Smart House

735722 - Mattia Curatitoli
737838 - Danilo Deponti

Introduzione

L'invecchiamento della popolazione ha un impatto sempre più significativo sul sistema sanitario. Per questo si cercano vie più efficienti per portare assistenza in casa delle persone bisognose, in particolare gli anziani, il cui numero secondo le stime é in costante aumento grazie alle migliori condizioni di vita e di cure.
Monitorare le attività umane (`ADL`) è diventato un aspetto fondamentale per costruire un ambiente intelligente atto a garantire il benessere in grado di aumentare sia la sicurezza, sia l'autonomia dei soggetti.
Uno degli approcci più promettenti ed economici è il monitoraggio delle attività tramite una rete wireless di sensori (`WSN`) disposti nell'ambiente abitativo, in quanto molto flessibile e di facile sviluppo. Sono stati sviluppati parecchi modelli che utilizzano sensori per monitorare le attività `ADL`, come ad esempio *Reti Bayesiane* o *Conditional Random Fields*. In particolare, si è notato che ***Hidden Markov Model (HMM)*** è un modello performante in questo dominio applicativo; alcuni articoli approfondiscono il modello proponendo soluzioni di HMM ibridi, noi invece abbiamo implementato una versione classica.

I sensori

Per costruire e testare il modello, sono stati utilizzati dataset presi da due appartamenti. Ogni appartamento fornisce due dataset: il primo con le rilevazioni generate dai sensori, il secondo con le attività rilevate dai sensori. I sensori utilizzati sono stati di vario tipo (magnetici, a pressione, elettrici etc), disposti in più zone della casa in maniera il meno invasiva possibile e hanno registrato e salvato i dati per diversi intervalli di tempo (due settimane circa).
Il dataset delle rilevazioni sensoristiche (figura 1) è una lista di righe corrispondenti all'intervallo di tempo nei quali i sensori sono stati attivi. Ogni riga é composta da:

```
Start time - End time - Location - Type - Place
```

Start time	End time	Location	Type	Place
2011-11-28 02:27:59	2011-11-28 10:18:11	Bed	Pressure	Bedroom
2011-11-28 10:21:24	2011-11-28 10:21:31	Cabinet	Magnetic	Bathroom
2011-11-28 10:21:44	2011-11-28 10:23:31	Basin	PIR	Bathroom
2011-11-28 10:23:02	2011-11-28 10:23:36	Toilet	Flush	Bathroom
2011-11-28 10:25:44	2011-11-28 10:32:06	Shower	PIR	Bathroom
2011-11-28 10:34:23	2011-11-28 10:34:41	Fridge	Magnetic	Kitchen
2011-11-28 10:34:44	2011-11-28 10:37:17	Cupboard	Magnetic	Kitchen
2011-11-28 10:38:00	2011-11-28 10:42:41	Toaster	Electric	Kitchen
2011-11-28 10:38:33	2011-11-28 10:38:40	Fridge	Magnetic	Kitchen
2011-11-28 10:41:29	2011-11-28 10:41:36	Cupboard	Magnetic	Kitchen
2011-11-28 10:41:43	2011-11-28 10:41:59	Cooktop	PIR	Kitchen
2011-11-28 10:41:59	2011-11-28 10:42:55	Microwave	Electric	Kitchen
2011-11-28 10:49:48	2011-11-28 10:51:13	Basin	PIR	Bathroom
2011-11-28 10:51:41	2011-11-28 13:05:07	Seat	Pressure	Living
2011-11-28 13:06:04	2011-11-28 13:06:06	Basin	PIR	Bathroom

Figura 1: Dataset dei sensori

dove la tripla `Location-Type-Place` indica il sensore che ha effettuato la rilevazione. Nel dataset delle attività ogni riga contiene l'intervallo di tempo e il nome dell'attività rilevata. Ogni attività è strettamente legata alla rilevazione effettuata dai sensori, ed é rappresentata (figura 2) su una riga come:

```
Start time - End time - Activity
```

Start time	End time	Activity
2011-11-28 02:27:59	2011-11-28 10:18:11	Sleeping
2011-11-28 10:21:24	2011-11-28 10:23:36	Toileting
2011-11-28 10:25:44	2011-11-28 10:33:00	Showering
2011-11-28 10:34:23	2011-11-28 10:43:00	Breakfast
2011-11-28 10:49:48	2011-11-28 10:51:13	Grooming
2011-11-28 10:51:41	2011-11-28 13:05:07	Spare_Time/TV
2011-11-28 13:06:04	2011-11-28 13:06:31	Toileting
2011-11-28 13:09:31	2011-11-28 13:29:09	Leaving
2011-11-28 13:38:40	2011-11-28 14:21:40	Spare_Time/TV
2011-11-28 14:22:38	2011-11-28 14:27:07	Toileting
2011-11-28 14:27:11	2011-11-28 15:04:00	Lunch
2011-11-28 15:04:59	2011-11-28 15:06:29	Grooming
2011-11-28 15:07:01	2011-11-28 20:20:00	Spare_Time/TV
2011-11-28 20:20:55	2011-11-28 20:20:59	Snack
2011-11-28 20:21:15	2011-11-29 02:06:00	Spare_Time/TV

