PADR 2018/2019

Praca domowa nr 4 (max. = 30 p.)

> W ramach niniejszego projektu zaimplementujesz i przetestujesz algorytm spektralny analizy skupień (*spectral clustering*) oparty na grafie kilku najbliższych sąsiadów punktów z wejściowego zbioru danych.

Termin oddania pracy: 25.01.2019, godz. 23:59.

Do przesłania na adres `M.Gagolewski@mini.pw.edu.pl` ze swojego konta pocztowego `*@*pw.edu.pl` – **jedno archiwum `.zip`**[1] o nazwie typu `Nick_Nazwisko_Imie_NrAlbumu_pd4.zip`. W archiwum znajdować się powinien jeden katalog, `Nick_Nazwisko_Imie_NrAlbumu_pd4`, dopiero w którym umieszczone zostaną następujące pliki:

- plik `spectral_aux.cpp` zawierający implementacje (w Rcpp) funkcji `Mnn()` oraz ewentualnie dodatkowych funkcji pomocniczych; [6 p.]
- plik `spectral.R` zawierający implementacje funkcji `Mnn_graph()`, `Laplacian_eigen()` oraz `spectral_clustering()`; [6 p.]
- plik `testy.Rmd` i `testy.pdf` – testy poprawności zaimplementowanych metod na przynajmniej trzech *własnych* zbiorach danych z $\mathbb{R}^2$ lub $\mathbb{R}^3$ (z ilustracjami m.in. w postaci wykresów); [3 p.]
- plik `raport.Rmd` i `raport.pdf` – raport z analizy danych benchmarkowych; [15 p.]
- pliki `.csv` zawierające wyniki benchmarków oraz pliki `.R` je generujące – to na ich podstawie stworzysz raport.

Nazwy plików nie powinny zawierać polskich liter diakrytyzowanych (przekształć *ą*→*a* itd.). Treść wysyłanego e-maila nie może być pusta. Tytuł wiadomości to `[PADR] Praca domowa nr 4`.

1 Zadanie analizy skupień

Niech dana będzie macierz $\mathbf{X} \in \mathbb{R}^{n \times d}$ reprezentująca n punktów $\{x_1, \ldots, x_n\}$ w $\mathbb{R}^d$. Zadanie analizy skupień[2] (ang. *cluster analysis*) jest przykładem uczenia bez nadzoru. W dużym uproszczeniu, jego celem jest *automatyczne* znalezienie takiego *podziału* zbioru danych na $k > 1$ (dane z góry) parami rozłącznych i niepustych podzbiorów – zwanych *skupieniami* – tak by obserwacje należące do tego samego skupienia były do siebie jak najbardziej *podobne* (np. leżały „blisko" siebie), zaś obserwacje z dwóch różnych skupień były możliwie jak najbardziej od siebie *odmienne*.

2 Ocena jakości podziału i zbiory benchmarkowe

Wynikiem działania wszystkich rozpatrywanych tutaj algorytmów analizy skupień będzie ciąg $\mathbf{z} \in \{1, \ldots, k\}^n$, taki że z_i określa, do którego z k skupień należy punkt x_i. Zachodzi oczywiście $(\forall j = 1, \ldots, k)\ (\exists i)\ z_i = j$.

Zbiory benchmarkowe należy pobrać z naszego repozytorium na GitHubie (folder `prace_domowe/pd4-zbiory-benchmarkowe`). Każdy z nich składa się z macierzy $\mathbf{X} \in \mathbb{R}^{n \times d}$ (plik `.data`) oraz ciągu referencyjnych (wskazanych przez ekspertów) etykiet $\mathbf{y} \in \{1, \ldots, k\}^n$ (plik `.labels0`) dla pewnych n, d, k.

[1] A więc nie: .rar, .7z itp.

[2] Zob. np. [Koronacki J., Ćwik J., *Statystyczne systemy uczące się*, EXIT, 2008, rozdz. 9] lub [Hastie T., Tibshirani R., Friedman J., *The Elements of Statistical Learning*, Springer, 2017, rozdz. 14.3] – http://web.stanford.edu/∼hastie/ElemStatLearn/

Zakładamy tutaj, że algorytm analizy skupień jest *dobry*, jeśli generuje podziały podobne do referencyjnych etykiet. Do oceny podobieństwa dwóch k-podziałów będziemy stosowali:

- indeks Fowlkesa–Mallowsa (FM)[3], zob. `dendextend::FM_index`;
- skorygowany indeks Randa (AR)[4], zob. `mclust::adjustedRandIndex`.

