Università degli Studi di Napoli Parthenope
Dipartimento di Scienze e Tecnologie
Corso di programmazione III

Data Mining

Titolo Progetto Data Mining

Studente: Giuseppe Accardo

Matricola: 0124000879

Anno Accademico: 2016/2017

Docente: Angelo Ciaramella

Indice

Traccia e descrizione del problema

Problema

Si vuole sviluppare un sistema di **Data Mining** per l'elaborazione di dati. Il Data Mining comprende un insieme di tecniche e metodologie che hanno per oggetto l'estrazione e visualizzazione di informazioni da grandi quantit'a di dati. Le fasi principali del sistema che si vuole sviluppare sono: **selezione delle caratteristiche, clustering e visualizzazione**.
Si suppone di avere un data set contnuto in un file .data. Il data set 'e ralativo alla classificazione di 3 tipi di rose (setosa, versicolour, virginica) mediante sue quattro carattrisiche (feature): lunghezza e larghezza del sepalo, lunghezza e larghezza del petalo.
Nella fase di selezione delle caratteristiche un utente pu'o scegliere il numero di caratteirstiche da selezionare per l'analisi (le colonne del data set). Nella fase di clustering viene usato un algoritmo per agglomerare dati simili. Nel caso specifico viene usato l'algoritmo K-Means. L'utente pu'o scegliere il numero di cluster da usare.
Nella fase di visualizzazione i dati agglomerati sono visualizzati in 2 e 3 dimensioni. Nel caso in cui il numero delle feature 'e piu' grande di 3 viene applicato un algoritmo di Analisi delle Componenti Principali per la visualizzazione.

Cos'è il Datamining

Il **data mining** è l'insieme di tecniche e metodologie che hanno per oggetto l'estrazione di un sapere o di una conoscenza a partire da grandi quantità di dati (attraverso metodi automatici o semi-automatici) e l'utilizzo scientifico, industriale o operativo di questo sapere.
Le tecniche di data mining sono fondate su specifici algoritmi. I pattern identificati possono essere, a loro volta, il punto di partenza per ipotizzare e quindi verificare nuove relazioni di tipo causale fra fenomeni; in generale, possono servire in senso statistico per formulare previsioni su nuovi insiemi di dati.
Un concetto correlato al data mining è quello di apprendimento automatico (**Machine learning**); infatti, l'identificazione di pattern può paragonarsi all'apprendimento, da parte del sistema di data mining, di una relazione causale precedentemente ignota, cosa che trova applicazione in ambiti come quello degli algoritmi euristici e dell'intelligenza artificiale. Tuttavia, occorre notare che il processo di data mining è sempre sottoposto al rischio di rivelare relazioni causali che poi si rivelano inesistenti.
In questo caso si utilizzerà la tecnica del Clustering mediante l'applicazione dell'algoritmo K-Means, un algoritmo del tipo "unsupervised classification" poichè le classi modello non vengono definite dall'utente, ma vengono ricostruite automaticamente. L'utente decide solo il numero di gruppi. Lo scopo del sistema sarà di "riconoscere" le rose in base alle loro caratteristiche.

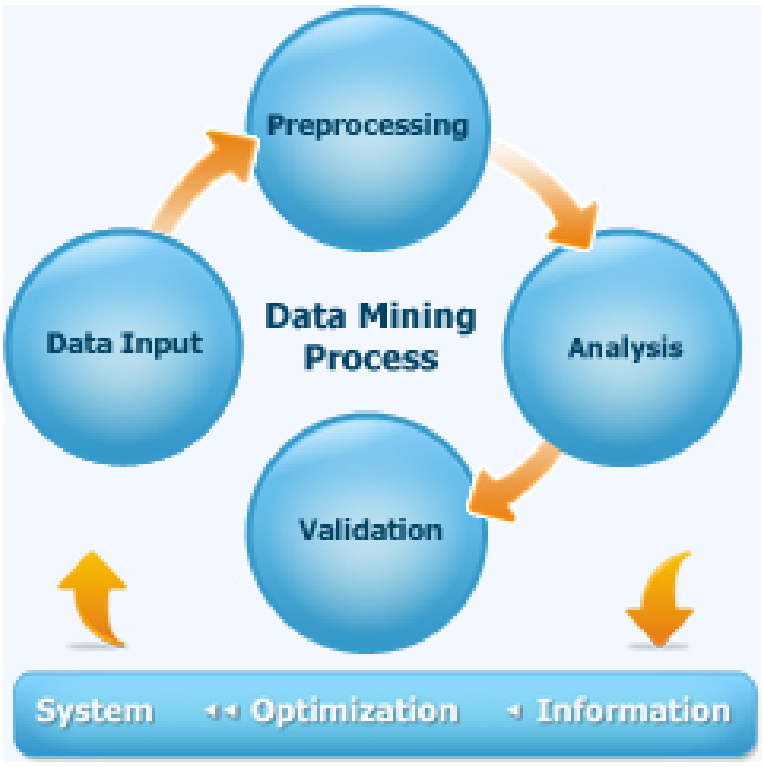


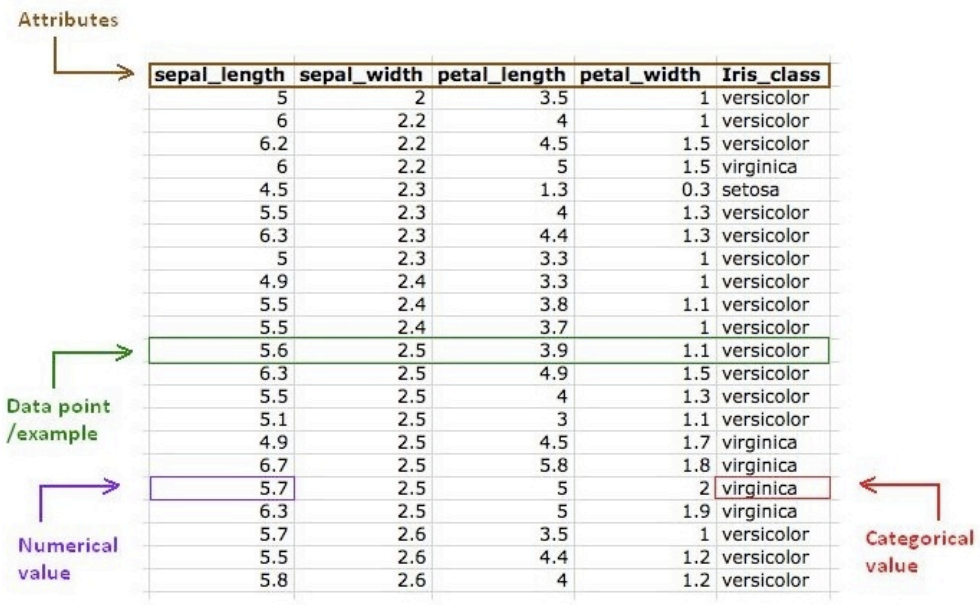
Figura 1: Datamining

Struttura del Dataset

Un **dataset** (o data set) è una collezione di dati.
Più comunemente un dataset costituisce un insieme di dati strutturati in forma relazionale, cioè corrisponde al contenuto di una singola tabella di database, oppure ad una singola matrice di dati statistici, in cui ogni colonna della tabella rappresenta una particolare variabile, e ogni riga corrisponde ad un determinato membro del dataset in questione.
La dimensione del dataset è data dal numero dei membri presenti (**osservazioni)**, che formano le righe, e dal numero delle variabili di cui si compone (**features**), che formano le colonne.

Dataset Iris

Il dataset Iris è un dataset multivariato introdotto da Ronald Fisher nel 1936. Consiste in 150 istanze di Iris misurate da Edgar Anderson e classificate secondo tre specie: Iris setosa, Iris virginica e Iris versicolor. Le quattro variabili considerate sono la lunghezza e la larghezza del sepalo e del petalo. A causa di errori, esistono diverse versioni del dataset utilizzate nella letteratura scientifica. Il dataset Iris viene utilizzato nell'ambito dell'apprendimento automatico come esempio di classificazione statistica.



sepal_length	sepal_width	petal_length	petal_width	Iris_class
5	2	3.5	1	versicolor
6	2.2	4	1	versicolor
6.2	2.2	4.5	1.5	versicolor
6	2.2	5	1.5	virginica
4.5	2.3	1.3	0.3	setosa
5.5	2.3	4	1.3	versicolor
6.3	2.3	4.4	1.3	versicolor
5	2.3	3.3	1	versicolor
4.9	2.4	3.3	1	versicolor
5.5	2.4	3.8	1.1	versicolor
5.5	2.4	3.7	1	versicolor
5.6	2.5	3.9	1.1	versicolor
6.3	2.5	4.9	1.5	versicolor
5.5	2.5	4	1.3	versicolor
5.1	2.5	3	1.1	versicolor
4.9	2.5	4.5	1.7	virginica
6.7	2.5	5.8	1.8	virginica
5.7	2.5	5	2	virginica
6.3	2.5	5	1.9	virginica
5.7	2.6	3.5	1	versicolor
5.5	2.6	4.4	1.2	versicolor
5.8	2.6	4	1.2	versicolor

Figura 2: Dataset iris

Descrizione del K-Means

L'algoritmo K-means è un algoritmo di **clustering partizionale** che permette di suddividere un insieme di oggetti (punti) in K gruppi sulla base dei loro attributi, ossia raggruppare per oggetti "simili".

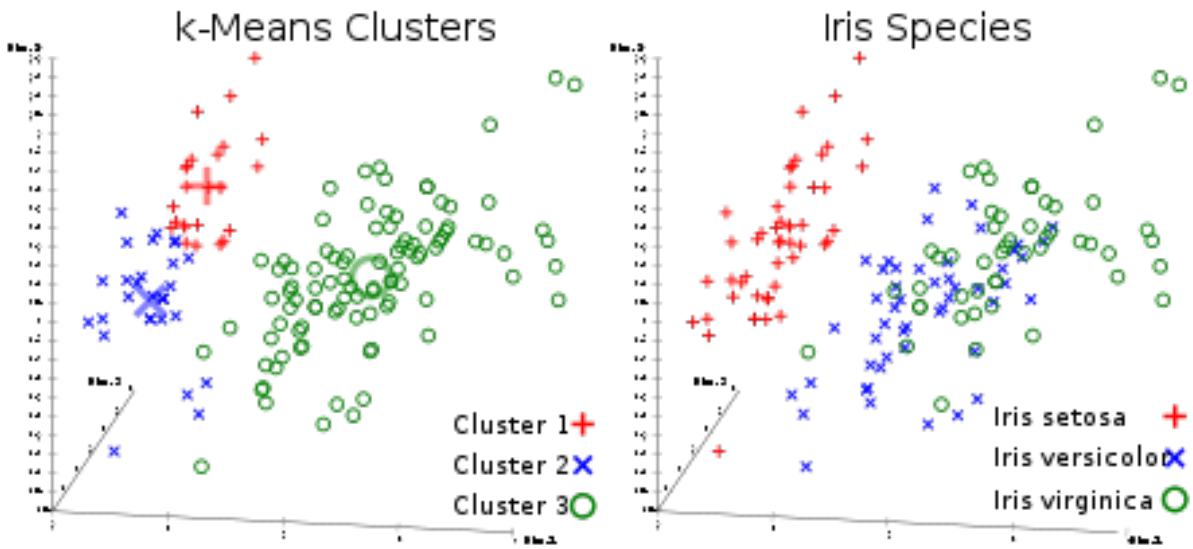


Figura 3: Visualizzazione del clustering mediante K-Means

Algoritmo

L'obiettivo che l'algoritmo si prepone è di minimizzare la varianza totale intra-cluster. Ogni cluster viene identificato mediante un **centroide** o punto medio. L'algoritmo segue una procedura iterativa: inizialmente crea K partizioni (K è l'unico valori di input) e assegna ad ogni partizione i punti d'ingresso o casualmente o usando alcune informazioni euristiche (conoscere a priori alcuni oggetti ed ecc) utilizzando la minima distanza euclidea (**MDM**).

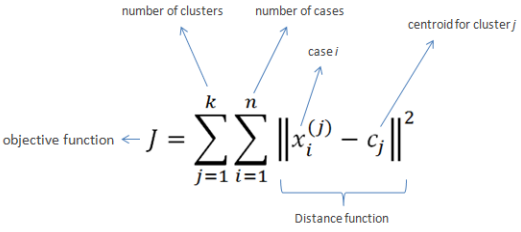


Figura 4: Applicazione della distanza euclidea

Quindi calcola il centroide di ogni gruppo costruendo così una nuova partizione associando ogni punto d'ingresso al cluster il cui centroide è più vicino ad esso. Quindi vengono ricalcolati i centroidi per i nuovi cluster come la media dei punti calcolati per ogni cluster e così via, finché l'algoritmo non converge.
Attenzione: la convergenza **non è sempre garantita** scegliendo K cluster poichè

potrebbero essere scelti dei centroidi così vicini, da creare meno gruppi di quelli che dovrebbero essere presenti.

