Documentazione Web App

Castelli Marco

Anno accademico 2021-2022

Indice

1 Introduzione

La web application sviluppata permette di generare, grazie al «LookBack Apriori Algorithm» un insieme di regole, dati come input un'insieme di dati e dei setting scelti dall'utente. Una volta avvenuta la generazione delle regole l'utente ha la possibilità di inserire una query. Il software effettuerà il match delle regole precedentemente generate e la query stessa.

Questa web application, che rappresenta il server, servirà poi per la creazione di un software di più grandi dimensioni. Il software sarà formato da una mobile app che manderà delle richieste API al server, esso genererà le regole prendendo i dati da un database e ritornerà un risultato all'utente.

1.1 LookBack Apriori Algorithm

L'algoritmo è un estensione del classico Algoritmo Apriori, infatti effettua un processo di data mining dei dati utilizzando anche il time slot dei dati generati.

I dati processati rappresentano le attività fisiche e la qualità del riposo dell'utente negli ultimi giorni, in particolare per ogni attività vi è la durata in minuti della stessa. L'algoritmo analizza i dati dell'utente e restituisce come risultato una lista di regole.

Queste regole vengono generate in base ai dati del singolo utente e rappresentano quali attività l'utente dovrebbe svolgere per ottenere un buon riposo.

2 Organizzazione del codice

Il file principale è «app.py» che rappresenta la routes delle varie pagine html, in esso vengono richiamate le varie funzioni per gestire i dati ottenuti dai form oppure i dati provenienti dalle api.

Tutte le funzioni utilizzate sono state organizzate in un file «utilities.py» mentre l'algoritmo di data mining, ossia «LookBack Apriori Algorithm», è definito in un file a parte .

Le pagine html si trovano all'interno della cartella «templates» mentre nella cartella «static» è presente il codice css.

All'interno della cartella «Setting» sono presenti tutte le regole che sono state generate durante l'utilizzo del software. Il nome di ogni file rappresenta il setting utilizzato per generare le regole, in particolare si specifica il dataset e i parametri: sleep value, temporal window, min support, min confidence.

Nella cartella «Data» sono presenti delle sottocartelle che contengono i dati di alcuni utenti.

3 Descrizione file principali

3.1 «app.py»

In un certo modo rappresenta il main del nostro programma.

Questo file permette di gestire sia i dati provenienti dai form (Front-end), sia quelli provenienti dalle api inviate dall'applicazione. Per differenziare la tipologia di richiesta, viene utilizzata la variabile «api», essa vale True se la richiesta è una api mentre False altrimenti.

In questo modo controllando il suo valore, so se restituire una pagina.html oppure richiamare un'altra funzione per poi restituire un oggetto json, ossia la risposta alla api.

Di seguito la descrizione mediante l'uso del front-end. Per le richieste api il funzionamento è lo stesso ma, al posto di richiamare la pagina html richiamo direttamente la funzione

Prima delle «routes» vengono dichiarate tutte le librerie utilizzate e dopodichè vengono definite delle variabili globali in modo che siano accessibili all'interno delle varie routes.

1. La prima route è «settingRules» in cui viene semplicemente richiamata la pagina «settingRules.html», in questa pagina html si compila il form con le informazioni per il setting: dataset , sleep value, temporal window, min support, min confidence.

2. Dopo aver compilato il form i dati vengono passatti alla route «rulesGeneration», qui vengono effettuati vari controlli sulla correttezza dei dati, qualora non lo fossero l'utente viene reindirizzatto alla pagina «settingRules.html». Se invece i dati fossero corretti si verifica se è già presente un file con quei setting, in modo da non dover rigenerare le regole, se non lo fosse, si controlla anche se è possibile generare le regole a partire da un file già presente ma con diversi setting.

   Se non è vero nessuno dei due casi precedenti allora, bisogna necessariamente richiamare la funzione di generazione delle regole «LookBack Apriori Algorithm.py» che restituisce una lista formata dalle regole generate, già ordinate e successivamente si salvano le regole su un file all'interno della cartella «Setting».

3. Avvenuta la generazione delle regole si viene reindirizzati alla pagina «rulesGeneration.html» che mostra le prime venti regole generate, ordinate per support, completezza e dimensione; vengono inoltre mostrati i setting utilizzati e il numero totale delle regole generate.

   Nella navbar è presente la voce «Match Query», una volta premuta, si viene reindirizzati alla pagina «matchQueryForm.html» dove l'utente ha la possibilià di inserire una query per il matching.

4. Inserita la query e premuto l'apposito bottone si viene reindirizzatti alla route «matchingQuery», qui viene controllata la correttezza della query e successivamente viene richiamata la funzione di matching che mostra il risultato all'interno della pagina «matchingQuery.html».

