ŁUKASZ STANISZEWSKI

METODY EKSPLORACJI DANYCH W ODKRYWANIU WIEDZY (MED)

PROJEKT – IMPLEMENTACJA ALGORYTMU ROCK

1. GRUPOWANIE DANYCH KATEGORYCZNYCH

Zadanie grupowania (ang. clustering) jest metodą uczenia nienadzorowanego, polegającą na grupowaniu próbek danych podobnych razem, jednocześnie rozdzielając od siebie te próbki, które znacząco różnią się względem siebie. Grupowanie stosuje się do np. strukturyzowania zbiorów pacjentów czy wyznaczania podobnych profili klientów w sklepie.

Mimo że, bardzo często, w przypadku grupowania, spotykamy się z zadaniami bogatymi w dane numeryczne lub przestrzenne, w przypadku tego projektu, interesujemy się grupowaniem bazującym na danych kategorycznych (lub logicznych, stanowiących specjalny przypadek danych kategorycznych), jak np. kolor czy płeć.

W zadaniu grupowania można spotkać się z różnymi rodzajami metod rozwiązujących te zadanie, natomiast w przypadku tego projektu, omawiana jest metoda hierarchicznego grupowania aglomeracyjnego (ang. agglomerative hierarchical clutering), która rozpoczyna się od przydzielenia każdemu punktowi własnej grupy a następnie, w kolejnych krokach, łączenia najlepszej pary grup w nową grupę, tworząc hierarchię.

Do poprawnego działania algorytmów bazujących na danych kategorycznych, konieczne jest odpowiednie przetworzenie danych. Stosowane jest tutaj **przerobienie reprezentacji** (zbioru wartości wszystkich atrybutów) **próbki w transakcję**. Takie podejście automatycznie rozwiązuje problem brakującej wartości dla atrybutu, poprzez zwyczajne nieuwzględnienie tego atrybutu w transakcji, przykładowo dla próbki o kolorze czarnym, krótkiej długości i nieznanej szerokości, transakcję można by było oznaczyć jako $\{kolor.czarny, dlugosc.krotka\}$.

Algorytmy grupowania hierarchicznego wymagają określenia konkretnej miary podobieństwa między dwoma punktami danych, pozwalającej stwierdzić jak bliskie sobie są dwie próbki. W przypadku, gdy posiadamy transakcje, wykorzystać możemy **indeks Jaccarda**, definiowany jako iloraz liczby wspólnych elementów dwóch transakcji do liczby cech występujących w obu transakcjach (bez powtórzeń), dla dwóch transakcji T_1 oraz T_2:

$$J(T_1, T_2) = \frac{|T_1 \cap T_2|}{|T_1 \cup T_2|}$$

2. ALGORYTM ROCK

Algorytm ROCK (Robust Clustering using links) został po raz pierwszy wprowadzony w publikacji [1]. ROCK jest algorytmem grupowania hierarchicznego. Aby móc lepiej badać podobieństwa między grupami, używa „globalnej" metody, bazującej na **połączeniach** (ang. links)**, których definiowanie zależy od liczby wspólnych sąsiadów** (ang. neighbours) między punktami, tzn. punkty A i B mogą być ze sobą połączone, jeśli punkt C jest sąsiadem zarówno punktu A jak i punktu B, nawet jeśli punkt A nie jest bezpośrednim sąsiadem B. Takie podejście pozwala osiągnąć grupy punktów posiadających wielu wspólnych sąsiadów. Aby móc stwierdzić, że dwa punkty (a dokładniej ich transakcje) „są podobne", konieczne jest ustalenie progu - **parametru $\boldsymbol{\theta}$** – mówiącego, ile musi wynosić, opisany w poprzednim rozdziale, indeks Jaccarda między nimi, żeby uznać je za wystarczająco bliskie sobie.

W przypadku tego algorytmu, połączenie dwóch grup w jedną ma największy sens, kiedy maksymalizowana jest liczba połączeń między punktami w każdej z grup przy jednoczesnym zapewnieniu, że punkty o małej liczbie połączeń będą znajdowały się w innych grupach. W związku z tym, **miarę jakości (ang. goodness measure) między dwiema grupami, mówiącą jak dobre jest połączenie tych grup** stanowi ułamek, którego licznik to suma połączeń między punktami w dwóch grupach, natomiast mianownik uwzględnia spodziewaną średnią liczbę połączeń w grupach (dzięki czemu nastąpi kara za punkty mające bardzo mało połączeń). Ostatecznie, algorytm ROCK, definiuje następującą **miarę jakości** (gdzie $link[A, B]$ – liczba połączeń między punktami z grup A i B; n_i – rozmiar grupy i; $f(\theta)$ – funkcja umożliwiająca aproksymację średniej liczby połączeń w grupie):

$$g(C_i, C_j) = \frac{link[C_i, C_j]}{(n_i + n_j)^{1+2f(\theta)} - n_i^{1+2f(\theta)} - n_j^{1+2f(\theta)}}$$

Ze względu na fakt, że możemy posiadać ogromną liczbę danych w ROCK najpierw następuje **próbkowanie ze zbioru**, w celu otrzymania losowej, mniejszej grupy punktów, na której to wykonywany jest algorytm. Następnie ma miejsce **przypisywanie pozostałych punktów do grup**, tzn. dla każdego punktu, sprawdzane jest, w której grupie posiada on najwięcej sąsiadów (uwzględniając normalizację przez spodziewaną liczbę sąsiadów) i tam jest on przypisywany.

