Systemy Obliczeniowe

Laboratorium 4 – Lista A (1 października 2024)

dr inż. Paweł Trajdos

1 Uwagi

1. Po zakończeniu realizacji zadań należy załadować na Eportal skrypy pythona. Format nazwy pliku `ZA.py`; `ZA.pyx` (`A` to numer zadania). Źle nazwane pliki nie są oceniane.
2. W przypadku niepewności lub zauważenia jakichkolwiek błędów w instrukcji należy niezwłocznie powiadomić prowadzącego laboratorium w celu wyjaśnienia sprawy. Reklamacje po zakończeniu zajęć nie będą uwzględniane.

2 Zadania

Zadanie 4.0(Pkt. 6.0):
Jako zadanie rozgrzewkowe należy zaimplementować sekwencyjną wersję algorytmu klasteryzacji **k**-średnich. Jako zbiór danych na potrzeby testowania implementacji, należy wykorzystać dane wygenerowane za pomocą funkcji **make classification**. Wygenerowane dane powinny zawierać 2 cechy informatywne.

Działanie sekwencyjnego algorytmu **k**-średnich przedstawia się następująco:

1. Jako centroidy początkowe wybieramy losowe instancje problemu.
2. Liczymy odległość wszystkich instancji problemu od wybranych centroidów (klasa **DistanceMetric** z biblioteki **sklearn**).
3. Przyporządkowujemy każdą instancję do najbliżej znajdującego się centroidu.
4. Wyliczamy nowy centroid jako średnią instancji należących do każdego z klastrów.
5. Wracamy do Punktu 2, aż pozycja centroidów przestanie ulegać zmianie.

Pomocna może okazać się wizualizacja działania implementowanego algorytmu. Może ona zostać wykonana za pomocą biblioteki **matplotlib** oraz funkcji **scatter**:

```
plt.scatter(centroids[:,0], centroids[:,1], c='r', s=80)

for c in range(n_clusters):
        plt.scatter(X[closest_centroid==c, 0],X[closest_centroid==c, 1], c=colors[c])
    plt.scatter(centroids[:,0], centroids[:,1], c='r', s=80)
```

Zadanie 4.1(Pkt. 6.0):
W drugim zadaniu należy zaimplementować zrównoleglоną wersję algorytmu **k**-średnich z wykorzystaniem MPI. Poniże opis przykładowego rozwiązania wraz z elementami kodu:

```
if rank == 0:
    # Data
    X, y = make_classification(n_samples=10000, n_features=2, n_informative=2, n_redundant=0,
    random_state=1410)
    plt.scatter(X[:, 0],X[:, 1], c="blue")

```

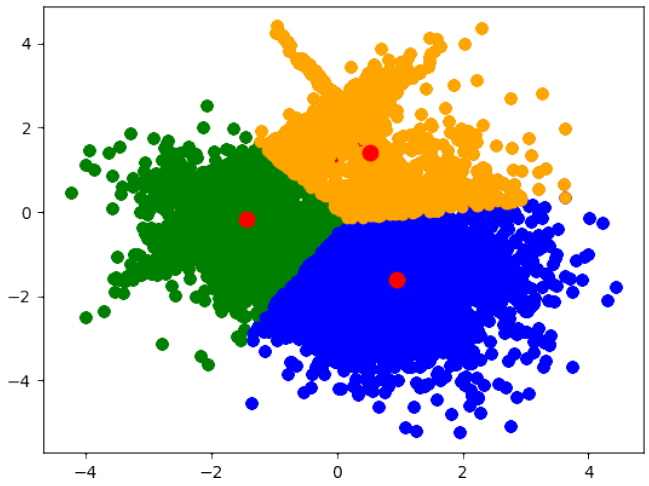
```
    # subsets
    ave, res = divmod(X.shape[0], size)
    counts = [ave + 1 if p < res else ave for p in range(size)]
    # print(counts)
    starts = [sum(counts[:p]) for p in range(size)]
    ends = [sum(counts[:p+1]) for p in range(size)]
    X = [X[starts[p]:ends[p]] for p in range(size)]

    # Membership
    ?

    # Centroids
    ?

else:
    X = None
    centroids = None
    membership = None
```

1. Na początku dostęp o danych ma wyłącznie Proces 0. Powinien on dokonać podziału danych z wykorzystaniem funkcji **scatter**, a następnie rozesłać je do pozostałych procesów. Należy także przygotować do rozesłania macierze przechowujące informacje o pozycji centroidów oraz przynależności instancji do klastrów.
2. Następnie należy rozesłać do wszystkich procesów przynależące im częsci zbioru danych, informacje o pozycji centroidów i o przynależności do klastrów.
3. Każdy z procesów liczy odległości od instancji w swoich podzbiorach danych do każdego z centroidów oraz przyporządkowuje instancje do klastrów.
4. Proces 0 zbiera z wykorzystaniem funkcji **gather** cały zbiór danych oraz informacje o przynależności instancji do klastrów i pozycji centroidów.
5. Proces 0 wylicza nowe centroidy jako średnią instancji należących do danego klastra.
6. Powrót do punktu 2.



Rysunek 1: Wizualizacja wyników działania algorytmu K-Means.

3 Dokumentacja

Dokumentacja:

- `MPI4Py`[1],
- `OpenMPI`[2],
- `MPIch`[3],
- `K-Means`[4],

[1] `https://mpi4py.readthedocs.io/en/stable/`
[2] `https://www.open-mpi.org/doc/`
[3] `https://www.mpich.org/documentation/guides/`
[4] `https://en.wikipedia.org/wiki/K-means_clustering`