**Analisi SilhouetteCalculator**

**Problema**

Considerato un data set ∈ {BrizziB, Jgalilee, Yhfyhf} si vuole calcolare l’indice di Silhouette.

Nota: per una descrizione dettagliata di tali data set, consultare il documento di analisi e progettazione di RandIndexCalculator.

**Soluzione**

Per risolvere il problema è necessario ottenere una rappresentazione, per ogni punto del data set in analisi, nella forma:

-80.32974400996152,63.67437888508667,1

Dove i primi due numeri reali, sono il punto bidimensionale considerato o, nel caso del data set Indian Pines, tutti gli attributi spettrali e non del punto, seguita da un numero intero che corrisponde al codice identificativo del cluster di appartenenza del punto nell’output dell’implementazione scelta.

In definitiva, la soluzione, per una istanza del problema, consiste dei seguenti passi:

1. Il caricamento in memoria del data set considerato, memorizzando gli attributi di tutti gli esempi in una matrice di numeri reali e il codice identificativo di tutti gli esempi in un vettore di interi; in modo tale che nella posizione i della matrice troviamo gli attributi dell’i-esima osservazione e nella posizione i del vettore troviamo il codice identificativo del cluster di appartenenza della stessa osservazione.
2. Il calcolo dell’indice di Silhouette partendo dalla matrice e dal vettore ottenuti al passo 1.

L’algoritmo SILHOUETTE utilizzato per il calcolo dell’indice di Silhouette prende in input la matrice A degli attributi e la il vettore C dei codici identificativi dei cluster di appartenenza, ottenuti al punto 1:

SILHOUETTE(A,C)

silhouette = 0

**for** i = 1 **to** rowlength(A)

e = A[i]

eWithin = 0

eWithinNum = -1

Sia eBetween un array di dimensione |distinct(C)|

Sia eBetweenNum un array di dimensione |distinct(C)|

**for** j = 1 to rowlength(A)

e2 = A[j]

**if** C[i] == C[j]

eWithin = eWithin + distance(e, e2)

eWithinNum = eWithinNum + 1

**else**

eBetween[C[j]] = eBetween + distance(e, e2)

eBetweenNum[C[j]] = eBetweenNum[C[j]] + 1

eWithin = eWithin / eWithinNum

**for** k = 1 **to** length(eBetween)

eBetween[k] = eBetween[k] / eBetweenNum[k]

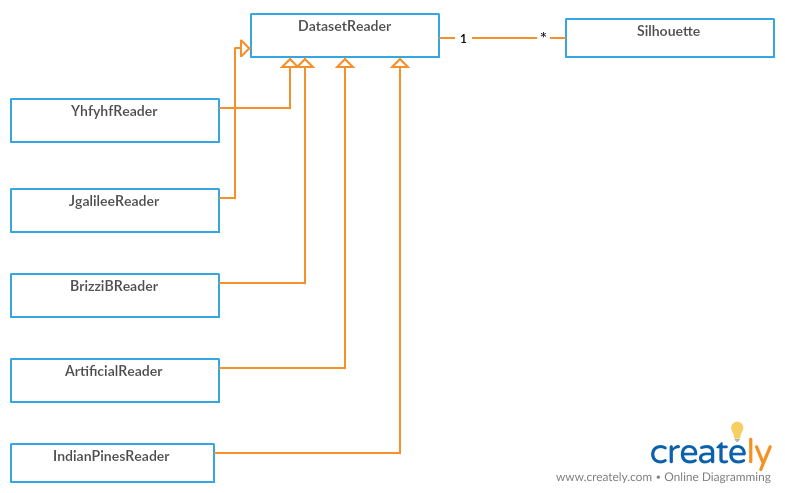
minBetween = min(eBetween)

silhouette = silhouette + (minBetween – eWithin) / max(minBetween, eWithin)

**return** silhouette / rowlength(A)

La complessità in tempo dell’algoritmo è O(n2) dove n corrisponde al numero di punti nel data set. Tale complessità risulta subottimale in quanto trascura la simmetria della funzione distanza; tuttavia la scelta di tale algoritmo deriva dalla necessità di ottenere una complessità in spazio tale da poter processare data set di numerosità superiore ad 1 milione di punti utilizzando una macchina di tipo consumer.

Come diagramma delle classi UML ad alto livello (o kite level) consideriamo il seguente:



dove le responsabilità sono così distribuite:

* DatasetReader legge i punti da file assumendo che tutte le componenti, di ogni punto, siano coinvolte nella precedente operazione di clustering
* YhfyhfReader legge il data set ottenuto dall’operazione di clustering effettuata dall’implementazione Yhfyhf.
* JgalileeReader legge il data set ottenuto dall’operazione di clustering effettuata dall’implementazione Jgalilee.
* BrizziBReader legge il data set ottenuto dall’operazione di clustering effettuata dall’implementazione BrizziB.
* ArtificialReader legge i dataset artificiali bidimensionali generati dal programma DatasetGenerator.
* IndianPinesReader: legge il data set Indian Pines per come presentato nel pacchetto del software S2Tec.
* Silhouette calcola l’indice di Silhouette per mezzo dell’algoritmo SILHOUETTE.