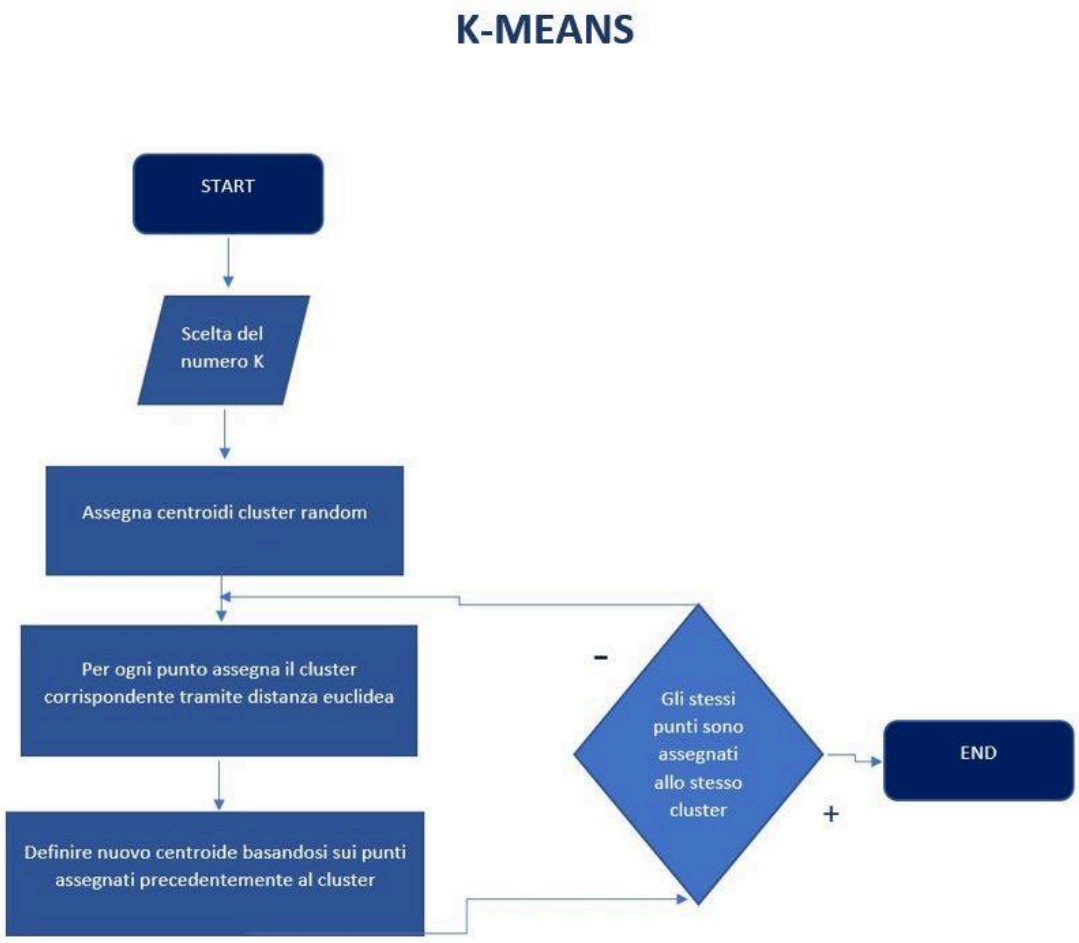


Figura 5: Algoritmo seguito per la realizzazione del K-Means

Analisi delle Componenti Principali (PCA)

L'analisi in componenti principali o PCA, dall'inglese principal component analysis, è una tecnica per la semplificazione dei dati utilizzata nell'ambito della statistica multivariata.
Lo scopo primario di questa tecnica è la riduzione di un numero più o meno elevato di variabili (rappresentanti altrettante caratteristiche del fenomeno analizzato) in alcune variabili latenti (feature reduction). Infatti tale è necessaria nel caso si volesse visualizzare un dataset con un numero di Features maggiori di 3, poichè non sarebbe possibile effettuare la visualizzazione, senza perderne il contenuto informativo.
Ciò avviene tramite una trasformazione lineare delle variabili che proietta quelle originarie in un nuovo sistema cartesiano nel quale la nuova variabile con la maggiore varianza viene proiettata sul primo asse, la variabile nuova, seconda per dimensione della varianza, sul secondo asse e così via. La riduzione della complessità avviene limitandosi ad analizzare le principali (per varianza) tra le nuove variabili.

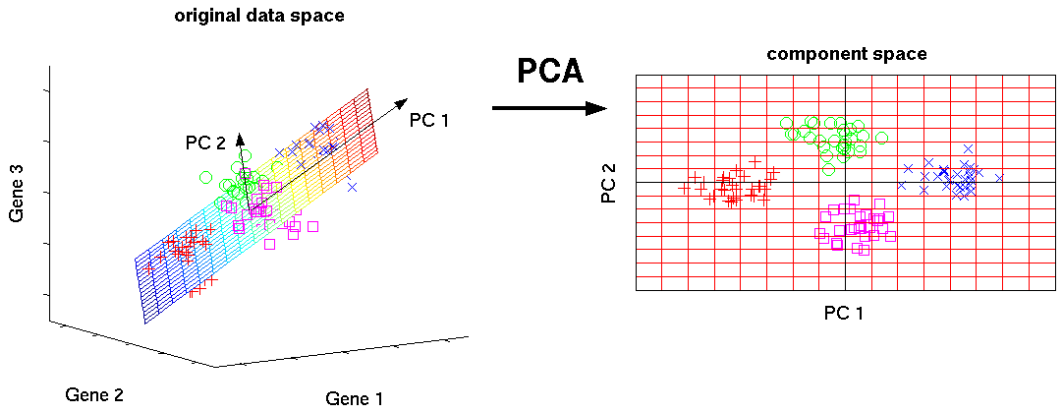


Figura 6: Realizzazione di una PCA

Ciò avviene tramite una trasformazione lineare delle variabili che proietta quelle originarie in un nuovo sistema cartesiano nel quale la nuova variabile con la maggiore varianza viene proiettata sul primo asse, la variabile nuova, seconda per dimensione della varianza, sul secondo asse e così via. La riduzione della complessità avviene limitandosi ad analizzare le principali (per varianza) tra le nuove variabili.

Progettazione

Analisi

Analizzando il problema si è deciso di sviluppare la progettazione di un sistema, suddividendo la modellazione e sviluppo in due macro-parti:

- La prima che si occuperà del **BACK-END**, nello specifico di *"forinire le strutture dati e le funzionalità necessarie alla creazione del Dataset e del Datamining"*.
  Il *Dataset* è caratterizzato dalle seguenti peculiarità:
  - **Informazioni fondamentali sul Dataset**, come il nome e il nome delle features (ad esempio Iris avrà lunghezza petalo, larghetta petalo ed ecc...);
  - **Caricamento del dataset**, cioè descrizione dei passaggi per caricare un determinato dataset. Nel caso di iris sarà un caricamento di un file .data, ma in generale può cambiare per ogni specifico dataset;
  - **Memorizzazione dai dati**, cioè come verranno memorizzati i record o osservazioni del dataset nella adeguata strutture dati che rappresenterà la tabella relazionale (o una matrice).

  Il *Data Mining* rappresenta un insieme di metodi e tecniche sintetizzati nel seguente modo:
  - **Decisione del Dataset** su cui operare;
  - **Selezione delle features**, cioè decidere quali caratteristiche o features scegliere per la clusterizzazione;
  - **Clusterizzazione mediante l'algoritmo K-Means**, indicando il numero K di cluster.
    NB la clusterizzazione non fornisce ancora la visualizzazione, perchè dipenderà dal gestore del front-end plottare i vari grafici.
- La seconda parte si occuperà del **FRONT-END**, nel dettaglio nel *"fornire un'adeguata GUI in JavaFX per una maggior esaltazione della user experience"*.
  - **Creazione di un menù** per garantire l'interazione con l'utente;
  - **Applicazione di una grafica adeguata**, cioè mediante una serie di applicazione di fogli di stile e vari accorgimenti grafici;
  - **Visualizzazione del risultato del clustering**, cioè la visualizzazione dei grafici 2D/3D utilizzati per l'output del DataMining.

Entrambe le macro-parti, sono stati analizzate utilizzando il paradigma della **programmazione orientata agli oggetti (OOP)**, individuando le caratteristiche principali descritte sopra.

Il linguaggio utilizzato per la progettazione è **Java 8 SE** (utilizzando anche nuovi costrutti), mentre è stato utilizzato unicamente **JavaFX** (oramai standard di java) per l'interfaccia grafica.

Novità di Java 8 utilizzate

Espressioni Lambda

Come visto durante il corso, tra le varie novità di Java 8 sono presenti il potenziamento delle interfacce, integrando metodi statici (non vengono ereditati) e definendo i metodi di default che rappresentano dei metodi concreti che saranno ereditati, facendo attenzione al Diamond Problem.
Per la progettazione sono state utilizzate le **Espressioni LAMBDA** che rappresentano una sintassi più semplice e leggibile per definire-creare un'istanza di una classe anonima che implementa un'interfaccia con un solo metodo astratto (interfaccia funzionale). Si consideri la seguante equivalenza per capirne il funzionamento:

```
lambda-parametri -> lambda-corpo
```

Si noti lo speciale segno "->". Questa è praticamente equivalente a:

```
new NomeI() {
    metodo(lambda-parametri) {
        lambda-corpo
    }
}
```

Nel progetto si è fatto un largo uso, ad esempio per registrare l'evento di evidenziare la colonna di una corrispondente checkbox:

```
/* Verifica se tale colonna dovrà essere selezionando controllando la corrispondente checkbox se è
     selezionata.*/
boxes[i].selectedProperty().addListener(/* Lambda expression con changeListener e changer */
        (obs, wasSelected, isNowSelected) -> //parametri disponibili per override
        {
        /* Se il box è stato selezionato aggiungilo tra le colonne da evidenziare, altrimenti
             toglielo da quelle evidenziate. */
            if (isNowSelected) {
                highlightColumns.add(col);
            } else {
                highlightColumns.remove(col);
            }
     });
```

In generale può essere utilizzata anche per riferimento a metodo o a costruttori mediante il simbolo '::'che non sono anonime, ma che si riferiscono ad un specifico metodo di una determinata classe (o di una istanza). Di seguito un esempio completo:

```
//Senza Lambda
List list1 = Arrays.asList(1,2,3,5);
for(Integer n: list1) {
    System.out.println(n);
}
/*foreach è metodo che accetta consumer, un'interfaccia funzionale che corrisponde all'azione da
    eguire per ogni elemento della lista*/
//Lambda Anonime
List list2 = Arrays.asList(1,2,3,5);
list2.forEach(n -> System.out.println(n));
//Lambda con Riferimenti anonimi,
list2.forEach(System.out::println);
```

JavaFX

JavaFX è una famiglia di software applicativi (**Framework**) divenuti ormai standard di Java8 SE, utilizzata soprattutto per la creazione di applicazioni web che hanno tutte

le caratteristiche e funzionalità delle comuni applicazioni per computer. Con JavaFX è possibile realizzare delle applicazioni per computer, cellulari (andoroid o iOS), dispositivi portatili di vario genere, televisori e altri tipi di piattaforme.

Lo spazio grafico di lavoro di JavaFX è un oggetto **Stage**, radice di ogni applicazione JavaFX 8 (finestra), che può contenere uno o piu oggetti **Scene**, che a loro volta contengono generici componenti grafici (**Node**) che presentano delle proprie proprietà (*Property*). Il concetto ricorda molto quello di DOM di una pagina HTML, in cui possono essere disposti opportuni tag con alcuni attributi.

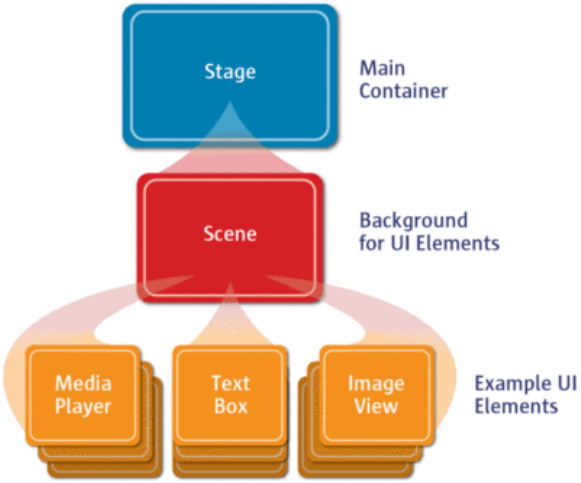


Figura 7: Idea dell'albero utilizzato da JavaFX

Java FX organizza la grafica un'insieme di nodi organizzati in una struttura ad albero, mediante un formato di Markup **FXML**, derivante da XML, che fornisce la struttura di un'interfaccia grafica separandola dal codice dell'applicazione consentendo di assemblare interfacce grafiche senza utilizzare codice Java. FXML non è un linguaggio compilatoe quindi non necessita di compilazione. Utilizzando il pattern **MVC** e un controller, diventa molto semplice la manipolazione.
Ricapitolando, queste sono le principali peculiarità che rendono molto interessante JavaFX:

- Immediato uso del **pattern Architetturale MVC**;
- Supporta il formato e i file **CSS**, ossia fogli di stile utilizzati per dare maggior enfasi grafica;
- Supporto a motore grafico 2D/3D;
- Integrazione di grafici scientifici;
- Portabilità e multipiattaforma;
- Facile costruzione di Layout Statici e Dinamici;

File .FXML, .CSS e utilizzo delle routine principali di JavaFX sono presenti nel progetto.

