Big Data Management – progetto finale

- Studente: Federico Berton
- Docente: Fabrizio Montecchiani

- A.A. 2022/2023
- Corso di Laurea Magistrale in Ingegneria Informatica e Robotica (curriculum data science)
- Link GitHub: https://github.com/bertonfederico/HeartDiseasePrediction

Sviluppo di un cluster Spark-Cassandra per la previsione di malattie cardiache

Il progetto si propone di sviluppare un modello per la previsione delle malattie cardiache a partire da un dataset iniziale (reperibile al seguente link) con l'utilizzo di un cluster Spark-Cassandra per attingere ai dati, elaborarli e aggiungere record al dataset. Il progetto inoltre fornisce un'interfaccia grafica per interagire in maniera semplice con i due endpoint del sistema (il primo per la previsione di disturbi cardiaci, e il secondo per l'inserimento di nuovi dati) e per visualizzare graficamente le informazioni statistiche riguardo il dataset di partenza.

In questo modo, quindi, è possibile costruire un sistema predittivo che si perfeziona incrementalmente aggiungendo di volta in volta informazioni al dataset utilizzato per l'allenamento del modello: ogni volta che un utente inserisce i propri dati questi vengono aggiunti al database distribuito e utilizzati, al successivo riavvio del server, per costruire nuovamente il modello predittivo.

1 - Dataset di partenza

Il dataset utilizzato (in formato .csv) contiene dei dati raccolti negli Stati Uniti durante l'anno 2020, e riporta, per ogni intervistato, alcune indicazioni riguardo le informazioni anagrafiche, lo stile di vita e la presenza di patologie. Nello specifico, il dataset contiene le seguenti informazioni:

- **HeartDisease**: intervistati che hanno dichiarato di aver avuto una malattia coronarica (CHD) o un infarto del miocardio (MI);
- **BMI**: indice di massa corporea;
- **Smoking**: hai fumato almeno 100 sigarette nella sua vita? [Nota: 5 pacchetti = 100 sigarette];
- **AlcoholDrinking**: forti bevitori (uomini adulti che bevono più di 14 drink a settimana e donne adulte che bevono più di 7 drink a settimana);
- **Stroke**: hai mai avuto un ictus?
- **PhysicalHealth**: pensando alla tua salute fisica, che include malattie fisiche e lesioni, per quanti giorni negli ultimi 30 giorni la tua salute fisica non è stata buona? (0-30 giorni);
- **MentalHealth**: Pensando alla tua salute mentale, per quanti giorni negli ultimi 30 giorni la tua salute mentale non è stata buona? (0-30 giorni);
- **DiffWalking**: hai serie difficoltà a camminare o a salire le scale?
- **Sex**: sei maschio o femmina?
- **AgeCategory**: categoria di età (14 livelli);
- **Race**: etnia;
- **Diabetic**: hai mai avuto il diabete?
- **PhysicalActivity**: adulti che hanno riferito di aver svolto attività fisica o esercizio fisico negli ultimi 30 giorni al di fuori del loro lavoro abituale;
- **GenHealth**: diresti che in generale la sua salute è...
- **SleepTime**: in media, quante ore di sonno riesce a dormire in un periodo di 24 ore?
- **Asthma**: ti hanno mai diagnosticato l'asma?
- **KidneyDisease**: esclusi i calcoli renali, le infezioni della vescica o l'incontinenza, ti è mai stata diagnosticata malattia renale?
- **SkinCancer**: ti è mai stato diagnosticato un cancro della pelle?

2 - Strumenti utilizzati

Il progetto è stato realizzato con l'utilizzo di diversi linguaggi: per la parte back-end e machine-learning sono stati utilizzati Python e script shell; per la parte front-end, invece, è stato fatto uso di HTML, CSS e Javascript.