   L'utente ha sempre a disposizione la navbar che può utilizzare per tornare a visualizzare le regole generate oppure per creare nuove regole con nuovi setting.

3.2 «LookBack Apriori Algorithm.py»

Il codice dell'algoritmo è stato messo sotto forma di funzione, il setting viene passato come parametro alla funzione e tali parametri vengono utilizzato dall'algoritmo per generare le regole secondo onerosi calcoli. Viene richiamata la funzione di ordinamento sulle regole ed infine vengono restituite tutte le regole generate.

4 Descrizione funzioni presenti nel file utilities

4.1 dataRange

Richiamata nella route «settingRules» verifica che i dati inseriti nel form appartengano ad un determinato range di valori.

Restituisce eventualmente un messaggio di errore e booleano.

4.2 isQueryValid

Richiamata nella route «matchingQuery» verifica che la query inserita sia ben formata, ossia contenga come attività HA,MA,LA,ZL,R con annesso un time slot (es: t2). Il time slot non deve superare la temporal window che l'utente ha inserito nei setting iniziali.

Restituisce un booleano e un messaggio.

4.3 addCriteria

Richiamata all'interno della funzione rulesSorting, altera la struttura di ogni elemento della lista di regole aggiungendo un campo completeness e size.

Utilità: facilità la successiva operazione di ordinamento.

Restituisce la lista di regole modificata.

4.4 rulesSorting

Richiamata all'interno del file «LookBack Apriori Algorithm.py» restituisce una nuova lista dove le regole sono ordinate per support, completeness, size.

4.5 splitFile

Richiamata in due diverse route quando abbiamo necessità di leggere le regole dal file, ad esempio inseriamo un setting precedentemente generato oppure è possibile utilizzare un'altro file da cui leggere le regole (setting implicito).

Scorre il file e memorizza tutte le informazioni in esso presenti, in particolare i valori dei setting, la sola lista di regole (campo "rule" della struttura) e l'intera struttura delle regole.

Restituisce le informazioni precedenti.

4.6 alreadySetting

Richiamata nella route «rulesGeneration».

Verifica se nella cartella «Setting» è già presente un file con il nome passato come parametro.

Restituisce un booleano

4.7 rulesImplicitGeneration

Richiamata nella route «rulesGeneration».

Dato un setting verifica se esiste un file dal quale è possibile prendere un sottoinsieme di regole (setting implicito).

Utilità: vogliamo generare un insieme di regole a partire da un setting ma, abbiamo a disposizione un file simile nel quale le regole sono già contenute.

4.8 saveSetting

Richiamata nella route «rulesGeneration».

Consente di creare un file con il nome del setting, in esso viene salvatosia il setting che tutte le regole generate (solo campo ”rule”).

4.9 takeRuleImplicit

Richiamata nelle routes «rulesGeneration» e «matchingQuery». Usata quando troviamo un file che contiene le regole specificate in un nuovo setting, per ottenere un sottoinsieme delle regole.

Consente di ottenere solamente le regole con support e confidence specificate.

Resituisce le sole regole e l'intera struttura delle regole.

4.10 findMatching

Funzione principale per il matching delle regole, in essa vengono fatte delle operazioni preliminari sulla query e sull'insieme di regole, dopodichè vengono richiamate le funzioni specifiche per il matching, una alla volta in ordine di importanza.

Il confronto tra la query dell'utente e le regole presenti, viene fatto solamente sulle attività diverse da t0.

Restituisce se trova la regola altrimenti null.

4.11 ExactMatch

Primo criterio per il matching: scorre tutta la lista di regole e se trova una regole che è identica alla query dell'utente la restituisce.

Esempio di ExactMatch:

Query utente: LA_3_t3 + MA_2_t2 + R_3_t1

Match: LA_3_t3 + MA_2_t2 + R_3_t1 + X_X_t0

4.12 Match

Secondo criterio per il matching: scorre tutta la lista di regole e se trova una regola in cui tutte le attività della "Match" appaiono nello stesso time slot nella query dell'utente, la restituisce.

Esempio di Match:

Query utente: MA_2_t3 + LA_3_t3 + MA_2_t2 + LA_1_t1 + R_3_t1

Match: LA_3_t3 + MA_2_t2 + R_3_t1 + X_X_t0

4.13 PartialMatch

Terzo criterio per il matching: ci sono delle stesse attività che appaiono nel medesimo time-slot, mentre le altre sono simili, cioè hanno la stessa attività ma intensità diverse. Scorre la lista e se trova la regola la restituisce.

