QUALITY THRESHOLD

Estensioni

1.1) Client con interfaccia sviluppata in javaFX

Si è progettato un client come applicazione desktop.
La realizzazione dell'interfaccia di tale applicazione è avvenuta tramite lo scene builder di javaFX, il quale crea dei file FXML per ogni interfaccia grafica.
Il vantaggio consiste nel disaccoppiare la creazione dei componenti grafici dal codice tramite i file FXML e di evitare di ricompilare il codice, visualizzando immediatamente i cambiamenti grafici operati.
Il codice sorgente in FXML è generato automaticamente dallo scene builder che permette la creazione di prototipi di interfacce interattive e facilmente testabili.
Per una buona progettazione dell'interfaccia è stato adoperato il pattern architetturale MVC (Model-View-Controller).

1.2) Model-View-Controller

Il pattern architetturale MVC consente un'organizzazione in package dell'interfaccia, in modo da separare le classi create a seconda del loro scopo.

Nel package "model" si inseriscono le classi che si occupano di acquisire le informazioni necessarie da un server o da una base di dati esterna ed eventualmente rimodellarle per l'interfaccia.

Nel package "view" si inseriscono le componenti grafiche che modellano l'interfaccia.
Nel caso specifico del progetto attuale la view è rappresentata principalmente dai file FXML.

Nel package "controller" si inseriscono tutte le classi che si occupano di trasformare le interazioni avvenute con l'interfaccia in adeguate richieste da effettuare al model.
Nel caso del progetto attuale questo package conterrà tutti gli ascoltatori associati all'interfaccia.

1.3) Elementi dell'interfaccia

1.3.1) Selezione delle tabelle presenti nel database.

È stato inserito un riquadro per selezionare direttamente una delle tabelle del database. Viene dunque effettuato un controllo per ottenere la lista di tutte le tabelle presenti nel database, permettendo di applicare l'algoritmo di clustering a qualsiasi tabella selezionabile.

1.3.2) Selezione dei file salvati.

Tramite un filtro è possibile mostrare una lista dei file .dmp salvati nel percorso di default. Tali file saranno selezionabili, consentendo di applicare l'algoritmo di clustering alle informazioni salvate anche se il database a partire dal quale è stato salvato il file non è più presente.
Qualora il file non dovesse essere idoneo, si genererebbe una finestra di errore al momento dell'apertura.

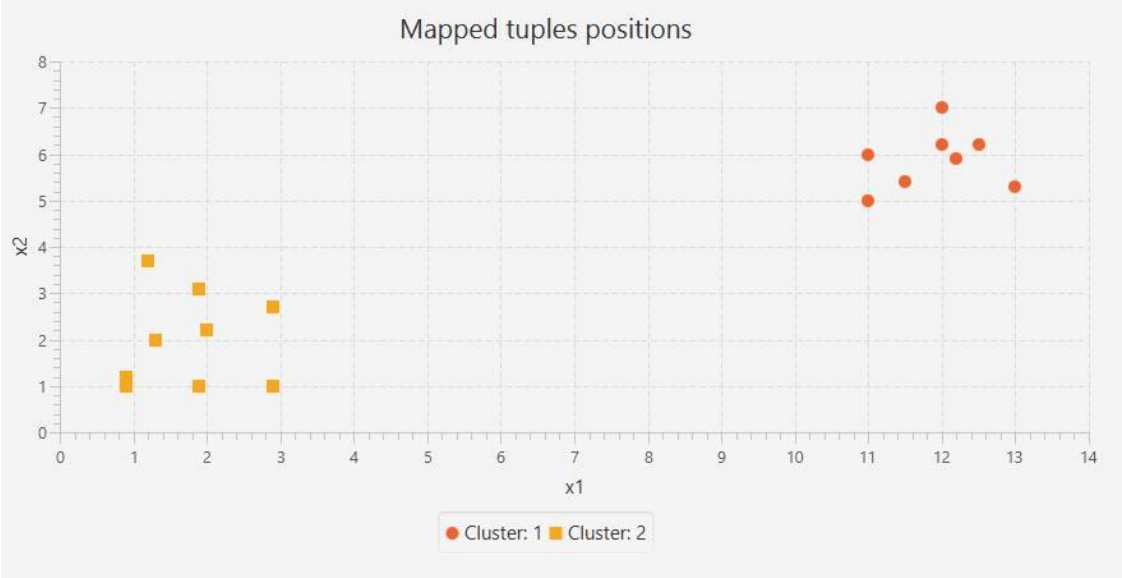
1.3.3) Visualizzazione della lista dei centroidi in una tabella.