Organizzazione e struttura progetto

La progettazione delle classi e la costruzione dell'applicazione è avvenuta utilizzando l'idee *"Eclipse Java EE IDE Versione: Mars.1 (4.5.1)"*.
Le principali librerie da linkare utilizzate per la progettazione sono:

- **JavaFX SDK;** per utilizzare javafx
- **colt-1.2.0.jar**, sviluppata dal CERN (Organizzazione europea per la ricerca nucleare) necessaria per la costruzione della classe 'PCAcolt', indispensabile per applicare la PCA.

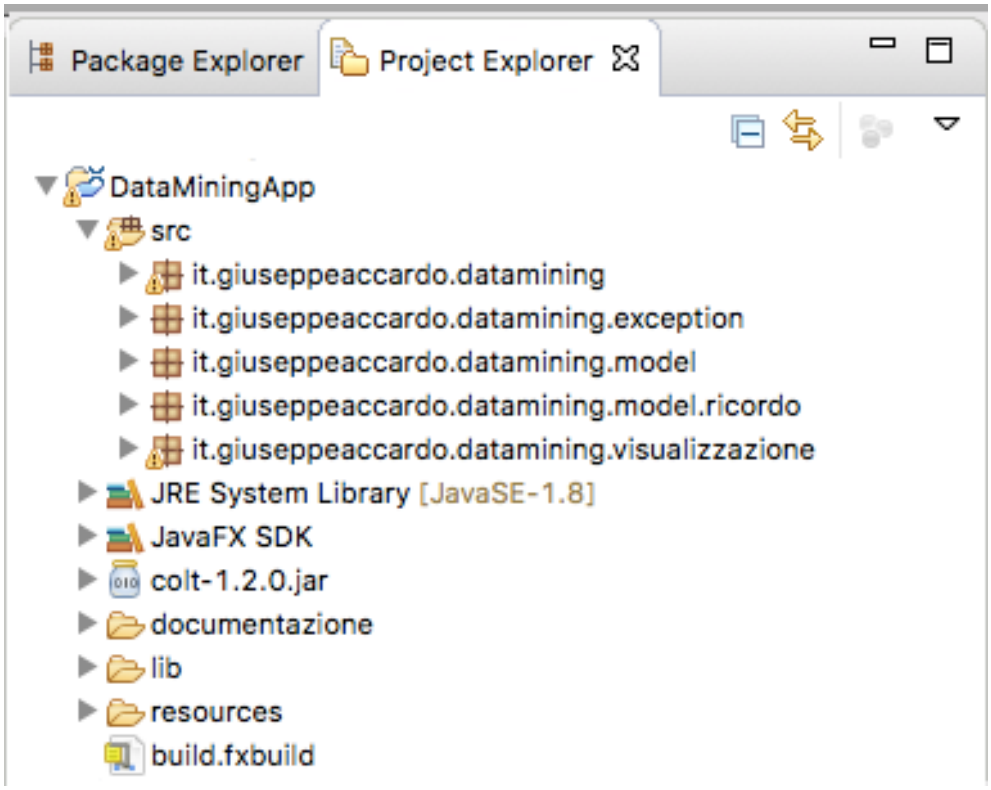


Figura 8: Struttura del progetto in Eclipse

Si può osservare che il primo package è quello contenente i file principali dell'applicazione. Tali apparterranno, come si vedrà successivamente, alla **'View'** e **'Controller'** utilizzati per visualizzare il menu, gestire le interazioni con esso e visualizzare il grafico (grazie all'uso delle classi all'interno dell'ultimo package).
Il secondo package è utilizzato per contenere particolare eccezioni utilizzare dal controller per la visualizzazione di alert o conferme.
I package che terminano con 'model' e 'model.ricordo' sono invece i **'Model'** utilizzati dal controller per il funzionamento dell'intera applicazione.
Inoltre sono presenti cartelle per la documentazione (JavaDOC) e rouserces per vari file (file dataset e immagini).
Il progetto presenta applicazioni di 'Ingegneria del Software' mediante l'utilizzo di Design Pattern per la risoluzione di varie problematiche più complesse. Infatti permettono di ottenere delle soluzioni, seguendo degli "schemi" o "pattern" che a priori permettono di risolvere i problemi.

In totale ne sono stati utilizzati 6 e sono i seguenti:

- **MVC**
- **Strategy**
- **Template Method**
- **Observer**
- **Memento**
- **Prototype**

Più avanti ne approfondiremo l'applicazione.

MVC

Il Model-View-Controller (MVC) è un **Pattern Architetturale** molto diffuso nello sviluppo di sistemi software (applicazioni J2EE, .NET ed ecc), in particolare nell'ambito della programmazione orientata agli oggetti, in grado di separare la logica di presentazione dei dati dalla logica di business.
Tale pattern è composto da 3 componenti che comunicano tra loro e sono:

- **Model**;
- **Controller**;
- **View**.

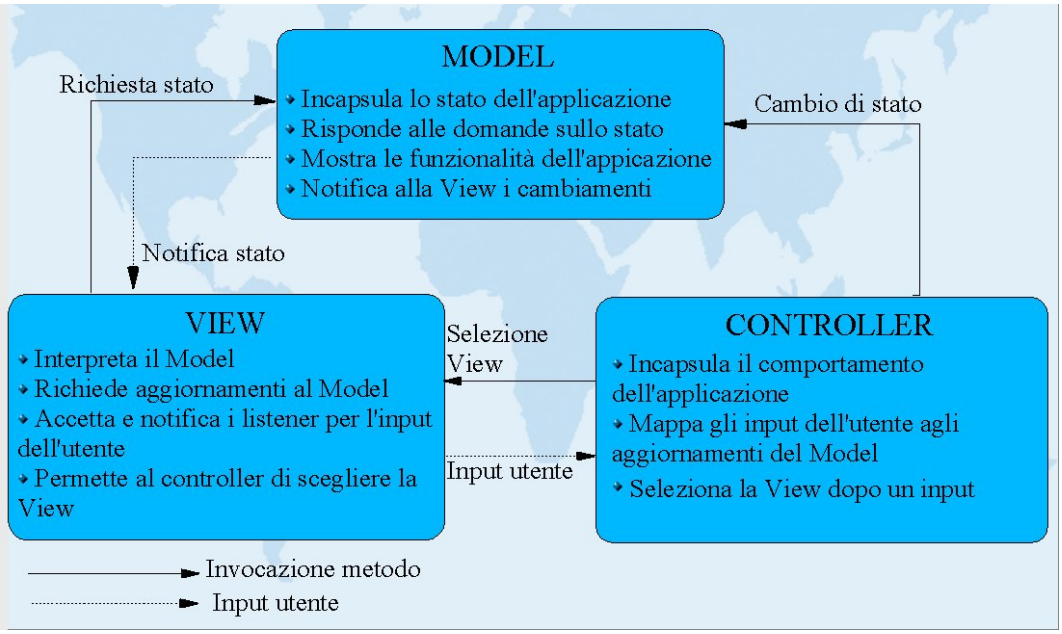


Figura 9: Tipico funzionamento dell'MVC

In questa applicazione il **Model** è realizzato mediante le classi 'Dataset' e 'Datamining' poichè forniscono l'entry per accedere ai dati e di memorizzarne uno stato in base alle richieste del client. Quindi il funzionamento del DataMining e del K-Means è completamente trasparente all'utente, cioè si dice che sia a livello *Back-End*.
Il model rappresenta l'insieme delle classi all'interno dei package *it.giuseppeaccardo.datamining.model* e *it.giuseppeaccardo.datamining.model.ricordo*.

Il **Controller** ha il compito di ricevere i comandi dall'utente (attraverso la **GUI** appartenente view) e di mapparne tutti gli aggiornamenti e cambiamenti di stato verso il model. Effettuate tale modifiche, il controller selezionerà la view su cui voler mostrare i gli eventuali risultati del model o effettuare una determinata azione (rendere il layout dinamico).
Il controller è realizzato mediante la classe *DataminingController.java* contenuta nel package principale *it.giuseppeaccardo.datamining*. Da notare che l'MVC effettua una divisione comportamentale e logica delle componenti, quindi ad esempio parte di View sarà contenuta anche nel controller.

La **View** si occupa di interpretare il model a livello di *Front-end* (interfaccia utente grafica o GUI). Da esso possono partire delle richieste verso il controller che risponderà, ma può richiedere anche semplicemente lo stato direttamente al model o ricevere da esso una notifica di cambiamento di stato.
Questo legame (in ambito di JavaFX è detto **Binding**) è effettuato mediante il design pattern ***"Observer"*** per definire un forte dipendenza tra view e model in modo tale che se una cambia, l'altra è aggiornata automaticamente.
In JavaFX tale design pattern è implementato direttamente mediante la collection interface **'ObservableList<E>'** che appunto tiene traccia di tutti i dati e dei suoi aggiornamenti.
Nel progetto la view è rappresentato dal layout sviluppato in **FXML**, comprendenti di fogli di stie **CSS** e dal **Main** che si occupa di lanciare lo stage su cui andranno le scene da far visualizzare all'utente. Tali moduli sono presenti sempre in *it.giuseppeaccardo.datamining*, appunto per le forti correlazioni che si hanno tra view e controller. Inoltre anche *it.giuseppeaccardo.datamining.visualizzazione* e *it.giuseppeaccardo.datamining.exception* vengono utilizzate dal controller per fornire alla view grafici o alert di attenzione (eccezioni).

Come detto la scelta di costruire l'applicazione solo con JavaFX è perchè presenta una serie di vantaggi (costruzioni di grafici, facile comunicazione con altri software ed ecc..), ma soprattutto perchè è già predisposto per la costruzioni di applicazioni seguendo il Design Pattern MVC. Infatti, come detto prima, fornisce già strutture che facilitano il compito di determinate operazioni, come l'aggiornamento dei dati che è direttamente implementato mediante l'ObservableList che presenta le funzionalità dell'Observer pattern.
Inoltre l'MVC garantisce una facile portabilità di sistema (Servlet ed ecc).

Principi SOLID

I principi **SOLID** sono stati introdotti da Robert C. Martin, uno dei padri delle metodologie agili, consentendo di avere un software estendibile e manutenibile, evitando di avere software rigido, difficile da cambiare, con cut and paste inutili e complesso **(code smells)**. L'acronimo corrisponde ai seguenti principi:

- **Single Responsability Principle (SRP)**
- **Open Closed Principle (OCP)**
- **Liskov's Substitution Principle (LSP)**
- **Interface Segregation Principle (ISP)**
- **Dependency Inversione Principle (DIP)**

Nello specifico tecnico:

1. **Single Responsability Principle (SRP)**
   Una classe dovrebbe avere un solo motivo per cambiare ogni elemento di un programma (classe, metodo, variabile) deve avere una sola responsabilit'a tale responsabilità debba essere interamente incapsulata dall'elemento stesso. Questo principio afferma che se una classe può cambiare per 2 motivi allora dobbiamo dividere le responsabilità in 2 classi, ogni classe gestisce solo una responsabilita'. Si consideri un modulo che compila e stampa un report, tale modulo puà cambiare per due motivi: il contenuto del report oppure il formato del report, questi due aspetti sono due responsabilita' distinte e vanno descritte in classi o moduli separati. Sarebbe cattiva progettazione accoppiare due cose che cambiano per motivi diversi in momenti diversi.
2. **Open Closed Principle (OCP)**
   Le entità software come classi, moduli e funzioni dovrebbero essere aperte per l'estensione, ma chiuse per le modifiche. Il principio stabilisce che la progettazione del software deve essere fatta in modo che le nuove funzionalita' sono aggiunte con il minimo cambiamento possibile nel codice esistente. Quando scriviamo una classe dobbiamo garantire che per una sua estensione non c''e bisogno di cambiare la classe stessa. Casi particolari di OCP: Template pattern, Strategy pattern, anche i pattern Decorator, Factory Method, Observer ci aiutano a rispettare OCP.
3. **Liskov's Substitution Principle (LSP)**
   I tipi derivati devono essere completamente sostituibili ai loro tipi base. Dobbiamo essere sicuri che nuove classi derivate stanno estendendo le loro classi base senza cambiare il comportamento. Le classi derivate dovrebbero essere capaci di rimpiazzare le classi base senza cambiamenti nel codice, altrimenti le nuove classi producono eetti indesiderati sul funzionamento del programma. Questo è un'estensione di OCP.
4. **Interface Segregation Principle (ISP)**
   Un client non dovrebbe dipendere da metodi che non usa. E' preferibile che le interfacce sia molte, specifiche e piccole (composte da pochi metodi) piuttosto che

poche, generali e grandi. Quando progettiamo software dobbiamo stare attenti a come definiamo le interfacce, se contengono molti metodi puo‘ capitare che vengono utilizzate da un modulo che contiene solo alcune di quelle funzionalita‘, siamo obbligati a implementare l'interfaccia completa e ritrovarci con metodi inutilizzabili.

