתמשים בה, וזה עוזר להבין מה קורה באלגוריתם
ולשלוט בו.
שימו לב שהאלגוריתם הזה תלוי בסדר הריצה על הקהילות, לכן הוא יכול להתכנס לערכי מינימום
שונים עבור הרצות שונות.