Ostatecznie algorytm ROCK opiera się o **iteracyjne łączenie mniejszych grup w większe, jednocześnie tworząc ich hierarchię, poprzez wybieranie scalania dającego najlepszą miarę jakości, do momentu aż zabraknie połączeń między grupami lub narzucona parametrycznie liczba grup ($\boldsymbol{k}$) zostanie osiągnięta**.

3. IMPLEMENTACJA

3.1. DODATKOWE UWAGI

Dla większości algorytmów grupowania, duży problem stanowią próbki danych odstające (ang. outliers). W przypadku ROCK można zauważyć, że takie punkty będą miały najczęściej brak lub bardzo mało sąsiadów, co oznacza, że nie biorą praktycznie udziału w grupowaniu i mogłyby zostać odrzucone. Dodatkowo, małe grupy punktów odstających mogą pozostać w izolacji niemal do końca działania algorytmu, więc w sytuacji, gdy zbliżamy się do określonej liczby grup, możemy zatrzymać się i wyeliminować te wyizolowane grupy (zawierające pojedynczą próbkę). W przypadku tej implementacji algorytmu ROCK, **grupy z tylko jednym punktem są eliminowane, gdy liczba grup zmniejszy się do 1/3 względem oryginalnej liczby grup**.

W ROCK, wykorzystywana jest **struktura sterty** (ang. heap). Dla każdej grupy i, istnieje jego **lokalna sterta** $q[i]$ – wskazująca każdą grupę j mającą z grupą i niezerową liczbę połączeń, a grupy w tej stercie są uszeregowane malejąco pod względem miary jakości. Dodatkowo, w algorytmie istnieje **sterta globalna** Q, wskazująca wszystkie grupy i szeregująca je malejąco co do ich najlepszej, wewnętrznej, miary jakości. Tak więc, po wykonaniu połączenia dwóch grup (i oraz j) w k, w pojedynczej iteracji, konieczne jest:

a) zamienienie, w lokalnej stercie każdej grupy, posiadającej połączenie z i lub j, tego połączenia na połączenie z grupą k i aktualizacja lokalnej sterty;
b) stworzenie lokalnej sterty dla grupy k;
c) zaktualizowanie globalnej sterty Q.

Zaimplementowany algorytm realizuje następujące kroki:

1. Inicjalizacja algorytmu:
   1.1. Losowy podział zbioru danych na dane treningowe i inferencyjne.
   1.2. Obliczenie sąsiedztwa między punktami na zasadzie „każdy z każdym”.
   1.3. Utworzenie macierzy połączeń między jedno-punktowymi grupami.
   1.4. Utworzenie lokalnych stert q dla każdej grupy i globalnej sterty Q.
2. Utworzenie grup ze zbioru treningowego:
   2.1. Realizuje pętlę, w której to w kolejnych krokach dwie grupy stanowiące najlepszą parę zostają scalone w nową grupę łączącą te dwie, do momentu aż pozostaną same grupy bez sąsiedztw lub spełniona zostanie zadana liczba grup.
   2.2. Na mocy każdego scalenia, następuje ponowne utworzenie lokalnych stert dla każdej grupy, mającej jakiekolwiek połączenia z grupami scalanymi. Dodatkowo, inicjalizowana jest lokalna sterta dla nowej grupy oraz globalna sterta Q.
   2.3. Gdy liczba grup sprowadzi się do $1/3$ inicjalnej liczby grup, usuwane są wszystkie grupy posiadające wyłącznie jedną próbkę.
3. Predykcja dla zbioru inferencyjnego - każdej próbce nie uwzględnionej w procesie tworzenia grup nadaje się grupę, w której posiada ona najwięcej sąsiadów.
4. Utworzenie nowego pliku z dodatkową kolumną mówiącą o grupach próbek.

3.2. Wykorzystane technologie

Do implementacji algorytmu wykorzystany został język Python (w wersji 3.11), natomiast zbiory danych zostały pobrane z serwisu UC Irvine Machine Learning Repository. Do przetworzenia danych wykorzystana została biblioteka pandas, a do implementacji algorytmu biblioteka numpy (obliczenia numeryczne) oraz heapq (implementacja sterty). Do mierzenia czasu zastosowano bibliotekę timeit, a do testów jednostkowych – bibliotekę pytest.

