ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Ingegneria e Architettura
Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

SISTEMI DI SUPPORTO ALLE DECISIONI IN AMBITO CLINICO: PREDIZIONE DEL RISCHIO "AS A SERVICE"

Relazione finale in
SISTEMI AUTONOMI

Relatore
Prof. ANDREA OMICINI

Co-relatori
Ing. STEFANO MARIANI

Presentata da
ANDREA DE CASTRI

Seconda Sessione di Laurea
Anno Accademico 2016 – 2017

PAROLE CHIAVE

Machine Learning

Decision Support System

Healthcare

REST

Angular

Historia magistra vitae

Indice

Introduzione

L'evoluzione tecnologica e l'ampliamento dei canali di comunicazione stanno giocando un ruolo fondamentale nelle aziende che hanno intenzione di evolvere la propria infrastruttura IT in un modello **as a service**. Il cloud sta svolgendo una funzione principale nel business aziendale portandone il mercato ad una crescita che oscilla tra il 18% e il 21%, nel solo 2017, e spingendo migliaia di aziende a integrare la vecchia IT con la nuova. [1]

Si deve inoltre tener conto dell'enorme volume di dati prodotto dai processi aziendali nell'ultimo ventennio, in quanto la richiesta di utilizzo di tali dati, per scopi aziendali, ha reso necessaria la creazione di tecniche avanzate per l'analisi degli stessi; ne sono un esempio la ricerca clinica e alcune pratiche mediche. Queste, infatti, 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.

Una delle tecniche utilizzate per la costruzione di algoritmi in grado di imparare da eventi passati e di predire eventi sconosciuti è il **machine learning**. Il risultato di un algoritmo di *learning* è rappresentato da un modello predittivo che, applicato ad un set di dati, permette di effettuare predizioni. Questi modelli possono essere rappresentati con formati differenti, introducendo nel sistema un ulteriore livello di difficoltà. Il sistema deve essere in grado di riconoscere la maggior parte dei formati di rappresentazione, ma, allo stesso tempo, cerca di promuovere la standardizzazione dei formati attraverso l'utilizzo dello standard *PMML*.

Questa standardizzazione ha lo scopo di permettere l'interoperabilità tra cliniche differenti, in modo da promuovere la cooperazione tra cliniche, potenziando l'abilità di analisi dei dati della singola clinica.

La sfida che ci si pone e di cui si tratterà in questo documento è la creazione di un sistema di supporto alle decisioni as a service che utilizzi una base di conoscenza elaborata da tecniche di machine learning. [10] Nel mondo sanitario queste tecnologie trovano applicazioni sia nella formulazione di diagnosi, sia nella predizione del rischio di un paziente affetto da determinate malattie.

Alcune tecniche, come l'utilizzo di reti neurali e della regressione logistica, riescono a identificare e classificare rispettivamente l'infarto del miocardio e la sindrome coronarica acuta con un'accuratezza dell'87.5% e del 72%. [2] La predizione del rischio può essere intesa come: probabilità di un paziente di entrare in ricovero nei successivi 30 giorni, probabilità di peggioramento oppure possibilità di morire.

Nonostante questi sistemi trovino applicazione, con ottimi risultati, in diversi campi della medicina, la loro integrazione risulta difficoltosa sia a causa dello scetticismo di alcune cliniche riguardo l'uso di queste metodologie, sia a causa della burocrazia. Per avvicinare le cliniche a queste nuove tecnologie e metodologie, il sistema deve essere in grado di lavorare con modelli predittivi 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.

Lo scopo che ci si prefigge di raggiungere con questo lavoro è quello di fornire un primo prototipo funzionante di un *Clinical Decision Support System*, focalizzandosi sulla costruzione di un'architettura di sistema che guidi l'analisi delle migliori tecnologie da utilizzare.

La tesi è strutturata in sette capitoli. Nel primo capitolo si fornirà un quadro generale dello stato dell'arte per quanto riguardano i sistemi di supporto alle decisioni, machine learning, sistemi healthcare e microservizi. Verranno, quindi, descritte le motivazioni e i goal dello sviluppo di questo sistema, seguite da un'accurata analisi dei requisiti.

Il passo successivo sarà quello di andare ad indicare quella che è l'architettura ideale per il sistema, mettendo in risalto alcuni pro e contro dell'infrastruttura. In seguito si discuterà delle tecnologie utilizzate per lo sviluppo

del sistema, ponendo l'attenzione sui due framework di sviluppo front-end e back-end.

Nel quinto e sesto capitolo verranno mostrate la struttura dell'intero sistema e le componenti principali, fornendo, dove ritenuto indispensabile, parti dell'implementazione e una breve documentazione.

Infine, si andrà ad analizzare lo specifico caso di studio con una descrizione dei test effettuati.

Capitolo 1

Stato dell'arte

1.1 Machine Learning

Il **Machine Learning** si sviluppa con lo studio dell'*intelligenza artificiale*[1] e rappresenta un insieme di metodologie che forniscono ai computer la capacità di apprendere. L'apprendimento è possibile grazie alla costruzione di algoritmi generici il cui scopo è quello di imparare e fare predizioni su un campione di dati; questi algoritmi non sono pre-programmati, ma il loro comportamento è regolato dai dati stessi.

1.1.1 Concetti chiave

Per capire al meglio questo ramo della *computer science* è necessario introdurre dei concetti fondamentali:

- **Pattern**: con il termine *pattern* vengono indicati i dati utilizzati per la fase di apprendimento.

- **Modello**: un *modello* è il risultato di un metodo di machine learning e dell'algoritmo usato con questo metodo. Il *modello* può essere utilizzato per effettuare predizioni sui dati.

[1] L'intelligenza artificiale è un'ampia disciplina informatica che studia le metodologie e le tecniche che consentono di sviluppare sistemi in grado di emulare il comportamento umano.

- **Features**: una *feature* è una proprietà su cui un modello viene allenato. Prendendo in esempio un sistema in cui si vuole capire quale mail sia spam, una *feature* potrebbe essere la frequenza con cui quella mail viene inviata.
- **Classe**: insieme di dati che hanno *features* comuni.

1.1.2 Metodi di apprendimento

Nel campo del *Machine Learning* ci sono due categorie principali di apprendimento che si differenziano in base al feedback che il sistema riceve durante il procedimento:

- **Apprendimento supervisionato**: per l'addestramento dell'algoritmo vengono utilizzati *pattern* di cui sono note le *classi*.
- **Apprendimento non supervisionato**: le classi dei pattern non sono note durante la fase di addestramento; gli algoritmi solitamente cercano di individuare **cluster**, ovvero gruppi di dati con *features* simili.

Classificazione e regressione

Il problema della **classificazione** nel dominio dell'*apprendimento supervisionato* è relativamente semplice. Partendo da un insieme di etichette (o *classi*) e un insieme di dati, a cui è già stata assegnata una classe corretta, è possibile predire l'etichetta per un nuovo dato ancora non etichettato. Un esempio di *classificazione* potrebbe essere la previsione del sesso maschile o femminile di una persona basandosi sull'altezza e il peso. Quindi dato un vasto **training set** di persone etichettate, l'algoritmo di classificazione può generare un modello. Questo modello può essere usato per predire se un persona è maschio o femmina.

La **regressione** è uno strumento diverso rispetto alla classificazione, questo perché nella *regressione* non viene predetto un valore categorico da assegnare ai dati, ma viene predetto un valore numerico. Riproponendo l'esempio precedente, con la regressione sarebbe possibile predire l'esatto valore del peso

o dell'altezza utilizzando come input il sesso e uno degli altri due parametri mancanti.

Clustering

A differenza della *classificazione* e della *regressione*, gli algoritmi di **clustering** fanno parte dell'apprendimento *non supervisionato*. Nel *clustering* i pattern non sono etichettati e spesso il numero di cluster non è noto a priori, quindi la natura non supervisionata del problema lo rende più complesso della classificazione. Il compito di questi algoritmi è quello di individuare **cluster**, ovvero gruppi di pattern dalle caratteristiche simili.

Apprendimento online-offline

Gli algoritmi sopra elencati posso effettuare a loro volta un apprendimento **online** oppure **offline**. Il primo metodo permette di aggiornare un modello pre-esistente con nuovi dati, mentre il secondo consiste nella generazione di un nuovo *modello* in cui sono utilizzati anche i nuovi *pattern*. Per questioni di performance, l'apprendimento *online* è preferibile, ma molti algoritmi non lo rendono applicabile.

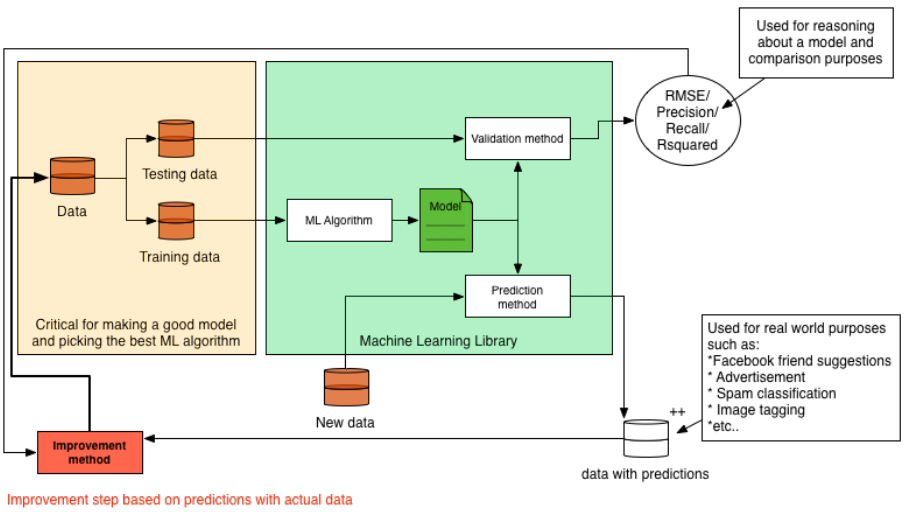

Figura 1.1: Fasi di apprendimento nel Machine Learning

1.1.3 Tool

Nel corso degli anni statistici, programmatori e ricercatori hanno sviluppato diverse librerie e framework per aiutare tutti coloro che volevano cimentarsi nel mondo del machine learning. La scelta del tool dipende molto dal linguaggio di programmazione e dalle preferenze dello sviluppatore. Alcuni dei tool più noti sono:

- **Weka**: scritto in Java, ma molto vecchio;
- **Smile**: in linguaggio *Scala* e *Java* e rilasciato di recente;
- **R** e **Caret**: linguaggio nato per la statistica.

La libreria **Smile** è stata utilizzata per sviluppare parte del progetto descritto in questo documento. Le performance elevate, la semplicità e una buona documentazione ha fatto si che la scelta ricadesse proprio su questa libreria.

1.1.4 Interoperabilità tra sistemi

Una problematica da affrontare è legata alla interoperabilità tra sistemi di *machine learning*. Ogni **tool** di learning produce un proprio modello predittivo, che però può essere compreso solo dallo stesso tool. 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**. Il descrittore di modelli *PMML* è molto diffuso e ha fondamenta più solide rispetto al *PFA* che è ancora in fase embrionale.

In ambito clinico sono diffuse diverse tecnologie di *machine learning*, quindi per permettere alle cliniche di cooperare tra loro è necessario effettuare una conversione di modelli attraverso l'utilizzo di uno dei formati di rappresentazione scritto precedentemente. 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.

1.1.5 Applicazione nel mondo clinico

Trattando il mondo clinico, possiamo parlare di **Clinical Machine Learning** che è una nuova disciplina in cui la statistica e l'informatica aiutano a migliorare la nostra comprensione della salute umana. Di recente sono stati sviluppati algoritmi per comprendere meglio la progressione delle malattie e facilitare nuove e precise strategie di trattamento per patologie come il diabete di tipo 2 e un raro cancro del sangue.

In futuro, con l'introduzione delle cartelle cliniche elettroniche (**EHR**) che descrivono in maniera dettagliata tutte le caratteristiche cliniche e fisiologiche di un paziente, sarà possibile predire con più precisione la probabilità di essere nuovamente ricoverato dopo un certo intervento, la probabilità di morte successivamente aver eseguito una determinata cura, ecc.

1.2 Healthcare System

Un **Healthcare System** è un sistema che si occupa della gestione delle risorse, dei dispositivi e dei metodi necessari per ottimizzare l'acquisizione e l'uso delle informazioni nell'ambito della salute e della biomedicina. Queste informazioni svolgono un ruolo sempre più importante nell'erogazione dei servizi sanitari moderni e nell'efficienza dei sistemi stessi.

Gli strumenti necessari per la costruzione di un *Healthcare System* includono linee guida cliniche, terminologie mediche, computer e altre tecnologie di informazione e comunicazione. I dati trattati da questi sistemi sono dati sensibili che riguardano le cartelle cliniche dei pazienti, le procedure di amministrazione ospedaliera e altre informazioni sulle risorse umane.

Questi sistemi nascono principalmente dalla necessità di aiutare paesi in via di sviluppo o lontani da servizi ospedalieri, ma anche per migliorare la qualità dei servizi offerti. Sono molteplici le ragioni per lo sviluppo di sistemi sanitari guidati dall'informazione tecnologiche:

- Erogazione del servizio in paesi ancora in via di sviluppo o lontani da zone con pochi servizi sanitari.

