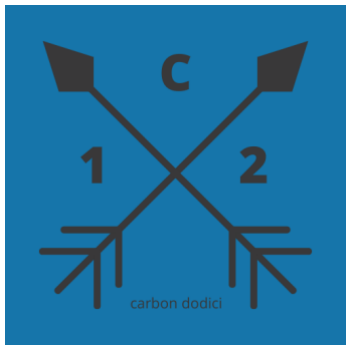


Carbon12 – Predire in Grafana

Manuale Sviluppatore

Informazioni sul documento

Versione	b.3.0
Stato	Approvato
Data di creazione	2020/03/24
Data di approvazione	2020/04/13
Redazione	Manuel De Franceschi Nicolò Fassina Francesco Gobbo Alessandro Lovo Veronica Pederiva
Verifica	Nicolò Fassina Andrea Longo
Approvazione	Francesco Gobbo
Uso	Esterno
Destinatari	Carbon12 Prof. Tullio Vardanega Prof. Riccardo Cardin
E-mail di riferimento	carbon.dodici@gmail.com

Scopo del documento

Manuale sviluppatore per la manutenzione e l'estensione del progetto Predire in Grafana realizzato dal team Carbon12.

Registro delle modifiche

Versione	Data	Descrizione	Nominativo	Ruolo
b.3.0-0	2020/04/13	Approvazione del documento	Francesco Gobbo	Responsabile
b.2.1-0	2020/04/12	Validazione del documento	Nicolò Fassina	Verificatore
a.2.0-9	2020/04/07	Verifica del documento	Andrea Longo	Verificatore
	2020/04/05	Redazione capitolo 8 – Test	Francesco Gobbo	Progettista
a.2.0-8	2020/04/04	Redazione sezione §7.1 – Aggiunta algoritmo ML	Veronica Pederiva	Progettista
a.2.0-7	2020/04/03	Redazione capitolo 6 – Architettura plug-in	Manuel De Franceschi	Progettista
a.2.0-6	2020/04/02	Redazione capitolo 5 – Architettura addestramento	Alessandro Lovo	Progettista
a.2.0-5	2020/04/01	Verifica del documento	Nicolò Fassina	Verificatore
	2020/03/30	Redazione capitolo 4 – Installazione	Veronica Pederiva	Progettista
a.2.0-4	2020/03/30	Redazione Capitolo 3 – Prerequisiti	Alessandro Lovo	Amministratore
a.2.0-3	2020/03/29	Redazione Capitolo 2 – Tecnologie interessate	Alessandro Lovo	Amministratore
a.2.0-2	2020/03/28	Redazione Capitolo 1 - Introduzione	Veronica Pederiva	Progettista
a.2.0-1	2020/03/24	Creazione del documento e definizione struttura	Nicolò Fassina	Amministratore

Indice

1 Introduzione

1.1 Scopo del documento

Il presente documento ha lo scopo di guidare gli sviluppatori nell'attività di manutenzione ed estensione del prodotto. Sia per il plug-in che per il programma di addestramento verranno descritte le tecnologie utilizzate, l'architettura e le scelte progettuali, alcuni suggerimenti utili all'estensione del programma e le modalità con cui sono stati condotti i test.

1.2 Scopo del prodotto

Lo scopo del prodotto è realizzare un plug-in per lo strumento Grafana che monitori, tramite un'analisi predittiva, un flusso di dati. Il plug-in deve essere accompagnato da un programma per l'addestramento degli algoritmi Support-Vector Machine[G] e Regressione Lineare[G] in grado di accettare un file CSV[G] con i dati di addestramento e creare un file in formato JSON[G] contenente i parametri per le previsioni. Il plug-in deve essere in grado di leggere dal file in formato JSON la definizione dell'algoritmo di previsione da applicare al flusso di dati ed eseguire i calcoli previsti, producendo dei valori che devono essere visualizzati in una dashboard contenente i grafici di previsione prodotti dal sistema di creazione di grafici di Grafana.

1.3 Glossario

Per evitare il presentarsi di ambiguità legati a termini tecnici o acronimi, si è deciso di riportare in appendice al presente documento un *Glossario* nel quale vengono definiti tali termini. I vocaboli riportati nel *Glossario* riportano una G maiuscola a pedice.

1.4 Riferimenti

- Grafana
  https://grafana.com
- NodeJS
  https://nodejs.org/en/
- ESLint
  https://eslint.org/
- Angular
  https://angular.io/
- NPM
  https://www.npmjs.com/
- EJS
  https://ejs.co/
- ExpressJS
  https://expressjs.com
- Yarn
  https://yarnpkg.com
- Jest
  https://jestjs.io

2 Tecnologie interessate

Il prodotto si compone di due parti, una applicazione web per addestrare i predittorIG e un plug-in per Grafana; queste due parti utilizzano varie tecnologie, che saranno elencate nei successivi paragrafi.

2.1 Addestramento

Le seguenti tecnologie sono utilizzate nell'addestramento:

- **JavaScript**: il principale linguaggio con cui è scritta la parte logica delle pagine;
- **HTML/CSS**: utilizzati per gestire la visualizzazione delle pagine web;
- **NodeJS**: è una tecnologia runtime Javascript costruita sul motore V8 di Chrome;
- **Yarn**: Utilizzato per gestire i pacchetti di NodeJS;
- **ExpressJS**: è un pacchetto NodeJS che consiste in un framework che aiuta ad organizzare le applicazioni web seguendo il modello MVC;
- **ESlint**: è uno strumento per l'analisi statica del codice per identificare eventuali problemi;
- **EJS**: linguaggio di template che permette di generare markup HTML con JavaScript.
- **Jest**: pacchetto NodeJS utilizzato per effettuare i test;
- **SVM/RL**: librerie utilizzate per generare i predittori allenati per il plug-in;

2.2 Plug-in

Le seguenti tecnologie sono utilizzate nel plug-in:

- **JavaScript**: il principale linguaggio con cui è scritto il plug-in;
- **AngularJS**: framework usato da Grafana per gestire la visualizzazione e l'interazione con le varie pagine dei plug-in;
- **HTML/CSS**: utilizzati insieme ad AngularJS per la gestione delle pagine web del plug-in;
- **NodeJS**: è una tecnologia runtime javascript costruita sul motore V8 di Chrome;
- **Yarn**: Utilizzato per gestire i pacchetti di NodeJS;
- **ESlint**: è uno strumento per l'analisi statica del codice per identificare eventuali problemi;
- **Grafana/toolkit**: pacchetto per NodeJS utilizzato per effettuare le build ed i test del plug-in;
- **Grafana**: software open-source per il monitoraggio e la visualizzazione di dati mediante grafici ed indicatori;
- **SVM/RL**: librerie utilizzate per la previsione dei dati.

3 Prerequisiti

3.1 Addestramento

3.1.1 NodeJS

NodeJS è necessario per avviare il server web che servirà poi le pagine web all'utente. Il Team ha utilizzato NodeJS in versione 12.

3.1.2 Yarn

Yarn è necessario per scaricare le dipendenze necessarie all'esecuzione del software, che sono specificate nel file package.json. L'installazione deve essere effettuata utilizzando il comando: `yarn install.`

3.1.3 Browser

Il software richiede un browser compatibile con Javascript. Il software è stato sviluppato con Chrome versione 80 e Safari versione 13.