Każdy z powyższych indeksów zwraca wartość równą 1, jeśli dwa dane k-podziały są równoważne. Im ich wartość jest dalej od 1, tym bardziej są one od siebie różne.

Będzie nas interesować zachowanie się wszystkich testowanych algorytmów na wszystkich badanych zbiorach benchmarkowych oraz trzech zbiorach własnej produkcji (np. ranking algorytmów względem uśrednionego po wszystkich zbiorach indeksu Randa).

3 Metody do przetestowania i raport z wykonanych badań

Należy zbadać następujące algorytmy analizy skupień:

- własną implementację algorytmu spektralnego wg opisu poniżej (dla różnych parametrów `M`);
- wszystkie algorytmy hierarchiczne z funkcji `hclust()`;

  Wyniki zwracane przez algorytmy hierarchiczne należy przetworzyć przy użyciu funkcji `cutree()`.
- algorytm *Genie* z pakietu `genie`;
- co najmniej jeden inny algorytm z jakiegoś pakietu na CRAN.

Aby uzyskać maksymalną liczbę punktów, należy:

- [`testy`] utworzyć (w postaci plików `.data` oraz `.labels0`) i osobno opisać co najmniej trzy zbiory benchmarkowe w $\mathbb{R}^2$ lub $\mathbb{R}^3$ wedle własnego uznania (różne kształty, różne sposoby generowania) i dołączyć je do ww. baterii zbiorów testowych;
- [`raport`] zbadać jakość (tj. policzyć indeksy FM, AM i AR dla generowanych podziałów względem etykiet referencyjnych) każdego algorytmu na wszystkich zbiorach benchmarkowych;

  Należy przedstawić nietrywialne podsumowania wyników (wykresy, tabele, ...). W raporcie nie umieszczamy wyników surowych. Raport ma być **czytelny**.

  Jeśli testowane algorytmy udostępniają różne parametry do tuningowania, należy zbadać ich wpływ na jakość wyników.
- [`raport`] zbadać wpływ standaryzacji zmiennych (kolumn w $\mathbf{X}$) na jakość analizy skupień;
- [`raport`] ... (autorskie pomysły mile widziane) ...

Uwaga 1: Brana będzie pod uwagę jakość kodu. Na przykład kod należy zamknąć w dobrze udokumentowane, wyspecjalizowane funkcje, tak by uniknąć powtórzeń itp.

Uwaga 2: Jeśli nie potrafisz czegoś zaimplementować samodzielnie, posłuż się gotowcem (w szczególności metoda spektralna oraz k-średnich jest już gdzieś zaimplementowana...) – uzyskasz przynajmniej choć kilka punktów (a i poćwiczysz pisanie raportu). Własne trzy zbiory benchmarkowe też możesz wygenerować bez implementacji poniższych.

[3] [Fowlkes E.B., Mallows C.L., A Method for Comparing Two Hierarchical Clusterings, *Journal of the American Statistical Association* **78**(383), 1983, 553–569]

[4] [por. Hubert L., Arabie P., Comparing Partitions, *Journal of the Classification* **2**, 1985, 193–218]

4 Algorytm spektralny i jego implementacja

Algorytm spektralny w wersji, którą tutaj zaimplementujesz, polega na zastosowaniu „zwykłej" procedury k średnich na odpowiednio zmodyfikowanej (poddanej różnym przekształceniom określonym przez widmo macierzy „bliskości" analizowanych punktów) macierzy $\mathbf{X}$.

Napisz funkcję `spectral_clustering(X, k, M)`, która dla $\mathbf{X} \in \mathbb{R}^{n\times d}$, $k \geq 2$ oraz $M \in \mathbb{N}$ zwraca k-podział zbioru danych $\mathbf{X}$ wyznaczony przy użyciu opisanych niżej podprocedur:

1. znajdowanie M najbliższych sąsiadów wszystkich punktów;
2. stworzenie grafu „sąsiedztwa" i uspójnienie go;
3. wyznaczenie odpowiednich k wektorów własnych jego laplasjanu;
4. zastosowanie algorytmu k średnich w nowej przestrzeni danych.