- Più qualità e sicurezza garantita ai pazienti.
- Smaterializzazione delle informazioni. Da questo consegue la possibilità di lavorare sempre con informazioni corrette, eliminando gli errori più comuni.
- Standardizzazione delle informazioni sanitarie.

In più, il rapido emergere di molte malattie croniche, che richiedono costose cure e trattamenti a lungo termine, sta facendo sì che molte cliniche sanitarie riesaminino le loro pratiche di assistenza. Per ridurre i costi della sanità, molte malattie al giorno d'oggi possono essere trattate direttamente da casa, monitorando sempre la patologia del paziente e controllando la corretta esecuzione della cura. Una delle malattie croniche possibili da curare con questi sistemi e che nel 2006 ha colpito 180 milioni di persone in tutto il mondo è il diabete.

Oltre ad aiutare i pazienti nella loro cura e nel loro monitoraggio, questi sistemi includono altre tecnologie capaci di supportare i professionisti sanitari, come i medici, nell'analisi della diagnosi e nella selezione della miglior cura da somministrare.

1.3 Decision Support System

Un **Decision Support System** (**DSS**) è un sistema informatico in grado di supportare l'attività di **decision making** nei processi aziendali. Questi sistemi sono nati a causa dell'enorme quantità di dati prodotta nell'ultimo ventennio e dalla necessità di analizzarli giornalmente. Un **DSS** facilità l'analisi di grandi moli di dati estraendo in poco tempo le informazioni utili ai processi decisionali e aiutano a prendere decisioni di fronte a problemi difficilmente risolvibili nell'immediato.

In un'architettura **DSS** le componenti fondamentali sono 4:

- **Base dati**: il sistema si appoggia su una base di conoscenza che ha lo scopo di raccogliere i dati e le informazioni utili all'utente. Le informazioni vengono estrapolate partendo dai dati stessi e andranno a

produrre la **conoscenza**, la quale permette di trovare relazioni tra i dati precedentemente sconosciute all'utente.

- **Base di modelli**: questa componente contiene tutte le procedure necessarie a risolvere i problemi dell'utente, quindi ha il compito di organizzare i dati per il processo di elaborazione.
- **Interfaccia utente**: GUI chiara e semplice per accedere a tutte le funzionalità del sistema.
- **Utente finale**: l'utente finale è indispensabile poiché la decisione finale è la combinazione della valutazione dell'utente con le informazioni estrapolate dal sistema.

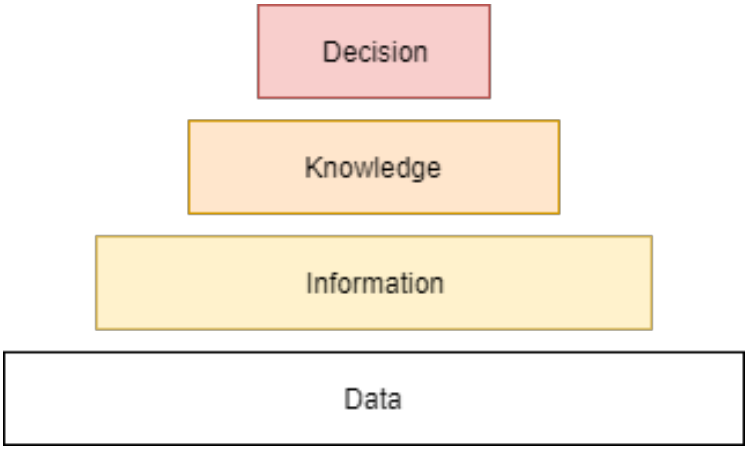

Figura 1.2: Processi di analisi dei dati per arrivare alle decisioni

L'utilizzo dei **DSS** non è limitato all'ambito aziendale, ma può essere utilizzato in ogni ambito in cui si ha la necessità di analizzare grandi quantità di dati e ricavare informazioni utili alla strategia decisionale. I punti di forza di questi sistemi sono la loro capacità di risolvere problemi mal strutturati e la loro flessibilità, essendo questi capaci di lavorare con dati di natura differente quali documenti, database, conoscenza personale ecc.

1.3.1 Clinical Decision Support System

Nel mondo clinico possiamo parlare di **Clinical Decision Support System** (**CDSS**), ovvero sistemi che migliorano l'assistenza sanitaria attraverso il supporto dato ai professionisti della salute nel compito del *decision making*. Lo scopo principale di questi sistemi è quello di assistere i medici sul punto di cura, in questo modo si riescono ad effettuare analisi migliori rispetto a quelle che un umano possa effettuare da solo. Oltre ad aiutare il medico nella scelta della cura e ad una analisi della diagnosi, i *CDSS* migliorano anche le prestazioni degli operatori sanitari[2], fornendo loro un supporto veloce da consultare prima e dopo dell'incontro con il paziente.

Tipi principali di CDSS

Esistono due tipi principali di CDSS, uno basato sulla conoscenza (**Knowledge-based**) e l'altro non basato sulla conoscenza (**Non-Knowledge-based**). La conoscenza del primo tipo contiene le regole e le associazioni di dati che spesso assumono la forma di regole **IF-THEN**. Per esempio, un sistema per determinare le interazioni tra farmaci potrebbe funzionare in questo modo:

1. il medico informa il sistema riguardo la diagnosi del paziente e i farmaci che attualmente il paziente sta assumendo;

2. il sistema confronterà tutti i farmaci utili e restituirà solo la lista di quelli che non interagiscono con gli altri già assunti;

[2]Molteplici studi indicano che con l'inserimento dei *CDSS* gli operatori sanitari hanno migliorato le loro prestazioni del 65%. [4]

I sistemi del secondo tipo usano una forma di intelligenza artificiale: il *Machine Learning*. Questo consente al sistema di imparare dai dati storici e trovare modelli nei dati clinici senza scrivere esplicitamente regole e input. Nonostante siano sistemi sicuri ed efficaci, i medici preferiscono evitarli per ragioni di affidabilità e responsabilità, dato che sistemi basati sull'apprendimento automatico non possono spiegare il motivo delle loro conclusioni. Tuttavia, questi possono essere utili come sistemi post-diagnosi per estrarre connessioni tra i pazienti e la loro storia clinica, utili per la ricerca e per prevedere eventi futuri. Inoltre, è stato affermato che in futuro il supporto decisionale clinico inizierà a sostituire i medici in compiti semplici.

Ostacoli da superare

Le istituzioni si stanno sforzando per produrre ed utilizzare questi sistemi, ma la maggior parte delle cliniche non li ha ancora accettati. Un ostacolo fondamentale è il complesso **workflow** seguito dalle cliniche, che rende difficile l'integrazione con questi sistemi. Le cliniche non hanno intenzione di abbandonare i loro sistemi attuali per passare ai *CDSS*, poiché dovrebbe reinserire tutti i dati nel nuovo sistema pagando molto tempo prezioso.

Un altro aspetto indispensabile per la buona riuscita di questi sistemi è la costruzione di cartelle cliniche elettroniche (**EHR**). Queste sono un modo per acquisire e utilizzare i dati in tempo reale per fornire assistenza di alta qualità ai pazienti. Incorporare **EHR** e **CDSS** nel processo medico cambierebbe il modo in cui la medicina è stata praticata fino al giorno d'oggi. La difficoltà nella costruzione di queste cartelle sta nella progettazione della struttura delle cartelle stesse, per mantenere dati di qualunque tipo. Inoltre, problemi legati alla privacy e alla confidenzialità con i pazienti sta rallentando di molto lo sviluppo di questa tecnologia.

Un'altra sfida da superare consiste nell'incorporare l'ampia quantità di ricerche cliniche pubblicate su base continua nei *CDSS*. In un anno vengono pubblicate decine di migliaia di studi clinici e, attualmente, ciascuno di questi studi deve essere letto e valutato manualmente.

1.4 Microservizi

Negli scorsi decenni l'*architettura multi-strato* era considerata l'architettura fondamentale da seguire per costruire sistemi software. Secondo questo pattern di progettazione le funzionalità del sistema software venivano separate su più strati che comunicano tra loro. In questa architettura il sistema software, anche se suddiviso in strati, risulta essere un *sistema monolitico*. I sistemi monolitici sono in termini di performance migliori rispetto ad altri tipi di sistemi, però risultano inefficaci nel momento il cui la complessità del sistema aumenti notevolmente. Moltissimi sono gli svantaggi nella costruzione di un sistema monolito:

- impossibile conoscenza dell'intero sistema;
- difficile la collaborazione nello sviluppo del sistema;
- difficoltà nell'aggiornare il sistema con nuove funzionalità;
- lievitazione dei tempi e dei costi per le modifiche del sistema;
- scalabilità del sistema ridotta o addirittura assente.

Con l'arrivo del *cloud computing*, l'organizzazione agile delle aziende in piccoli team di sviluppo e l'aumento della complessità dei sistemi ha fatto si che l'*architettura multi-strato* viene sostituita da un'**architettura a microservizi**. Questo stile architetturale consiste nello sviluppare una singola applicazione come insieme di piccoli servizi, ciascuno dei quali viene eseguito da un proprio processo e comunica con gli altri servizi attraverso la rete utilizzando, solitamente, il protocollo **HTTP**. A differenza delle applicazione monolitiche, ad ogni servizio viene affidata una sola funzionalità da eseguire, promuovendo il motto **less is more**.

Ogni microservizio, quindi, è un'entità separata che viene generalmente pubblicata su una piattaforma *Paas*. La comunicazione tra i servizi avviene attraverso la rete in modo da evitare qualsiasi tipo di accoppiamento tra servizi. Il microservizio offre all'esterno solo un **Application Programming Interface**

(**API**), nascondendo il dettaglio di come le funzionalità sono implementate, lo specifico linguaggio o tecnologia utilizzati. Questo mira a far si che il cambiamento di ciascun microservizio non abbia impatto sugli altri o su chi sta utilizzando il servizio.

1.4.1 Vantaggi

I vantaggi portati con l'introduzione di questa architettura sono molti.

- Rilascio incrementale del software semplificato. In questo modo il cliente ha la possibilità di provare il sistema già nelle prime fasi di sviluppo.
- Riduzione dei tempi di rilascio del software. Ogni singolo servizio è autonomo rispetto agli altri, di conseguenza può raggiungere l'ambiente di produzione in modo indipendente dagli altri, senza che tale attività abbia effetti drammatici sul resto del sistema.
- Facile l'aggiornamento delle tecnologie per stare al passo con il mercato. Essendo la grandezza del singolo servizio ridotta è possibile riscriverlo in poco più di due-tre settimane di lavoro.
- L'utilizzo di tecnologie ad hoc per lo sviluppo di specifiche funzionalità. Per esempio, ogni servizio potrebbe utilizzare un database diverso in base alle proprie caratteristiche, in modo da promuovere l'efficienza.
- L'intero sistema deve essere **fault tolerant**, ovvero deve essere in grado di funzionare in presenza di guasti in uno dei servizi. Questa caratteristica è molto importante, poiché questi sistemi comunicano attraverso la rete che è un canale di comunicazione non sicuro.
- Scalare un microservizio è più semplice ed economico rispetto a scalare un sistema software monolitico.
- L'attività di deployment è un'attività meno rischiosa rispetto a sistemi con la vecchia architettura. Ogni singolo servizio può essere *messo online* in modo indipendente e qualora si verificasse un problema sarebbe facile isolarlo e intraprendere azioni di rollback.

1.4.2 Svantaggi

Sicuramente l'approccio a mircoservizi non è una panacea, ma espone anche degli svantaggi.

- Il testing di questi sistemi risulta più complesso, in quanto per testare un servizio è necessario considerare anche tutti gli altri da cui esso dipende.

- È abbastanza complesso capire quale sia la grandezza migliore di un servizio, in termini di funzionalità e di linee di codice. Un servizio troppo grande non è corretto, dato che esce fuori dalle linee guida dell'architettura, però costruire troppi piccoli servizi complicherebbe le interazioni tra essi.

1.4.3 Approcci architetturali: SOA, ROA

Esistono due approcci architetturali per poter sviluppare servizi web e, anche se l'obiettivo è pressoché identico con l'adozione del Web come piattaforma di elaborazione, la loro soluzione offerta è totalmente differente. La differenza principale è la visione del Web come piattaforma di elaborazione: l'architettura applicativa SOA (*Service Oriented Architecture*), basata sul protocollo **SOAP** (*Simple Object Access Protocol*), ha una visione del Web sotto forma di servizi, mentre l'architettura **ROA** (*Resource Oriented Architecture*) propone una visione del Web incentrata sul concetto di risorsa.

Un Web Service **ReSTful** immagazzina un insieme di risorse sulle quali un *client* può effettuare delle operazione attraverso i metodi offerti dal protocollo *HTTP*. Si contrappone la struttura dei Web Service basati su *SOAP*, poiché questi ultimi prevedono un insieme di metodi richiamabili da remoto da parte di un client. Il protocollo SOAP definisce una struttura dati per lo scambio di messaggi tra applicazioni, riproponendo parte di quello che il protocollo HTTP fa già. SOAP utilizza HTTP come protocollo di comunicazione, ma non è vincolato da esso, dal momento che può far uso di altri protocolli come, ad esempio, *FTP*. ReST invece sfrutta e utilizza a pieno le potenzialità del

protocollo applicativo HTTP, non aggiungendo nulla a quanto è già esistente sul Web per consentire ad applicazioni remote di interagire.