5. **Dependency Inversione Principle (DIP)**
   I moduli di alto livello non dovrebbero dipendere dai moduli di basso livello. Entrambi dovrebbero dipendere dalle astrazioni. Le astrazioni non dovrebbero dipendere dai dettagli. I dettagli dovrebbero dipendere dalle astrazioni. In un cattivo sviluppo del software le classi ad alto livello dipendono pesantemente da quelle a piu‘ basso livello. Quando progettiamo un software definiamo classi di basso livello quelle che implementano le operazioni di base (accesso al disco, protocollo di rete ecc.)be classi di alto livello quelle che incapsulano la logica di business, un approccio naturale di sviluppo ‘e quello di scrivere prima le classi di basso livello e poi quello di alto livello, ma questo non ‘e un modo di progettare flessibile: nel caso in cui dobbiamo sostituire una classe di basso livello abbiamo che le classi di alto livellodipendono pesantemente da quelle di basso livello e questo comporta la modifica anche nelle classi di alto livello. Per evitare questo tipo di problemi possiamo introdurre uno strato intermedio (astrazione) tra le classi di alto livello e quelle di basso livello: le classi di alto livello non devono dipendere da quelle di basso livello lo strato intermedio non deve essere creato basandosi su le classi di basso livello, le classi di livello basso sono create basandosi sullo strato intermedio.

Il progetto è stato sviluppato ponendo attenzione al rispetto dei principi SOLID riporto un esempio per ognuno dei principi:

1. **Single Responsability Principle (SRP)**
   Ogni singola classe è costruita in modo tale da avere classi con singole riesponsabilitò.
   Ad esempio visualizzazione grafica è una classe separata rispetto dalla classe Datamining. Infatti in base al contenuto di un Datamining, viene generato in un apposita classe una nuova classe Visualizzazione che presenta delle responsabilità distinte.

2. **Open Closed Principle (OCP)**
   Le classi sono molto più predisposte all'estensione, che alla modifiche delle stesse. Come detto in precedenza, utilizzando classi astratte o determinati pattern, come il Template o lo Strategy, riusciamo a garantire facilmente questo principio. Un ottimo esempio è il Template utilizzato per fornire un Framework di Dataset, in cui il Dataset concreto (come Iris) non necessita di effettuare cambiamenti alla classe stessa, poichè è aperta all'estensione. Stessa considerazione vale per Visualizzazione poichè utilizza lo Strategy Pattern.

3. **Liskov's Substitution Principle (LSP)**
   Il principio di Liskov è ampiamente garantito.
   Basta osservare che le implementazioni concrete di Visualizzazione o Dataset, potrebbero essere completamente sostituite dal altri tipi derivati, senza perderne la logica. Questo è garantito, oltre dai pattern applicati, dall'uso di interfacce e classi astratte. Il principio non è applicato in modo "severo". Teoricamente,

anche ArrayList<E> dovrebbe essere dichiarato come un List<E> poichè è la collection da cui estende.

4. **Interface Segregation Principle (ISP)**
   Tale principio è soddisfatto poichè vengono utilizzate un numero consistente di interfacce che presentano pochi metodi.
   In particolare non sono presenti casi in cui è stato dovuto applicare (ad esempio un'interfaccia con tanti metodi scissa in più interfacce).

5. **Dependency Inversione Principle (DIP)**
   I moduli di alto livello non dipendono da quelli di basso livello, mentre i dettagli si.
   Il caricamento del dataset viene effettuato da una sua implementazione concreta (Iris in questo caso) che ne implementa un comportamento a basso livello, mentre dataset fornisce presenta giusto la logica da seguire. Un ottimo pattern da utilizzare sarebbe stato il DAO, non approfondito nel corso.

Diagrammi UML

Questo è il diagramma UML completo delle 21 classi dell'intero sistema di JavaFX, contenente le componenti MVC.
Le classi sono rigorosamente commentate e spiegate in modo dettagliato nella documentazione del JavaDOC.

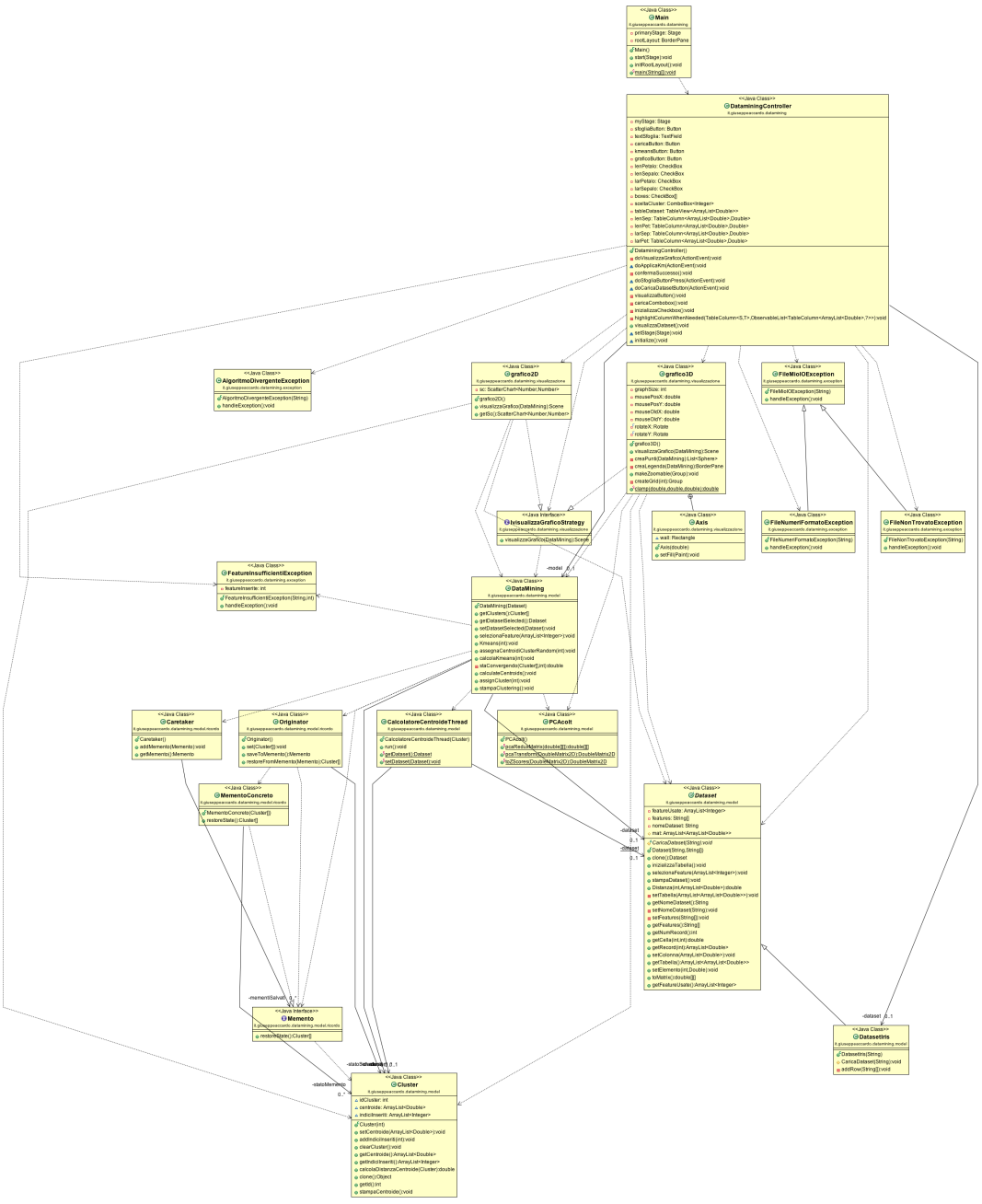


Figura 10: UML completo di tutte le classi utilizzate

Suddividendo le classi per package:

- ***it.giuseppeaccardo.datamining***
  Package contenente l'insieme dei file necessari per il funzionamento di JavaFX.
  Main è la classe principale dell'applicazione in JavaFX. Esso rappresenta la parte "View" che si occupa di impostare un "Layout" predefinito in formato FXML, ossia la parte statica dell'applicazione, con l'opportuno foglio di stile CSS. Main estende application e il metodo che eredita più importante è il metodo start(Stage primaryStage). Questo è automaticamente chiamato quando l'applicazione viene lanciata dall'interno del metodo main. Lo stage indica la finestra su cui agganciare le scene che a loro volta conterranno la disposizione degli oggetti grafici (per definire la GUI).
  Il layout da cui caricare l'interfaccia è il file FXML *Datamining.fxml* (formato di javaFX non compilato), mentre lo stile CSS è caricato dal file *application.css*.

  Ad un file FXML associato un controllore, rappresentato dallla classe **dataminingController**, che è una classe java che gestisce gli elementi grafici ad esempio impostando l'azione richiesta quando si preme un pulsante. Quindi è il pilastro dell'interazione e della user-interface dell'utente.
  Per gestire aggiornamenti tra View-Controller e notifiche di stato, viene utilizzata la collection ***ObservableList<E>*** che auto-implementa il design pattern **Observer**.

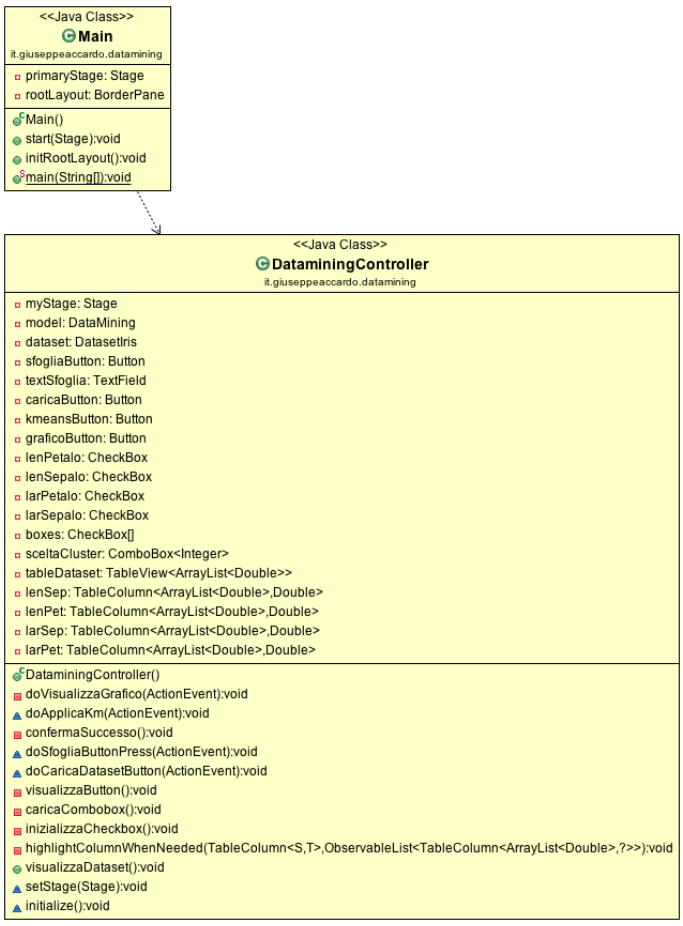


Figura 11: UML delle classi principali per gestire javaFX

- ***it.giuseppeaccardo.datamining.exception***
  Questo package contiene tutte le eccezioni create ed utilizzate per gestire degli eventi anomali.
  Eccezioni sono spiegate nel dettaglio nella documentazione, ma in generale vengono utilizzate per gestire la divergenza del K-Means, la scelta giusta delle features (minimo due), l'esistenza e formattazione giusta del file.
  NB tutte le eccezioni hanno come super padre la classe **Exception**

