Wizualizacja Dużych Zbiorów Danych

ShapeVis Raport 1

autorzy: Ilona Tomkowicz, Maciej Tobiasz

Akademia Górniczo-Hutnicza
Wydział Informatyki, Elektroniki i Telekomunikacji,
Informatyka, II stopień, I semestr

4 maja 2020

Contents

1. Przegląd literatury

W pierwszym etapie realizacji projektu przystąpiono do analizy materiałów źródłowych dotyczących algorytmu ShapeVis. Głównym źródłem wiedzy na ten temat okazał się artykuł ShapeVis: High-dimensional Data Visualization at Scale [1], gdzie algorytm został szczegółowo opisany i dzięki temu można go odtworzyć. Artykuł jest opublikowany stosunkowo niedawno, jednak jak się okazuje metoda wizualizacji danych nazywana ShapeVis była znana już w 1999 roku. Publikacja, która przestawia jej porównanie z metodą Magic Eye View nie przytacza jednak dokładnych szczegółów tego algorytmu [2]. Może okazać się przydatna w testowaniu jakości implementacji, jednak na etapie pisania kodu nie przewiduje się korzystania z niej.

Pomimo poszukiwań nie znaleziono w dostępnych źródłach istniejącej implementacji tego algorytmu. Z tego powodu konieczne było rozpoczęcie własnej implementacji programu w języku Python.

2. Struktura programu

Implementację podzielono na kilka etapów, które przedstawiono symbolicznie na diagramie przepływu danych 2.1. Najpierw moduł zajmujący się procesowaniem danych wejściowych załaduje je do programu. W zależności od użytego zbioru konieczna będzie modyfikacja oryginalnego wejścia tak, aby wydobyć trzy informacje: wielowymiarowe wartości danych, odpowiadające danym wartościom klasy, nazwy klas. Następnie dane są przekierowane do modułów odpowiedzialnych za kolejne kroki algorytmu. Dokładniejszy opis w postaci pseudokodu poszczególnych kroków znajduje się w rozdziale 3. Na końcu niskowymiarowa reprezentacja danych trafia do estymatora jakości, który na podstawie metryki odległości oblicza jakość reprezentacji. W celu porównania wyników z innymi transormajami w programie na etapie testowania został przewidziany moduł AlgoComparer, którym można sprawdzić działanie innych metod takich jak t-SNE czy UMAP dla tego samego zbioru danych.

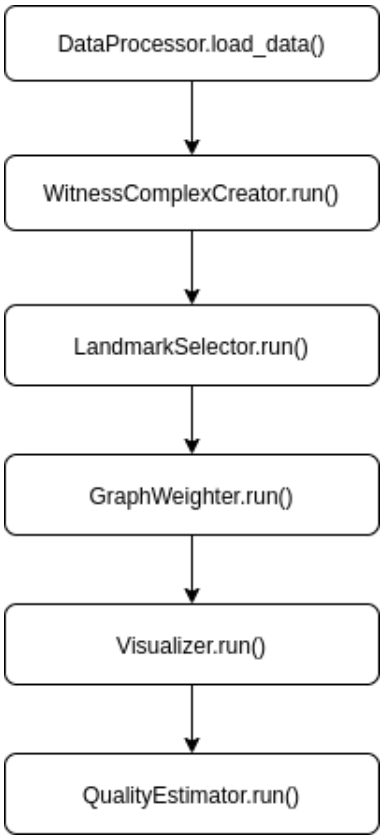

Figure 2.1: Wysokopoziomowy schemat przepływu danych w programie

3. Pseudokod algorytmu

3.1 Stworzenie grafu k-NN powiekszonego o dodatkowe połączenia

```
Data: Dane wysokowymiarowe: data
Result: Spróbkowane dane wejściowe: samples
r = random number from range 0:data.length;
while r > 0 do
    randomIndex = random number from range 0:data.length;
    sampledProbe = data[randomIndex];
    add sampledProbe to samples;
    decrement r;
end
return samples
```

Algorithm 1: Próbkowanie

```
Data: Spróbkowane dane w postaci listy: samples, ilośc sąsiadów: k
Result: Graf w postaci słownika sąsiedztwa: adjacencyDict, i słownika odległości: adjacentDistancesDict
adjacencyDict = is empty;
adjacentDistancesDict = is empty;
foreach node1 in samples do
    foreach node2 in samples do
        if node1 == node2 then
            continue;
        end
        if adjacencyDict[node1].length < k then
            add node2 to adjacencyDict[node1];
            distanceBetweenTheseNodes = calculateDistance(node1, node2);
            add distanceBetweenTheseNodes to adjacentDistancesDict[node1];
        end
        else
            maxDistanceBetweenCurrentNeighboursAndNode1 = max(adjacentDistancesDict[node1]);
            newCandidateNodeDistance = calculateDistance(node1, node2);
            if newCandidateNodeDistance < maxDistanceBetweenCurrentNeighboursAndNode1 then
                exchange the distanced node for node2 in adjacencyDict[node1];
                update the distance for this node;
            end
        end
    end
end
return adjacencyDict, adjacentDistancesDict
```