La lista dei centroidi visualizzata sotto forma di stringa nel progetto originale, viene immessa in una tabella.
Non viene più inviato dal server l'output separatamente tramite il toString del QTMiner, bensì viene ricavato dal Transfer Object che si occupa di comunicare tutte le informazioni necessarie al client.
Ogni riga della tabella rappresenta un centroide e ha un ascoltatore associato che consente la generazione istantanea dell'istogramma delle distanze associato al cluster.

1.3.4) Grafici

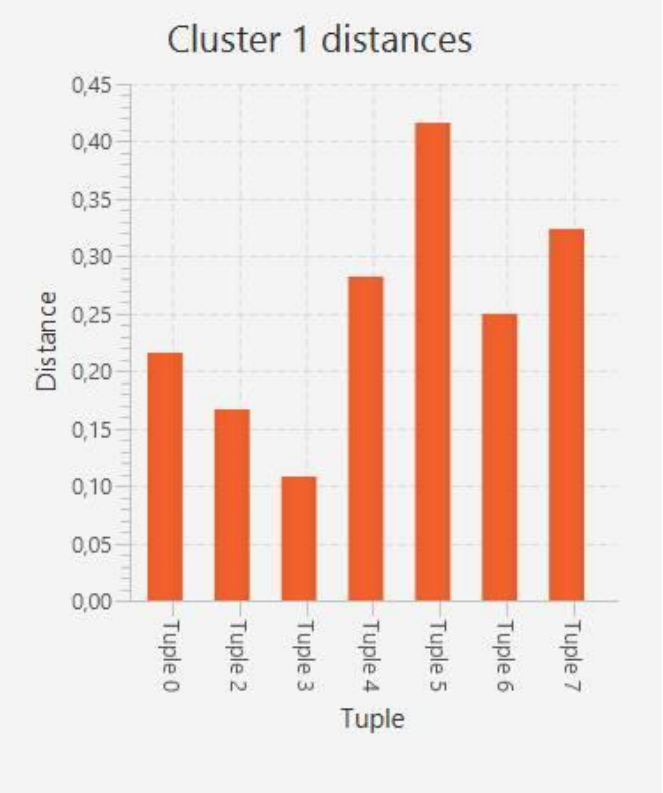
Scatter chart

Il grafico mostra la dispersione delle tuple raggruppate in clusters all'interno di un piano cartesiano, considerati due attributi distinti.



Bar chart

Il grafico è un istogramma che mostra la distanza di ogni tupla dal relativo centroide.
Si crea un differente bar chart per ogni cluster individuato.



1.4) Visualizzazione come Applet

L'applicazione in javaFX è resa visualizzabile tramite browser web sotto forma di Applet, come richiesto dalla presentazione al caso di studio. Il main dell'applicazione javaFX estende la classe FXApplet che a sua volta estende JApplet.
La classe FXApplet contiene:

- I metodi standard da sovrascrivere di JApplet.
- L'aggiunta opzionale del metodo initApplet() che consente di aggiungere componenti grafici all'applet.
- L'aggiunta del metodo initFX() che serve per caricare i file FXML aggiungendoli ad un JFXPanel. Esso viene sovrascritto dal main dell'applicazione javaFX.

2) Transfer Object

Il pattern Transfer Object è utilizzato per trasferire informazioni con diversi attributi dal server al client. Il Transfer Object è una classe POJO serializzabile contenente metodi getter e setter e che viene popolata dalle business class del server per poi essere spedita al client. I Transfer Object non hanno dunque alcun comportamento e le informazioni non vengono in alcun modo modificate dal client.
I Transfer Object presenti nel progetto sono:

- ItemTO che rappresenta la classe Item del progetto base
- TupleTO che rappresenta la classe Tuple del progetto base
- OutputTO che incapsula i risultati richiesti dal client
- Attribute, DiscreteAttribute e ContinuousAttribute sono già serializzabili e contengono solo metodi getter e setter, dunque non c'è bisogno di creare dei Transfer Object ad hoc.