Oltre a quanto precedentemente detto, i Web Service basati su SOAP prevedono lo standard **WSDL**, *Web Service Description Language*, per definire l'interfaccia di un servizio. Ciò conferma il tentativo di adattare al Web l'approccio di interoperabilità basato su chiamate remote. ReST non prevede esplicitamente nessuno standard per descrivere come interagire con una risorsa, ma le operazioni sono implicite nel protocollo HTTP.

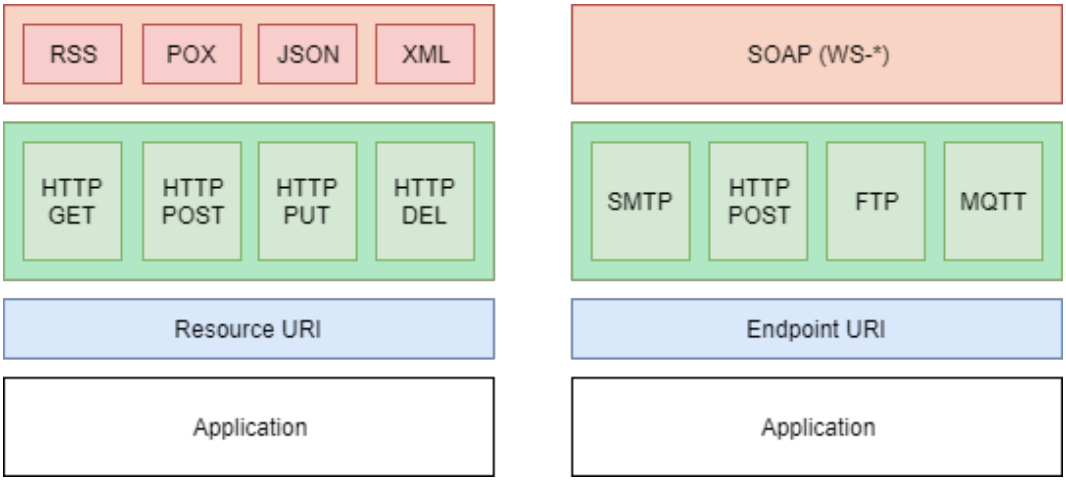

Figura 1.3: Differenze tra architettura REST e SOAP

1.4.4 Java API per servizi Web ReSTful

Nella versione Java EE 6 è stato introdotto **JAX-RS**, che sono delle API per supportare la creazione di servizi Web che seguendo i concetti del pattern *ReST*. Queste api fanno uso di annotazioni, introdotte in Java SE 5, in grado di semplificare lo sviluppo e il deployment dei servizi Web e dei loro client.

JAX-RS 2.0 è la versione attuale di queste API ed è stata rilasciata il 12 Giugno 2013.

JAX-RS attualmente è il core di moltissimi framework, nati per semplificare ulteriormente lo sviluppo di servizi Web ReST. Di seguito viene mostrato un elenco dei framework più diffusi e famosi.

- Jersey, implementazione open source di Sun, compagnia acquistata da Oracle.
- RESTeasy, implementazione di JBOSS.
- Restlet, framework open source leggero e completo.
- Apache CXF, framework opensource.

Capitolo 2

Motivazioni, obiettivi e requisiti

2.1 Motivazioni

La necessità di sviluppare strumenti di modellazione predittiva per il supporto decisionale clinico nasce dal fatto che questi strumenti fanno fatica a rimpiazzare i sistemi attuali, dato che suscitano scetticismo ai professionisti sanitari.

In più, a fronte di questo stato dell'arte, si evince che l'utilizzo del *machine learning as a service* è una delle tecnologie più all'avanguardia e tra le migliori da seguire in ambito software.

2.2 Obiettivi

L'obiettivo di questo lavoro è quello di progettare un'infrastruttura che possa permettere lo sviluppo di un DSS in ambito clinico sotto forma di insieme di servizi. Il sistema deve essere in grado di effettuare una valutazione del rischio affrontato dai pazienti con patologie particolari che seguono specifiche cure.

Lo sviluppo di questo sistema ha il fine di raggiungere diversi obiettivi.

- Supportare sia la valutazione del rischio degli aspetti sanitari della popolazione, sia la valutazione del rischio dei pazienti.

- Garantire ai medici la possibilità di accedere al servizio nelle rispettive workstation.
- Utilizzare librerie open-source per permettere il miglioramento continuo del software e per dare flessibilità alla politica di licenza e alla ri-configurazione del sistema.
- Consentire ai data scientist e ai medici di lavorare fianco a fianco o separatamente.
- Permettere ai medici di utilizzare modelli predittivi già convalidati e presenti nei loro sistemi correnti.
- Dare la possibilità ai data scientist di sperimentare nuovi modelli di predizione, indipendenti da quelli inseriti nel CDSS. I modelli creati potranno essere in futuro valutati per poter essere utilizzati dalle cliniche.

2.3 Requisiti

Un'infrastruttura di questo tipo offre molteplici funzionalità agli utenti, intesi come data scientist e professionisti sanitari. Prima di poter progettare un'architettura di sistema e passare alla fase implementativa, è indispensabile effettuare un'accurata analisi dei requisiti. In questa sezione vengono descritti i requisiti che il sistema deve rispettare, suddividendoli, per una miglior comprensione, in 3 categorie: **funzionali**, **non-funzionali** e **tecnici**.

2.3.1 Requisiti funzionali

Caricamento di un modello

Il DSS deve consentire il caricamento di modelli già allenati. Il sistema deve essere in grado di supportare la maggior parte dei modelli; ma nella fase iniziale si è deciso di accettare solamente modelli con formato R o PMML. Per rendere l'operazione facile da eseguire, il caricamento dovrebbe avvenire attraverso l'upload di un singolo file da parte dell'utente. Oltre al file del

modello, l'utente dovrà fornire una descrizione dettagliata dello stesso in modo tale che sia di facile comprensione e utilizzabile da altri utenti.

Questa funzionalità risulta particolarmente adatta a tutte le cliniche che hanno intenzione di usufruire del sistema facendo uso solamente di propri modelli pre-addestrati. In questo modo ogni clinica potrebbe testare i propri modelli sfruttando i servizi offerti attraverso il cloud e valutare i risultati ottenuti.

Conversione di un modello

Attualmente sul mercato sono disponibili diversi framework per la costruzione di modelli predittivi e ogni modello ha una propria rappresentazione; questo rende difficoltosa l'interazione tra cliniche che sfruttano framework differenti, poiché un modello può essere riconosciuto ed utilizzato solamente dallo stesso framework con cui è stato creato.

Il sistema che si vuole costruire mira ad una maggior interazione tra sistemi diversi, in modo da permettere a cliniche che lavorano con framework differenti di collaborare.

Quindi, una funzionalità indispensabile è quella relativa alla traduzione del formato dei modelli. Ogni modello, infatti, prima di essere memorizzato, deve essere tradotto nel formato di rappresentazione PMML, ovvero un formato standard che consente ad un modello di essere condiviso tra sistemi eterogenei.

Nel nostro caso specifico, qualora il modello caricato fosse di tipo R, questo verrebbe convertito automaticamente dal sistema nel corrispondente modello PMML.

Recupero dei modelli

Il sistema devo fornire la lista dei modelli disponibili e le loro informazioni di base quali tipologia e significato delle variabili indipendenti e dipendenti, algoritmo utilizzato per la costruzione del modello e istruzione per applicare il modello.

Questa funzionalità consente ad una clinica di selezionare il modello migliore da utilizzare per il proprio scopo. Ad esempio può accadere che una

clinica, specializzata in cardiologia, metta a disposizione un modello cosi che un'altra clinica, specializzata in ambito diverso, possa utilizzarlo nel momento in cui ha bisogno di valutare dati provenienti da un'analisi cardiologica e non ha un modello a disposizione. In questo caso il modello della prima clinica risulta essere di grande utilità alla seconda.

Download dei modelli

Tutti i modelli possono essere recuperati in qualsiasi momento da chiunque attraverso l'operazione di download. Ogni modello deve essere scaricabile attraverso un unico file; nel caso in cui il modello fosse rappresentato da più file questi dovrebbero essere impacchettati in un unico file zip.

Il download di un modello permette agli utenti di poter effettuare modifiche locali allo stesso senza influenzare il corrispettivo online; in questo senso l'utilità sta nel fornire ai data scientist la possibilità di effettuare esperimenti su modelli addestrati da altri provider, valutandone i risultati.

Cancellazione di un modello

I modelli possono essere eliminati qualora venissero scoperti errori accidentali nei dati o quando il loro utilizzo risulterebbe essere inutile. Per ragioni di sicurezza l'operazione di cancellazione può essere effettuata solamente dal provider che ha caricato o creato il modello, oppure può essere notificata al provider da parte di un altro utente.

Applicazione di un modello

Il DSS deve consentire all'utente di applicare uno dei modelli disponibili a un set di dati compatibili con la struttura del modello. Il risultato della valutazione deve essere restituito come file, che contiene i dati presi in input e la loro predizione.

I dati in input per la fase di valutazione di un modello potrebbero essere utilizzati per aggiornare il modello stesso. In questo modo i modelli potrebbero effettuare un apprendimento continuo, rendendolo più stabile e affidabile.

Creazione di un modello

Oltre alla possibilità di caricare un modello pre-addestrato, l'utente può creare modelli predittivi attraverso il sistema, fornendo ad esso il file di training set dei dati. Per la costruzione del modello, si dà la possibilità all'utente di selezionare quale algoritmo utilizzare tra quelli disponibili per la fase di training. Il modello può essere riaddestrato in qualsiasi momento da parte del provider.

Snapshot di un modello

Il provider può richiedere di effettuare uno snapshot a un suo modello in qualsiasi momento. Lo snapshot corrisponde a una copia di un modello: il modello continuerà ad essere allenabile con altri dati, mentre lo snapshot sarà congelato alla stato attuale e potrà essere solamente applicato. Questa funzionalità è utile nel momento in cui si ritiene che i risultati dell'applicazione di un modello tendano all'ottimo.

Accesso locale o globale

Ogni modello può avere un accesso *locale* o *globale*; nel primo caso, il training del modello può essere effettuato solo dal provider che lo ha caricato o creato, nel secondo caso, il modello è addestrabile con dati provenienti da qualsiasi fonte. Per ovvie ragioni di affidabilità di un modello, l'accesso di default sarà quello *locale*.

Apprendimento online o offline

L'apprendimento di un modello può essere impostato come online oppure offline. Nel primo caso, il modello non effettua il training usando solamente il training set, ma anche i dati risultanti dalle predizioni consentendo di creare un apprendimento continuo. Nel secondo caso, un modello viene congelato appena terminata la fase di apprendimento.

2.3.2 Requisiti non funzionali

Un requisito non funzionale di rilievo è il rispetto della privacy e della sicurezza, visto che i dati dei pazienti, utilizzati per la fase di training, sono dati sensibili che non possono essere divulgati pubblicamente. I dati utilizzati per il training rimarranno nascosti a chi usufruisce dei modelli e la comunicazione utente-servizio sarà effettuata attraverso canali sicuri costruiti con l'utilizzo di apposite tecnologie.

2.3.3 Requisiti tecnici

I requisiti tecnici, talvolta indispensabili in grandi sistemi come questo, sono di varia natura. Sarebbe utile costruire un sistema il più flessibile possibile, ovvero cercare di mantenere il codice open-source il più possibile, in modo da permettere ad altri sviluppatori di contribuire e coadiuvare lo sviluppo. Il sistema dovrebbe essere in grado di lavorare con modelli costruiti in qualsiasi linguaggio e con qualsiasi framework di machine learning. Questo requisito nasce dall'esigenza di rendere indipendente la rappresentazione dei modelli da parte dei framework addetti alla costruzione di modelli, come R e Weka. Per questa ragione è stato deciso di lavorare il più possibile con modelli rappresentati dal modello PMML, che è stato brevemente citato nelle sezioni precedenti.

Infine, per rendere il servizio usufruibile da qualsiasi client indipendentemente dal suo linguaggio di implementazione, le API ReST lavorano utilizzando l'XML e il JSON per la rappresentazione dei dati.

Capitolo 3

Architettura di Sistema

In questo capitolo viene descritta l'architettura di sistema seguendo e rispettando l'analisi dei requisiti effettuata nel capitolo precedente. La descrizione fornita mira a mostrare il design infrastrutturale del sistema complessivo, tralasciando il più possibile i dettagli implementativi.

Un'architettura ben progettata consente di mantenere il sistema il più modulare possibile, inoltre semplifica le fasi di modifica ed estensione nel processo di implementazione.

Il sistema è stato pensato come un'infrastruttura in esecuzione sul cloud, capace di fornire diversi servizi con lo scopo finale di aiutare i data scientist e i professionisti sanitari nell'uso di modelli predittivi per i propri scopi. L'architettura si ispira a quelle di micro-servizi, introdotte nello stato dell'arte.