Figura 2: Dataset delle attività

HMM

Un *HMM (Hidden Markov Model)* è un modello probabilistico definito da variabili osservabili x_t e variabili nascoste y_t, dove t indica il tempo. In questo caso le variabili osservabili sono i sensori, mentre le variabili nascoste sono le attività e dipendono come in figura 3.

In un `HMM` standard le variabili nascoste y_t (y al tempo t) dipendono solo dallo stato delle variabili y_{t-1} (y al tempo $t-1$), mentre le variabili osservabili x_t (x al tempo t) dipendono solo dallo stato delle variabili y_t (y allo stesso istante t).

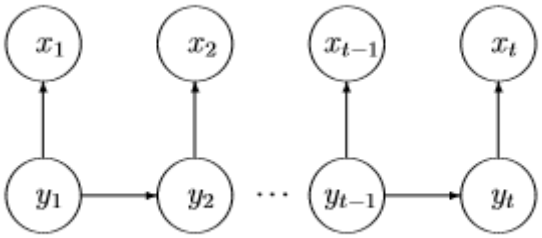

Figura 3: Dipendenze in un Hidden Markov Model

Necessari in un `HMM` sono:

- i dati relativi a $p(y)$, ovvero la probabilità a priori delle variabili y;
- $p(y_t|y_t-1)$ le probabilità di transizione da uno stato y al successivo;
- $p(x_t|y_t)$ le probabilità di una variabile x_t data y_t.

Il Software

Il software è stato sviluppato in `Python`, e permette di creare un `HMM` partendo da una sequenza di rilevazioni prese dal dataset di un appartamento.
Si è deciso di trasformare le rilevazioni dal formato testuale fornito `.txt` in un formato più compatto ed ordinato, il `csv`. Come verrà spiegato dettagliatamente nei successivi paragrafi, il software analizza i sorgenti del dataset e calcola le probabilità iniziali, di transizione degli stati e di emissione degli eventi in base alle occorrenze all'interno del file.
Successivamente, il software si avvale di 2 librerie gratutite di nome `GHMM` e `Pomegranate`, in grado di creare modelli `HMM` e molte altre funzionalità, tra cui l'utilizzo dell'algoritmo di Viterbi.
Con `GHMM` é stato possibile solo costrutire il modello HMM, mentre con `Pomegranate` si é utilizzato l'algoritmo di Viterbi ed é stato possibile fare considerazioni sul modello creato.
Entrambi i modelli vengono però salvati in appositi file di testo generati a runtime.

Le librerie

GHMM

`GHMM` (General Hidden Markov Model) é una libreria gratuita, disponibile sotto licenza LGPL (GNU Lesser General Public License), scritta in C, ma disponibile in Python

grazie a un wrapper che funge da interfaccia. Essa é in grado di costruire modelli base o estesi HMM con emissioni discrete o continue.
É disponibile all'indirizzo `http://ghmm.org/` ed é sotto la supervisione del gruppo di ricerca di Alexander Schliep presso l'Università di Rutgers. Ha inoltre a disposizione una sezione all'indirizzo `http://sourceforge.net/projects/ghmm/` dove é possibile scrivere su un forum per eventuali richieste.
Presenta però due svantaggi:

- Non é frequentemente aggiornata e supportata (ultimo update 09/2015)
- É facilmente installabile sono in ambiente Unix, in particolare sulla distibuzione Ubuntu; in ambiente OSX é molto complesso installarla a causa di diversi errori in fase di compilazione del sorgente in C; in ambiente Windows non é per niente supportata, ma solo disponibile attraverso l'utilizzo di Cygwin.

Per generare un HMM

```
model = HMMFromMatrices(sigma, DiscreteDistribution(sigma), t_adls,
o_sens_adls, p_adls)
```

con il dizionario delle emissioni `sigma`

```
sigma = IntegerRange(0, len(list_sens))
```

e con `t_adls` matrice delle transizioni, `o_sens_adls` matrice delle osservazioni e `p_adls` lista delle probabilità a priori delle variabili nascoste.