3.3. Podział modularny

Implementację algorytmu można rozdzielić na następujące moduły:

a) *Dataprocessor* – moduł przetwarzający dane tabelaryczne w punkty obsługiwane przez algorytm, rozdzielający próbki ze zbioru danych na próbki treningowe oraz inferencyjne w sposób losowy, a także sprowadzający atrybuty i ich wartości do transakcji. Dodatkowo, odpowiada za ponowne przetworzenie wyjścia algorytmu grupującego tak, aby mógł zostać zapisany jako dane tabelaryczne.
b) *Cluster* – reprezentuje pojedynczą grupę jako zbiór punktów, implementuje lokalną stertę grupy i funkcję scalającą dwie grupy w jedną. Oblicza miarę jakości pomiędzy dwiema grupami, a także liczbę połączeń między nimi.
c) *Metrics* – implementuje zarówno funkcje mierzące jakość rezultatów algorytmu, jak i funkcje estymujące średnią wielkość grupy, a także indeks Jaccarda.
d) *RockAlgorithm* – zawiera pełną implementację algorytmu – proces inicjacji, tworzenia grup i inferencji punktów w zależności od przekazanych parametrów. Zarządza stertami grup, a także stertą globalną. Realizuje proces usuwania punktów odstających.
e) *Runner* – umożliwia uruchomienie algorytmu z konsoli poprzez przekazanie odpowiednich parametrów.

3.4. Uruchomienie algorytmu

Aby zainstalować projekt wystarczy, z głównego folderu, pobrać wszystkie niezbędne paczki Pythonowe, a następnie pomóc mu wskazać miejsce z wszystkimi modułami:

- `pip install -r requirements.txt`
- `export PYTHONPATH=$PYTHONPATH:$CWD`

Uruchomienie algorytmu następuje wraz z uruchomieniem modułu *runner* z odpowiednimi argumentami, przykładowo:

```
python3 rock/runner.py --dataset mushroom  --theta 0.8 --k 2 \
--approx_fn rational_add --split_train 0.25 --skip_outliers  \
--calculate_metrics
```

Po jego uruchomieniu, w folderze *results/* znajdować będzie się plik z próbkami algorytmu wraz z przypisanymi numerami grup oraz, ewentualnie w folderze *metrics/*, plik z parametrami i odpowiadającymi wartościami metryk.

Możliwe do przekazania argumenty:

a) **dataset** – nazwa zbioru danych, do wyboru są zbiory *mushroom*, *congressional* lub *csv* (dowolny plik .csv, składający się z samych danych kategorycznych; gdy wybrany, konieczne jest wyspecyfikowanie ścieżki do niego przy użyciu argumentu *dataset_csv_path*);
b) **theta** – wartość parametru θ wskazującego próg dla indeksu Jaccarda uznającego sąsiedztwo dwóch punktów, stanowi liczbę z zakresu $[0,1]$;
c) **k** – spodziewana ostateczna liczba grup – stanowi liczbę całkowitą;
d) **approx_fn** – nazwa funkcji aproksymującej średnią wielkość grupy, do wyboru: *rational_add (*$1+x/1-x$*), rational_sub (*$1-x/1+x$*), rational_exp (*$\exp x/1-x$*), rational_sin (*$\sin x/1-x$*);*
e) **calculate_metrics** – jeśli wybrane, na koniec algorytmu wyliczone zostaną wartości metryk i zapisane do pliku w folderze *metrics/*.
f) **skip_outliers** – jeśli wybrane, próbki odstające nie zostaną uwzględnione w wyniku działania algorytmu.

3.5. Testy jednostkowe

Dodatkowo, utworzone zostały testy jednostkowe sprawdzające działanie pojedynczych funkcjonalności. Znajdują się w folderze *test/*, a uruchomić je można przy użyciu komendy:

- `pytest test/`

Testy te sprawdzają niezawodność najważniejszych funkcjonalności rozwiązania takich jak: metryki, funkcje matematyczne, tworzenie grup wraz ze stertami, obliczanie sąsiedztw i macierzy połączeń czy miar jakości.

4. Eksperymenty

4.1. Zbiory danych

Jako zbiory danych do eksperymentów wybrane zostały zbiory te, które wykorzystane zostały w oryginalnym artykule naukowym, ponieważ przeprowadzenie eksperymentów na tych dwóch zbiorach pozwoli sprawdzić, czy implementacja jest poprawna:

a) Zbiór danych Mushroom [2] - zawiera opisy próbek odpowiadających 23 gatunkom grzybów skrzelowych z rodziny Agaricus i Lepiota, posiadając zmienną wyjaśnianą stwierdzającą czy grzyb jest trujący. Posiada on 8124 próbek danych opisywanych za pomocą 22 kolumn kategorycznych.
b) Zbiór danych Congressional Voting Records [3] - zestawiający głosy dla każdego z kongresmenów Izby Reprezentantów Stanów Zjednoczonych w 16 kluczowych głosowaniach określonych przez CQA, posiadając zmienną wyjaśnianą jako partię, na którą został oddany głos. Posiada on 435 próbek opisywanych za pomocą 16 kategorycznych kolumn.