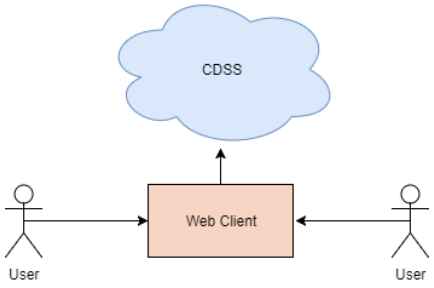

Figura 3.1: Comunicazione tra utente e CDSS

3.1 Componenti

Il sistema è costituito da diversi moduli, ognuno dei quali è un servizio con un compito specifico e limitato accessibile da end-point ReSTful. I moduli interagiscono tra loro, offrendo un'interfaccia accessibile sia dall'esterno sia dagli altri servizi. La somma di tutte le funzionalità va a costituire il CDSS.

3.1.1 Servizio per l'apprendimento

Il modulo adibito al learning permette la costruzione di modelli predittivi basati sui dati dei pazienti e ha anche il compito di effettuare *online-learning*. Questo servizio è in comunicazione con quello che si occupa di salvare dati su disco, poiché una volta prodotto un modello, quest'ultimo deve essere salvato nel sistema.

3.1.2 Servizio per la predizione

Il servizio per l'applicazione dei modelli ha il compito di effettuare predizioni sui dati passati in input attraverso l'utilizzo di un modello caricato nel sistema o creato direttamente da quest'ultimo. Questo modulo interagisce, da un lato, con il modulo di learning per effettuare, qualora fosse possibile, online-learning e dall'altro con il servizio database per recuperare i modelli salvati.

3.1.3 Servizio database

Il modulo in questione gestisce il caricamento, il salvataggio, il download e la cancellazione dei modelli e delle loro informazioni. Il servizio lavora salvando dati su file e su un database e accetta in input nella fase di caricamento solo modelli pre-addestrati che rispettino il formato R o PMML. Nel momento in cui viene fornito un modello R, questo servizio comunica con quello di traduzione per tradurre il modello nel corrispettivo modello PMML.

3.1.4 Servizio per la traduzione dei modelli

Il servizio che si occupa della traduzione ha il compito di tradurre modelli R in modelli PMML. Questo è in comunicazione solamente con il servizio database per salvare modelli tradotti.

3.2 Attori

È possibile identificare i principali attori all'interno di questa architettura in base alle entità che usufruiscono dei servizi. Lo sono sicuramente i data scientist e i medici poiché usufruiscono del sistema, ma lo è anche un software esterno che accede ai servizi attraverso le API.

Per di più, anche ogni servizio potrebbe essere considerato un attore, dato che per svolgere il proprio compito, nella maggior parte dei casi, deve interagire con un altro servizio.

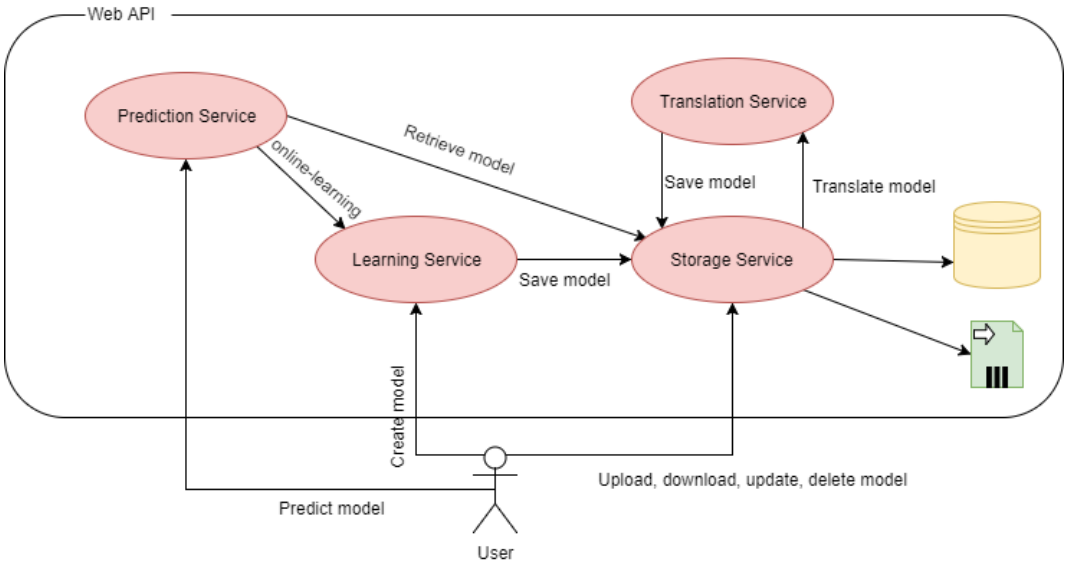

Figura 3.2: Architettura complessiva del sistema

3.3 Vantaggi dell'architettura

I vantaggi dell'architettura descritta nella sezione precedente possono essere divisi in due categorie.

- Vantaggi legati alla natura intrinseca dei micro-servizi.
- Vantaggi derivati dal dominio applicativo in cui si applica l'architettura.

Basandosi su un'architettura a micro-servizi, i vantaggi rispecchiano quelli di un'architettura ReSTful, di conseguenza il sistema è facilmente scalabile e modulare. Infatti, nel caso in cui ci si accorgesse che un servizio è in sovraccarico, si potrebbe duplicare un'istanza di un servizio senza che il sistema ne risenta o se ne accorga. Nel caso in cui un servizio sia indisponibile, cosa che accade ad esempio se il servizio di learning sta lavorando con big-data, gli altri servizi rimangono disponibili. Inoltre la fase di deployment di ogni modulo risulta essere indipendente dalle altre, in questo modo è possibile costruire un ambiente ideale per l'esecuzione di ogni servizio.

Vantaggi importanti sono legati anche al dominio applicativo in cui è utilizzato il sistema. Nel caso clinico si dà la possibilità ai medici di utilizzare i loro modelli pre-addestrati, ma allo stesso tempo è concesso ai data scientist di sperimentare e testare nuovi modelli, promuovendo il principio di **separation of concerns**.

Capitolo 4

Panoramica sulle tecnologie

In questo capitolo viene fatta una panoramica sulle tecnologie utilizzate per lo sviluppo del sistema. La scelta di ogni soluzione mira alla qualità del sistema in termini di performance, scalabilità, manutenibilità e anche per garantire maggior portabilità tra sistemi eterogenei.

4.1 Framework ReST

Sviluppare un servizio Web ReSTful in Java che supporti una gran varietà di rappresentazione dei tipi di dato e che gestisca la comunicazione client-server non è molto semplice. Per semplificare questo sviluppo sono state progettate le **JAX-RS API**.

Jersey è un framework di ottima qualità per lo sviluppo di servizi ReSTful che si basa su JAX-RS e ne fornisce il supporto. Inoltre, il framework estende le caratteristiche di JAX-RS aggiungendo nuove caratteristiche ed utilità per semplificare ulteriormente lo sviluppo di servizi e client.

4.1.1 Come utilizzarlo

Jersey può essere recuperato in diversi modi poiché è una libreria Java, ma in questo progetto è stato utilizzato attraverso l'uso di Maven, cioè un tool per la gestione dei progetti basato sul concetto di **project object model**

(**POM**). Molti sviluppatori utilizzano Maven per *buildare* i loro progetti e per includere e gestire dipendenze con altri progetti.

Per creare un servizio, importando la libreria con Maven, si ha bisogno di inserire nel file POM la seguente dipendenza.

```
<dependency>
    <groupId>org.glassfish.jersey.containers</groupId>
    <artifactId>jersey-container-servlet</artifactId>
    <version>2.26</version>
</dependency>
```

Il framework Jersey è stato progettato come insieme di moduli componibili, che offrono funzionalità differenti in base alle necessità di cui un servizio ha bisogno. Infatti applicazioni differenti potrebbero richiedere moduli differenti. In più, le applicazioni sviluppate con questo framework devono includere, dove necessario, tutte le librerie su cui i moduli di Jersey si basano a loro volta.

4.1.2 Creare un server

Creare un server con Jersey è veramente molto semplice e richiede pochissime righe di codice. Per poter creare un servizio si ha bisogno di un *URI* da contattare, ovvero un indirizzo ed una porta in cui il server è in ascolto. Inoltre, siccome si sta sviluppando un servizio ReST, è necessario indicare al servizio quali sono le risorse del nostro applicativo a cui un client può accedere.

```
ResourceConfig resourceConfig = new ResourceConfig(MyResource.class);
final HttpServer server =
    GrizzlyHttpServerFactory.createHttpServer(BASE_URI,
    resourceConfig, false);
Runtime.getRuntime().addShutdownHook(new Thread(new Runnable() {
  @Override
  public void run() {
    server.shutdownNow();
  }
}));
server.start();
```

La classe **ResourceConfig** si occupa di raccogliere tutte le informazioni legate alle risorse disponibili. In questo esempio, è stato detto al servizio che ha a disposizione una sola risorsa, rappresentata dalla classe **MyResource**. È possibile passare in input alla classe *ResourceConfig* un *Set* di classi che rappresentano l'insieme delle risorse.

4.1.3 Risorse e componenti

Una risorsa è identificata da una semplice classe Java che fa uso di annotazioni speciali. L'annotazione indispensabile per il riferimento alla risorsa è **@Path**, che indica il percorso relativo per raggiungere una determinata risorsa. Si può accedere ad ogni risorsa attraverso chiamate HTTP ed ogni metodo del protocollo può determinare operazioni differenti. Le operazioni effettuate dalle risorse sono rappresentate da metodi, anche loro associati ad una o più annotazioni.

Le annotazioni che si possono associare a metodi sono molteplici, le più comuni sono:

- **@Path**: aggiunge un percorso più specifico a quello della risorsa;
- **@GET, @POST, @PUT, @DELETE**: definiscono con quale metodo HTTP può essere richiesta un'operazione;
- **@Consume**: specifica il tipo di rappresentazione che una risorsa può ricevere e consumare.
- **@Produce**: specifica il tipo di rappresentazione che una risorsa può produrre e mandare al client. Solitamente vengono restituiti file testuali con rappresentazione JSON o XML.

Per rispettare le linee guida di un buon servizio ReSTful è preferibile inserire nei percorsi delle risorse solamente sostantivi e non verbi, questo perché l'architettura si basa sul concetto di risorsa. Ad esempio, per scaricare un file in formato zip sarebbe da evitare parole come *download*. L'URI ideale da utilizzare potrebbe essere "http://example.com/file/zip".

Un esempio più o meno completo di una risorsa utente potrebbe essere:

```
@Path("user")
public class UserResource {

  private Map<String,User> map = new HashMap<>();

  @GET
  @Produces(MediaType.TEXT_PLAIN)
  public String getName(@FormParam("id") String userID) {
    return this.map.get(userID).getName();
  }

  @PUT
  @Consume(MediaType.APPLICATION_JSON)
  @Produce(MediaType.APPLICATION_JSON)
  public Response addUser(User user){
   String id = generateRandomID();
   this.map.put(id, user);
   return Response.ok().build();
  }

}
```

Per fornire servizi attraverso il Web è necessario garantire anche sicurezza per tutelare gli utenti e per mantenere le informazioni confidenziali. Jersey dispone anche di molteplici moduli per costruire uno strato di sicurezza, che sono molto utili nella costruzione di servizi che mettono a disposizione molte funzionalità basate sulla separazione dei permessi. Infatti, attraverso il framework è possibile stabilire quali sono i permessi associati ad ogni utente e quali permessi sono necessari per accedere a una determinata funzionalità. Nel mondo clinico, queste pratiche sono molto diffuse, per esempio un medico è più privilegiato in termini di permessi rispetto ad un infermiere che a sua volta è più privilegiato di un oss.

4.2 Framework front-end

La fase di design e di sviluppo di un client Web può essere effettuata attraverso diversi framework. Uno dei framework che ha riscosso molto successo negli ultimi anni è **Angular**. Questo framework open-source è sviluppato da Google e da una vasta community ed è utile per la costruzione di applicazioni Web complesse. Angular è un'evoluzione di AngularJS ed è basato sul linguaggio di programmazione TypeScript.

Angular sfrutta il patter **Model-View-Controller** (**MVC**) per progettare applicazioni Web. I componenti principali di un'applicazione sono 4.

- **View**: rappresenta quello che l'utente vede, l'interfaccia grafica generata a partire da un template HTML e dai suoi file associati.
- **Direttiva**: è un componente che estende l'HTML con tag e attributi personalizzati ed è l'unico componenti autorizzato a manipolare il **Document Object Model** (**DOM**) via Javascript.
- **Componente**: un componente rappresenta il controller della view, quindi ha lo scopo di passare i dati alla View e di offrire funzionalità.
- **Servizio**: questo è un oggetto che fornisce funzionalità indipendenti dalla GUI, come ad esempio richieste al server via HTTP. Solitamente un servizio deve essere utilizzato per effettuare qualsiasi operazione che potrebbe richiedere del tempo.

Le caratteristiche che lo hanno reso uno dei migliori framework sono molteplici.

- **Modularità**: il sistema è diviso in moduli in modo da rendere il core più leggero e veloce;
- **Riuso**: le entità di un progetto Angular sono costituite da componenti. La divisione in componenti permette un miglior riuso del codice, aumentando la manutenibilità e la leggibilità.

- **Reattività**: Angular supporta la programmazione reattiva attraverso l'utilizzo di RxJS.
- **Asincrono**: la compilazione di una pagina Web è completamente asincrona.