3.2 Plug-in

3.2.1 Yarn

Yarn è necessario per scaricare le dipendenze necessarie e per effettuare la build del software. Le dipendenze sono specificate nel file package.json, ed è necessario installarle mediante il comando `yarn install` prima di poter effettuare la build del plug-in.

Nel file package.json sono specificati anche gli script necessari ad eseguire la build, che possono essere chiamati con i comandi: `yarn build` per effettuare la build in modalità di produzione, `yarn dev` per quella in modalità sviluppatore.

Yarn è inoltre necessario a svolgere i test, con il comando `yarn test.`

3.2.2 Grafana

Per utilizzare il plug-in è necessario il software Grafana in versione 6.5.3. Versioni superiori a questa potrebbero esibire malfunzionamenti nella UI.

3.2.3 InfluxDB

Influx è il database con cui si interfaccia il plug-in.

3.2.4 Telegraf

Telegraf uno strumento di monitoraggio si interfaccia con InfluxDB per inserire i dati.

4 Installazione

Il prodotto è disponibile all'indirizzo

https://github.com/carbondodici/Predire_in_Grafana_Product

dal quale è possibile effettuare il download del plug-in e del programma di addestramento. Le due componenti sono rese disponibili in due cartelle separate.

4.1 Installazione applicazione di addestramento

Spostare la cartella addestramento nella posizione desiderata sul proprio dispositivo in modo che sia accessibile agli utenti che devono farne uso.
Da terminale, spostarsi nella directory addestramento e utilizzare il seguente comando per installare le dipendenze:

```
yarn install
```

oppure:

```
npm install
```

4.2 Installazione plug-in

Spostare la cartella `predire-in-grafana-app` nella cartella `grafana/data/plugins`, successivamente, da terminale spostarsi nella directory del plug-in, installare le dipendenze e creare la build utilizzando i comandi seguenti:

```
yarn install
yarn build
```

oppure:

```
npm install
npm build
```

5 Architettura addestramento

5.1 Visione generale

Il programma di addestramento è costituito da un'applicazione web che si basa sulla tecnologia Node.js e sul framework Express. Viene utilizzato poi il linguaggio di markup HTML in accoppiata con il template engine EJS per la realizzazione delle viste con cui l'utente interagisce. Per la sua realizzazione è stata adottata un'architettura MVC pull model e la progettazione è stata fatta nel rispetto dei solid principle per offrire possibilità di estensione dell'applicazione.
Il server funge da raccordo e gestisce l'integrazione tra i pacchetti npm che soddisfano le dipendenze del programma, le classi che consentono l'input e l'output dei file e i modelli di machine learning, che costituiscono il model.

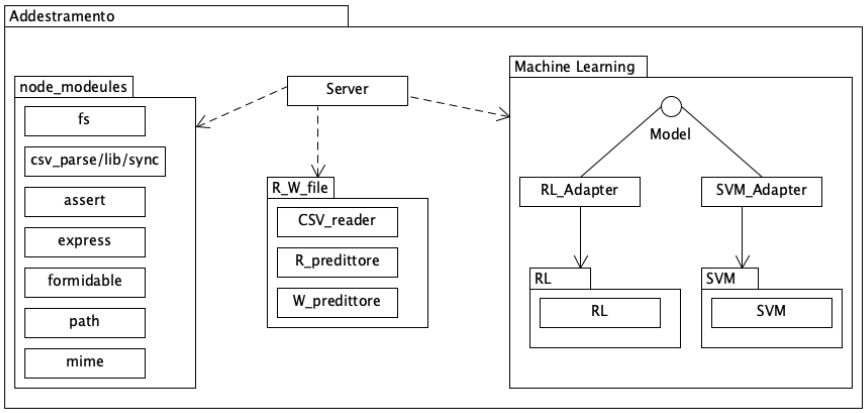


FIGURA 1 - DIAGRAMMA DEI PACKAGE ADDESTRAMENTO

Le classi che consentono l'input e l'output dei file, permettendo di compiere operazioni su di essi e di verificarne la validità di struttura e contenuti sono: Csv_reader, R_Predittore e W_Predittore.
Csv_reader si occupa di leggere i dati di addestramento contenuti in file CSV implementando un'interfaccia che indica i metodi attraverso cui il server accede alle informazioni di cui necessita.
La gestione del predittore, in formato JSON, è invece affidata alle classi R_Predittore per la sua lettura e W_Predittore per la scrittura del predittore con i risultati dell'allenamento.

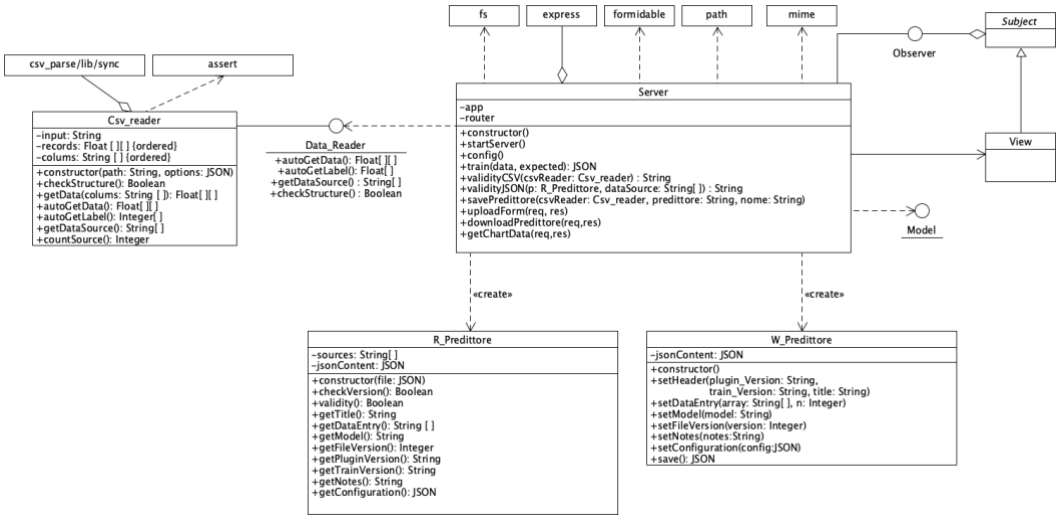


FIGURA 2 - DIAGRAMMA DELLE CLASSI ADDESTRAMENTO – GESTIONE FILE

Per la gestione delle librerie esterne che implementano gli algoritmi per RL e SVM si è utilizzato il design pattern object adapter. Sono stati raccolti quindi in un'unica interfaccia i soli metodi necessari al server per leggere i risultati dell'allenamento e per configurare i modelli con la configurazione ottenuta da un eventuale predittore in ingresso nel caso si tratti di un aggiornamento. Tale interfaccia è opportunamente implementata da due classi adapter: RL_Adapter e SVM_Adapter.