Sama implementacja wspiera wszelkie zbiory składające się z samych danych kategorycznych.

4.2. Porównanie do oryginalnej implementacji

4.2.1. Zbiór Mushroom

Pierwszym krokiem eksperymentów było porównanie wyników uzyskanych przez oryginalną implementację na tle wyników z oryginalnego artykułu naukowego. Dla zbioru *Mushroom*, autorzy artykułu naukowego otrzymali 21 grup, w tym 20 grup czystych (zawierających próbki o tylko jednej klasie) oraz jedną grupę zawierającą 72 grzyby trujące i 32 grzyby jadalne.

Symulacja algorytmu zaimplementowanego w ramach tego projektu, dla parametrów: $\theta = 0.8$, $k = 2$, $f(\theta) = \frac{1+x}{1-x}$ wraz z wydzieleniem zbioru do tworzenia grup jako $\frac{1}{4}$ pełnego zbioru została przedwcześnie zatrzymana na 22 grupach, w tym 21 grupach czystych oraz jednej, małej grupie zawierającej 16 grzybów jadalnych i 7 trujących.

Aż 18 z 22 grup pokrywa się pomiędzy tymi dwoma rozwiązaniami. Poza różnicami w grupach nieczystych, tutejszej implementacji brakuje jednej grupy o 8 grzybach trujących i jednej o 16 grzybach jadalnych, natomiast wprowadza ona jedną grupę o 192 grzybach jadalnych, jedną o 32 grzybach jadalnych oraz jedną o 32 grzybach trujących. Pomimo utworzenia o jedną więcej grup przez zaimplementowany algorytm, za istotne należy uznać to, że jedyna grupa nieczysta jest tutaj znacznie mniejsza niż przy oryginalnych wynikach, a także posiada minimalnie lepsze rozdzielenie klas. Dodatkowo, algorytm zdolny był przypisać grupy 151 więcej próbkom, co czyni go bardziej odpornym na próbki odstające.

Nr grupy	(# jadalnych, # trujących)	Nr grupy	(# jadalnych, # trujących)	Nr grupy	(# jadalnych, # trujących)
1	(96,0)	8	(0,32)	**15**	**(32, 72)**
2	(0, 256)	9	(0,1296)	16	(0,1728)
3	(704,0)	**10**	**(0, 8)**	17	(288,0)
4	(96,0)	11	(48,0)	18	(0,8)
5	(768,0)	12	(48,0)	19	(192,0)
6	(0,192)	13	(0,288)	**20**	**(16, 0)**
7	(1728,0)	14	(192,0)	21	(0,36)

Tabela 1 – wyniki algorytmu ROCK dla zbioru Mushroom z oryginalnej implementacji.
Pogrubionym kolorem zaznaczono grupy nieosiągane przez implementację wykonaną tutaj.

Nr grupy	(# jadalnych, # trujących)	Nr grupy	(# jadalnych, # trujących)	Nr grupy	(# jadalnych, # trujących)
1	(96,0)	9	(0,1296)	17	(288,0)
2	(0, 256)	**10**	**(192, 0)**	18	(0,8)
3	(704,0)	11	(48,0)	19	(192,0)
4	(96,0)	12	(48,0)	**20**	**(16, 7)**
5	(768,0)	13	(0,288)	21	(0,36)
6	(0,192)	14	(192,0)	**22**	**(32, 0)**
7	(1728,0)	**15**	**(0, 32)**	--	--
8	(0,32)	16	(0,1728)	--	--

Tabela 2 - wyniki algorytmu ROCK dla zbioru Mushroom dla implementacji wykonanej w ramach tego projektu.
Pogrubionym kolorem zaznaczono grupy, którą dodała implementacja wykonana tutaj.

Gdy natomiast zastosowane zostało wydzielenie zbioru do tworzenia grup jako $\frac{1}{2}$ pełnego zbioru, wtedy udało się zreprodukować wynik z oryginalnego artykułu (podział jak w tabeli 1.).

4.2.2. ZBIÓR CONGRESSIONAL VOTING RECORDS

Dla zbioru *Congressional Voting Records*, autorzy artykułu naukowego otrzymali 2 grupy, kolejno grupę ze 144 republikanami i 22 demokratami oraz grupę z 5 republikanami oraz 201 demokratami.