In sintesi Angular è un framework versatile che permette di creare applicazioni Web che possono essere visualizzate correttamente su qualsiasi tipo di device, sia Desktop sia mobile.

Figura 4.1: Logo del framework Angular

Capitolo 5

Architettura del software

Dopo un'analisi dell'architettura di sistema si procede con la descrizione delle componenti software che la realizzano. In questo capitolo vengono descritte le classi principali evidenziandone il comportamento e l'interazione con le altre componenti.

Ogni classe ha il compito di rappresentare le funzionalità di un determinato servizio, promuovendo la modularità, l'estendibilità e la comprensibilità del progetto. Il dettaglio della descrizione è ad un livello tale da far comprendere la struttura del codice, senza scendere troppo nei particolari implementativi.

5.1 Organizzazione in Package

Il progetto è composto da diversi package, ognuno dei quali racchiude le classi che implementano tutte le funzionalità di un singolo servizio. Questo aumenta la modularità del progetto, rendendo più semplice il cambiamento di singole parti, e semplifica l'esecuzione di eventuali modifiche da apportare.

I package del progetto sono:

- **database**: contiene i sorgenti delle classi che hanno il compito di gestire la scrittura e lettura dei dati sia sul file system, sia sul database;
- **learning**: questo package contiene le classi per effettuare la creazione di modelli attraverso algoritmi di learning;

- **translator**: all'interno del package sono presenti le classi che permettono di effettuare la traduzione di modelli R in modelli PMML;
- **evaluator**: contiene i sorgenti delle classi per applicare un modello esistente a dei dati;
- **main**: questo package contiene semplicemente la classe **Main**, che ha il solo compito di far avviare il servizio ReSTful;
- **model**: contiene le classi che rappresentano gli oggetti principali usati dall'applicazione;
- **parser**: contiene i sorgenti delle classi per recuperare i dati presenti in file testuali;
- **resorces**: contiene tutte le classi che rappresentano le risorse del servizio ReSTful;
- **utils**: le classi al suo interno hanno il compito di fornire utilità aggiuntive;
- **view**: le classi contenute da questo package consentono di visualizzare i dati presenti all'interno dei dataset. I dati sono mostrati attraverso l'utilizzo di scatter-plot, che forniscono una buona rappresentazione della distribuzione spaziale dei dati;

5.2 Libreria Smile e classi di utilità

Prima di fornire una descrizione delle principali classi, è necessario delineare alcune componenti della libreria *Smile* [8] e altre componenti di utilità che sono di fondamentale importanza per il funzionamento del sistema.

5.2.1 AttributeDataset

La maggior parte degli algoritmi di learning messi a disposizione da questo framework prendono in input valori di tipo array di double, ma questi spesso sono incapsulati all'interno della classe **AttributeDataset**.

Questa classe rappresenta la collezione di dati sotto forma matriciale, in cui ogni colonna della tabella rappresenta una particolare variabile e ogni riga corrisponde ad un record del dataset. Inoltre, ad ogni colonna della matrice è associata una classe **Attribute** che ne descrive alcune informazioni. I metadati che si possono ricavare da quest'ultima classe sono il nome della variabile e una sua descrizione, il tipo di attributo (*nominale*, *ordinale*, *stringa*) e il peso della variabile. Alcuni algoritmi permettono di considerare per ogni variabile un peso, cioè l'influenza della variabile sul calcolo finale.

Infine, il fatto che i dati vengano salvati in una matrice consente l'accesso diretto, e quindi è migliore se si vuole accedere frequentemente a diverse istanze.

5.2.2 Parser

È stata realizzata una classe **Parser** per l'estrapolazione dei dati contenuti in un file. Il costruttore di questa classe prende in input l'InputStream del file, il tipo di file e, se presente, l'indice della colonna che rappresenta la variabile dipendente. Il tipo di file è rappresentato da un **enum FileType**, che, attraverso l'estensione del file, riesce a riconoscere il *FileType* associato.

Il parser accetta solo due tipi di formato: **CSV** e **ARFF**. Sono entrambi file testuali, ma con regole sintattiche differenti. Nel primo caso i dati vengono delimitati solitamente da virgole e/o punti e virgole, mentre nel secondo formato i dati hanno la struttura definita dal framework *Weka*[1].

La classe *Parser* fornisce un'interfaccia che mette a disposizione un solo metodo: **parse**. La chiamata a questo metodo inizia la fase di parsing e una volta terminata restituisce un *AttributeDataset*.

[1]Weka è un tool utilizzato principalmente per svolgere compiti di data mining. Questo tool contiene una vasta collezione di algoritmi di machine learning che possono essere applicati ai dati oppure utilizzati attraverso Java. `https://www.cs.waikato.ac.nz/ml/index.html`

5.3 Classi del sistema

In questa sezione vengono descritte le classi principali che compongono il sistema, ponendo attenzione alle classi che modellano il comportamento di ogni servizio.

5.3.1 Salvataggio dei dati

Il package **database** contiene due classi che hanno il compito di gestire la memorizzazione e la cancellazione delle informazioni dei modelli. **DBManager** consente di creare e restituire la connessione con il database; al suo interno troviamo diversi metodi per effettuare tutte le query disponibili: salvataggio delle informazioni di un nuovo modello, aggiornamento delle informazioni, cancellazione ecc.

Ogni metodo prende in input la connessione al database e i dati necessari per l'esecuzione della query. Il corpo di ogni metodo costruisce la query attraverso un PreparedStatement, in questo modo si riesce a evitare un attacco di tipo **SQL Injection**[2]. Evitare un attacco di questo genere è fondamentale in sistemi che lavorano con dati sensibili, dato che non possono essere divulgati pubblicamente.

I modelli che vengono caricati nel sistema devono essere salvati nel file system. La classe **FileManager** realizza questo compito gestendo le operazioni di lettura, scrittura e cancellazione dei file. La classe lavora su un percorso specifico definito dall'utente e al suo interno crea una serie di sotto-cartelle per salvare i vari modelli. Il nome di ogni sotto-cartella è rappresentato dall'id del modello e al suo interno vengono salvati i modelli predittivi generati.

La lettura e la scrittura dei file, in questo caso di modelli, vengono effettuate attraverso la libreria **XStream** che si occupa di effettuare marshalling e unmarshalling di qualsiasi oggetto Java. L'operazione di marshalling consente di

[2]Un attacco di SQL Injection consiste nell'inserimento di stringhe SQL malevole all'interno di campi di input in modo che queste ultime vengano eseguite. Lo scopo di questo attacco, per esempio potrebbe essere inviare all'attaccante il contenuto del database.

serializzare un oggetto Java in un file XML, mentre l'un-marshalling permette di creare un oggetto Java partendo dal file XML.

5.3.2 Traduzione di un modello

Il package **translator** contiene la classe **ModelTranslator** che ha il compito di tradurre un modello R nel corrispettivo modello PMML. La classe ha un costruttore che prende in ingresso l'InputStream del file del modello e implementa un'interfaccia con il solo metodo **translate**. La traduzione viene eseguita attraverso la libreria **JPMML-R**, che mette a disposizione classi di utilità per questo compito.

Per effettuare la traduzione viene creato un oggetto **RExpParser** che prende in input il file del modello R; il parser genera una *regular expression*[3] che viene utilizzata da un **converter** per creare l'oggetto della classe PMML. Il metodo **translate** restituisce l'oggetto PMML creato, che può essere salvato su file attraverso la classe di utilità **MetroJAXBUtil**.

5.3.3 Funzionalità di learning

La classe **ModelTrainer** ha il compito di implementare le funzionalità per la creazione di modelli partendo dai dati forniti dall'utente. Implementando l'interfaccia **ITrainer**, un'istanza di questa classe consente di recuperare il modello allenato e la tipologia del modello creato. Nel prototipo creato, il modello è rappresentato da un insieme di cluster a cui è assegnato un classificatore. (Capitolo 7)

Il costruttore accetta in input i dati per la fase di apprendimento e il tipo di modello che si vuole creare. Gli algoritmi di clustering al momento supportati sono due (**KMeans** e **XMeans**), ma il supporto è facilmente estendibile a tutti gli algoritmi di clustering partizionale offerti da *Smile*. Gli algoritmi di clustering appartenenti a questa famiglia consentono di partire da un unico cluster e suddividerlo nel più corretto numero di cluster.

[3]Una regular expression è una sequenza di simboli che identificano un insieme di stringhe.

5.3.4 Applicazione di un modello

L'applicazione di un modello viene effettuata dalla classe **ModelApplier** che implementa l'interfaccia **IApplier**. Anche questa interfaccia contiene un solo metodo che permette di valutare i dati in input e di ottenere la relativa predizione attraverso un oggetto di tipo **AttributeDataset**.

La seguente classe permette l'applicazione dei modelli costruiti con il framework Smile oppure modelli PMML, infatti la classe fornisce due costruttori: uno che prende in input modelli di Smile, mentre l'altro accetta un oggetto PMML. L'operazione di valutazione è abbastanza differente in base al formato del modello, ma il risultato della valutazione dello stesso input su modelli uguali in formati differenti è identico.

Per eseguire l'applicazione di un modello *Smile* viene chiamato il metodo **predict** sul modello; il metodo prende in ingresso un array di double e restituisce in output il risultato della valutazione. Il processo viene iterato per tutti i dati da valutare. Il risultato restituito è un numero intero che indica la classe assegnata al dato in input.

La valutazione di un modello PMML è leggermente differente. Prima di tutto viene creato un **ModelEvaluatorFactory** che ha il compito di creare un **Evaluator**, cioè l'oggetto che effettivamente valuterà gli input dell'utente. Si ricorre alla creazione di una Factory, poiché l'applicazione di un modello dipende dall'algoritmo utilizzato per la creazione del modello. L'Evaluator costruito mette a disposizione il solo metodo **evaluate**, che prende in ingresso una mappa di valori in cui la chiave corrisponde al nome della colonna, mentre il valore è il valore numerico associato alla chiave. Il risultato della valutazione è a sua volta una mappa di valori in cui è stata aggiunta la predizione.

Infine, terminata la procedura di valutazione, i risultati sono aggiunti ad un *AttributeDataset* e successivamente possono essere scritti in un file CSV. Per semplificare e rendere veloce la costruzione di un file CSV è stata creata la classe **CsvBuilder** che accetta in input un *AttributeDataset* e restituisce il file sotto forma di array di byte.

5.3.5 Model

Il modello si basa su poche classi di cui una essenziale: **Model**. Più nel dettaglio, la classe *Model* descrive un modello predittivo con tutte le informazioni associate:

- **id**: rappresenta l'identificativo di un modello;
- **name**: nome dato ad un modello per poter comprendere la sua utilità;
- **providerID**: l'identificativo del soggetto che ha caricato o creato il modello;
- **provider**: nome del soggetto;
- **description**: descrizione relativa ad un modello. Nella descrizione è utile inserire tutte le caratteristiche dei dati per la creazione del modello e quale è il suo scopo nell'utilizzo.
- **xVariables**: variabili indipendenti di un modello;
- **yVariable**: variabile predicibile attraverso le variabili indipendenti;
- **trainable**: questo booleano indica se il modello è allenabile oppure congelato;
- **hasGlobalAccess**: indica se il modello è allenabile da tutti oppure solamente dal provider che lo ha creato;
- **onlineTrainable**: variabile booleana che indica se il modello è allenabile online, ovvero ogni volta che viene effettuata una valutazione, i dati in output vengono utilizzati per aggiornare il modello effettuando un'altra fase di training.
- **creationDate**, **lastUpdate**: queste due variabili sono due oggetti **Date** che rappresentano la data di creazione del modello e la data dell'ultimo aggiornamento.

5.4 Web Client

Una componente del sistema, utile specialmente per la fase di testing, è il client Web realizzato in Angular 4. Il pattern chiave del progetto client è il pattern MVC (model-view-controller), così facendo si facilita la manutenzione del codice, il suo riuso e l'eventuale aggiunta di ulteriori funzionalità. La View è costituita da file HTML che mostrano la lista dei modelli, diversi pulsanti per accedere alle funzionalità e le informazioni del modello selezionato. Attraverso fogli di stile, scritti in linguaggio CSS, la view tende ad assomigliare a un menu di uno smartphone in cui il menu è sulla sinistra, mentre sulla destra troviamo tutte le informazioni di un modello.

I controller sono rappresentati dai componenti, che consentono di mostrare i dati nella View. Un oggetto che può essere considerato parte del controller è il servizio; i servizi hanno il compito di interrogare il back-end attraverso chiamate HTTP e di restituire le informazioni al controller.

Le richieste HTTP vengono effettuate con l'utilizzo della classe **http** fornita da Angular, inoltre si fa uso della libreria **rxjs** per effettuare richieste asincrone, in moda da non bloccare il client durante una richiesta che potrebbe richiedere diverso tempo. Per risolvere richieste asincrone vengono utilizzate le Promise, cioè oggetti segnaposto che rappresentano un valore che è in attesa di essere restituito da un'operazione asincrona. Su tale oggetto possono essere definite le operazioni da effettuare una volta terminata la procedura asincrona.