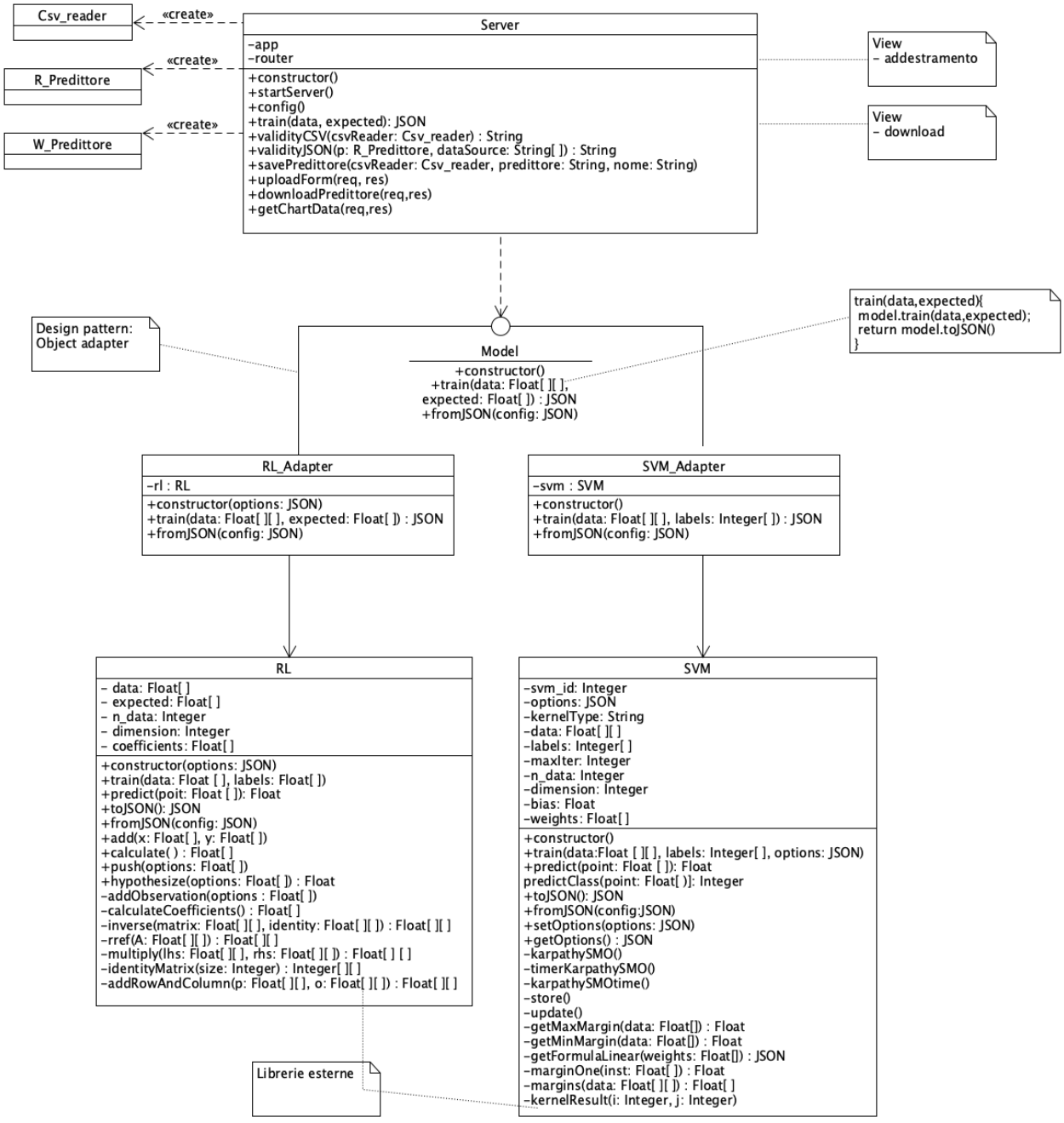


FIGURA 3 - DIAGRAMMA DELLE CLASSI ADDESTRAMENTO - MODELLI MACHINE LEARNING

5.2 Csv_reader

Dopo aver selezionato il file CSV e completato il form, l'utente premendo un pulsante nella View avvia il processo di addestramento. In particolare, viene chiamato il metodo uploadForm nel Server che istanzia un oggetto di tipo Csv_Reader passandogli il path del file CSV. Il metodo interno al server ValidityCsv effettua i controlli sulla validità del contenuto e della struttura del file. Se non vengono restituiti errori vengono chiamati i metodi autoGetData e autoGetLabel dell'oggetto Csv_Reader che ritornano rispettivamente una matrice contenente i dati e un vettore contenente le labels. Infine, viene chiamato getDataSource per ottenere i nomi delle sorgenti di dati.

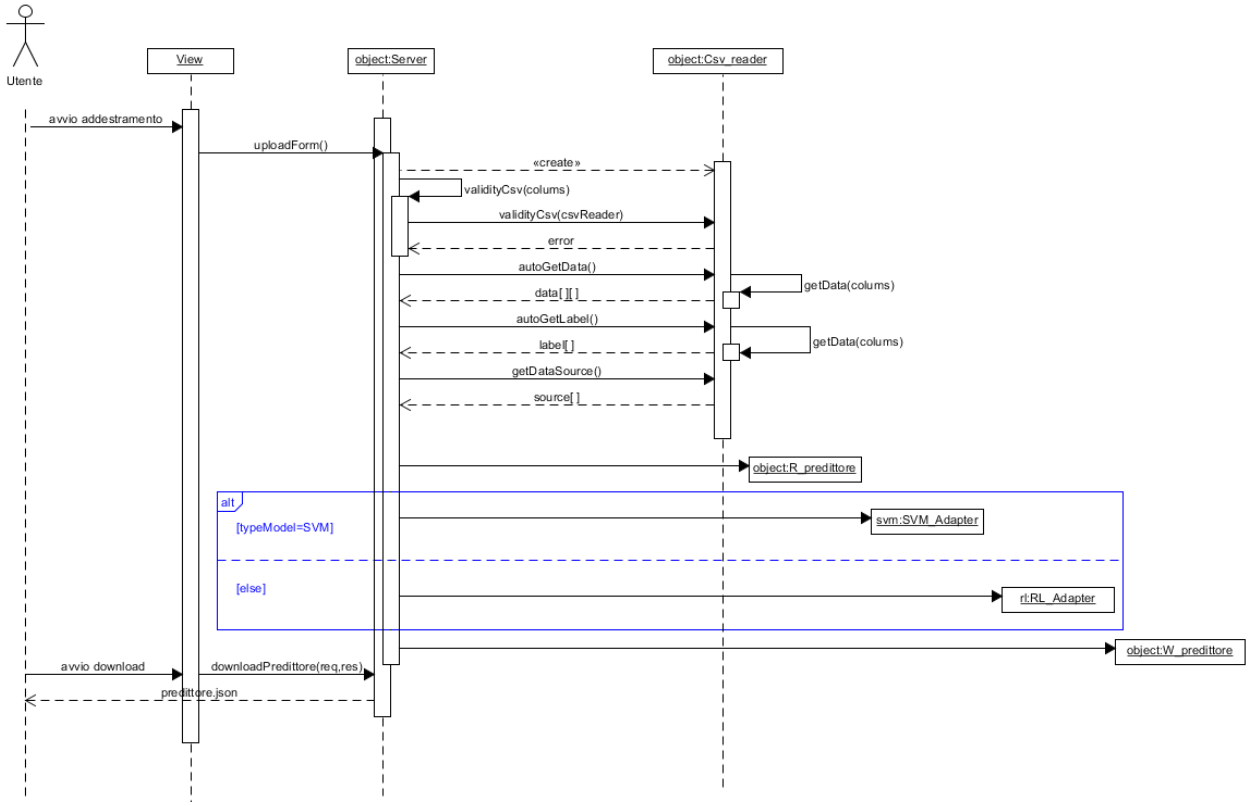


FIGURA 4 - DIAGRAMMA DI SEQUENZA ADDESTRAMENTO - CSV_READER

5.3 R_predittore

Il metodo uploadForm procede poi verificando anche il file JSON del predittore se l'utente ne ha selezionato uno. In questo caso viene creato l'oggetto R_Predittore passandogli il contenuto del JSON e viene chiamato il metodo ValidityJson interno al server che ne verifica la struttura, controlla se le versioni del plugin e del programma di addestramento sono compatibili con quelle in uso, se il modello coincide con quello scelto, e se le data entry coincidono con quelle del CSV. Se non vengono restituiti errori, il metodo getConfiguration di R_Predittore ritorna la configurazione che sarà utilizzata dai modelli di addestramento.