Sono state utilizzate inoltre le seguenti tecnologie:

- **Apache Hadoop 3.3.4**: utilizzato tramite il suo file system distribuito, viene impiegato nella fase di pre-processing e inserimento nel database del dataset;
- **Apache Cassandra**: utilizzato come database distribuito per la memorizzazione dei dati (sia quelli del dataset di partenza che quelli raccolti successivamente tramite interfaccia);
- **Apache Spark 3.2.3**: utilizzato in una versione meno recente rispetto all'ultima release, in quando le versioni successive non sono supportate dalla libreria Pyspark2pmml. Spark, assieme alla sua libreria

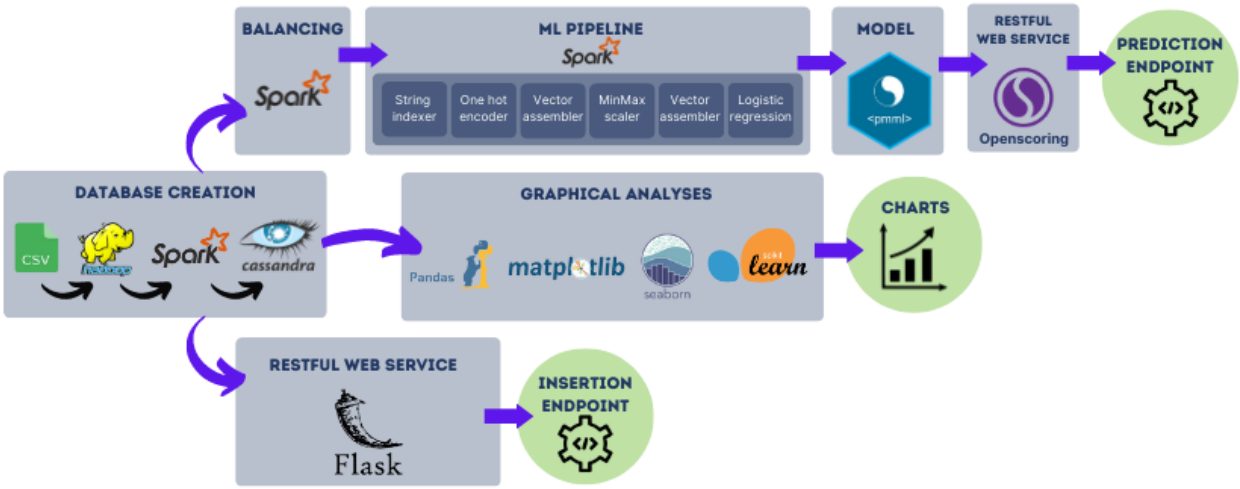
MLlib, viene utilizzato per la manipolazione dei dati, la creazione della ML-pipeline e del modello predittivo;

- **Pyspark2pmml**: libreria utilizzata per esportare il modello predittivo *PySpark* in formato .pmml;
- **OpenScoring-server-jar 2.1.1**: utilizzato per creare un web service su cui caricare il file .pmml in modo da fornire un endpoint per la previsione di disturbi cardiaci;
- **Java 8**: utilizzato da Pyspark2pmml per la conversione del modello in .pmml e per eseguire il file .jar di OpenScoring;
- **Flask**: libreria Python utilizzata per la creazione di un web service che fornisce un endpoint per l'inserimento di nuovi dati nel database Cassandra;
- **Cassandra.cluster:** libreria Python per la connessione con databases Cassandra;
- **matplotlib, pandas, seaborn, scikit-learn:** librerie utilizzate per la creazione di grafici che riportano le informazioni statistiche sul dataset di partenza;

3 - Funzionamento del sistema

3.1 – Back-end

La parte software back-end si basa su un passo preliminare che viene eseguito solamente la prima volta che deve essere avviato il server, ovvero la creazione del database distribuito Apache Cassandra a partire dal dataset in formato .csv. Successivamente il database appena creato viene utilizzato per analizzare graficamente i dati, creare il modello predittivo e creare un endpoint per l'inserimento di dati nel database. Tutto ciò è raffigurato nel seguente diagramma di flusso:



3.1.1 – Creazione database distribuito

In prima istanza è necessario procedere alla creazione del database; questo viene fatto nello script '*Server/0_install/cassandra/cqlscript.cql*', in cui viene creato un KEYSPACE 'heartdisease' (per semplicità con replication factor pari a 1), e successivamente una tabella 'features' con attributi pari a quelli del dataset, in cui viene aggiunta una colonna alfanumerica 'id' come chiave primaria.

Una volta creato il database e scaricato il dataset in formato .csv, come prima cosa esso deve essere caricato in hdfs per essere pre-processato dallo script '*Server/0_install/cassandra/preProcessing.py*'; il passo di pre-processing, eseguito tramite *PySpark*, consiste nell'aggiunta dell'attributo 'id' a tutti i record, in questo caso posto pari al numero di riga all'interno del file .csv. Successivamente il dataframe ottenuto viene inserito nel database.