Nella seguente porzione di codice viene mostra la chiamata HTTP con metodo **POST** per la creazione di un modello. Il corpo della richiesta contiene i dati di una *Form* e il risultato della richiesta è un oggetto di tipo **Result**. Quest'ultimo indica se la richiesta è andata a buon fine o meno. Il risultato viene restituito nel momento in cui si riceve la risposta dal server, senza però bloccare il client.

```
createModel(form: FormData): Promise<Result> {
  return this.http.post("http://localhost:8080/model/", form)
      .toPromise()
      .then(response => response.json() as Result)
}
```

5.5 Comunicazione tra i componenti

In questa sezione vengono illustrate le interazioni che avvengono tra i componenti per offrire alcune delle funzioni principali del sistema. Saranno utilizzati dei diagrammi di sequenza per mostrare al meglio lo scambio di informazione tra le diverse componenti.

5.5.1 Creazione di un modello

Il client Web permette, attraverso l'interfaccia fornita, di creare un nuovo modello. Per la creazione è necessario compilare una Form con i seguenti dati: il nome da assegnare al modello, una descrizione esaustiva e il training-set dei dati. Inoltre è necessario indicare al sistema qual'è l'indice della colonna della variabile dipendente, in questo modo, durante la fase di recupero dei dati, il parser riuscirà ad effettuare una separazione tra le variabili dipendenti e quella dipendete.

Una volta inviata la richiesta, il servizio di learning si occuperà prima di tutto di recuperare i dati dal training-set, successivamente contatterà il servizio per lo storage dei dati. Quest'ultimo genererà un identificativo per il nuovo modello, salverà le informazioni principali nel database e salverà il modello nel file-system.

Per concludere verrà notificata all'utente la buona riuscita dell'operazione richiesta e il risultato sarà mostrato all'utente attraverso la Web UI messa a disposizione dell'utente.

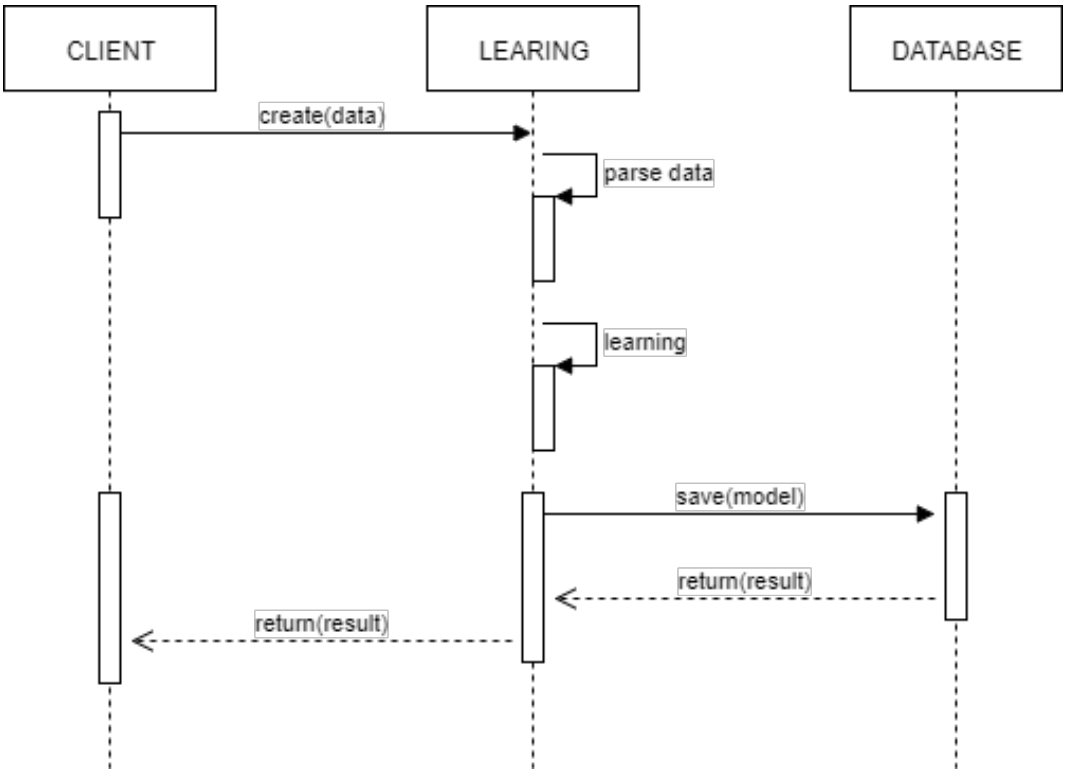

Figura 5.1: Diagramma di sequenza - Creazione di un modello

5.5.2 Caricamento di un modello

Il client permette di caricare, per il momento, solamente due formati: formato R e formato PMML. Il sistema in futuro dovrà essere in grado di accettare la maggior parte dei formati dei modelli predittivi costruiti con i più diffusi tool di machine learning, inoltre per aumentare l'interoperabilità tra le cliniche ogni modello, dove necessario, dovrà essere tradotto nel formato standard di rappresentazione PMML.

Il client invierà una richiesta al servizio per il salvataggio dei dati contenente nel body il modello pre-addestrato. Prima di effettuare il salvataggio del file, il servizio controllerà il formato di rappresentazione del modello. Nel caso in cui il modello fosse di tipo R, questo verrebbe immediatamente inviato al servizio di traduzione attraverso le API fornite. Il servizio di traduzione tradurrà il modello e lo restituirà in formato PMML al servizio per la gestione dei dati per effettuare il salvataggio.

Terminata la procedura di salvataggio, l'utente sarà notificato sull'interfaccia dedicatagli riguardo la buona riuscita dell'operazione.

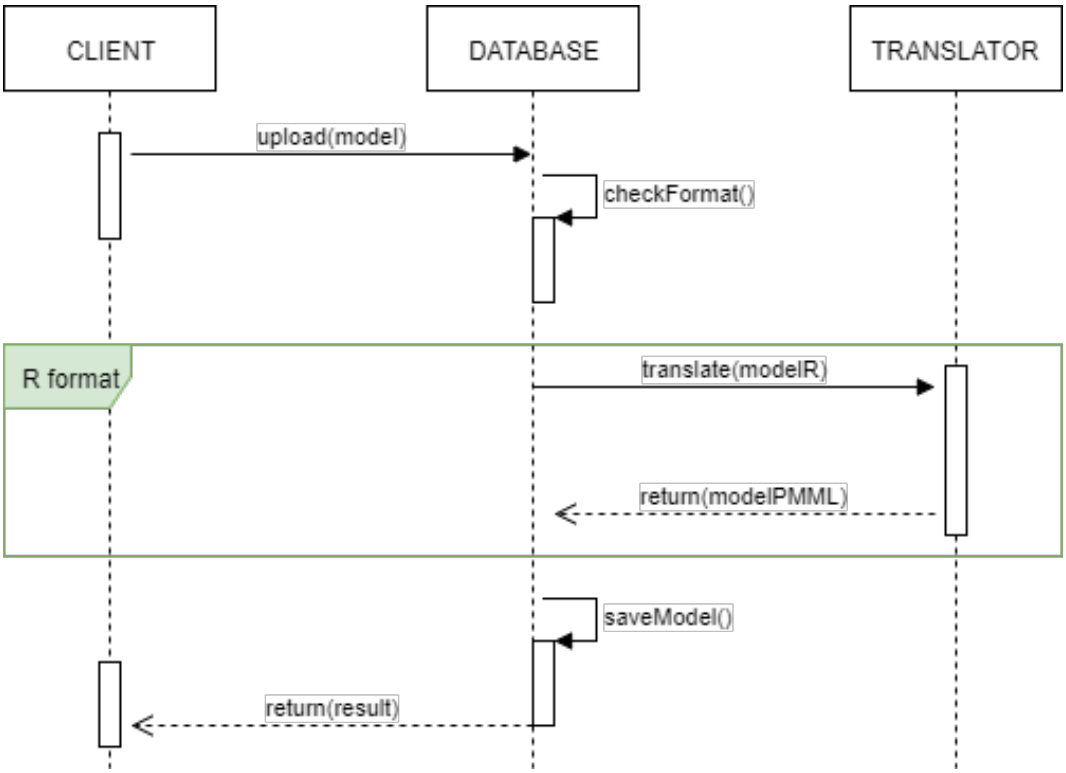

Figura 5.2: Diagramma di sequenza - Caricamento di un modello

5.5.3 Applicazione di un modello

Prima di poter applicare un modello è necessario che l'utente selezioni un modello presente nel sistema; una volta selezionato il modello desiderato l'utente potrà decidere di applicarlo ad un set di dati con l'utilizzo di apposite Form.

I dati inviati dall'utente saranno destinati al servizio di predizione che, attraverso le interazioni con il servizio adibito alla gestione dei dati, recupererà il modello selezionato e tutte le relative informazioni. Successivamente i dati in input vengono valutati con l'utilizzo del modello e viene creato un file CSV con i risultati da inviare all'utente attraverso un **StreamingOutput**.

In seguito viene verificato se il modello è abilitato o meno per effettuare online learning. Nel caso in cui fosse abilitato, possono verificarsi due diversi scenari affinché il modello venga aggiornato con nuovi dati.

- L'utente utilizza un modello di un altro provider che ha impostato un

accesso globale;

- L'utente utilizza un proprio modello.

Per concludere la procedura, il servizio di predizione invia i dati in input al servizio di learning, che contatterà il database per recuperare il vecchio training-set. Una volta uniti i vecchi dati con i nuovi, il servizio di learning aggiornerà il modello corrente e lo salverà nel file system. Purtroppo i modelli utilizzati per l'apprendimento non consentono di effettuare un apprendimento incrementale, ovvero un apprendimento in cui il modello viene aggiornato fornendo in input solo i nuovi dati, ma è necessario effettuare la procedura di learning da zero. Smile implementa un algoritmo di online learning solo per quanto riguarda i SVM[4], che aggiornano il proprio modello attraverso i dati passati in input al metodo **learn**.

[4]Un algoritmo di Support Vector Machine appartiene alla famiglia dei classificatori lineari. Il suo compito consiste nel trovare un iper-piano di separazione che divida i pattern nel miglior modo possibile.

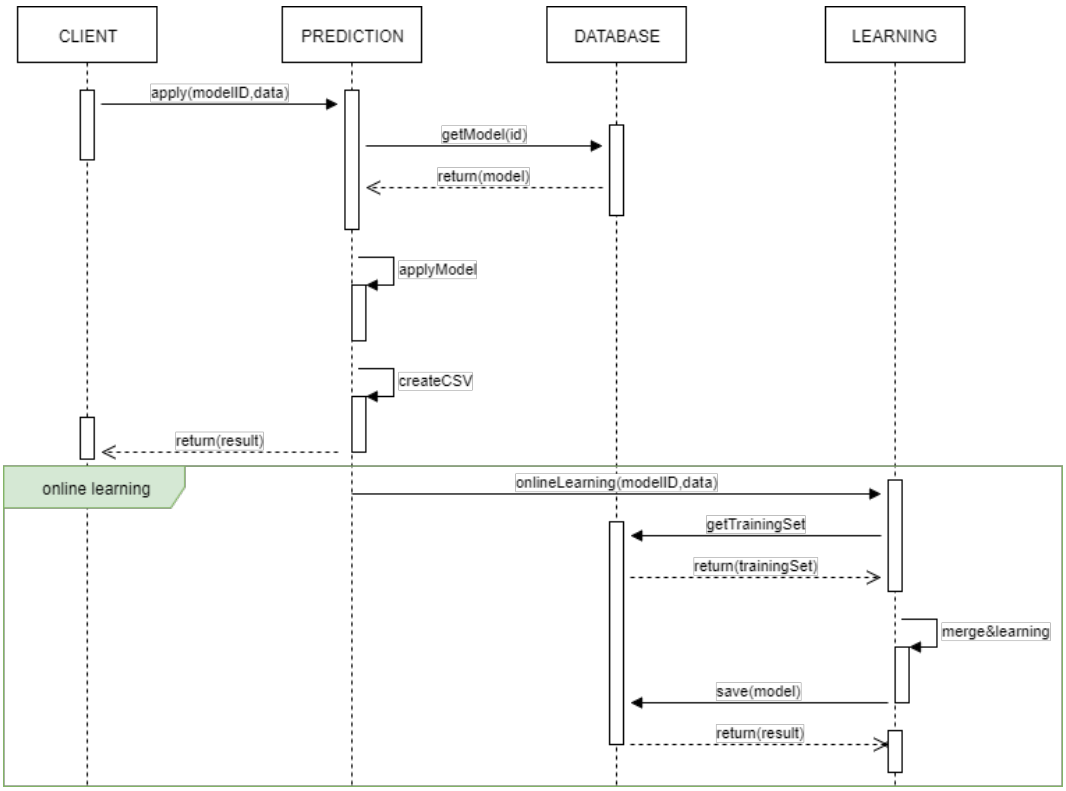

Figura 5.3: Diagramma di sequenza - Applicazione di un modello

Capitolo 6

Implementazione

In questo capitolo vengono illustrate le parti principali relative all'implementazione del progetto, evidenziando le API fornite e le classi di rilievo per il funzionamento del sistema. I linguaggi di riferimento sono Java, HTML, Javascript e CSS. Java è stato utilizzato per l'implementazione del servizio ReSTful e per tutte le componenti del back-end, mentre HTML, Javascript e CSS sono stati utilizzati per l'implementazione del client Web.