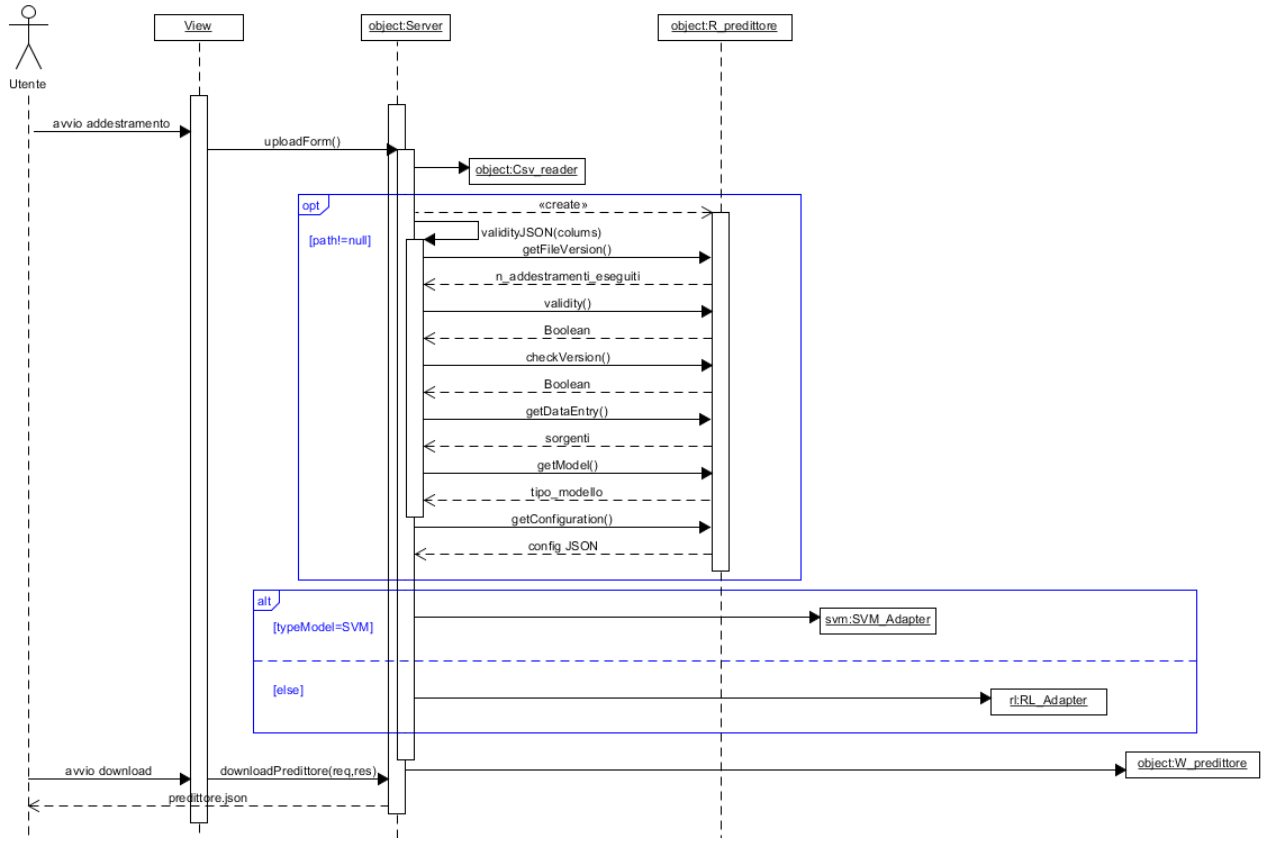


FIGURA 5 - DIAGRAMMA DI SEQUENZA ADDESTRAMENTO - R_PREDITTORE

5.3 SVM_Adapter e RL_Adapter

A seconda del modello selezionato dall'utente viene istanziato un oggetto SVM_Adapter o RL_Adapter. Se è stato selezionato un file JSON del predittore allora viene impostata la sua configurazione passandola all'adapter con il metodo fromJSON che richiama il corrispondente metodo fromJSON interno alle librerie esterne utilizzate. Viene poi lanciato l'addestramento vero e proprio con il metodo train che in modo analogo chiama il metodo train della libreria passandogli i dati e le labels. Terminata l'esecuzione viene ritornata nel server la configurazione da salvare nel JSON del predittore.