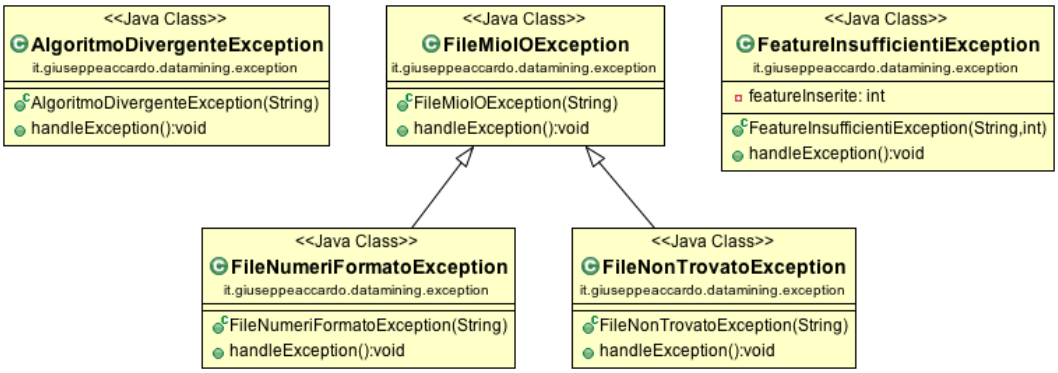


Figura 12: UML delle classi per gestire le eccezioni

- ***it.giuseppeaccardo.datamining.model***
  Questo package contiene il Model.
  Dataset è una classe astratta (CaricaDataset è un metodo astratto) utilizzata fondamentalmente per realizzare uno 'Scheletro' iniziale di un dataset. Il dataset costituisce un insieme di dati strutturati in forma relazionale, cioè corrisponde al contenuto di una singola tabella sottoforma di una matrice, dove le righe rappresentano i 'record' e le colonne le 'variabili' dette 'features'. Tale classe è posta in tal modo poichè si vuole rinviare la responsabilità di "come caricare un dataset reale" alla classe che estende dataset, utilizzando la stessa struttura di dataset. Il principio seguito è quello del design pattern ***Template Method*** che definisce lo scheletro di un algoritmo in un'operazione, rinviando alcuni passi alle sottoclassi client, cioè consente alle sottoclassi di ridefinire alcuni passi senza cambiare la struttura dell'algoritmo. Altri design pattern che potevano essere realizzati erano il "Factory Method" o "Decorator". Da notare che tale classe utilizza anche il design pattern ***Prototype***, mediante l'implementazione dell'interfaccia di 'Cloneable' (override di clone). Questo perchè quando si selezionano le features di un dataset, si ha la necessita di creare nuovi oggetti copiando il dataset non selezionato di partenza in tal modo da non perdere lo stato iniziale. Copiato il dataset, si selezioneranno solo le features richieste.
  Dataset Iris è l'implementazione concreta di Dataset, in cui il metodo astratto "caricaDataset" (Template Method) viene ridefinito.
  Cluster rappresenta il "grappolo" o gruppo, in cui sono agglomerati i punti del dataset "piu simili". Un cluster è rappresentato da un centroide e dall'indice del record della tabella. Da notare che il "Prototype" è utile anche in questo caso poichè si ha la necessità di clonare un cluster in un determinato stato.
  CalcolatoreCentroideThread è necessaria per l'esecuzione dei thread in parallelo. Tale implementa l'interfacca Runnable di cui si riscrive il metodo run(), necessario per il thread poichè saranno inserite le operazioni che tale dovrà ese-

guire. In questo caso, i thread in parallelo, riceveranno un cluster su cui effettaure il calcolo del centroide utilizzando la media aritmetica e tenendo conto di quanti e quali elementi sono inseriti per ogni cluster.
La classe **DataMining** rappresenta l'insieme di tecniche e metodologie che hanno per oggetto l'estrazione di un sapere o di una conoscenza a partire da grandi quantità di dati e l'utilizzo scientifico, industriale o operativo di questo sapere. Le fasi principali del sistema sviluppate sono: selezione delle caratteristiche, clustering e visualizzazione. La selezione delle caratteristiche (features) prevede quali e quante features vogliamo analizzare del dataset. Datamining può effettuare una scrittura da console utilizzando *PCAcolt*, classe utilizzata per effettuare l'Analisi delle Componenti principali.
Nella fase di clustering viene usato un algoritmo per agglomerare dati simile, ossia il *K-Means* che prenderà in input il numero di cluster K su cui operare. La visualizzazione di tale operato è eseguita dalle classi presenti nel package "Visualizzazione", anche se tale classe può effettuare una visualizzazione semplice da console, e la memorizzazione dello stato dei cluster precedenti avviene mediante il design pattern ***Memento***, implementato nel package *it.giuseppeaccardo.datamining.model.ricordo*.

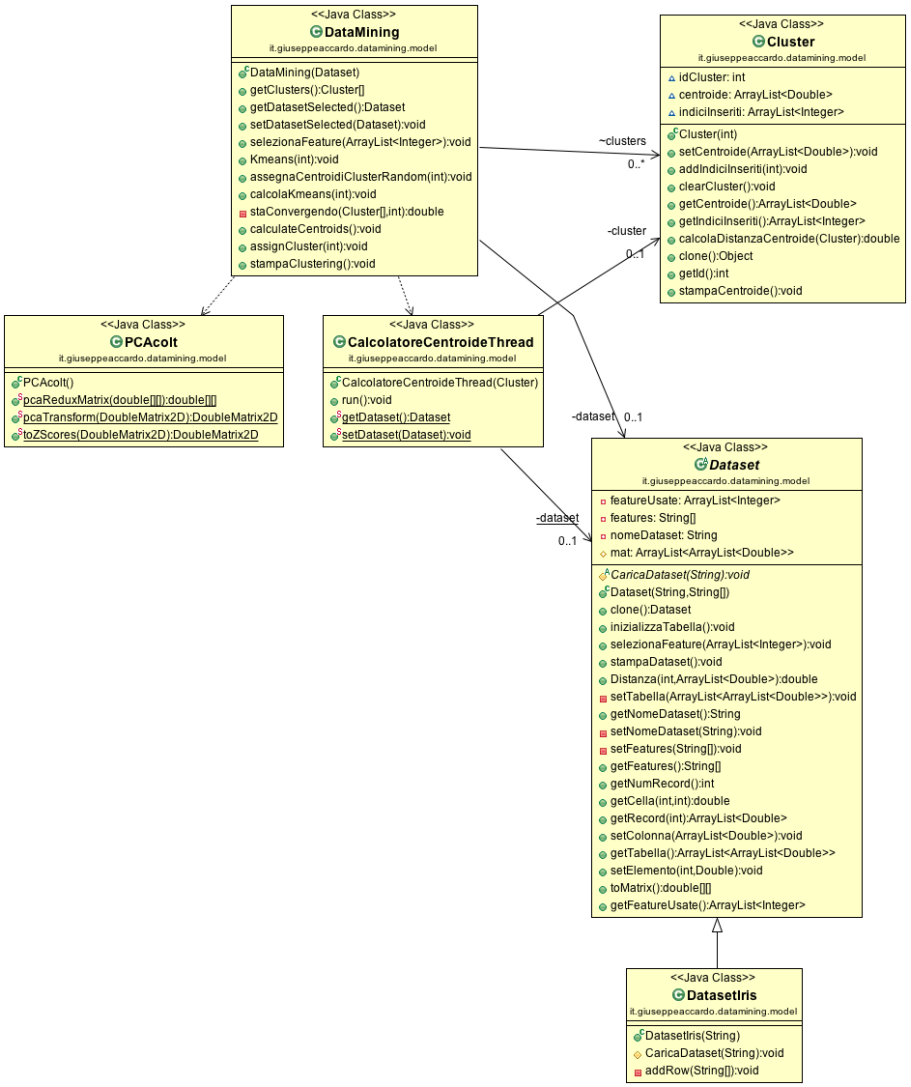


Figura 13: UML delle classi appartenenti al Model

- ***it.giuseppeaccardo.datamining.model.ricordo***
  In questo package sono presenti le componenti fondamentale per l'utilizzo del design pattern ***Memento***.

  Memento rappresenta l'interfaccia che il **ConcreteMemento** debba implementare per salvare un memento, cioè per immagazzinare lo stato generato da **Originator**. Quest'ultimo rappresenta l'istanza grazie alla quale viene generato uno stato fisico da salvare in Memento.
  Le istanze di ConcreteMemento saranno conservate in **Caretaker**.

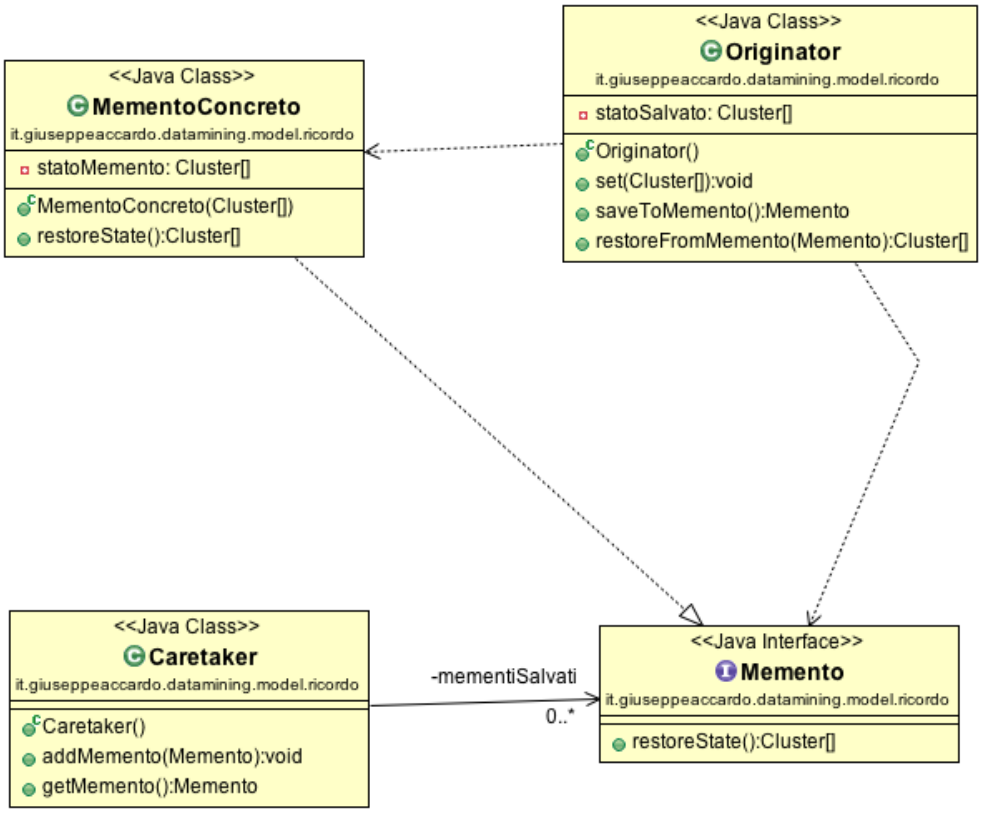


Figura 14: UML delle classi appartenenti all'implementazione di memento

- ***it.giuseppeaccardo.datamining.visualizzazione***
  Nel seguente package sono contenute le classi necessarie utilizzate da *DataminingController* per creare le View che rappresenteranno i grafici 2D/3D della clusterizzazione.
  I Grafici 2D sono stati realizzati mediante le classi *Chart* offerte da JavaFX, mentre i grafici 3D sono stati realizzati utilizzando il motore grafico tridimensionale fornito sempre da JavaFX, mediante la classe *Shape3D*.
  Per craere un'intercambiabilità di grafici, ossia la necessita di modificare dinamicamente gli algoritmi, è stato applicato il design pattern ***Strategy***.
  Questa realizzazione avviene mediante l'interfaccia funzionale **IVisualizzaGraficoStrategy** che viene implementata sia da **grafico2D** che da **grafico3D**. In modo tale, utilizzando come reference tale interfaccia, il metodo *visualizzaGrafico* sarà 'virtuale', ossia a run time assumerà l'implementazione dell'oggetto corrispondente istanziato. Ad esempio, istanziando grafico2D, il metodo che sarà realmente invocato a run time sarà quello appartenente a grafico2D.

Da notare che grafico3D utilizza la classe Axis per il disegno degli assi e PCAcolt per effettuare l'analisi delle componenti principali nel caso di features maggiori di 3.
Il package, oltre a contenere le classi, contiene anche i file *grafico2D* e *grafico3D* per la definizione dello stile dei grafici.