6.1 API ReSTful

La comunicazione con il servizio avviene attraverso l'utilizzo del protocollo HTTP. Per abilitare la comunicazione tra client e back-end viene utilizzato il framework Jersey. Attraverso le annotazioni di Jersey è possibile definire il metodo con cui deve avvenire la richiesta HTTP, il tipo dei parametri in entrata e il formato della risposta inviata al client.

In questo prototipo, tutte le richieste sono gestite in maniera sincrona dal servizio, ma alcune operazioni, come quella della valutazione di un modello, potrebbero essere sostituite in modo da essere eseguite in maniera asincrona. L'esecuzione asincrona lato back-end risulta particolarmente utile per gestire operazioni che potrebbero richiedere del tempo e quindi potrebbero rallentare, o addirittura bloccare, il funzionamento del servizio.

Il servizio ReSTful ha al suo interno due risorse: **ModelResource** e **ModelsResource**. Attraverso la prima risorsa è possibile effettuare azioni su un

singolo modello, mentre la seconda risorsa permette il recupero di tutti i modelli. Le risorse sono modellate da semplici classi Java a cui sono assegnate specifiche annotazioni.

6.1.1 Caricamento e creazione di un modello

Il caricamento il un modello avviene attraverso una chiamata HTTP con metodo PUT. I parametri richiesti sono il file del modello, il nome del modello e una descrizione esaustiva per facilitarne l'utilizzo. Data la varia natura dei dati presi in input è necessario che il tipo di input sia MULTIPART-FORM-DATA. I file accettati possono avere estensione .rds oppure .pmml. Conclusa l'operazione di update e di salvataggio del modello, viene inviata al client una response in cui viene indicata la buona riuscita dell'operazione. La risposta è inviata seguendo il formato di rappresentazione JSON.

URL: /model/
Method: PUT
Params: file, modelName, description
Response: risultato dell'operazione in JSON

```
@PUT
@Consumes(MediaType.MULTIPART_FORM_DATA)
@Produces(MediaType.APPLICATION_JSON)
public Response uploadModel(
  @FormDataParam("file") InputStream uploadedInputStream,
  @FormDataParam("file") FormDataContentDisposition fileDetail,
  @FormDataParam("modelName") String modelName,
  @FormDataParam("description") String description){...}
```

Le API per la creazione di un modello sono simili a quelle descritte per il caricamento di un modello pre-esistente; la differenza principale è nel metodo utilizzato per la chiamata HTTP. Nel caso della creazione viene utilizzato il metodo POST e il file contenente i dati deve essere di tipo CSV o ARFF. Inoltre, viene richiesto il parametro aggiuntivo che indica l'indice della colonna contenente la variabile dipendente.

6.1.2 Applicazione di un modello

L'applicazione di un modello avviene attraverso la chiamata HTTP con metodo POST. Il modello da applicare al set di dati preso in input viene selezionato attraverso l'identificativo presente nell'URL della richiesta. In questo caso si parla di parametro URL, ovvero un dato testuale che non è presente nel corpo della richiesta ma nell'URL della richiesta.

Il servizio restituirà al client lo stream di dati della valutazione oppure un messaggio in formato JSON per indicare la presenza di un errore.

URL: /model/:model-id/
Method: POST
URL Params: model-id
Params: file
Response: stream di dati oppure oggetto JSON per mostrare il messaggio di errore

```
@Path("{model-id}")
@POST
@Consumes(MediaType.MULTIPART_FORM_DATA)
@Produces({MediaType.APPLICATION_OCTET_STREAM,
    MediaType.APPLICATION_JSON})
public Response apply(
  @FormDataParam("file") InputStream inputStream,
  @FormDataParam("file") FormDataContentDisposition fileDetail,
  @PathParam("model-id") String modelID){...}
```

L'operazione di valutazione di un modello potrebbe richiedere del tempo ed, essendo l'operazione gestita in maniera sincrona, potrebbe rallentare o bloccare l'esecuzione del servizio di prediction. In questo prototipo tutte le operazioni lato back-end sono svolte in maniera sincrona, ma per svolgere operazioni che potrebbero richiedere del tempo è preferibile gestirle in maniera non bloccante.

Jersey permette di gestire, attraverso l'utilizzo di specifici moduli, operazioni asincrone, in questo modo è possibile inviare la risposta al client una volta terminata la procedura. Le operazioni che richiedono tempo e risorse devono essere eseguite da un Executor che consiste in un pool di thread. Terminata la procedura asincrona è possibile rispondere al client attraverso l'oggetto AsyncResponse messo a disposizione da Jersey.

```
ExecutorService TASK_EXECUTOR = Executors.newCachedThreadPool();

@GET
@Path("async/example")
public String asyncOperation(@Suspended final AsyncResponse ar) {
  TASK_EXECUTOR.submit(new Runnable() {

    @Override
    public void run() {
      // Long operation
      ar.resume("Response");
    }
  });
}
```

6.1.3 Snapshot di un modello

L'operazione di Snapshot consiste nel duplicare un modello, in cui una versione viene congelata allo stato attuale e non potrà più essere allenata, mentre la copia continuerà ad essere allenabile.

Questa operazione può essere richiesta dal client attraverso una chiamata HTTP con metodo GET. Nell'URL della richiesta deve essere presente l'identificativo del modello, seguito dalla keyword *snapshot*. Infine viene restituito al client il risultato dell'operazione effettuata in formato JSON.

URL: /model/:model-id/snapshot
Method: GET
URL Params: model-id
Response: risultato dell'operazione in JSON

```
@Path("{model-id}/snapshot")
@GET
@Produces(MediaType.APPLICATION_JSON)
public Response apply(@PathParam("model-id") String modelID){...}
```

6.1.4 Download di un modello

Il client può richiedere il download di qualsiasi modello presente nel sistema. Per recuperare un modello si deve effettuare una chiamata HTTP con metodo GET. L'URL, a differenza di quello per effettuare una richiesta di snapshot, è composto dall'identificativo del modello seguito dalla keywork *zip*.

Una volta che il servizio di database restituisce il modello attraverso l'identificativo, questo viene aggiunto ad uno zip. Terminata la fase di compressione del modello questo viene inviato come stream di dati al client.

URL: /model/:model-id/zip
Method: GET
URL Params: model-id
Response: stream di dati

```
@Path("{model-id}/zip")
@GET
@Produces(MediaType.APPLICATION_OCTET_STREAM)
public Response apply(@PathParam("model-id") String modelID){...}
```

6.2 Web UI

Per l'implementazione dell'interfaccia Web sono stati utilizzati HTML, Javascript e CSS. L'interfaccia è stata creata in modo da rendere la navigazione semplice e intuitiva. Nella parte sinistra troviamo la lista dei modelli e il bottone per la creazione di uno nuovo. Nella parte destra dell'interfaccia sono mostrate tutte le informazioni relative ad un modello e tutti i bottoni per effettuare azioni come il download, cambio della tipologia di accesso al modello, snapshot del modello, eliminazione e applicazione del modello. La tipologia di accesso indica quali utenti hanno il permesso di modificare un modello; un accesso locale permette di effettuare modifiche solo da parte del provider, mentre un accesso globale permette l'azione di modifica a tutti gli utenti. Cliccando sui pulsanti **Create Model** e **Update Model** vengono aperte due finestre: la prima mostra la form da compilare per la creazione del modello, mentre la seconda permette di cambiare la tipologia di accesso al modello attraverso **checkbox** e consente di effettuare lo snapshot.

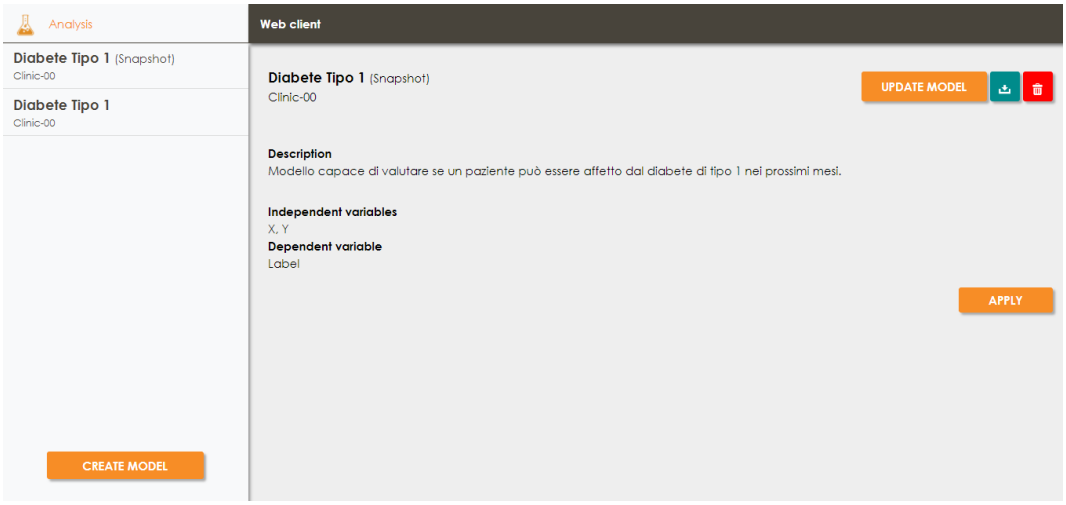

Figura 6.1: Client Web

Capitolo 7

Caso di studio

In questo capitolo viene descritto l'approccio utilizzato dal sistema per effettuare le fasi di apprendimento e di valutazione. Questo approccio è ottimizzato in termini di tempo e precisione, qualità indispensabili in un ambiente clinico che richiede l'analisi di grandi quantità di dati e permette il minimo errore possibile.

7.1 Switching Regression Model e Fuzzy Clustering

In *Switching Regression Model e Fuzzy Clustering* [7] gli autori propongono un nuovo approccio per la costruzione di modelli predittivi; l'approccio consiste nel suddividere i dati in gruppi in modo da trovare modelli che rispecchino al meglio ogni gruppo.

Per effettuare la valutazione dei dati è necessario cercare la funzione migliore che riesca ad associare ad ogni dato in input il corrispettivo output. Nei problemi più semplici, in cui $S = \{(x_1, y_1), ...(x_n, y_n)\}$ è un set di dati dove ogni variabile indipendente x_k è associata la variabile dipendente y_k, l'obiettivo è trovare una singola funzione che trovi le relazioni tra le x e le y tra tutti i dati. Molti problemi complessi nel mondo clinico fanno uso di numerose variabili e questi problemi necessitano di trovare soluzioni ottimali, nonostante la natura del problema.

Nel caso di studio i dati, invece di essere rappresentati da un unico modello, vengono rappresentati da P modelli. Questo vuol dire che ogni coppia di dati (x_k, y_k) è rappresentata da un modello specifico, ma prima di poter applicare la valutazione è necessario selezionare il modello corretto. Questo approccio è basato su tecniche di **fuzzy clustering** che dividono i dati in gruppi, per poi assegnare ad ogni pattern una probabilità di appartenenza ad ogni gruppo e infine selezionare il gruppo con probabilità maggiore.

La divisione in gruppi può essere effettuata con l'uso di algoritmi di clustering; questi algoritmi creano cluster di dati con caratteristiche simili tra loro e i pattern appartenenti ad un cluster andranno ad identificare il training-set del classificatore relativo al cluster. Quindi, dopo la procedura di clustering e di creazione dei classificatori, avremo un classificatore per ogni gruppo identificato dall'algoritmo di clustering.

Quando si riceve un nuovo dato, questo viene passato al clustering che predice il miglior gruppo di appartenenza e una volta trovata l'etichetta del gruppo si andrà ad applicare il classificatore corrispondente.

Lavorando con dataset di grandi dimensioni, questo procedimento risulta essere migliore rispetto all'applicazione di un unico modello di predizione, questo perché suddividere i dati in gruppi di pattern con caratteristiche simili permette al modello di rappresentare al meglio la collezione dei dati; così che l'utilizzo di questo approccio è preferibile soprattutto all'interno di un contesto sanitario.

7.1.1 Algoritmi utilizzati

Nel caso specifico, per dividere in sottogruppi il training set vengono utilizzati algoritmi di clustering. Al momento il sistema può utilizzare due differenti algoritmi per effettuare clustering: **XMeans** e **GMeans**. Entrambi gli algoritmi seguono una metodologia partizionale e stimano il numero di cluster presenti nella distribuzione dei dati.

Nella seconda parte dell'algoritmo i dati di ogni gruppo vengono passati in input ad un classificatore, che creerà un modello in grado di rappresentare i pattern appartenenti a tale gruppo.

Allo stato attuale del sistema, la fase di classificazione viene eseguita da una funzione di **regressione logistica**, che attraverso un modello di regressione binomiale, riesce a stimare la probabilità dell'occorrenza di un certo evento. In altre parole, si riesce a stimare in modo abbastanza preciso la classe di appartenenza di un dato. Questo classificatore riesce a lavorare in maniera ottimale quando la mole dei dati del training set è molto elevata.

