Sistemi Autonomi

Andrea De Castri

Alma Mater Studiorum – University of Bologna

1 Introduzione

L'evoluzione tecnologica e l'ampliamento dei canali di comunicazione stanno giocando un ruolo fondamentale in molti ambiti lavorativi. Ad esempio, la ricerca clinica e alcune pratiche cliniche stanno subendo un cambiamento radicale attraverso l'introduzione di algoritmi di apprendimento che facilitano l'analisi di una enorme mole di dati relativa alle informazioni dei pazienti.

Il progetto **H2020 CONNECARE**, ad esempio, ha lo scopo di creare un nuovo sistema per gestire malati cronici con l'utilizzo di pratiche innovative, come la valutazione dei rischi di un paziente attraverso l'analisi di modelli predittivi. A causa dello scetticismo di alcune cliniche riguardo questi sistemi e a causa della burocrazia, la costruzione di modelli predittivi attraverso il **machine learning** risulta impossibile data la mancanza di dati dei pazienti. Il consorzio, così, ha deciso di sviluppare un sistema in grado di lavorare con modelli già allenati, pronti per essere applicati su nuovi dati. In questo modo, le cliniche potranno testare con i propri modelli le funzionalità e le potenzialità del sistema. Per descrivere modelli predittivi verranno utilizzati formati standard come **PMML** e il **PFA**, in grado di descrivere modelli di machine learning e modelli statistici.

Questo documento andrà ad analizzare e descrivere un sistema capace di caricare ed utilizzare modelli predittivi già allenati, sfruttando come descrittore di modelli lo standard *PMML*.

1.1 Motivazioni

Negli ultimi anni è cresciuta la necessità di costruire sistemi a supporto delle decisioni con lo scopo di aumentare l'efficacia dell'analisi dei dati, in quanto fornisce supporto a coloro che devono prendere decisioni in breve tempo. Un sistema a supporto delle decisioni si appoggia su un'ampia base di conoscenza che aiuta l'utilizzatore a prendere la miglior decisione possibile nel minor tempo possibile.

A questo punto entrano in gioco i *modelli predittivi*, che sono modelli costruiti con l'analisi di dati storici e attuali per fare predizioni sul futuro o su eventi sconosciuti. Nel mondo clinico, questi modelli trovano relazioni tra molti fattori che permettono la valutazione del rischio potenziale associato a determinate condizioni del paziente.

I modelli sopra citati possono essere costruiti con diversi linguaggi di programmazione, ma questo pone il vincolo che un modello creato con un determinato linguaggio può essere utilizzato solo attraverso lo stesso linguaggio. In questo modo la interoperabilità tra sistemi differenti risulta molto difficile. Per ovviare al problema, sono stati sviluppati dei formati di rappresentazione dei modelli in grado di essere condivisi da sistemi eterogenei. I formati principali sono **PMML** e **PFA**, scritti corrispettivamente con formattazione *XML* e *JSON*. Nel seguente progetto è stato scelto di utilizzare come descrittore di modelli il PMML, dato che è più diffuso e con fondamenta più solide rispetto al PFA.

La trasformazione di un modello nel corrispondente PMML permette, nel nostro caso, di far cooperare più cliniche che utilizzano software di analisi differenti. In questo modo una clinica che lavora con un linguaggio X riesce ad utilizzare un modello costruito da un'altra clinica che utilizza il linguaggio Y.

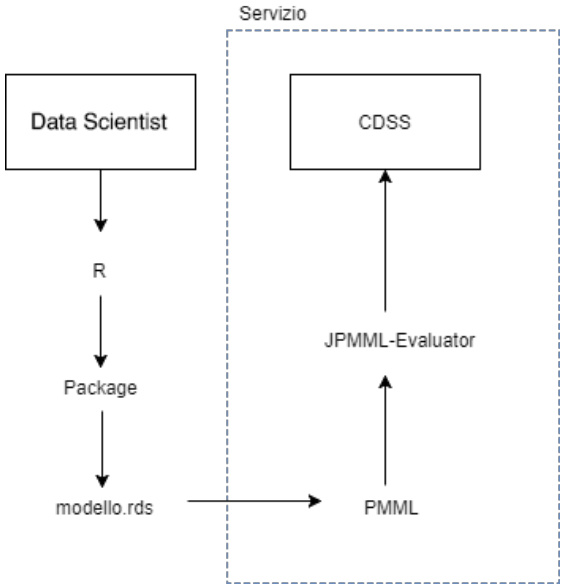

Fig. 1. Ciclo di trasformazione di un modello

1.2 Goal

L'obiettivo principale è la costruzione di un sistema in grado di caricare e convertire *modelli predittivi* **R** in modelli **PMML**, valutare l'input dell'utente utilizzando un modello pre-caricato e validare l'output generato da un modello *PMML* con l'output del modello *R* corrispondente.