3.1.2 – Analisi grafica del dataset

Come prima cosa, dopo la creazione e il popolamento del database, i dati vengono analizzati graficamente per esaminare la distribuzione delle risposte tramite *PySpark* e le librerie *matplotlib, pandas, seaborn, scikit-learn* all'interno dello script '*Server/1_start_server/createModel.py*'.

La colonna '*ageCategory*', che nel dataset si presenta come categorica ma che di fatto è una caratteristica continua, viene trasformata in continua sostituendo ogni valore categorico con la media della determinata categoria.

Per quanto riguarda i features di tipo continuo, viene creato un grafico di tipo kernel density estimate che riporta la distribuzione dei valori negativi per la colonna '*heartdisease*', di quelli positivi, e della somma dei due.

Per quanto riguarda invece i features di tipo categorico, viene creato un grafico a barre per visualizzare la distribuzione dei valori negativi per la colonna '*Heartdisease*' e di quelli positivi, e successivamente un grafico a torta per analizzare la percentuale di riposte per ogni categoria.

Viene creata inoltre una prima heatmap per visualizzare la matrice di correlazione tra tutti i features del dataset, e dopo la creazione del modello una seconda heatmap per analizzare la quantità di predizioni corrette ed errate che vengono fatte dal modello.

3.1.3 – Creazione del modello predittivo

Sempre all'interno dello script '*Server/1_start_server/createModel.py*' viene creato il modello predittivo tramite *PySpark* e la libreria per il Machine Learning MLlib.

Come prima cosa è necessario eliminare dal dataframe la colonna '*id*', non utile ai fini della creazione del modello. Successivamente è fondamentale bilanciare il dataset di partenza, in quanto il rapporto di sbilanciamento tra le risposte negative a '*heartdisease*' (in netta maggioranza) e quelle positive (in minoranza) risulta essere di 10 a 1. Per non diminuire drasticamente il numero di campioni tramite puro undersampling delle risposte negative, e allo stesso tempo per non aumentare di circa dieci volte la taglia del dataset tramite puro oversampling delle risposte positive, sono state combinate le due soluzioni in modo da mantenere più o meno identica la taglia dell'istanza da analizzare. È stato quindi effettuato un undersampling delle risposte negative, riducendo il loro numero alla metà circa, e un oversampling delle risposte positive, aumentando il loro numero di cinque volte circa.

Successivamente si procede con la creazione della pipeline di processamento, che si sviluppa nei seguenti passi:

- **StringIndexer (categoricalIndexer):** indicizzazione delle caratteristiche categoriche (inclusa la feature 'heartdisease'), ognuna delle quali viene mappata in una seconda colonna denominata con lo stesso nome di quella di partenza, con l'aggiunta del suffisso "*_indexed*";
- **OneHotEncoder (categoricalEncoder):** codifica delle caratteristiche categoriche appena indicizzate (ad eccezione di '*heartdisease*', che essendo la caratteristica obiettivo deve essere di tipo booleano, ovvero con valori 0/1), ognuna delle quali viene mappata in una seconda colonna denominata con lo stesso nome di quella di partenza, con la modifica del suffisso da "*_indexed*" a "*_encoded*;
- **VectorAssembler (continuousAssembler):** assemblaggio delle caratteristiche continue in un unico vettore contenuto nella colonna "*assembledFeatures*";
- **MinMaxScaler (continuousScaler)**: normalizzazione delle caratteristiche appena assemblate;
- **VectorAssembler (totalAssembler):** assemblaggio di tutte le caratteristiche contenute nelle colonne "_encoded" e quelle contenute nella colonna "assembledFeatures" all'interno di un unico vettore nella colonna "final_features";
- **LogisticRegression (regressor):** Creazione di una regressione logistica a partire dalle caratteristiche "final_features" e la caratteristica da prevedere 'heartdisease_indexed'.

Una volta creata la pipeline, essa viene allenata su una porzione del dataset (70%) e poi viene testato nella porzione di dataset rimanente. Successivamente viene valutata l'accuratezza delle previsioni effettuate sul dataset di test tramite un MulticlassClassificationEvaluator, che risulta all'incirca di 0.775.

Il modello appena creato tramite *PySpark* viene convertito in formato .pmml tramite pyspark2pmml, e successivamente caricato nel web service di Openscoring già messo precedentemente in esecuzione tramite il file '*Server/lib/openscoring-server-executable-2.1.1.jar*'.