Również dla algorytmu zaimplementowanego w ramach tego projektu, dla parametrów: $\theta = 0.73$, $k = 2$, $f(\theta) = \frac{1-x}{1+x}$ wraz z wydzieleniem zbioru do tworzenia grup jako $\frac{1}{2}$ pełnego zbioru otrzymano 2 grupy, pierwsza stanowiąca 133 republikanów i 10 demokratów oraz druga stanowiąca 16 republikanów i 207 demokratów

W przypadku pierwszych grup, implementacja wykonana w ramach tego projektu osiągnęła lepiej rozdzieloną grupę, natomiast, w przypadku drugich grup, oryginalna implementacja osiągnęła lepszy podział.

Nr grupy	(# republikanów, # demokratów)	Nr grupy	(# republikanów, # demokratów)
1	(144,22)	2	(5,201)

Tabela 3 – wyniki algorytmu ROCK dla zbioru Congressional Voting Records z oryginalnej implementacji.

Nr grupy	(# republikanów, # demokratów)	Nr grupy	(# republikanów, # demokratów)
1	(133,10)	2	(16,207)

Tabela 4 - wyniki algorytmu ROCK dla zbioru Congressional Voting Records dla implementacji wykonanej w ramach tego projektu.

Biorąc pod uwagę fakt, że algorytm posiada w sobie czynnik losowy (rozdzielenie danych na treningowe i inferencyjne) bliskie wyniki, jakie osiągnęła implementacja wykonana tutaj, wskazują, że jest ona poprawna i zgodna z tą zaproponowaną przez oryginalnych autorów.

4.3. WPROWADZENIE DO EKSPERYMENTÓW

Działanie algorytmu zbadane zostało pod względem szybkości działania oraz skuteczności. Dzięki wybraniu zbiorów ze zmienną wyjaśnianą, jesteśmy w stanie policzyć nie tylko wewnętrzne miary ewaluacji grupowania, ale także zewnętrzną. Do sprawdzenia skuteczności algorytmu, obliczona została:

a) Wewnętrzna miara ewaluacji Silhouette.
b) Zewnętrzna miara ewaluacji Purity.

Sprawdzone zostało, jaki wpływ na skuteczność i szybkość działania algorytmu ma dobór parametrów algorytmu, tzn.:

a) Liczby θ stanowiącej próg sąsiedztwa przy indeksie Jaccarda.
b) Doboru funkcji $f(\theta)$ umożliwiającej aproksymowanie średniej liczby połączeń w grupie.
c) Doboru parametru k mówiącego o spodziewanej liczbie grup.

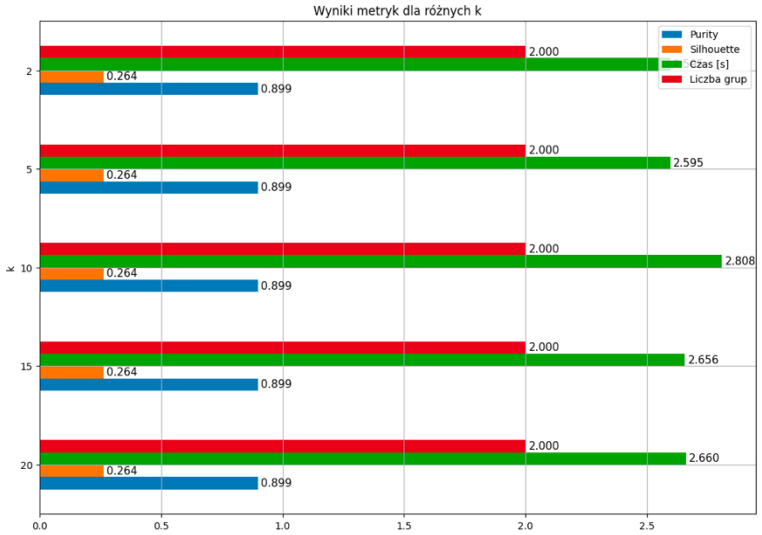
4.4. WYNIKI EKSPERYMENTÓW

Badanie zmiany wpływu parametru wykonane zostało względem parametrów wykorzystanych w punkcie 4.2. Badane wartości parametrów zestawiono w tabeli:

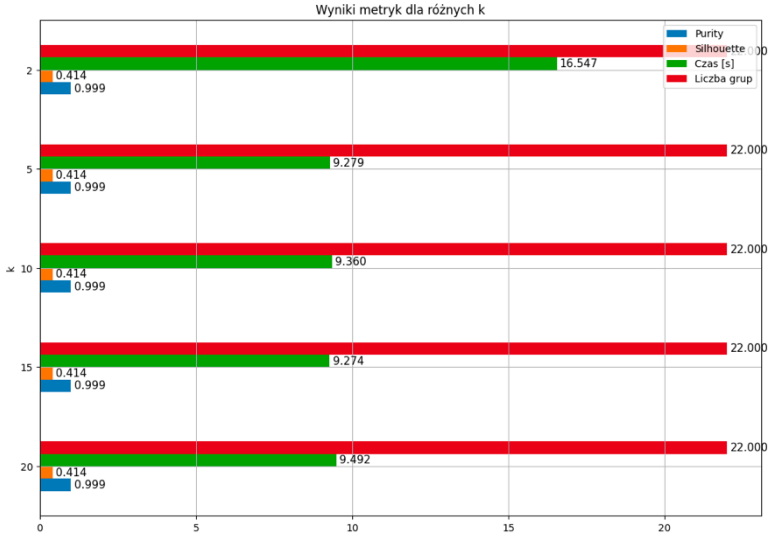
a) Parametr θ: $0.5, 0.6, 0.65, 0.7, 0.8, 0.9$
b) Parametr $f(\theta)$: $1-x/1+x$, $1+x/1-x$, $\exp x/1-x$, $\sin x/1-x$
c) Parametr k: $2, 5, 10, 15, 20$

4.4.1. PARAMETR K

Parametr k wskazujący satysfakcjonującą liczbę grup nie wpływa na jakość działania algorytmu. Dla każdego eksperymentu otrzymane zostały te same wartości metryk. Wyniki różnią się jedynie w czasie, natomiast różnica wynika prawdopodobnie z nie-determinizmu mierzenia czasu. W ostateczności za każdym razem otrzymywane są odpowiednio dla zbiorów: 2 lub 22 grupy. W przypadku pierwszym wynika to z procesu usuwania próbek odstających, a w przypadku drugiego – zatrzymania po wyczerpaniu się sąsiedztw między punktami.



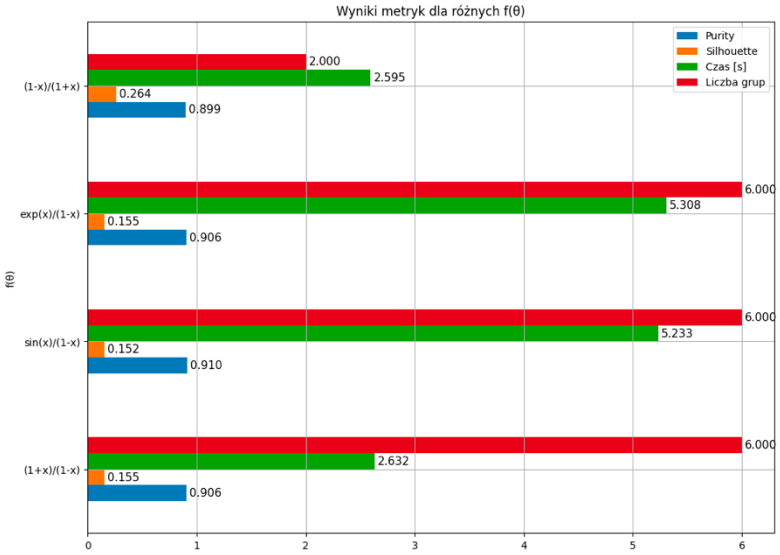
Wykres 1 – wyniki metryk Purity i Silhouette wraz z czasem wykonania algorytmu oraz liczbą ostatecznie osiągniętych grup dla różnych wartości parametru k dla zbioru Congressional Voting Records



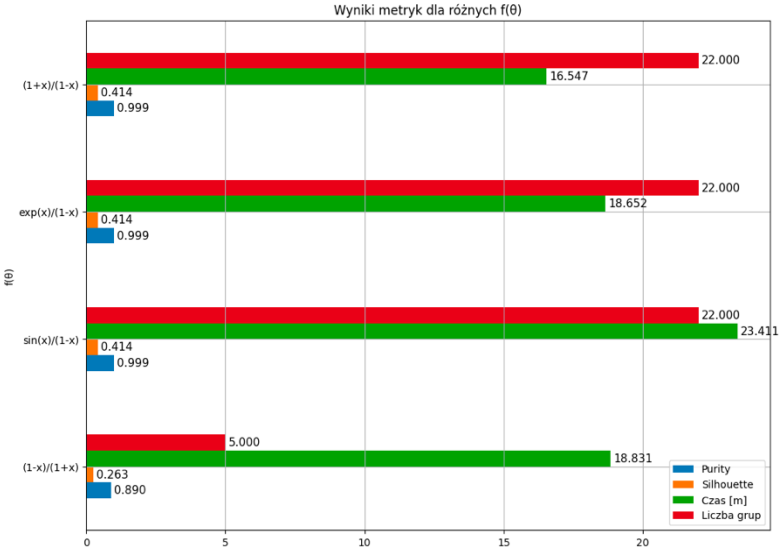
Wykres 2 – wyniki metryk Purity i Silhouette wraz z czasem wykonania algorytmu oraz liczbą ostatecznie osiągniętych grup dla różnych wartości parametru k dla zbioru Mushroom