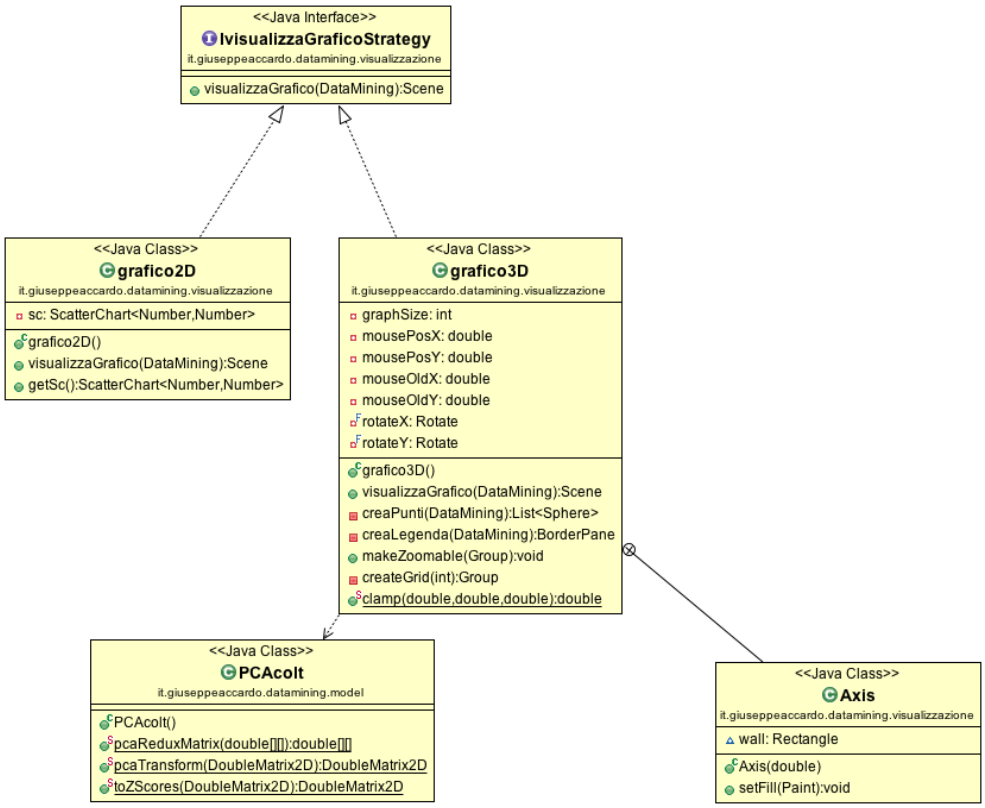


Figura 15: UML delle classi appartenenti all'implementazione dello Strategy

Design Pattern

Il Design Pattern è un concetto che puo‘ essere definito come una soluzione progettuale generale ad un problema ricorrente. 23 Design Patterns sono stati raccolti e formalizzati concettualmente nel 1994 nel libro Design Patterns: Elements of Reusable Object-Oriented Software scritto da 4 autori successivamente noti come **Gang of Four (GoF)**.
l'utilizzo dei Pattern Designs comporta numerosi benefici:

- Forniscono un approccio industriale standard per risolvere un problema ricorrente consentendo di risparmiare tempo.
- Agevolano il riutilizzo di codice robusto e facile manutenere.
- L'utilizzo dei Design Patterns rende il codice facile da capire e da debuggare agli sviluppatori permettendo a nuovi membri del team di sviluppo di capire il codice piu‘ velocemente.

Java Design Patterns si suddividono in 3 categorie: **creational, structural and behavioral**. Inoltre è stato applicato il design pattern architetturale MVC, già discusso precedentemente.
Nel progetto *sono stati applicati 5 di questi pattern, più uno architetturale (MVC).*

Template Method

Scopo: Definire lo scheletro di un algoritmo in un'operazione, rinviando alcuni passi alle sottoclassi client. Consente alle sottoclassi di ridefinire alcuni passi senza cambiare la struttura dell'algoritmo.
Motivazione: necessita di modificare dinamicamente gli algoritmi utilizzati da un'applicazione e.g., visite in una struttura ad albero, possibilita di selezionare a tempo di esecuzione una tra le visite
Applicabilità: quando si vuole implementare la parte invariante di un algoritmo una volta sola e lasciare che le sottoclassi implementino il comportamento che puo variare. Quando il comportamento comune di piu classi puo essere fattorizzato in una classe a parte per evitare di scrivere piu volte lo stesso codice. Per avere modo di controllare come le sottoclassi ereditano dalla superclasse, facendo in modo che i metodi template chiamino dei metodi "gancio" (hook) e impostarli come unici metodi sovrascrivibili.

Caso d'uso: La necessità di applicare tale pattern nasce dal fatto che si *vuole creare uno "scheletro" o uno schema di una classe (framework) che è il 'Dataset' e il dataset concreto, che in questo caso è 'DatasetIris' rappresenterà la sua implementazione concreta (ogni dataset concreto avrà il proprio caricamento).*
Dataset è posta in tal modo poichè si vuole rinviare la responsabilità di "come caricare un dataset reale" alla classe che estende dataset, utilizzando la stessa struttura di dataset. Altri design pattern che potevano essere applicati erano il "Factory Method" o "Decorator", ma si è optato per questo perchè il codice che ne usciva fuori era molto più pulito, semplice e rispettava i principi solid (ad esempio in decorator avremo violato il DIP).

```
// Scheletro o Template
public abstract class Dataset implements Cloneable {
    ...
```

```
    protected abstract void CaricaDataset(String path);
}
//Concreto
public class DatasetIris extends Dataset{
    protected void CaricaDataset(String pathFile){
        ...
    }
        ...
}
```

Il vantaggio principale di questo pattern è che garantisce una facile estensione e riuso del codice per realizzare altri eventuali dataset concreti.

Prototype

Scopo: Specifica il tipo di oggetti da creare usando un'istanza prototipale e creando nuovi oggetti copiando questi oggetti
Motivazione: Permette ad un oggetto di creare oggetti senza conoscere la loro classe o i dettagli per crearli
Applicabilità: sistema indipendente da come i suoi prodotti sono creati, composti e rappresentati. Le classi da istanziare sono specificate a run-time per evitare di scrivere una gerarchia di classi. E' piu conveniente copiare un'istanza esistente che crearne una nuova.

Caso d'uso:
Il "Prototype" è stato utilizzato per diversi compiti ed è stato usato mediante l'implementazione dell'interfaccia di ***'Cloneable'***.
Ad esempio è presente in Dataset poichè viene utilizzato (invocato) dal Datamining quando si selezionano le features di un dataset poichè si ha la necessita di creare nuovi oggetti copiando il dataset non selezionato di partenza in tal modo da non perdere lo stato iniziale. E' stato utilizzato anche per imagizzinare lo stato di un cluster.

```
// Implementazione di Cloneable
public abstract class Dataset implements Cloneable {
...
    @Override
        public Dataset clone(){
        try{
                /* Shallow copy --> Copia tutto ciò che resterà immutato ->reference
                   Deep copy --> Crea le istanze che cambieranno->Valore */
                /* Crea un clone del dataset. Da notare che si necessita di effettuare anche un
                     clone di mat che è una collection e
                 * ciò lo renderà "Deep", necessario per la seleziona delle features senza
                     modificare il dataset iniziale  */
                Dataset datasetClone = ((Dataset) super.clone());
                datasetClone.setTabella( ((ArrayList<ArrayList<Double>>) datasetClone.getTabella()
                    .clone()) );
                return datasetClone;
        }
        catch(CloneNotSupportedException e) {
                System.out.println(e.getMessage());
                e.printStackTrace();
                return null;
        }
        ...
}
...
// Applicazione di datamining
/* Clona il DATASET di PARTENZA che dovrà essere 'immune' alla selezione */
Dataset datasetSel = (Dataset) dataset.clone();
/* Setta nuovo dataset clonato e di questo effettua la selezione delle feature richieste */
this.setDatasetSelected(datasetSel);
```

Si noti che la clonazione definita è di tipo **"Deep Copy"** perchè clone di default effettua una **"Shallow Copy"**, ossia una copia del reference di oggetti o collection presenti.

Strategy

Scopo: Definire una famiglia di algoritmi, incapsularli e renderli intercambiabili. L'algoritmo cambia indipendentemente dai client che lo usano.
Motivazione: necessita di modificare dinamicamente gli algoritmi utilizzati da un'applicazione.
Applicabilità: lassi correlate differiscono solo per il loro comportamento. Abbiamo bisogno di varianti di un algoritmo. Un algoritmo usa dati che i client non dovrebbero conoscere. Una classe definisce diversi comportamenti

Caso d'uso:
Lo "Strategy" è stato utilizzato in DataminingController per la visualizzazione di un corrispondente grafico 2D o 3D in base a delle determinate circostanze: il numero delle features selezionate.
Per *creare un'intercambiabilità di grafici 2D e 3D*, ossia la necessita di modificare dinamicamente gli algoritmi, è stato applicato tale design pattern.
L'UML è chiaramente disponibile in *it.giuseppeaccardo.datamining.visualizzazione*.

```
// interfaccia per strategy
@FunctionalInterface
public interface IvisualizzaGraficoStrategy {
        public Scene visualizzaGrafico(DataMining model);
}
...
// Grafici che implementano strategy
public class grafico2D implements IvisualizzaGraficoStrategy {
    ...
    public Scene visualizzaGrafico(DataMining model){
        ...
    }

}
...
public class grafico2D implements IvisualizzaGraficoStrategy {
    ...
    public Scene visualizzaGrafico(DataMining model){
        ...
    }
}
...
//Applicazione di DataminingController
IvisualizzaGraficoStrategy grafico;
/* La scena conterrà il grafico corrispondente */
Scene sceneGrafico;
...
/* Istanziamo il grafico 2D o 3D, in base alle features utilizzate */
if(model.getDatasetSelected().getFeatureUsate().size()<3)
    grafico = new grafico2D();
else
        grafico = new grafico3D();

/* Con questo metodo otteniamo direttamente la scena corrispondente senza che sia il programmatore
     a doverlo riprogrammare */
sceneGrafico = grafico.visualizzaGrafico(model);
...
```

In MVC è molto frequente l'utilizzo dello Strategy, per gestire la mappatura tra algoritmi che regolano il processo tra Controller e View.

Memento

Scopo: L'intento di questo modello e di catturare lo stato interno di un oggetto senza violare incapsulamento e fornendo cosi un mezzo per ripristinare l'oggetto allo stato iniziale quando necessario.
Motivazione:A volte si rende necessaria l'acquisizione dello stato interno di un oggetto in un certo istante e ripristinare successivamente l'oggetto a quello stato. Il

processo e utile in caso di errori, e.g., calcolatrice che mantiene la lista delle operazioni precedenti
Applicabilità: viene utilizzato quando uno stato di un oggetto deve essere catturato in modo che possa essere ripristinato a quello stato piu tardi. In situazioni in cui il passaggio in modo esplicito dello stato dell'oggetto violerebbe incapsulamento.

Caso d'uso:
Il design pattern "Memento" è stato utilizzato in Datamining durante la fase di clustering in cui viene usato un algoritmo per agglomerare dati simile, ossia il K-Means che prenderà in input il numero di cluster K su cui operare. In questo caso si ha la *necessità di dover salvare lo stato precedente dei cluster* per poterli confrontare successivamente con lo stato successivo e verificarne l'eventuale convergenza.

Memento rappresenta l'interfaccia che il ***ConcreteMemento*** debba implementare per salvare un memento, cioè per immagazzinare lo stato generato da ***Originator***. Quest'ultimo rappresenta l'istanza grazie alla quale viene generato uno stato fisico da salvare in Memento. Le istanze di ConcreteMemento saranno conservate in ***Caretaker***.
Nel package *it.giuseppeaccardo.datamining.model.ricordo* l'uml composto dalle componenti fondamentale per l'utilizzo del design pattern Memento.

```
// interfaccia per Memento
public interface Memento{
        public Cluster[] restoreState();
}
// Memento concreto
public class MementoConcreto implements Memento {
        /** Stato del memento **/
        private Cluster[] statoMemento;
        /**
         * Registra uno stato di originator
         * @param statoDaSalvare stato che originator vuole salvare im memento
         */
   public MementoConcreto(Cluster[] statoDaSalvare) {
                this.statoMemento = statoDaSalvare;
        }
        /** Ripristina uno stato di originator
         * @return  statoMemento ritorna lo stato di memento*/
   public Cluster[] restoreState() {
        return statoMemento;
        }
}
// Originator
public class Originator {
           /** Stato transitorio salvato **/
           private Cluster[] statoSalvato;
           /*Setta lo stato da salvare*/
           public void set(Cluster[] statoDaSalvare) {
                        statoSalvato = new Cluster[statoDaSalvare.length];
                        for(int i = 0;i<statoDaSalvare.length; i++)
                                statoSalvato[i] = (Cluster) statoDaSalvare[i].clone();
           }
           /**crea e restituisce il memento con lo stato salvato
            * @return MementoConcreto restituisce il memento concreto contenento lo stato salvato
                 in originator**/
           public Memento saveToMemento() {
               return new MementoConcreto(this.statoSalvato);
           }
           /**Ripristina uno stato da memento
            * @param m memento da cui ripristinare lo stato
            * @return statoSalvato ritorna lo stato ripristinato da memento**/
           public Cluster[] restoreFromMemento(Memento m) {
                        statoSalvato = m.restoreState();
                  return statoSalvato;
           }
        }
// Caretaker
public class Caretaker {
         /** Lista dei memento salvati. La lista è sottoforma di Coda **/
```

```
            private Queue<Memento> mementiSalvati = new LinkedList<Memento>(); //queue è
                interfaccia
            /** Aggiungi un memento nella Coda
            * @param m memento da salvare**/
            public void addMemento(Memento m) {
                    this.mementiSalvati.add(m);
            }
            /** Estrai memento dalla coda
                  * @return memento**/
            public Memento getMemento() {
                    return this.mementiSalvati.poll();
            }
}
...
...
// Salvataggio di un memento
/* Origina stato e custodiscilo nel memento */
originator.set(clusters);
caretaker.addMemento(originator.saveToMemento() );
// Ripristino di un memento
/*Aggiorna lo stato originator e restituisce lo stato precedente*/
Cluster[] oldClusters = originator.restoreFromMemento( caretaker.getMemento());
```