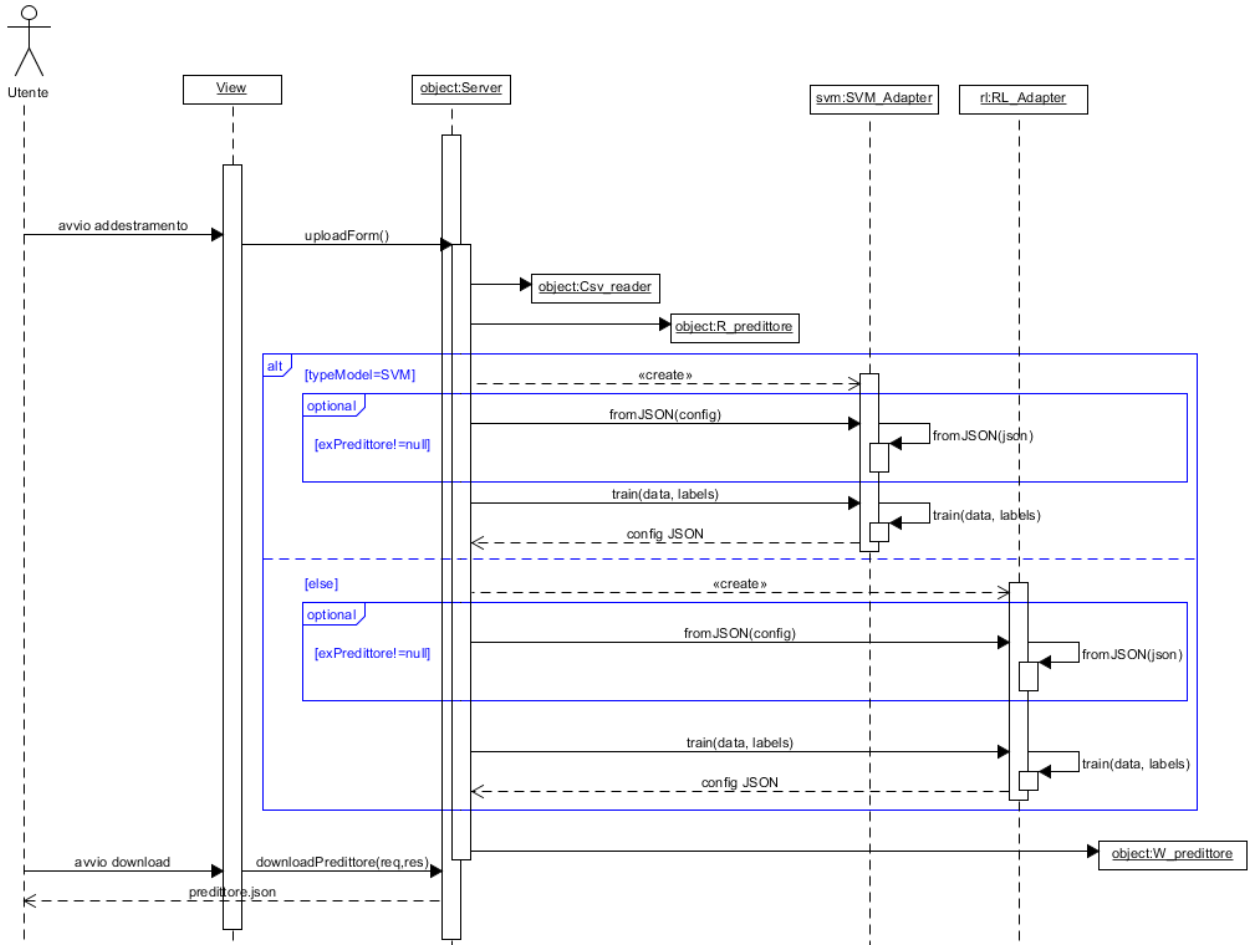


FIGURA 6 DIAGRAMMA DI SEQUENZA ADDESTRAMENTO - TRAIN

5.4 W_predittore

Terminata la procedura di addestramento, nel server viene creato l'oggetto W_Predittore. Su di esso vengono chiamati vari metodi per inserirvi le informazioni da salvare e infine il metodo save che ritorna il contenuto in formato stringa da scrivere sul file sul disco.

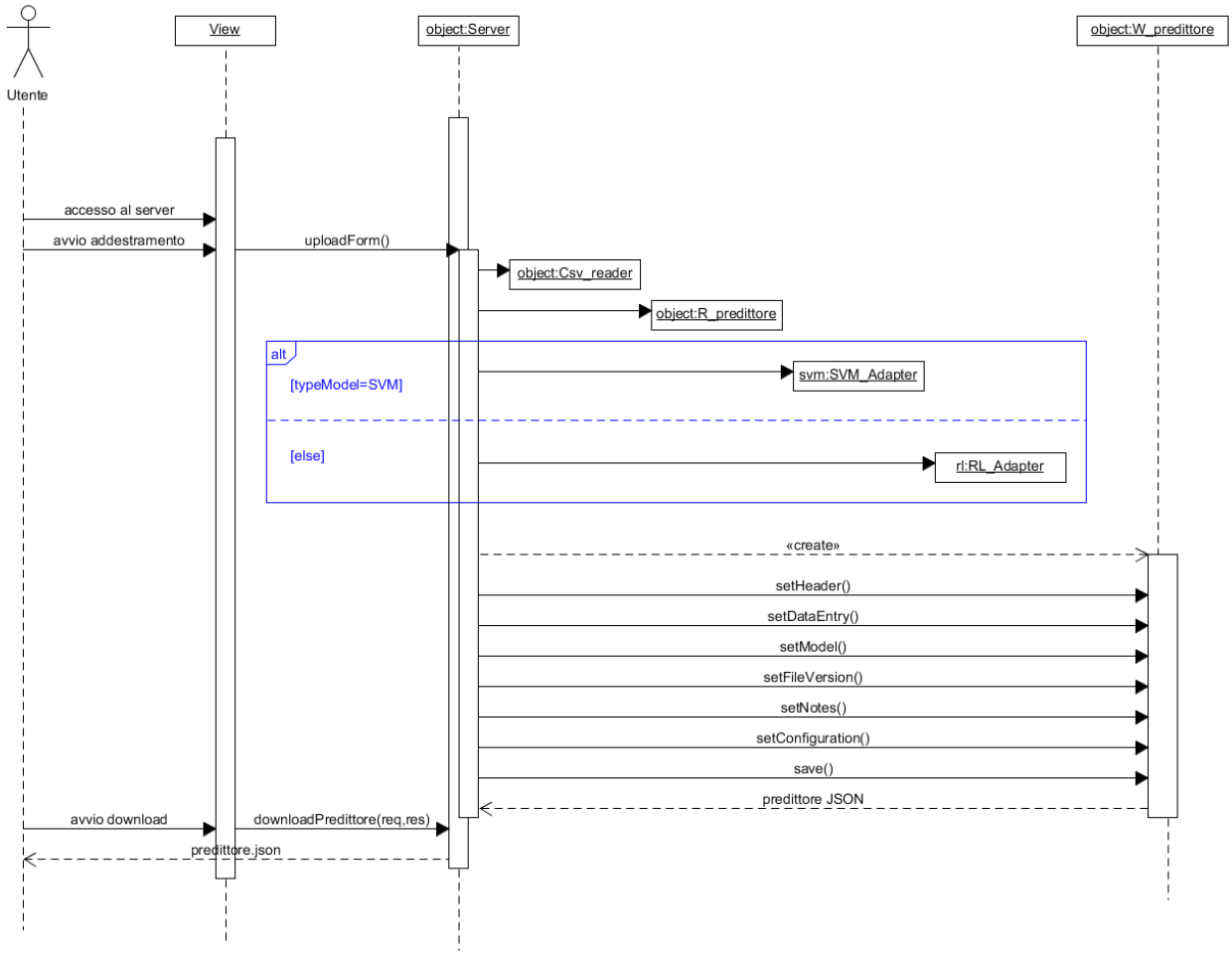


FIGURA 7 - DIAGRAMMA DI SEQUENZA ADDESTRAMENTO - W_PREDITTORE

6 Architettura plug-in

6.1 Visione generale

Il sistema si divide in tre parti principali, che sono:

- Il plug-in;
- Grafana;
- InfluxDB.

L'interazione tra i tre è la seguente: una volta avviato, il plug-in legge i dati che vengono inseriti su influx da fonti esterne, effettua delle previsioni su di essi e inserisce queste ultime nel database. Grafana poi si occupa di visualizzarle.

Per le pagine web che vengono visualizzate dall'utente viene usato AngulaJS, di conseguenza viene usato il pattern MVVM, con le pagine HTML che fungono da view, ImportCtrl e PredictCtrl da view-model ed InfluxDB, SVM_Adapter e RL_Adapter da model.