4.4.2. Parametr $f(\theta)$

Parametr $f(\theta)$ wskazuje funkcję aproksymującą średnią wielkość grupy i służy do zniechęcania do łączenia się w zbyt duże grupy przez podgrupy scalane. W przypadku obu zbiorów widać, że najwięcej czasu zajmuje funkcja $\frac{\exp x}{1-x}$, która jest nieco bardziej skomplikowaną funkcją niż pozostałe. Co ciekawe, najlepsza dla zbioru Congressional Voting Records – bo dająca najbardziej odseparowane grupy, zwracająca oczekiwaną liczbę grup i wykonująca się najszybciej, jest funkcja $1 - x/1 + x$, natomiast dla zbioru Mushroom – funkcja $\frac{1+x}{1-x}$ (wykonując się najkrócej przy jednoczesnym zachowaniu najlepszych wartości obu metryk).



Wykres 3 – wyniki metryk Purity i Silhouette wraz z czasem wykonania algorytmu oraz liczbą ostatecznie osiągniętych grup dla różnych wartości parametru $f(\theta)$ dla zbioru Congressional Voting Records

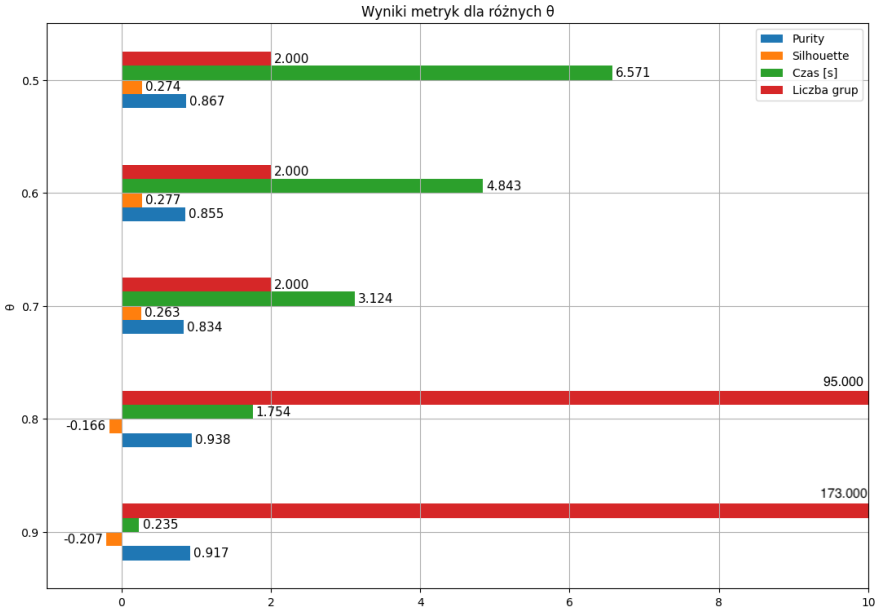


Wykres 4 – wyniki metryk Purity i Silhouette wraz z czasem wykonania algorytmu oraz liczbą ostatecznie osiągniętych grup dla różnych wartości parametru $f(\theta)$ dla zbioru Mushroom

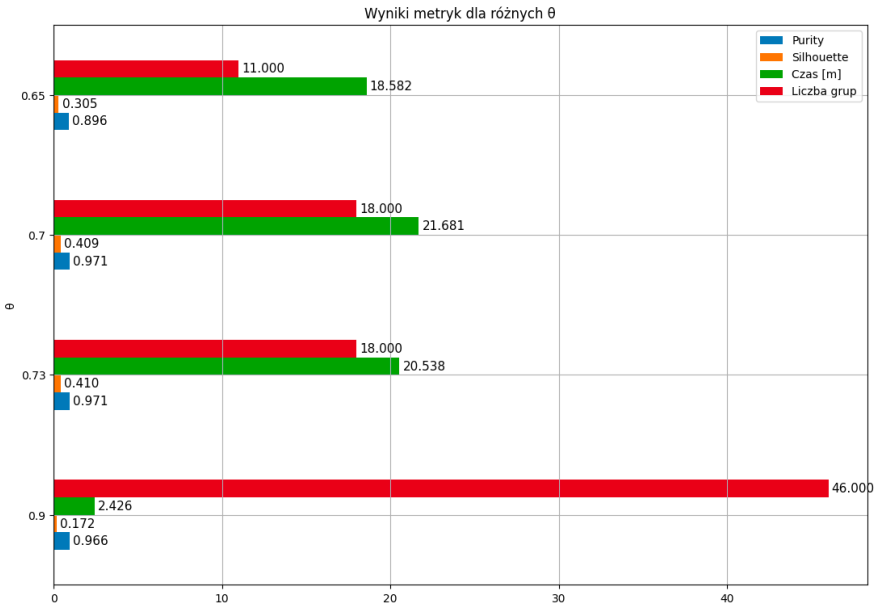
Przy okazji tych przykładów, zauważyć można ciekawą rozbieżność pomiędzy metrykami Purity a Silhouette – często jest tak, że lepszy podział pod względem metryki Purity oznacza gorszy podział pod względem metryki Silhouette.