Observer

Scopo: Definisce una dipendenza una a molti tra oggetti, tale che se un oggetto cambia stato, tutte le sue dipendenze sono notificate e aggiornate automaticamente
Motivazione: Un effetto del partizionamento di un sistema in una collezione di classi cooperanti e la necessita di mantenere le consistenze tra gli oggetti. Le classi non devono essere fortemente accoppiate perche riducono la loro riusabilita
Applicabilità: quando un'astrazione ha due aspetti che dipendono l'uno dall'altro. L'incapsulamento di questi aspetti in oggetti separati permette il loro riuso in modo indipendente. Quando il cambiamento di un oggetto richiede il cambiamento di altri e non si conoscono quanti oggetti hanno bisogno di essere cambiati. Si puo usare nell'ambito comunicazione numeri

Caso d'uso:
In JavaFX e in MVC, si utilizza molto spesso questo pattern poichè crea un legami e relazioni tra view-controller e view-model.// Come già detto, dalle View possono partire delle richieste verso il controller che risponderà, ma può richiedere anche semplicemente lo stato direttamente al model o ricevere da esso una notifica di cambiamento di stato. Questo legame (in ambito di JavaFX è detto **Binding**) è effettuato mediante il design pattern ***"Observer"*** per definire un forte dipendenza tra view e model o view e controller in modo tale che se una cambia, l'altra è aggiornata automaticamente. In JavaFX tale design pattern è implementato direttamente mediante la collection interface **'ObservableList<E>'** che appunto tiene traccia di tutti i dati e dei suoi aggiornamenti.
In certe situazioni, JavaFX **obbliga** l'utilizzo, ad esempio se si vuole creare una tabella che si relazione con il model oppure cliccando su un checkbox, si aggiorna automaticamente lo stato delle colonne delle tabelle per essere così evidenziate (vedi ad esempio nella classe DataminingController).

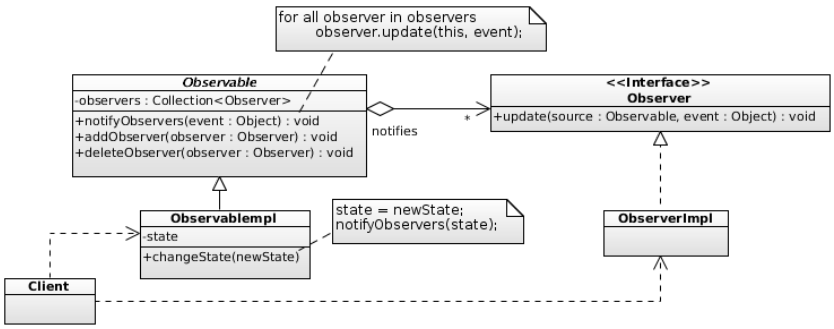


Figura 16: UML dell'observer che viene fatto uso direttamente dall'ObservableList

Documentazione -JavaDOC

La documentazione comprende i Class Diagram e i commenti inseriti durante tutte le fasi dello sviluppo, sono stati inseriti commenti per package, classi, metodi e altro, questi vengono trasformati in documentazione da javadoc che ‘e uno strumento che genera un insieme di pagine HTML tra loro collegate che rispecchiano la struttura del progetto. Si possono raccogliere informazioni navigando nella pagine della documentazione.

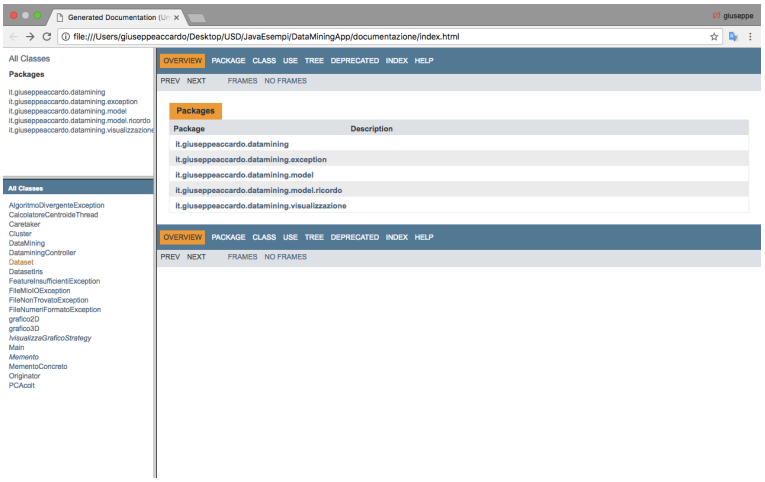


Figura 17: index

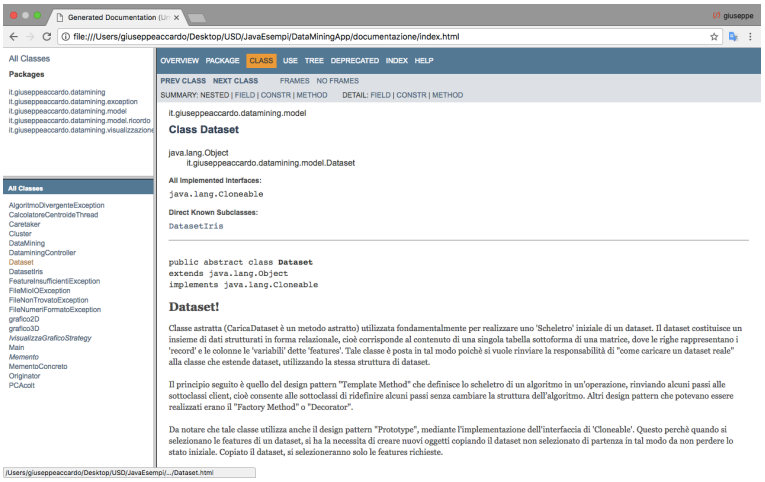


Figura 18: Ad esempio dataset

Funzionamento della GUI

Il deplyoment è il processo di impacchettamento e distribuzione verso gli utenti del software che abbiamo sviluppato. Si tratta di una parte cruciale del processo di sviluppo poichè costituisce il primo contatto che gli utenti avranno con il nostro software grazie alla quale.
L'output può essere sottoformato .exe per gli utenti windows o .dmg per gli utenti os. In generale l'impacchettamento .jar è usufruibile per qualunque JVM.

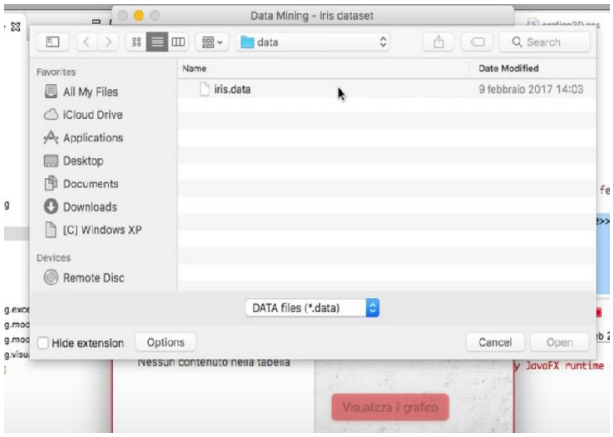
User-Experience

Al lancio dell'applicazione, viene visualizzato un menù realizzato interamente in **FXML** e **CSS**. La potenzialità di javaFX è di offrire un confortante e immediata interfaccia utente.

Figura 19: Inizio

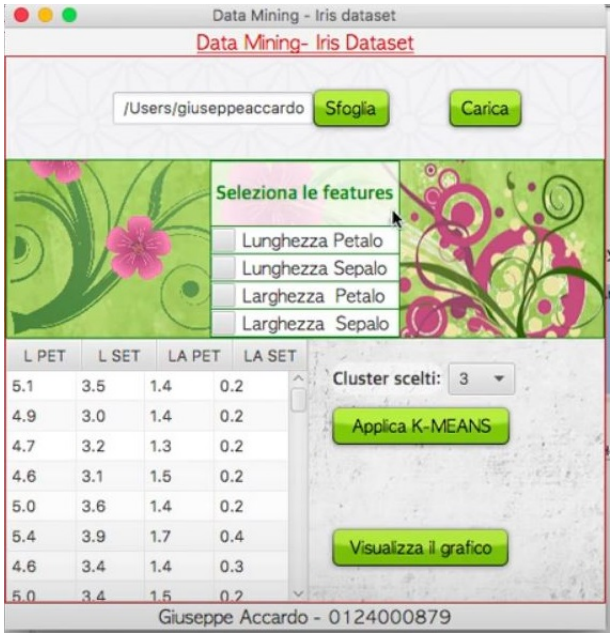
Serie di passaggi per effettuare il caricamento da file

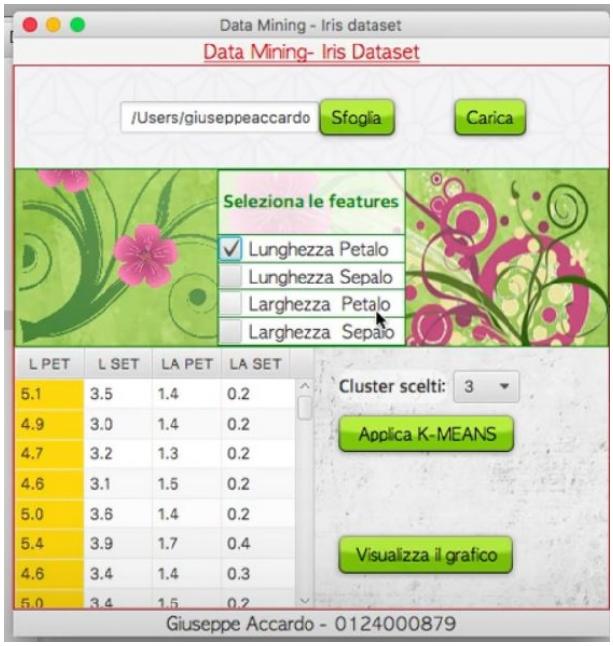

Data Mining - Iris dataset
Data Mining- Iris Dataset
Sfoglia
Carica
L PET
L SET
LA PET
LA SET
Cluster scelti:
Applica K-MEANS
Nessun contenuto nella tabella
Visualizza il grafico
Giuseppe Accardo - 0124000879

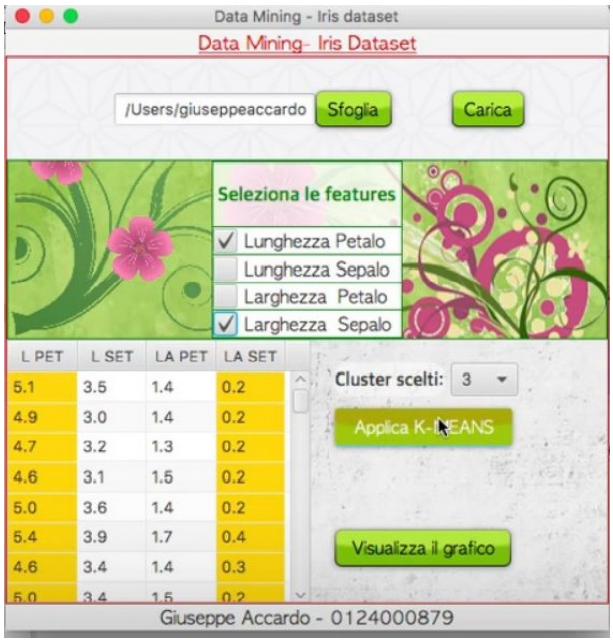

Data Mining - Iris dataset
data
Search
Favorites
All My Files
iCloud Drive
Applications
Desktop
Documents
Downloads
[C] Windows XP
Devices
Remote Disc
Name
Date Modified
iris.data
9 febbraio 2017 14:03
DATA files (*.data)
Hide extension
Options
Cancel
Open
Visualizza il grafico


Data Mining - Iris dataset
Data Mining- Iris Dataset
/Users/giuseppeaccardo
Sfoglia
Carica
L PET
L SET
LA PET
LA SET
Cluster scelti:
Applica K-MEANS
Nessun contenuto nella tabella
Visualizza il grafico
Giuseppe Accardo - 0124000879

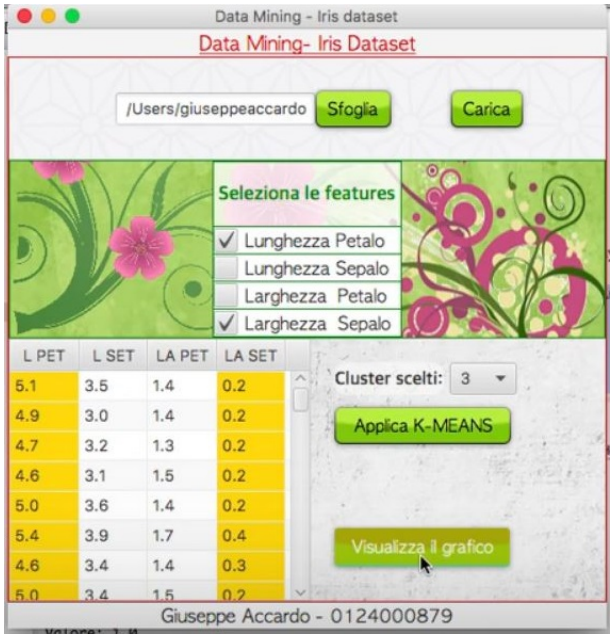
Il caricamento del file, se è tutto ok, produrrà il dataset nel formato della TableView e verranno visualizzate le possibili checkbox da selezionare. Per ogni checkbox selezionato, sarà evidenziata la corrispondente colonna (grazie all'ObservableList).
A questo punto vengono selezionate le Features su cui vogliamo applicare il datamining mediante K-Means, il numero di cluster K e si cliccherà su "Applica K-Means".