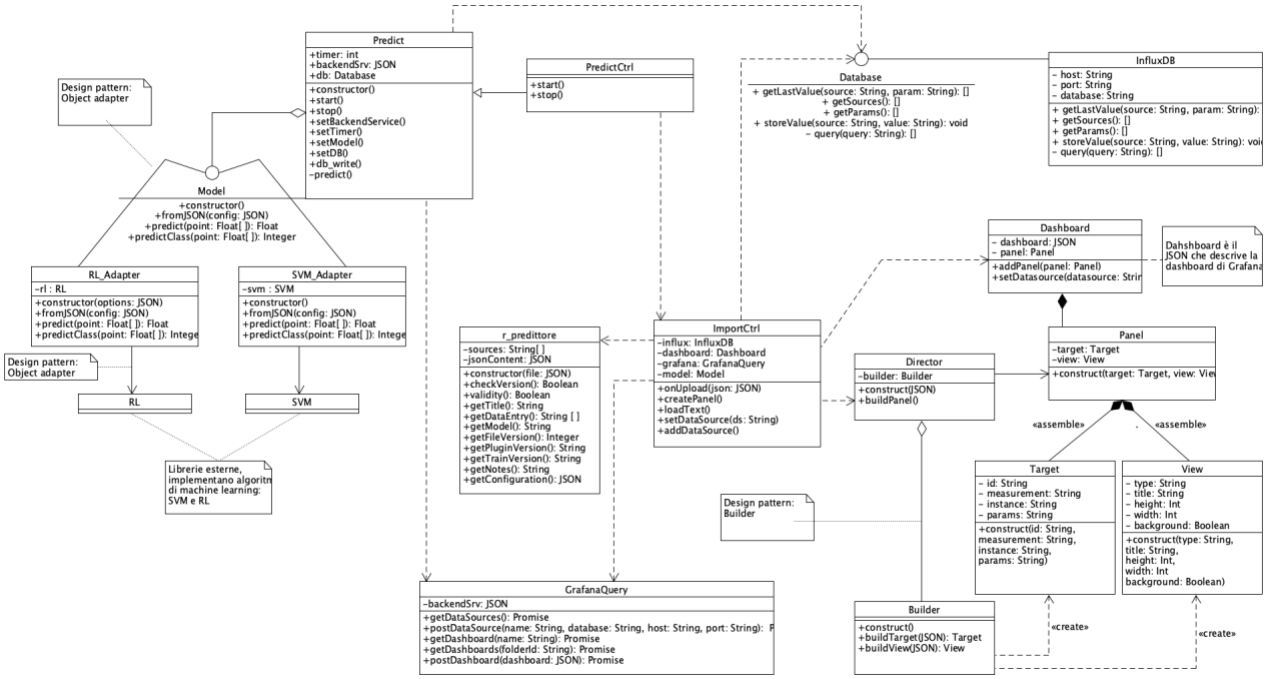


FIGURA 8 - DIAGRAMMA DELLE CLASSI PLUG-IN

Il plug-in si divide in due parti principali: ImportCtrl, che contiene la logica per importare predittori e impostare Grafana per visualizzare i dati, e PredictCtrl, che contiene la logica per effettuare la previsione dei dati.

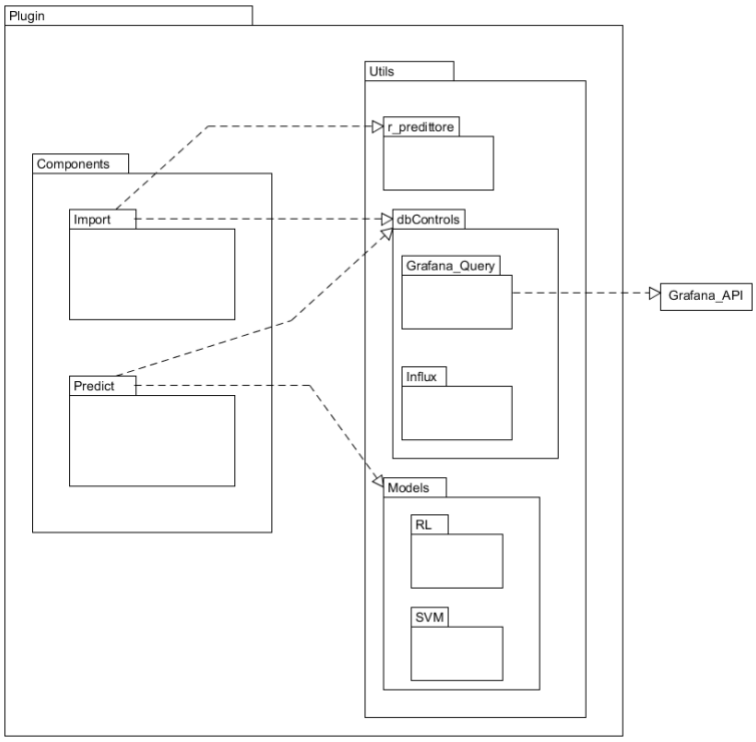


FIGURA 9 - DIAGRAMMA DEI PACKAGE PLUG-IN

6.2 ImportCtrl

La parte di ImportCtrl usa varie componenti per permettere all'utente di importare i predittori allenati, associarli con le sorgenti presenti su influx e creare un pannello ed una dashboard su cui visualizzare dati e previsioni.
Lo schema di sequenza per l'import dei predittori è il seguente:

- ImportCtrl è la classe principale con cui interagisce l'utente ed utilizza le altre per svolgere le funzioni richieste.
- r_predictior si occupa di controllare la validità del json ricevuto e di estrarne i dati.
- GrafanaQuery si occupa di effettuare richieste a Grafana utilizzando le sue API HTTP.
- InfluxDB implementa l'interfaccia Database e implementa le funzioni necessarie ad interagire con il database Influx.