Pomegranate

`Pomegranate` é una libreria per Python, implementata in Cython (un compilatore C per Python), nata dal pre-esistente, e non piú supportato, progetto `YAHMM`, che permette la gestione di reti Bayesiane, Catene di Markov, HMM, Macchine a stati finiti e altro ancora.
É disponibile la documentazione ben aggiornata all'indirizzo
`http://pomegranate.readthedocs.io/en/latest/` ed é facilemnte installabile tramite il pacchetto `pip` eseguendo da terminale

```
pip install pomegranate
```

Per generare un HMM

```
model = HiddenMarkovModel( name="Model-name" )
```

Il package caratterizza ogni stato nascosto e le sue emissioni con la seguente sintassi:

```
ADL1 = State( DiscreteDistribution( 'sens1':  0.1, 'sens2':  0.4,
'sens3':  0.5 ), name='ADL1' )
```

Dopodichè, è necessario definire le probabilità iniziali come segue:

```
model.add_transition( model.start, ADL1, 0.6 )
```

e le probabilità di transizione tra gli stati nel seguente modo:

```
model.add_transition( ADL1, ADL2, 0.65 )
```

Successivamente, dopo la chiamata `model.bake()` che permette l'effettiva elaborazione dei dati forniti per la creazione del `HMM`, è possibile fare diversi tipi di inferenze sul modello. Come `forward`,`backward`, `forward-backward` e `Viterbi`.

Le funzioni

Di seguito sono descritte le funzioni principali del software, in modo da illustrare dettagliatamente il metodo usato per creare le probabilità utilizzate per la creazione del modello.

check_and_generate_csv

Prende in input il path di un file testuale `.txt` ed esamina riga per riga (evitando le prime due che nel dataset fornito sono di intesazione) e per ognuna verifica che lo `Start time` sia antecedente al `End time`; sono stati anche implementati, ma non funzionanti, i controlli tra gli `Start time` e `End time` tra righe consecutive. Se non sono presenti errori genera un file `.csv` con i dati controllati, altrimenti lancia un messaggio di errore che indica in quale righe sono state rilevate incorrettezze.
Questo controllo ci ha permesso di scoprire un errore nel file originale rispetto alcuni `Start` e `End time`

```
check file >> OrdonezA_ADLs.txt <<
        start/end time mismatch at line 72
         ['2011-12-01 19:28:51', '2011-12-01 16:29:59', 'Toileting']
        start/end time mismatch at line 81
         ['2011-12-02 12:20:41', '2011-12-01 10:20:59', 'Grooming']
        start/end time mismatch at line 83
         ['2011-12-02 12:27:47', '2011-12-01 11:35:49', 'Breakfast']
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
[!] there are errors in file:
[!]     >> OrdonezA_ADLs.txt <<
[!] please fix them before continuing.
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

normalize_list e normalize_matrix

Queste funzioni prendono in input rispettivamente liste e matrici e normalizzano i dati contenuti, utilizzando la divisione fornita della libreria `numpy`. In questo modo ogni lista e ogni riga della matrice ha come somma 1.0.

csv_list e csv_matrix

Come facilmente intuibile, generano file in formato `.csv` a partire dai dati in input: lista dei nomi delle variabili (su righe e colonne per la matrice) e lista (o matrice) dei valori da inserire.

obtain_p_adls

Permette di ricavare la lista delle attività rilevate, le probabilità iniziali di ogni singola attività e la sequenza di attività presente nel dataset. Questa funzione prende in input il percorso del file csv contenente le attività da analizzare e semplicemente conta le occorrenze di ciascuna stessa attività all'interno del file incrementando un contatore ad ogni rilevazione, dopodichè normalizza i dati.

obtain_t_adls

Permette di ricavare la matrice delle transizioni per le `ADLs`. Conta, ogni volta che un attività accade al tempo t, quante volte ogni singola attività compare al tempo $t-1$, in modo da generare una matrice $n \times n$ con la somma, per ogni attività, delle volte che le altre attività sono accadute successivamente. Infine normalizza i dati.

obtain_list_sens

A partire dal file in input ricava la lista dei sensori attivati durante le rilevazioni (composti dalla tripla univoca `Location-Type-Place`) e la sequenza con cui sono stati attivati.

obtain_o_sens_adls

La funzione permette di ricavare le probabilità di emissione dalle `ADLs` ai sensori. Viene tenuto conto di tutte le volte che l'intervallo di tempo di una rilevazione del sensore cade all'interno della rilevazione di un'attività. La matrice ottenuta é poi normalizzata.

Esecuzione e stime

Da ogni dataset vengono calcolate le matrici di transizione degli stati nascosti, di osservazione e le probabilità a priori e dati in pasto alla libreria pomegrante, la quale genera il modello come descritto in precedenza.
Una volta generato il nostro modello, utilizzeremo *l'algoritmo di Viterbi* con il comando