Algorithm 2: Tworzenie grafu k-NN

Data: Graf w postaci słownika sąsiedztwa: adjacencyDict, i słownika odległości: adjacentDistancesDict, lista wierzchołków nie spróbkowanych: unsampledNodes, lista wierzchołków spróbkowanych: sampledNodes
Result: Graf w postaci słownika sąsiedztwa powięszony o dodatkowe połaczenia: adjacencyDict
foreach *unsampledNode in unsampledNodes* **do**
    distancesToSampledNodes is empty;
    foreach *sampledNode in sampledNodes* **do**
        distance = calculateDistance(unsampledNode, sampledNode);
        add distance to distancesToSampledNodes;
    end
    find two min distances in distancesToSampledNodes: minDist1, minDist2;
    and corresponding nodes: nearestNode1, nearestNode2;
    if *not checkIfNodeIsAdjacentTo(adjacencyDict[nearestNode1], nearestNode2)* **then**
        connect them by adding nearestNode2 to adjacencyDict[nearestNode1];
    end
end
return adjacencyDict

Algorithm 3: Powiekszenie grafu o połączenia najbliższych sąsiadów danych niespróbkowanych.

3.2 Tworzenie podzbiorów charakterystycznych

Data: Graf w postaci listy wierzchołków grafu: nodes oraz słownika sąsiedztwa: adjacencyDict
Result: Lista landmarków: landmarks, lista sąsiadów odpowiadających landmarkom: revNeigh
landmarks, revNeigh is empty;
while *nodes has contents* **do**
    randomNodeIdx = random number from range 0:nodes.length;
    randomNode = nodes[randomNodeIdx]; add randomNode to landmarks; add adjacencyDict[randomNode] to revNeigh; remove randomNode from nodes; remove all occurrences of randomNode from adjacencyDict;
end
return landmarks, revNeigh

Algorithm 4: Landmarking - wybór punktów charakterystycznych

3.3 Stworzenie grafu ważonego używając łańcuchów Markova

Data: Lista landmarków: landmarks, lista sąsiadów odpowiadających landmarkom: revNeigh
Result: Macierz wag połączeń między węzłami: weights
visitCounter is a dictionary with nodes as keys, each with list of revNeigh.lengths filled with 0s;
minLen, maxLen are min and max values for random walk length;
for *b times* **do**
    walkLength = random number from range minLen:maxLen;
    currentNode = random node from landmarks;
    currentNodeIdx = index of currenNode in landmarks;
    walkLength > 0 update prevNode and prevNodeIdx as currentNode and currentNodeIdx;
    currentNode = rand node from revNeigh[prevNodeIdx];
    currentNodeIdx = check in landmarks;
    increment visitCounter[prevNode] on index of currentNode;
    decrement walkLength;
    increment visitCounter[prevNode] on index of currentNode;
end
calculate matrix a from formula a = visitCounter[i][j] / sum(0:k) visitCounter[i][k]; calculate weights from formula: a[i][j] + a[j][i] - a[i][j]*a[j][i]; **return** weights

Algorithm 5: Stworzenie grafu ważonego

4. Implementacja

Do tej pory zaimplementowana została pierwsza część alorytmu, opisana w rodziale 3.1. Poza tym powstała klasa procesująca dane, pozwalająca na wybór zbioru. Na chwilę obecną używany jest zbiór MNIST o ograniczonej liczebności z biblioteki sklearn. Zbudowany został też moduł oceny jakości algorytmu na podstawie metryki odlgłości oraz moduł pozwalający na porównanie kocowego efektu działania ShapeVis z inną transformacją. Poza tym powstają testy sprawdzające poprawność zaimplementowanego algorytmu.

Link do repozytorium na GitHubie: `https://github.com/ilonatommy/ShapeVis`

Bibliography

[1] N. Kumari, R. Siddarth, A. Rupela, P. Gupta, and B. Krishnamurthy, "Shapevis: High-dimensional data visualization at scale," *ArXiv*, vol. abs/2001.05166, 2020.

[2] M. Kreuseler and H. Schumann, "Information visualization using a new focus+context technique in combination with dynamic clustering of information space," in *NPIVM '99*, 1999.