Esempio di PartialMatch:

Query utente: R_2_t3 + LA_2_t3 + MA_2_t2 + LA_2_t1 + R_2_t1
Match: LA_3_t3 + MA_2_t2 + R_3_t1 + X_X_t0

4.14 SimilarMatch

Quarto criterio per il matching: ogni time slot contiene un'attività che è simile ad un'altra nella match rule. Scorre la lista e se trova la regola la restituisce.

Esempio di SimilarMatch:
Query utente: R_3_t3 + LA_2_t3 + MA_3_t2 + LA_2_t1 + R_1_t1
Match: LA_3_t3 + MA_2_t2 + R_3_t1 + X_X_t0

4.15 isSimilar

Funzione utilizzata all'interno delle funzioni PartialMatch e SimilarMatch, consente di verificare se un insieme di attività, in un dato time slot (t2), è simile rispetto ad un altro insieme nello stesso time slot.

Due attività sono simili se hanno la stessa attività (es: HA) ma con intensità diverse.

Resituisce una lista con i valori "Similar", "Partial" e "False", che serviranno per controllare il tipo di match.

Se l'array risultato contiene solo "Similar" allora il match è di tipo SimilarMatch, se invece oltre a "Similar" c'è anche "Partial" allora è PartialMatch.

4.16 splitActivity

Richiamata in tutte le funzioni di match, consente a partire da una stringa (la regola) di ottenere una lista formata dalle attività in t1,t2,t3, ..., tn

Esempio: ([[HA_1_t3, MA_2_t3],[LA_2_t2, MA_1_t2],[R_2_t1, ZL_2_t1]])

4.17 ruleAntecedent

Consente di ottenere l'antecedente della regola cioè tutto ciò prima di '->', si veda un file all'interno di «Setting» per capire meglio. Viene utilizzata per il matching delle regole.

5 Api

5.1 Introduzione

Per rendere il server completo per il suo funzionamento, lo si ha integrato per gestire anche richieste api.

Queste richieste provengono da un'app sul telefono e specificano i parametri di setting e la query, ossia i medesimi parametri inseribili da front-end.

Il server è in grado di sapere grazie alla variabile global «api» se i dati provengono da un api oppure da front-end. Nel primo caso risponderà tramite un oggetto json nel secondo, restituirà una pagina html.

5.2 Struttura api e formato risposta

5.2.1 Richiesta api

La richiesta api viene inviata tramite POST e il suo body è formato da un oggetto JSON così definito:

```
{
"data": "pm1",
"sleep_value":3,
"temporal_window":2,
"min_confidence":1.0,
"min_support":0.03,
"my_query":"HA_3_t2 + MA_3_t1 + R_1_t1",
"filterT0":"on"
}
```

Tutti i significati delle variabili sono facilmente intuibili. «filterT0» può assumente valore «on,off» e consente di fare il matching con regole che hanno attività in t0 oppure no.

5.2.2 Risposta api

La risposta invece ha il seguente formato JSON:

```
{
"description": "",
"typeMatch": "",
"ruleMatch": ""
}
```

«description» assume valore «correct» se non ci sono stati errori di formato nella richiesta api, altrimenti contiene un messaggio che specifica qual'è errore.

«typeMatch» assume valore solo se description è «correct», specifica qual'è il matching trovato per la query scritta nella api (es: EXACT_MATCH, PARTIAL_MATCH,...)

«ruleMatch» assume valore solo se description è «correct», specifica qual'è la regola trovata nel matching con la query scritta nella api.

Esempi di risposte:

- Richiesta api corretta ma query inserita non trova matching:

  {"description": "correct", "typeMatch": "NO RULES FOUND", "ruleMatch": "NULL"}

- Richiesta api corretta e query inserita trova matching:

  {"description": "correct", "typeMatch": "EXACT MATCH", "ruleMatch": "HA_2_t2 + R_2_t2 + LA_3_t1 + R_3_t0 -> ZL_3_t0"}

- Richiesta api errata:

  {"description": "Query has not valid activity!", "typeMatch": "", "ruleMatch": ""}

5.3 Modalità di utilizzo

- Uso definitivo:

  L'applicazione sviluppata crea delle richieste api, con il formato definito precedentemente e le invia all'indirizzo: *https://lookbackapriorialgorithm.herokuapp.com/*.

- Testing:

  Tramite Postman al medesimo indirizzo, Postman è una piattaforma API per gli sviluppatori per progettare, creare, testare e iterare le API.

  Passaggi:

  1. Si crea una nuova HTTP request
  2. Si seleziona il metodo POST
  3. Si specifica il seguente URL *https://lookbackapriorialgorithm.herokuapp.com/*
  4. Si clicca su body e si seleziona la voce «raw»
  5. Si clicca su «text» e si specifica JSON
  6. Nel form sottostante si inserisce la richiesta api
  7. Si invia la richiesta premendo su SEND