7.1.2 Implementazione

In questa sezione viene mostrata una parte fondamentale dell'implementazione per effettuare la fase di training su un set di dati preso in input. Prima di tutto viene applicato uno dei due algoritmi di clustering per suddividere il dataset in gruppi, successivamente ogni cluster (o gruppo) viene utilizzato per creare il modello corrispondente.

```
private PartitionClustering<double[]> clustering;
private LogisticRegression[] logisticRegressions;

// Divisione dei dati in cluster. Il numero massimo di cluster
// in cui puo' essere diviso il dataset e' indicato da MAXCLUSTER.
// Attraverso il parametro passato in input e' possibile scegliere
// quale algoritmo di clustering utilizzare.
private void clustering(ClusterType clusterType){
  switch (clusterType){
    case XMEANS:
      this.clustering = new XMeans(this.data, MAX_CLUSTER);
      break;
    case GMEANS:
      this.clustering = new GMeans(this.data, MAX_CLUSTER);
      break;
  }
}
```

```
// Fase di classificazione
private void classification(){
  // Recupero del numero di cluster
  int numClusters = this.clustering.getNumClusters();
  // Recupero delle etichette assegnate ad ogni cluster
  int[] labels = this.clustering.getClusterLabel();
  // Recupero il numero di dati presenti in ogni cluster
  int[] clustersSize = this.clustering.getClusterSize();

  // Creazione di N classificatori, dove N e' il numero di cluster
  this.logisticRegressions = new LogisticRegression[numClusters];

  // I dati di ogni cluster allenano il corrispettivo classificatore
  for(int i=0; i<numClusters; i++){
    double[][] x1 = new double[clustersSize[i]][];
    int[] y1 = new int[clustersSize[i]];
    for(int j=0, k=0; j<this.data.length && k<clustersSize[i]; j++){
      if(labels[j] == i){
        x1[k] = this.data[j];
        y1[k] = this.classification[j];
        k++;
      }
    }
    this.logisticRegressions[i] = new LogisticRegression(x1, y1);
  }
}
```

7.2 Testing dell'algoritmo

Nella fase di test l'intento è quello di testare il funzionamento della piattaforma software di *predizione del rischio as a service*, piuttosto che l'efficacia degli algoritmi, per cui la natura dei dati utilizzati è piuttosto ininfluente.

Per testare l'algoritmo, data la difficoltà nel reperire dati sanitari reali, sono stati utilizzati dataset che si discostano dall'ambito sanitario. Il primo dataset contiene dati riguardanti il download e l'upload di una rete e permette di individuare quale tra due ISP sta fornendo il servizio.

Il secondo dataset è più articolato ed è composto da circa 1300 valutazioni di banconote vere e contraffatte. I dati sono stati estratti da immagini catturate da videocamere industriali e i loro valori si riferiscono alle caratteristiche dei pixel dell'immagine. Gli attributi su cui è basata l'analisi sono:

- **varianza**: indica la misura di quanto i valori di ogni pixel si discostino quadraticamente rispettivamente dalla media aritmetica di tutti i valori;
- **simmetria**: indica il punto in cui una funzione è simmetrica;
- **indice di curtosi**: indica la forma di una distribuzione di frequenza e misura il grado di appiattimento di una funzione di densità;
- **entropia**: indica il grado di entropia o di disordine dell'immagine.

In base ai valori assegnati a questi attributi è possibile valutare se una banconota è vera oppure contraffatta.

In entrambi i casi l'algoritmo ha avuto ottimi risultati calcolando il valore corretto per quasi la totalità dei dati del testing set.

Durante la fase di test è stato molto utile rappresentare a video la distribuzione spaziale di ogni dataset. Nella maggior parte dei casi, i dati sono rappresentati nello spazio multidimensionale, quindi, prima di poterli rappresentare, è stato necessario ridurre la dimensionalità dello spazio mappando i dati in uno spazio 2D.

Prendendo in esempio il secondo dataset utilizzato, lo spazio è stato ridotto di due dimensioni, passando da uno spazio a quattro dimensioni ad uno spazio bidimensionale.

Nell'immagine seguente si può notare il risultato dell'applicazione di un algoritmo di clustering al dataset delle banconote. I dati, rappresentati dai puntini colorati, sono stati suddivisi in 5 gruppi, ognuno dei quali colorato in maniera differente. Le ascisse e le ordinate, in questo caso, non hanno un vero e proprio significato, poiché, proiettando i dati in uno spazio di dimensionalità minore, lo spazio di riferimento è cambiato.

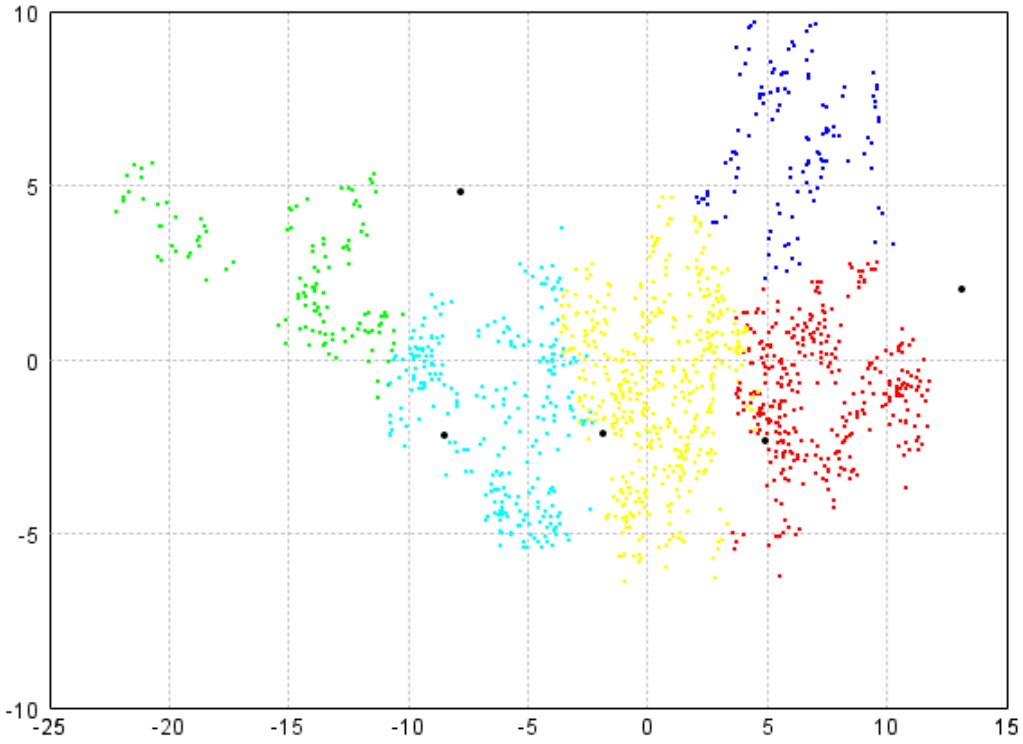

Figura 7.1: Suddivisione in cinque gruppi del dataset

Ogni singolo diagramma rappresenta la classificazione interna ad ogni gruppo identificato precedentemente. I puntini blu e rossi identificano rispettivamente i dati di banconote classificate come vere o contraffatte.

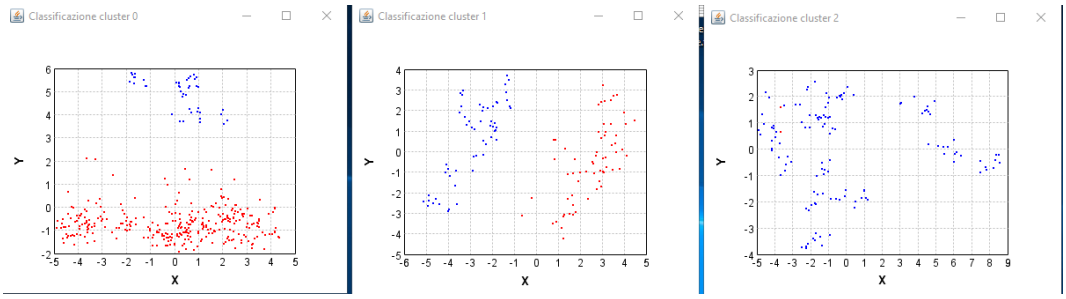

Figura 7.2: Classificazione dei primi tre gruppi

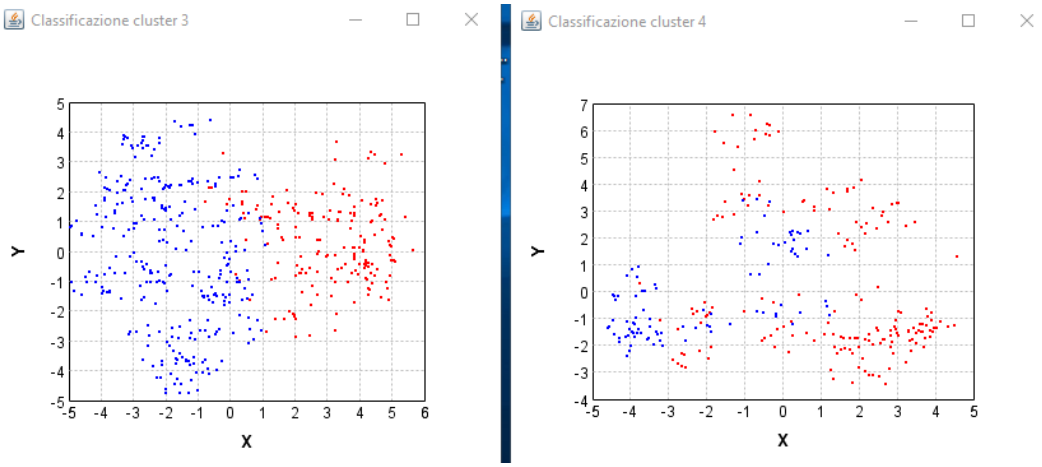

Figura 7.3: Classificazione degli ultimi due gruppi

Conclusioni

Il lavoro svolto offre un interessante punto di partenza per poter sviluppare un DSS as a service. Nei decenni a venire, sarà sempre più necessario adottare soluzioni innovative, affidabili e usufruibili in qualsiasi momento, soprattutto per quanto riguarda la sanità. La tempestività di eseguire una diagnosi o la somministrazione di cure migliori potrebbe salvare moltissime vite.

L'architettura è stata suddivisa il più possibile in moduli, così da facilitare l'aggiunta di funzionalità mancanti nel minor tempo possibile e con il minor costo possibile. Inoltre, l'applicativo sfrutta tecnologie moderne e all'avanguardia che gli permettono di fornire maggiore scalabilità e sicurezza.

In sviluppi futuri, si potrebbe pensare di effettuare il deployment di ogni singolo servizio su *container* virtuali; un container è un ambiente di esecuzione che mette a disposizione alle applicazioni solo il necessario per essere eseguite, inoltre stesse applicazioni eseguite su container differenti sono totalmente separate.

Un altro aspetto fondamentale da valutare riguarda la sicurezza. Tutti i dati dei pazienti usati per la fase di apprendimento, prima di essere salvati all'interno del sistema devono essere criptati con qualche algoritmo di codifica in modo da evitare fughe di dati in caso di attacco.

Per rendere il sistema utilizzabile dalla maggior parte delle cliniche mondiali, sarà sicuramente necessario costruire un software che accetti la maggior parte, se non tutti, dei modelli predittivi utilizzati dalle cliniche.

Lo studio effettuato è stato, per lo più, circoscritto all'ambito sanitario, ma questi sistemi trovano soluzioni anche in altri domini applicativi come la domotica, la finanza, la scienza ecc.

In conclusione, il lavoro ha l'obiettivo di proporre un'infrastruttura di un

sistema innovativo in esecuzione sul cloud per offrire funzionalità di predizione, con l'aspirazione di innovare e migliorare il sistema sanitario attuale.

Ringraziamenti

Ringrazio prima di tutto il mio relatore Andrea Omicini, ma soprattutto il mio co-relatore Stefano Mariani che mi ha seguito in tutte le fasi di sviluppo del progetto. Ringrazio i miei genitori per avermi permesso di studiare e di raggiungere questo traguardo importante. Infine ringrazio tutti i miei amici che mi hanno accompagnato in questa avventura fantastica. Grazie di cuore a Alex Collini, Cristian Paolucci, Matteo Aldini, Filippo Berlini, Brando Mordenti, Davide Foschi, Lorenzo Righi, Luca Battistini.

Bibliografia

[1] Emanuela Teruzzi, *Verso il cloud* *http://www.silicon.it/cloud/verso-il-cloud-quali-passaggi-da-compiere-per-fare-il-salto-103665*

[2] Saima Safdar, Saad Zafar, Nadeem Zafar, Naurin Farooq Khan, *Machine learning based decision support systems for heart disease diagnosis.*

[3] Wikipedia, *Machine learning* *https://en.wikipedia.org/wiki/Machine_learning*

[4] Wikipedia, *Clinical Decision Support System* *https://en.wikipedia.org/wiki/Clinical_decision_support_system*

[5] Chris Richardson, *Microservices* *http://microservices.io/patterns/microservices.html*

[6] Sun, *Jersey* *https://jersey.github.io/*

[7] Richard J. Hathaway, James C. Bezdek, *Swiching Regression Models and Fuzzy Clustering*

[8] Smile, *Libreria di machine learning* *http://haifengl.github.io/smile/index.html*

[9] Hamad R., Modrek S., Kubo J., *Using 'big data' to capture overall health status: properties and predictive value of a claims-based health risk score*

[10] Ribeiro M., Grolinger K., Capretz M.A.M., *MLaaS: Machine Learning as a Service* 14^a International Conference on Machine Learning and Applications

[11] Pivarski J., Bennett C., Grossman R.L., *Deploying Analytics with the Portable Format for Analytics*