2 Processo di sviluppo e piano di lavoro

2.1 Analisi strumenti software esistenti

Prima di procedere con la costruzione del sistema, sono state prese in esame diverse soluzioni, ognuna con i propri pro e contro. Questa analisi ha permesso di esaminare i diversi strumenti necessari, ma soprattutto ha evidenziato i vantaggi e le problematiche che conseguivano dalla scelta di una soluzione al posto di un'altra.

Il primo approccio consisteva nel trovare delle librerie Java capaci di eseguire codice scritto in linguaggio R, in questo modo si potevano utilizzare modelli R in Java senza effettuare nessun tipo di conversione dei modelli. Questo approccio non è ottimale, inoltre dovrebbe essere ripetuto ogni volta che il sistema deve lavorare con un modello costruito con un nuovo linguaggio.

Il secondo approccio prevedeva di utilizzare delle librerie Java o R capaci di convertire modelli R in modelli PMML. Per garantire più libertà ai *Data Scientist* nello sviluppo di modelli, si è preferito focalizzarsi su librerie Java. In questo modo il Data Scientist non è vincolato da nessuna libreria, ma il sistema si occuperà della comprensione e della conversione del modello R in modello PMML.

Per quest'ultimo approccio sono state trovate diverse librerie, descritte in dettaglio in seguito. Prima di utilizzarle all'interno del progetto sono stati effettuati dei semplici test per valutarne l'effettivo funzionamento. Principalmente sono stati eseguiti gli esempi forniti dalla documentazione, inoltre per un primo utilizzo sono stati adattati anche dei sorgenti.

2.2 Conversione modelli

Il primo passo è stato trovare una libreria che fosse in grado di comprendere e convertire modelli R in modelli PMML. La libreria avrebbe dovuto riconoscere alcuni dei principali package (es. Random forest, Caret) per la costruzione di modelli. I package possono essere intesi come dei raccoglitori di algoritmi per la creazione dei modelli, quindi uno stesso modello R può essere creato con package differenti.

La libreria utilizzata è stata **JPMML-R**, scritta in Java e oltre ad essere molto performante, grazie al suo caching, è compatibile con molte altre librerie di cui parleremo più avanti. JPMML-R è utilizzabile solo per programmi Java, ma esiste una versione utilizzabile in R: **r2pmml**. Dopo varie ricerche si è riscontrato che r2pmml utilizzi al suo interno JPMML-R.

Gli sviluppatori di questa libreria hanno rilasciato il loro lavoro open-source su GitHub[1], ma non hanno rilasciato esempi per l'utilizzo. L'unico aiuto che ci viene fornito è una conversione di un modello R in uno PMML attraverso linea di comando. Per poter utilizzare direttamente le classi Java è stato necessario esplorare il codice sottostante in modo da individuare quali fossero le classi interessate alla conversione di un modello.

2.3 Valutazione di modelli

Effettuata la conversione è stato indispensabile avere una libreria che potesse utilizzare questi PMML per valutarli con input dell'utente. Gli stessi sviluppatori di JPMML-R hanno svilupatto **JPMML-Evaluator**[2], una libreria capace di valutare e validare risultati di modelli PMML. Questa libreria utilizza al suo interno **JPMML-Model**, che ha lo scopo di effettuare murshalling di modelli.

Anche in questo caso non sono state fornite molte indicazioni per il suo utilizzo e non è fornita una documentazione adeguata, infatti per effettuare una valutazione sono utilizzate alcune classi di cui è difficile capirne il significato. L'unica documentazione disponibile è composta da due esempi eseguibili da linea di comando: il primo mostra come effettuare una conversione di modelli, il secondo spiega la validazione di un modello. Oltre ai due esempi, sono forniti dei frammenti di codice che consentono il recupero di alcune informazioni di un modello, come nome dei campi, tipo di dato, tipo operazionale ecc...

Per eseguire una valutazione viene caricato un modello PMML attraverso un InputStream, successivamente viene effettuato il murshalling per avere la relativa classe Java del modello. In seguito viene creato un Evaluator, che alla fine valuterà l'input dell'utente. L'input dell'utente deve essere un file CSV di cui vengono letti tutti i record attraverso un parser apposito. Infine l'output restituito è un file CSV con tutte le valutazioni dei record.

2.4 Validazione di un modello

La validazione è molto simile alla valutazione, ma per poter essere eseguita ha bisogno di prendere in input il modello PMML, l'input da valutare e l'output atteso. Una volta effettuata la valutazione, l'Evaluator confronterà i record dell'output con quelli dell'output atteso e per ogni differenza verrà mostrato un conflitto sulla console.