```
model.viterbi(sequence)
```

dove `sequence` é la sequenza di sensori fornita dal dataset.

Esso genera la sequenza di attività ottima generata in base alla `sequence` inserita; in questo modo possiamo verificare se la sequenza di `ADLs` risultante è coerente con quella della fornita dal dataset originale.

Parametri di affidabilità

Per ricavare informazioni dal confronto tra la sequenza di attività orginale (che chiamiamo per semplicità `seq1`) e quella generata a partire dal modello HMM (che chiamiamo `seq2`) abbiamo implementato tre differenti metodi.

diff_base

Calcola un rate in base alla differenza punto-a-punto tra le due sequenze inserite; il punteggio ideale é prossimo allo zero, poiché:

- inizializzo un contatore di diff a zero
- ogni volta che le `ADLs` tra le due sequenze alla stessa posizione non sono uguali incremento di `1` un contatore
- calcolo il rapporto tra il numero di diff e la lunghezza della sequenza di riferimento `seq1`

quindi più il rapporto é vicino allo zero, meno sono le differenze punto-a-punto tra le due sequenze.

diff_A

Assegna un rate in base all'uguaglianza o meno tra un'attività al passo x di `seq1` e l'intervallo attorno al passo x di `seq2`.

- inizializzo un contatore di diff a zero
- ogni volta confronto la `ADL` in posizione x di `seq1` con le `ADL` nell'intervallo attorno a x in `seq2`
- assegno un punteggio alla distanza dal centro dell'intervallo (`0` se il match é in posizione x, 1 se il match é in $x+1$ o $x-1$ e così via
- calcolo il rapporto tra il rate ottenuto e la lunghezza della sequenza di riferimento `seq1`

Più il rapporto é prossimo allo zero, migliore é la somiglianza tra le due sequenze.

diff_B

Assegna un rate in base allo scostamento dei match tra le due sequenze;

- ad ogni passo confronto la `ADL` in x della `seq1` e scorro la `seq2` fino a quando non trovo un match in y
- il passo successivo cerca il match tra $x+1$ della `seq1` a partire da $y+1$ della `seq2`
- calcolo il rapporto tra la lunghezza della sequenza di riferimento `seq1` con il punto di arrivo in `seq2`

Più il rapporto é vicino ad `1` e più le due sequenze sono simili e intervallate da poche differenze.

Conclusioni e sviluppi futuri

Le analisi sono fatte su ogni singolo dataset (ovvero appartamento) fornito e dai risultati ottenuti si può dire che:

- a partire dal modello costruito su *OrdonezA*, si ottiene una sequenza di attività abbastanza diversa putno-a-punto da quella originale, ma con poche differenze se si considerano gli *intorni* e la sequenza complessiva scartando eventuali *dati sporchi* (figura 4)
- a partire dal modello costruito su *OrdonezB*, si ottiene una sequenza di attività ancora più differente putno-a-punto da quella originale, ma con poche differenze se si considerano gli *intorni* e ancora meno se si considera la sequenza complessiva scartando i *dati sporchi* (figura 5)

```
punctual difference: 0.778225806452 (excellent ~= 0)
index A difference: 2.74596774194 (excellent ~= 0)
index B difference: 0.888888888889 (excellent ~= 1)
```

Figura 4: output OrdonezA

```
punctual difference: 0.855983772819 (excellent ~= 0)
index A difference: 3.68965517241 (excellent ~= 0)
index B difference: 0.989959839357 (excellent ~= 1)
```

Figura 5: output OrdonezB

Per migliorare i risultati potrebbe essere interessante approfondire l'associazione tra rilevazioni sensoristiche ravvicinate e simili, che sono identificate da un'unica attività. Si nota infatti che la nostra sequenza di `ADLs` generata con l'algoritmo di Viterbi, nonostante venga *ripulita* da ripetizioni di attività non significative, si presenta comunque

più lunga rispetto alla sequenza originale, proprio perchè va a generare un attività a partire da ogni rilevazione da sensore, a differenza dell'output fornito dal dataset.
Sarebbe anche interessate integrare con tool grafici disponibili con la libreria `Pomegranate` per poter indagare anche ad occhio nudo i procedimenti dell'algoritmo.
Un altra possibilità, avendo a disposizione un dataset di maggiori dimensioni, potrebbe essere di indagare l'attendibilità di previzione di `ADLs` non solo per ogni appartamento singolo, ma per un insieme di appartamenti, e magari inferire eventuali differenze.