Infine viene mostrato un log in console che indica varie informazioni, tra cui il rapporto di sbilanciamento, il tipo di estimatore, il tipo di valutatore, l'accuratezza della predizione:

```
##########################################################################################################
##########################################################################################################
##########################################################################################################
########  --> Unbalancing ratio: 10.683                                                          #########
########  --> Balanced by by keeping the same size of original input dataset                     #########
########  --> Estimator: logistic regression                                                     #########
########  --> Evaluator: MulticlassClassificationEvaluator                                       #########
########  --> Prediction accurancy: 0.777                                                        #########
########  --> Prediction served on: http://localhost:8080/openscoring/model/HeartDisease         #########
########  --> Client page: ../../Client/homePage.html                                            #########
##########################################################################################################
##########################################################################################################
##########################################################################################################
```

3.1.4 – Creazione dell'endpoint per l'inserimento di nuovi dati

L'ultimo step per la realizzazione del lato back-end è la creazione di un web service per l'inserimento di nuovi dati nel database, in modo da poter rendere sempre più accurato il modello tramite l'acquisizione di altre risposte. In questo modo, alla successiva messa in esecuzione del server, il sistema di machine learning prenderà come input l'intero contenuto del database, che include, oltre ai dati analizzati nella precedente esecuzione, anche tutti i dati inseriti successivamente.

Per far ciò nello script '*Server/1_start_server/insertEndPoint.py*' è stato fatto uso della libreria *Flask* per la creazione del web service, e la libreria *cassandra.cluster* per la connessione al database e l'inserimento dei dati.

3.2 – Front-end

Lato front-end, l'applicazione si sviluppa in una pagina web contenente tre visualizzazioni diverse:

1. Heatmap della matrice di correlazione (per visualizzare il legame tra tutte le diverse caratteristiche) e grafico a torta della distribuzione dei valori della colonna '*heartdisease*' (per visualizzare in maniera immediata lo sbilanciamento del dataset accennato in precedenza);
2. Grafici per ognuna delle caratteristiche per visualizzare la relazione tra essa e la presenza di problemi cardiaci (tramite grafici a torta, grafici a barre e kernel density estimate plot);
3. Heatmap della matrice di confusione (per dare un'idea dell'accuratezza del modello) e form tramite il quale inserire i propri dati, visualizzare la predizione effettuata dal modello assieme alla percentuale di affidabilità, ed eventualmente inserire i propri dati nel database aggiungendo l'informazione mancante, ovvero l'effettiva presenza o meno di disturbi cardiaci nel soggetto compilante. La predizione e l'inserimento vengono effettuati tramite richieste di tipo POST nello script '*Client/js/requests.js*'. Nel caso dell'inserimento dei novi dati nel database, oltre alle informazioni inserite dall'utente viene inserito nella richiesta http il parametro '*id*' costruito come concatenazione tra l'indirizzo IP da cui viene la richiesta e il timestamp in cui avviene la richiesta.

4 – Utilizzo

4.1 – Installazione componenti necessari

In caso alcuni o tutti i componenti necessari (specificati al punto 2) non fossero già stati installati, è possibile installarli tramite lo script '*Server/0_install/0_installComponents.sh*'.

4.2 – Creazione e popolamento del database

La creazione e il popolamento del database descritti al punto 3.1.1, da eseguire solo alla prima esecuzione del server, vengono realizzati tramite lo script '*Server/0_install/1_createCassandraDb.sh*'.

4.3 – Esecuzione del server

Per creare il modello predittivo dai dati contenuti nel database, produrre i grafici e creare i web service utili alla predizione e all'inserimento, è sufficiente eseguire lo script '*Server/1_start_server/0_runServer.sh*'.

4.4 – Interruzione del server

In caso sia necessario interrompere il server, ad esempio per poi eseguire nuovamente il server in modo da creare nuovamente il modello predittivo a partire dai nuovi dati inseriti nel database, è sufficiente eseguire lo script '*Server/2_stop_server/stopServer.sh*'.

5 – Possibili miglioramenti

La sezione di machine learning per la creazione del modello predittivo potrebbe essere migliorata modificando il modello statistico utilizzato, ovvero la regressione logistica, con altri modelli e comparare i risultati in termini di performance e accuratezza. Ad esempio, utilizzando il Gradient-Boosted Trees come modello si riesce ad ottenere un'accuracy migliore, a discapito però dell'efficienza.