[1] https://github.com/jpmml/jpmml-r
[2] https://github.com/jpmml/jpmml-evaluator

3 Implementazione

Tutte le funzionalità principali del progetto sono delegate alla classe **ModelManager**, che può essere considerata come un wrapper di funzionalità utilizzabile all'interno di un servizio. Questa classe ha le dipendenze delle librerie sopra citate e queste a loro volta sfruttano altre librerie. Per dare un quadro completo, viene fornito un diagramma:

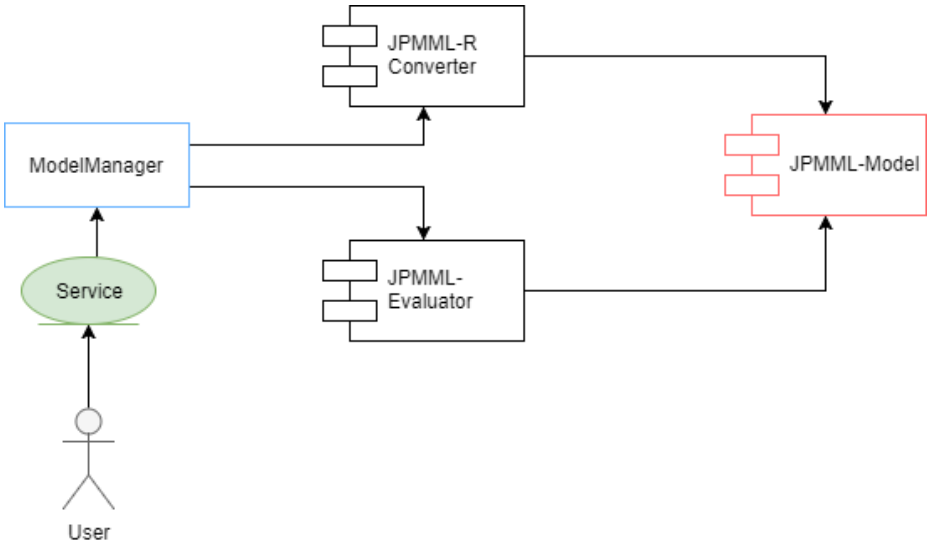

Fig. 2. Dipendenze del progetto e delle librerie

3.1 Conversione dei modelli

Prima di poter effettuare la conversione di un modello, è indispensabile creare un modello R. Di seguito è stato riportato un frammento di codice per creare un modello R attraverso il *package* **randomForest**. Il seguente modello descrive diversi tipi di fiori attraverso 4 variabili: lunghezza petalo, larghezza petalo, lunghezza stelo e larghezza stelo.

```
library("pmml")
library("randomForest")

data("iris")

irisInput = iris[, 1:4]
write.table(irisInput, file = "InputIris.csv",
    col.names = TRUE, row.names = FALSE, sep = ",",
    quote = FALSE)

iris.randomForest = randomForest(Species ~ ., iris)
saveRDS(iris.randomForest, "IrisModel.rds")

# saveXML(pmml(iris.rpart), file =
    "DecisionTreeIris.pmml")

iris.class = predict(iris.randomForest, newdata =
    iris, type = "class")
iris.prob = predict(iris.randomForest, newdata = iris,
    type = "prob")

irisOutput = data.frame(iris.class, iris.prob)
names(irisOutput) = c("Species",
    "probability(setosa)", "probability(versicolor)",
    "probability(virginica)")
write.table(irisOutput, file = "ExpectedOutput.csv",
    col.names = TRUE, row.names = FALSE, sep = ",",
    quote = FALSE)
```

Inizialmente, per poter convertire modelli *R* in modelli *PMML*, è stato necessario modificare il metodo **run** appartenente alla classe Main del progetto JPMML-R. Il metodo era stato modificato aggiungendo come parametri l'**inputStream** del modello R e l'**outputStream** del modello PMML.

Successivamente, esplorando meglio la libreria, è stato possibile utilizzarla direttamente per la conversione di modelli. Per convertire un modello è necessario creare un **Parser** per ottenere la *regular expression* che rappresenta il modello

R. In seguito, attraverso la regular expression, si creerà un **Converter** capace di generare un oggetto che rappresenta il modello PMML corrispondente. Infine, attraverso la classe **MetroJAXUtil**, si può effettuare il salvataggio sul file del modello ottenuto.

```
RExpParser parser = new RExpParser(content);
REXP rexp = parser.parse();

ConverterFactory converterFactory =
   ConverterFactory.newInstance();
Converter<REXP> converter =
   converterFactory.newConverter(rexp);

PMML pmml = converter.encodePMML();
MetroJAXBUtil.marshalPMML(pmml, os);
```

3.2 Valutazione e validazione di modelli

L'implementazione della valutazione e della validazione è stata triviale, dato che, nel repository in cui viene fornita la libreria, sono presenti dei semplici esempi su come effettuare queste operazioni.