4.1 Macierz najbliższych sąsiadów

Funkcję tę implementujesz w Rcpp (ręcznie).

Napisz funkcję `Mnn(X, M)` (*M-nearest neighbors*), która dla $\mathbf{X} \in \mathbb{R}^{n\times d}$ oraz $M \in \mathbb{N}$ wyznacza macierz $\mathbf{S} \in \mathbb{N}^{n\times M}$, taką że $s_{i,j}$ jest indeksem j-tego najbliższego sąsiada x_i względem metryki euklidesowej.

W szczególności ma zachodzić $(\forall i)\ s_{i,1} = \arg\min_{j\neq i} \|x_i - x_j\|$ (przy założeniu, że odległości się nie powtarzają).

4.2 Macierz sąsiedztwa

Funkcji tej nie musisz implementować w Rcpp. Jednakże pewne realizowane tu podzadania możesz zlecić funkcjom pomocniczym, które już w Rcpp zaimplementować jest warto (dla chętnych).

Napisz funkcję `Mnn_graph(S)`, która jako argument przyjmuje macierz $\mathbf{S} \in \mathbb{N}^{n\times M}$ wygenerowaną przy użyciu powyższej funkcji.

Funkcja ta generuje symetryczną macierz $\mathbf{G} \in \{0,1\}^{n\times n}$, taką że $g_{i,j} = 1$, jeśli $(\exists u)\ s_{i,u} = j$ lub $s_{j,u} = i$.

$\mathbf{G}$ jest więc macierzą sąsiedztwa reprezentującą graf nieskierowany $\tilde{G}$ o n wierzchołkach, taki że i-ty wierzchołek jest połączony z j-tym, jeśli x_i jest wśród M najbliższych sąsiadów x_j lub x_j jest wśród M najbliższych sąsiadów x_i.

Z oczywistych względów n nie może być zbyt duże (powiedzmy większe niż 50,000). W praktyce funkcja `Mnn_graph(S)` powinna zwracać macierz rzadką, zob. `scipy.sparse` w Pythonie lub pakiet `Matrix` (klasa `dsRMatrix`) w R. W niniejszym projekcie nie jest to wymogiem, ale zachęcam do poszerzenia swojej wiedzy i rozwoju nowych umiejętności.

Należy wykryć wszystkie składowe spójne (na przykład przy użyciu algorytmu przeszukiwania wszerz (BFS) lub w głąb (DFS)). Jeśli graf $\tilde{G}$ jest spójny, zwracamy $\mathbf{G}$ bez dalszych modyfikacji.

W przeciwnym przypadku, zakładając, że w $\tilde{G}$ jest p składowych spójnych, należy dodać do $\tilde{G}$ dokładnie $p-1$ (nieskierowanych) krawędzi (w dowolny poprawny sposób), tak by $\tilde{G}$ uspójnić. Dopiero tak zmodyfikowaną macierz sąsiedztwa zwracamy w wyniku działania funkcji.

4.3 Laplasjan i jego wektory własne

Funkcja `Laplacian_eigen(G, k)` dla $k > 1$ i macierzy $\mathbf{G}$ jak wyżej:

1. wyznacza laplasjan grafu $\tilde{G}$, tj. $\mathbf{L} = \mathbf{D} - \mathbf{G}$, gdzie $\mathbf{D}$ jest macierzą diagonalną taką, że $d_{i,i}$ jest stopniem i-tego wierzchołka w $\tilde{G}$;

2. wyznacza macierz $\mathbf{E} \in \mathbb{R}^{n \times k}$, której kolumny składają się z wektorów własnych macierzy $\mathbf{L}$ odpowiadających 2., 3., ..., (k+1) co do wielkości wartości własnej;
3. zwraca $\mathbf{E}$ jako wynik.

   Do wyznaczania wektorów własnych używamy oczywiście funkcji „wbudowanej".

4.4 Algorytm k-średnich

Na tak wyznaczonej macierzy $\mathbf{E}$, należy uruchomić algorytm k-średnich. Jego gotową implementację znajdziesz w jednej z bibliotek.