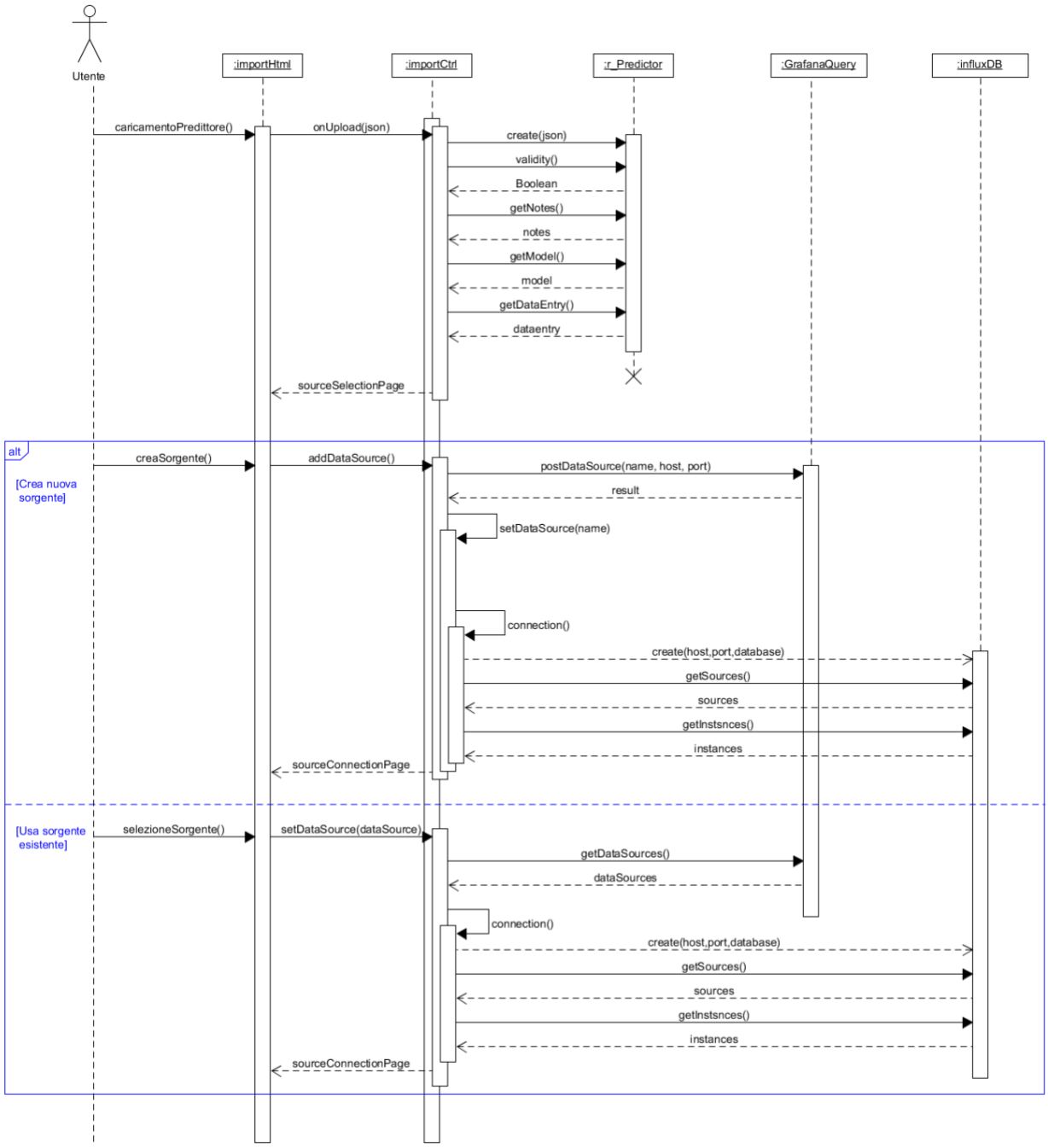


FIGURA 10 - DIAGRAMMA DI SEQUENZA PLUG-IN – IMPORTCTRL PT1

ImportCtrl si occupa anche della creazione dei pannelli e delle dashboard per visualizzare i dati. L'insieme delle classi Director, Builder, Target, View e Panel implementano il design pattern Builder per costruire i pannelli. Una volta costruiti quest'ultimi vengono aggiunti alle dashboard, che sono rappresentate dalla classe Dashboard. Lo schema di sequenza per la creazione dei Pannelli e delle Dashboard è il seguente:

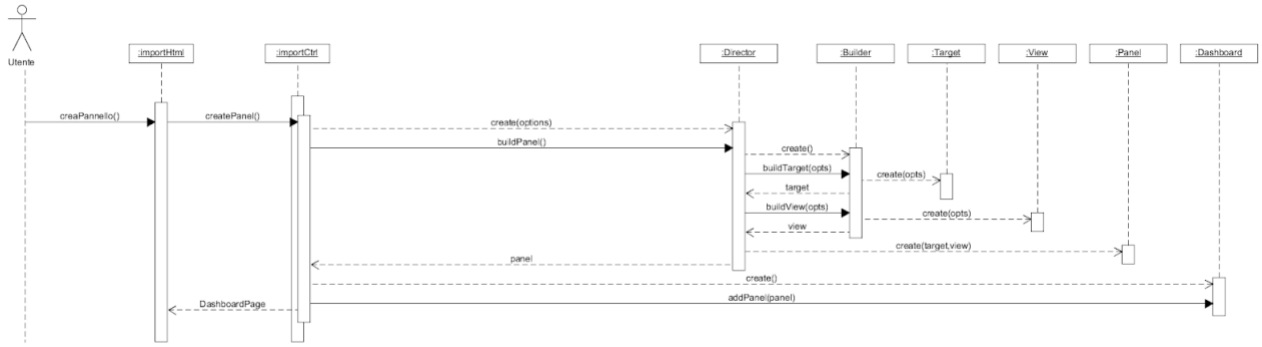


FIGURA 11 - DIAGRAMMA DI SEQUENZA PLUG-IN - IMPORTCTRL PT2

6.3 PredictCtrl

La parte di PredictCtrl si occupa di gestire l'avvio e l'arresto della predizione, oltre ad eseguire quest'ultima. PredictCtrl è la classe principale con cui interagisce l'utente per avviare o fermare la predizione. La classe Predict si occupa di effettuare in modo asincrono le predizioni, utilizzando un Model, e poi di salvarle nel database Influx utilizzando InfluxDB. Le classi SVM_Adapter e RL_ Adapter implementano Model e usano il design pattern Object Adapter per adattare rispettivamente la libreria SVM e RL. Il diagramma di sequenza è il seguente:

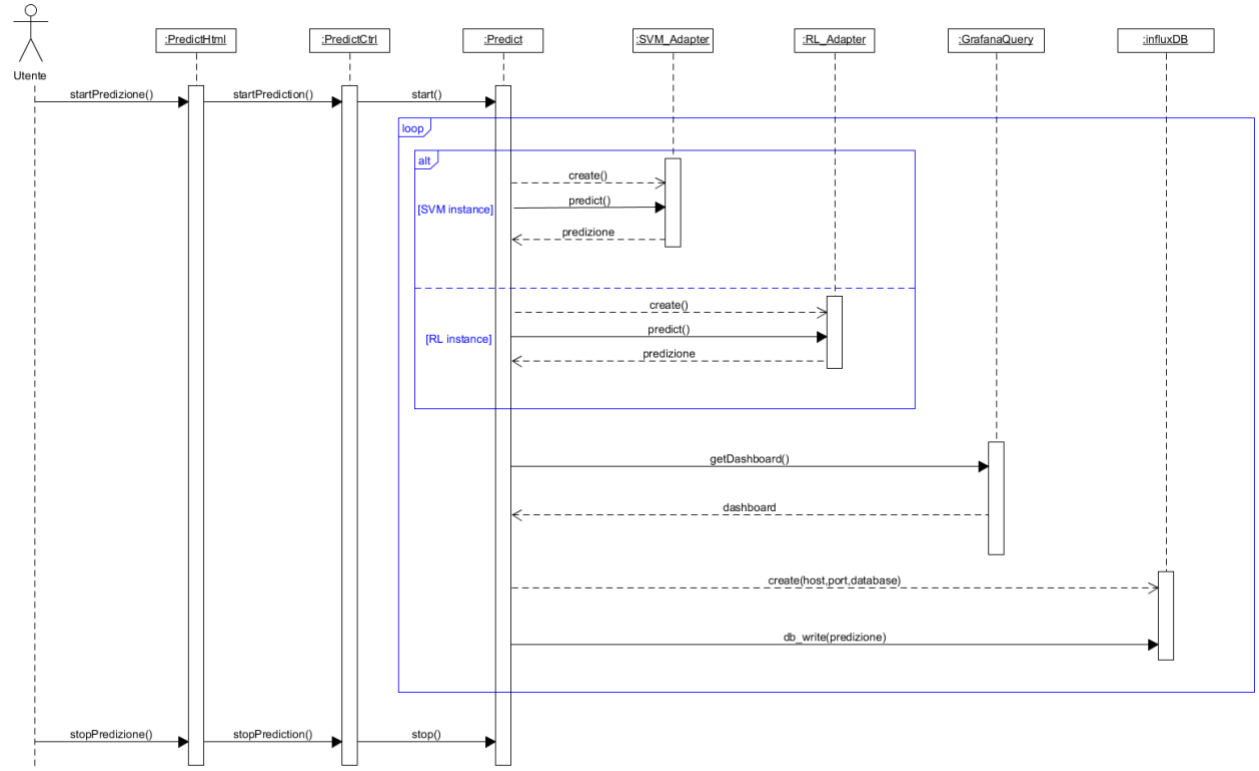
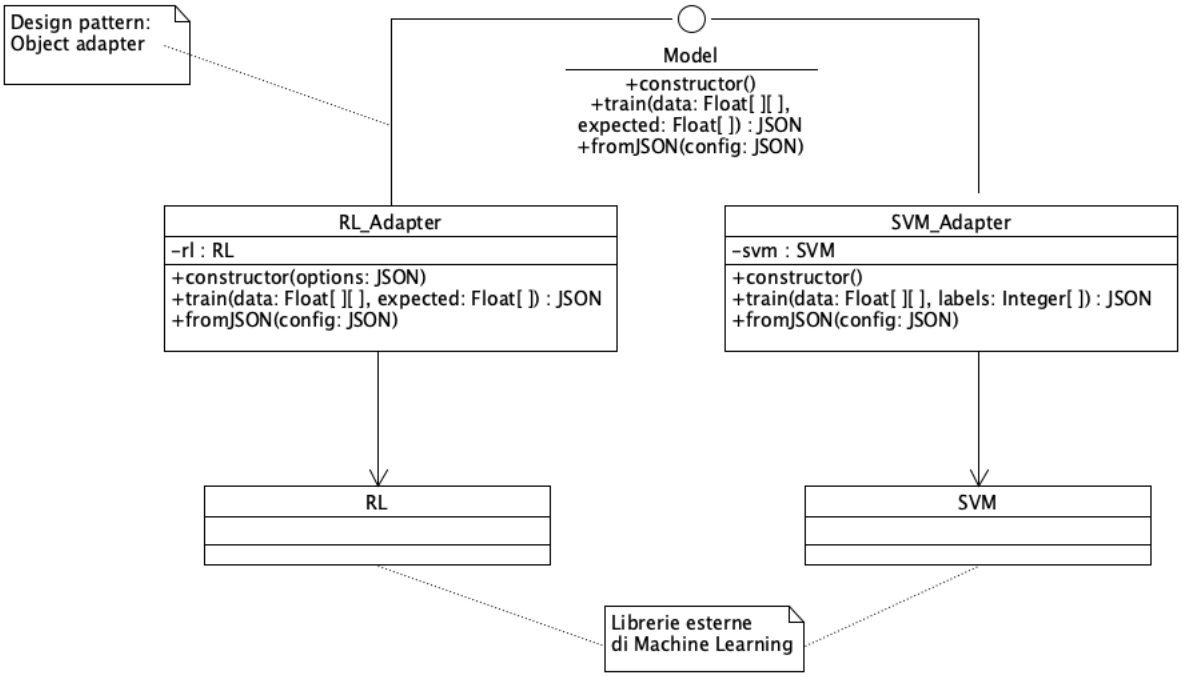


FIGURA 12 - DIAGRAMMA DI SEQUENZA PLUG-IN - PREDICTCTRL

7 Estensione funzionalità

7.1 Addestramento: aggiunta di un algoritmo di machine learning

Come presentato nel capitolo 5, gli algoritmi di Machine Learning sono stati gestiti utilizzando il Design Pattern Object Observer. In figura x viene riportato il diagramma delle classi relativo agli algoritmi di Machine Learningg. Non essendo possibile in `node.JS` una nativa implementazione delle interfacce, l'interfaccia `model` è stata implementata come una classe che lancia delle eccezioni se tutte le sue sottoclassi non implementano uno e entrambi i metodi che prevede. In questo modo ci si può assicurare che tutte le classi che implementano l'interfaccia contengano i metodi richiesti. La classe `Algorithm_Adapter` dovrà implementare i metodi richiesti, `fromJSON` e `train` e possedere un riferimento ad una libreria esterna che implementi l'algoritmo di Machine Learning desiderato che dovrà essere opportunamente inclusa. Il primo metodo dovrà accettare in ingresso un oggetto JSONg contenente i parametri di configurazione del modello di Machine Learning, in modo che sia possibile anche aggiornare un certo predittoreg legato a dei dati per i quali è già stato fatto un allenamento invece che farlo ripartire da zero. Il metodo `train` invece riceve come parametri un array multimensionale contenente i dati di allenamento e un array labels con i valori attesi per effettuare un allenamento supervisionato del modello. Il metodo train deve restituire un oggetto JSON contenente la configurazione del modello che verrà salvata nel file predittore.json risultato dell'applicazione di addestramento e che dovrà essere dato in ingresso al plug-in di previsione.



8 Test

Per testare il prodotto sia per quanto riguarda la parte addestramento sia la parte plug-in è stato usato il framework di test JavaScript, chiamato Jest.

8.1 Installazione

Per installarlo per la parte addestramento sarà sufficiente digitare da terminale uno di questi due comandi:

- Installazione Jest usando yarn:

```
yarn add --dev jest
```

- Installazione Jest usando npm:

```
npm install --save-dev jest
```

Per la parte plug-in invece basta far uso del pacchetto `npm grafana/toolkit`, questo pacchetto al suo interno contiene Jest, ed esso viene configurato automaticamente. Quindi facendo la build vengono eseguiti anche tutti i test automaticamente (facendo uso di yarn).

8.2 Avviare i test

Per avviare solamente i test per la parte addestramento sarà sufficiente digitare da terminale il comando:

```
npm test
```

Per avviarli per poter osservare il livello del code coverage sarà sufficiente avviare il seguente comando:

```
npm run testcoverage
```

Per la parte plug-in per far avviare solamente i test sarà sufficiente digitare il seguente comando:

```
yarn grafana/tookit plugin:test
```

A Glossario

CSV
CSV (Comma-separeted values) è un formato basato su file di testo utilizzato per l'esportazione e l'importazione di una tabella di dati, specialmente da fogli elettronici o database.

JSON
JSON (JavaScript Object Notation) è un semplice formato per lo scambio di dati indipendente dal linguaggio di programmazione usato.

MACHINE LEARNING
Branca dell'Intelligenza Artificiale che si basa sull'idea che i sistemi possono imparare dai dati e prendere decisioni autonomamente o con un intervento umano limitato.

REGRESSIONE LINEARE (RL/LR)
Metodo di previsione statistica per stimare un valore numerico atteso condizionato dalla relazione esistente tra due o più fattori.

PREDITTORE
File contenente i dati relativi ad un modello di predizione addestrato. Conterrà quindi l'indicazione del modello e la mappatura dei pattern riconosciuti.

SUPPORT VECTOR MACHINE (SVM)
Modelli di apprendimento supervisionato associati ad algoritmi di apprendimento per la regressione e la classificazione.