L'unico problema incontrato riguarda la validazione: infatti durante le prime prove si incorreva in un errore, in cui esistevano molti conflitti tra l'output del modello R e l'output del modello PMML corrispondente. Questi conflitti dipendevano dal fatto che i nomi delle variabili in output del programma R erano differenti da quelle create dalla libreria *JPMML-Evaluator*. Per ovviare a questo problema è necessario restituire i nomi delle variabili del programma R nel seguente modo: [3]

probability(NOME VARIABILE)

Durante la fase di valutazione dei modelli, JPMML-Evaluator definisce in maniera automatica il nome delle variabili in output: il nome delle variabili viene definito seguendo il formato scritto in precedenza. Nel momento in cui viene effettuata la validazione di un modello, qualora le variabili in output non avessero lo stesso identico nome si avrebbe per ogni record un numero di conflitti pari al numero di campi.

[3] Vedi codice R inserito nella sezione precedente

4 GUI

Per testare tutte le funzionalità del sistema è stata costruita una GUI composta da 3 pannelli.

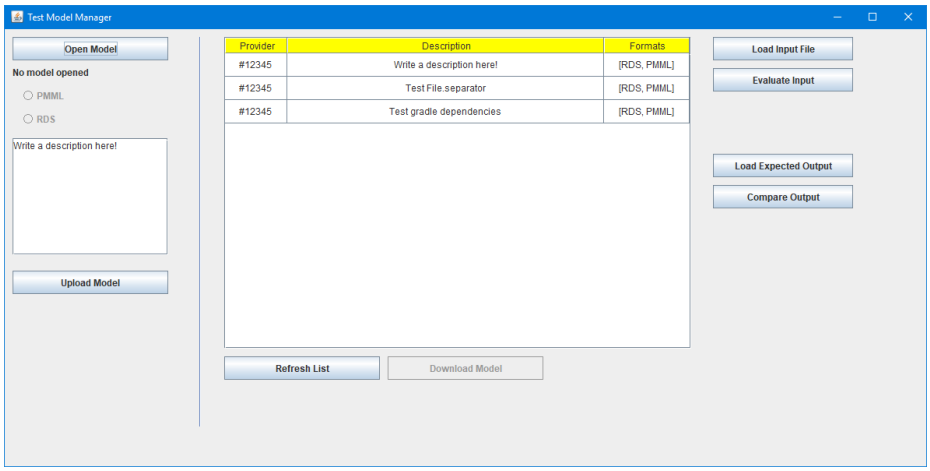

Fig. 3. GUI dell'applicazione

4.1 Pannello sinistro

Questo menù mette a disposizione funzionalità come il caricamento di modelli R o PMML. Nel caso in cui venga caricato un modello R, questo viene salvato su file sia come modello R sia come modello PMML. Una volta selezionato il modello da caricare viene riconosciuta in maniera automatica la tipologia di modello in base all'estensione del file. Per caricare un modello si deve cliccare sul pulsante **Open Model**, scrivere una descrizione facoltativa del modello e infine cliccare il pulsante **Upload Model**.

4.2 Pannello centrale

Il panello centrale è composto quasi esclusivamente da una JTable che mostra tutti i modelli caricati sul sistema e le informazioni essenziali. Di ogni modello è possibile effettuare il download; il download è simulato attraverso la copia del file di un modello in un'altra directory del sistema. Per scaricare un modello si deve prima di tutto selezionare il modello, cliccando il record relativo nella JTable, infine si deve cliccare sul bottone **Download Model**.

4.3 Pannello destro

Due sono le funzionalità principali di questo pannello: la valutazione e la validazione. Queste funzionalità sono gestite da 4 pulsanti. Per valutare l'input dell'utente con un modello si deve selezionare il modello nella JTable, caricare l'input attraverso il pulsante **Load Input File**, infine cliccare sul pulsante **Evaluate Input**. Il risultato viene salvato nella cartella **src** del progetto in formato *CSV*.

Invece per confrontare un output di un modello R con il corrispettivo output del modello *PMML*, oltre ad effettuare i passaggi sopra elencati per la valutazione, si deve caricare l'output atteso attraverso il bottone **Load Expected Output**. Il risultato della valutazione viene mostrato in standard output sulla console cliccando il bottone **Compare Output**.

5 Sviluppi futuri

Il sistema corrente può essere modificato e ampliato in modo da salvare tutte le informazioni su un database relazionale o non, invece di salvare tutte le informazioni su file. Le funzionalità fornite sono disponibili solo in locale, ma potrebbero essere fornite attraverso servizi **Rest**. I modelli predittivi, oltre ad essere salvati sul sistema potrebbero essere direttamente costruiti dal sistema attraverso algoritmi di apprendimento. In questo modo il sistema è in grado di modificare il modello ogni volta che un nuovo dato venga inserito nel database.