Se il datamining andrà a buon fine, verrà visualizzato un alert di successo. A questo punto può essere visualizzato il risultato andando a cliccare su "Visualizza Grafico". Il grafico che ci si aspetto sarà in 2D poichè si sono scelete solo 2 Features. Si osservi come lo strategy renda trasparente l'intercambiabilità tra il grafico 2D e 3D



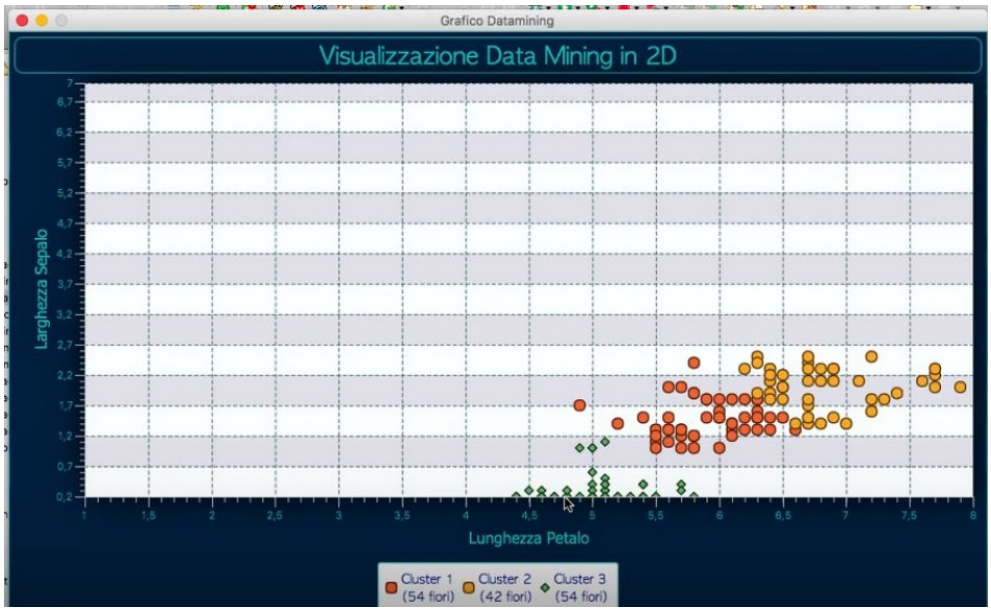
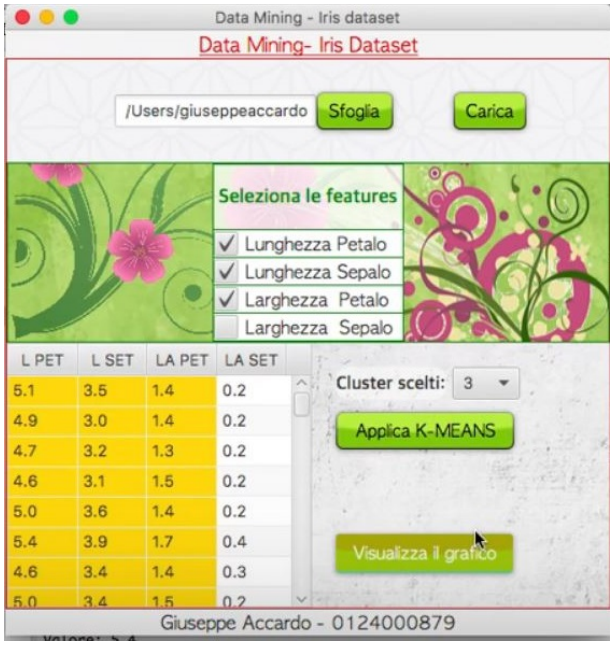
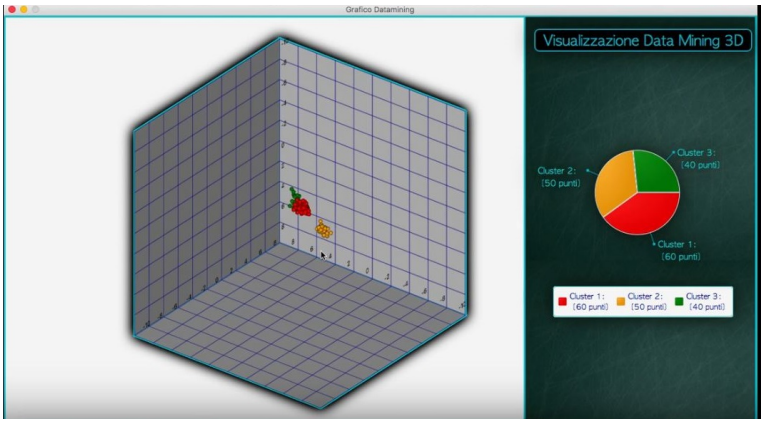
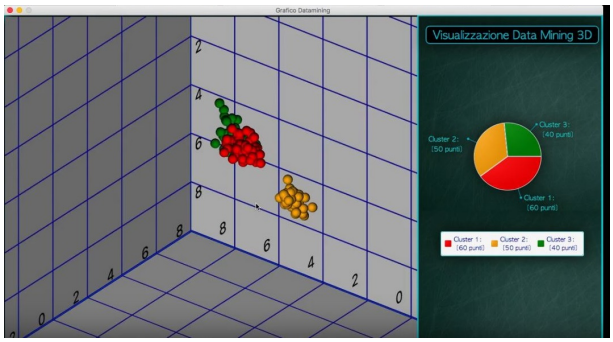


Figura 20: Grafico 2D: Osserviamo come i punti siano stati raggruppati, in base al colore, ai loro corrispondenti cluster. Questo visualizzato è un ottimo risultato.

Ora si effettua una prova selezionando 3 features per avere il rsultato finale in un grafico 3D (ci si aspetta il solito lavoro dallo strategy)







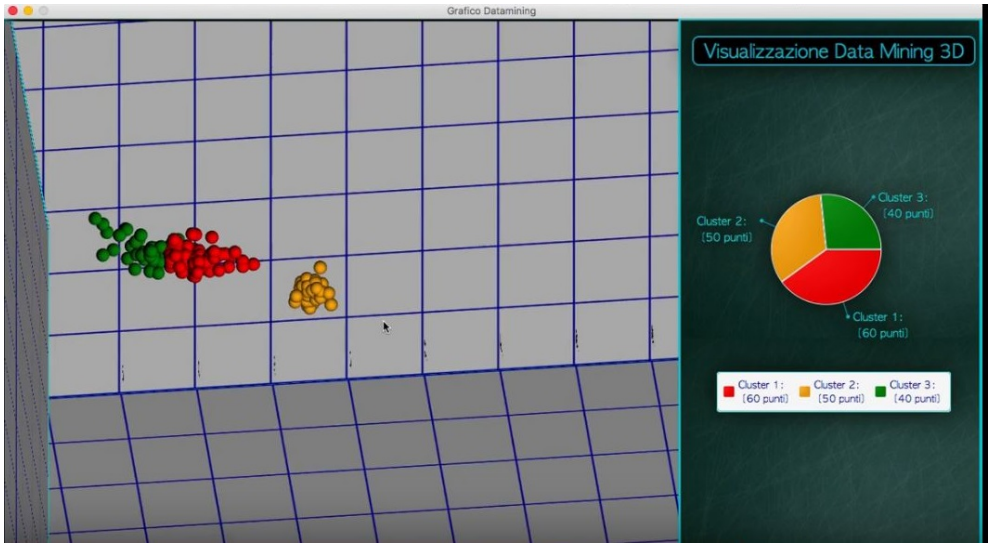


Figura 21: Grafico 3D: Osserviamo come i punti siano stati raggruppati, in base al colore, ai loro corrispondenti cluster. Anche questo visualizzato è un ottimo risultato.

Eccezioni: gestione anomalie

Osserviamo le eccezioni sviluppato per gestire alcuni eventi anomili.
Il k-means potrebbe generare un numero di cluster inferiore a quelli richiesti poichè potrebbero essere scelti dei centroidi "troppo vicini" che non riescono a creare il gruppo di cluster.

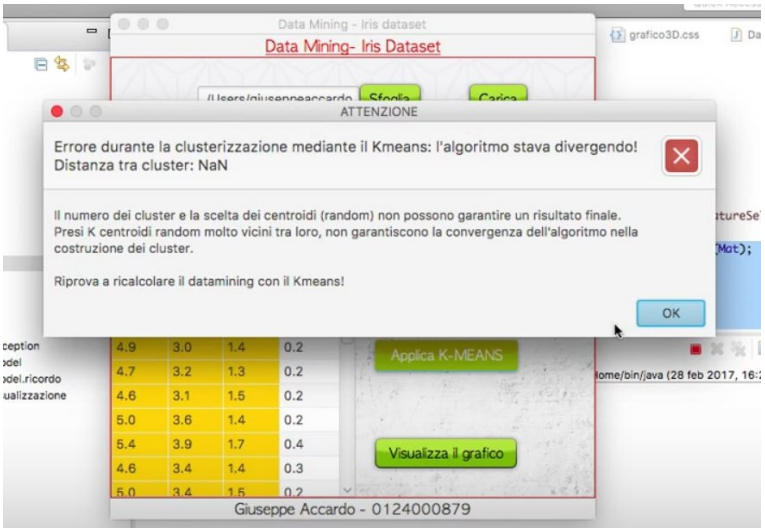


Figura 22: Errore del kmeans. L'eccezione lanciata è "AlgoritmoDivergenteException".

Il numero delle features devono essere almeno due, altrimenti il datamining non avrebbe senso.

Figura 23: Errore selezione features.L'eccezione lanciata è "FeatureInsufficientiException"

Caso in cui si hanno errori sulla reperibilità del file Il file non è formato da valori

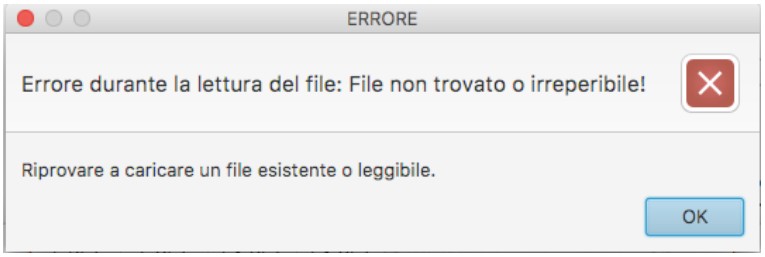


Figura 24: Errore apertura file.L'eccezione lanciata è "FileNonTrovatoException"

adeguati (solo numeri).

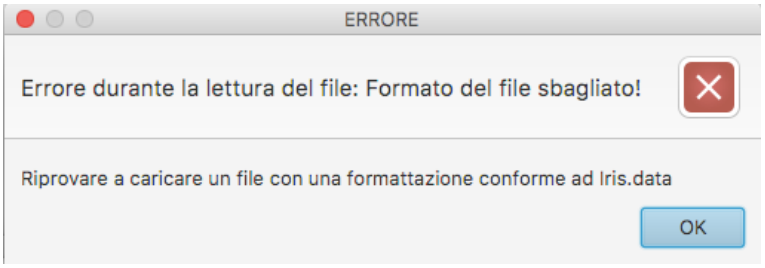


Figura 25: Errore formato interno file.L'eccezione lanciata è "FileNumeriFormatoException"

Bibliografia e strumenti di sviluppo

La seguente relazione è stata realizata in LaTeX utilizzando l'omonimo linguaggio ed è stato utilizzato SublimeText come editor poichè si è utilizzato un Macbook.

Fonte di studio
- Slide del corso di di programmazione 3
- Manuale di Java di Claudio De Sio
- Getting Started with JavaFX
- Object Oriented Design

Strumento hardware di sviluppo Macbook Air 2015.

Software usati per la progettazione di schemi ObjectAid UML Explorer.

IDEE e ambiente di sviluppo IDE: Eclipse Mars.1 plugin: e(fx)clipse.