4.4.3. PARAMETR θ

Parametr θ wskazuje próg, jaki musi przekroczyć indeks Jaccarda między dwiema próbkami, aby uznać ich za sąsiednie. Na podstawie wyników widać zależność, gdzie większa wartość tego parametru oznacza mniejszy czas wykonania, natomiast większą liczbę grup końcowych. Wynika to z faktu, że grupy, w trakcie działania algorytmu, wraz ze zwiększeniem θ, posiadają między sobą coraz to mniejszą liczbę sąsiadów, przez co algorytm kończy się szybciej (brak rozdzielnych grup mających jakiekolwiek połączenia między sobą). Dodatkowo, sam proces reinicjalizacji sterty lokalnej każdego sąsiada dla obu grup, po ich scaleniu, trwa coraz krócej, bo dla każdej grupy istnieje coraz mniej innych grup mających jakiekolwiek połączenia z nimi. W przypadku symulacji dla obu zbiorów, również zauważyć można rozbieżność metryk Purity oraz Silhouette. Widać też, że miara Silhouette cierpi najbardziej tam, gdzie jest najwięcej grup, podczas gdy Purity nie ma z tym problemu.



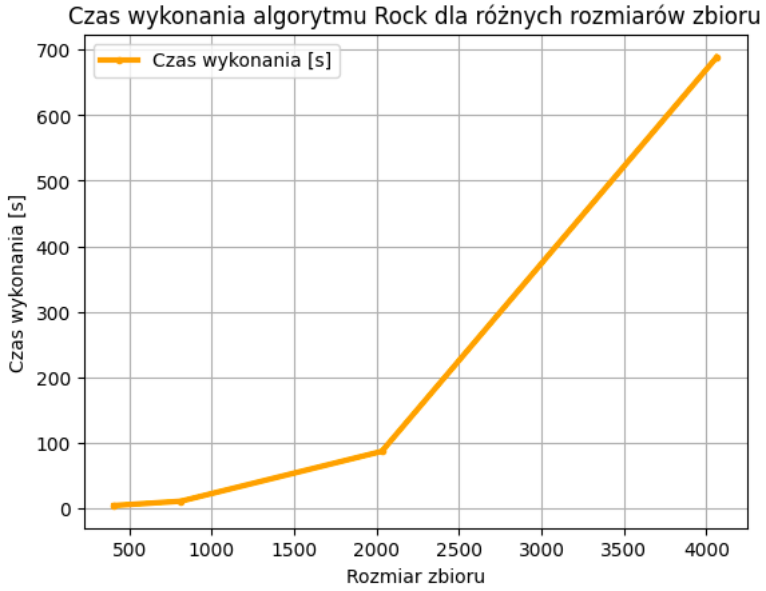
Wykres 5 – wyniki metryk Purity i Silhouette wraz z czasem wykonania algorytmu oraz liczbą ostatecznie osiągniętych grup dla różnych wartości parametru θ dla zbioru Congressional Voting Records



Wykres 6 – wyniki metryk Purity i Silhouette wraz z czasem wykonania algorytmu oraz liczbą ostatecznie osiągniętych grup dla różnych wartości parametru θ dla zbioru Congressional Voting Records

4.4.4. CZAS WYKONANIA A WIELKOŚĆ ZBIORU

Ostatnim eksperymentem do wykonania było sprawdzenie szybkości wykonania algorytmu w zależności od wielkości zbioru do tworzenia grup. Ze względu na fakt, że sam algorytm posiada dużą złożoność, ograniczono się do zbioru o wielkościach od 130 przykładów do 4000. Jak widać, szybkość działania algorytmu znacząco zależy od liczby punktów wykorzystywanych przy tworzeniu grup. Po przekroczeniu 2000 liczby próbek, czas jego wykonania drastycznie podskoczył do góry, natomiast różnica ta jest mało widoczna dla mniejszych zbiorów.



Wykres 7 – czas wykonania (w sekundach) zaimplementowanego algorytmu ROCK dla różnych rozmiarów zbioru danych.

BIBLIOGRAFIA

[1] Guha S., Rastogi R., Shim K, "ROCK: A robust clustering algorithm for categorical attributes," in *Proceedings of the International Conference on Data Engineering*, Sydney, 1999.

[2] U. M. L. Repository, „Mushroom," 1987. [Online]. Available: https://doi.org/10.24432/C5959T. [Data uzyskania dostępu: 14 01 2024].

[3] U. M. L. Repository, „Congressional Voting Records," 1987. [Online]. Available: https://doi.org/10.24432/C5C01P. [Data uzyskania dostępu: 14 01 2024].