UNIVERSITÀ
DEGLI STUDI
FIRENZE

Scuola di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea in Informatica

Tesi di Laurea

TITOLO ITALIANO

TITOLO INGLESE

NOME CANDIDATO

Relatore: *Relatore*
Correlatore: *Correlatore*

Anno Accademico 2014-2015

INDICE

ELENCO DELLE FIGURE

"Inserire citazione"
— Inserire autore citazione

FASI PER IL RICONOSCIMENTO DELLA POSIZIONE ALL'INTERNO DEGLI EDIFICI

L'identificazione della posizione all'interno di un edificio si svolge in 2 fasi:

1. Scansione dell'ambiente
2. Ricerca della posizione

SCANSIONE DELL'AMBIENTE

Analisi statica dell'ambiente chiuso, nel quale il software raccogliera' le onde magnetiche per ogni intervallo di tempo, le classifichera' con una semplice label la quale rappresentera' la zona di appartenenza.
La scansione a sua volta composta da diverse sotto-fasi cioè:

1. Raccoglimento dei dati
2. Estrazione del magnitudo
3. Raggruppamento
4. Estrapolazione delle *features*

Raccoglimento dei dati

Innanzitutto esaminiamo la composizione dei dati che stiamo andando ad estrarre: si tratta di onde magnetiche quindi strutturate nel seguente modo:

$$(x, y, z)$$

I tre valori sono espressi in μT (micro Tesla), unita' di misura della densita' di un flusso magnetico. Ad ogni onda magnetica viene assegnata una *label*: una stringa od un numero che identifica univocamente una parte dell'ambiente chiuso

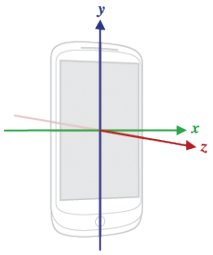


Figura 1: Raffigurazione grafica delle onde raccolte

Estrazione del magnitudo

Per estrarre l'intensità di ogni onda magnetica eseguiamo semplicemente la norma euclidea di un vettore:

$$\sqrt{x^2 + y^2 + z^2}$$

Raggruppamento

Le onde magnetiche con la stessa *label* vengono raggruppate in *fingerprints*, insiemi di dimensione prefissata. A livello logico, ogni *fingerprint* cerca di identificare univocamente un punto all'interno di una zona, identificata con una *label*. L'insieme di *fingerprints* quindi, cerca di distinguere, tramite le caratteristiche dei campi elettromagnetici di ciascun punto, ogni *label* dall'altra.

Estrapolazione delle features

Per ogni *fingerprint*, l'estrazione delle *features* consiste nell'estrazione di variabili statistiche. In questo specifico caso sono:

- Media
- Varianza
- Deviazione standard
- Mediana
- Media troncata
- Coefficiente di variazione
- Massimo
- Minimo
- $1°, 5°, 95°, 99°$ percentile
- $1°, 2°, 3°$ quartile

RICERCA DELLA POSIZIONE

Dopo aver scansionato l'ambiente questa fase viene eseguita dal cliente durante l'utilizzo dell'applicazione. La ricerca consiste nel creare sul momento una *fingerprint* e poi, tramite un algoritmo di apprendimento automatico, cercare di inferire la *label* su cui ci troviamo. Nello specifico, l'algoritmo utilizzato e' il *k nearest neightbours*.

APPRENDIMENTO SUPERVISIONATO

Gli algoritmi di apprendimento usati rientreranno tutti sono la categoria di apprendimento supervisionato, quindi dovremo fornire degli esempi con un valore associato. In termini piu' formali si parla di un insieme di addestramento $\mathbf{X}$, $\mathbf{y}$ dove $\forall x_i \in \mathbf{X}$, x_i e' un insieme di attributi mentre $y_i \in \mathbf{y}$ rappresenta il valore associato a quell'insieme di attributi. Quest'ultimo puo' essere discreto che continuo: nel primo caso si parla di classificazione mentre nel secondo di regressione.

K-NEAREST-NEIGHTBOURS

Uno degli algoritmi piu' semplici di apprendimento automatico, e' il *k-nearest-neighbours* dove l'input consiste in k elementi presi dal *training set* piu' vicini in base ad un criterio scelto da chi utilizza l'algoritmo (per esempio la distanza euclidea o di Mahalanobis).

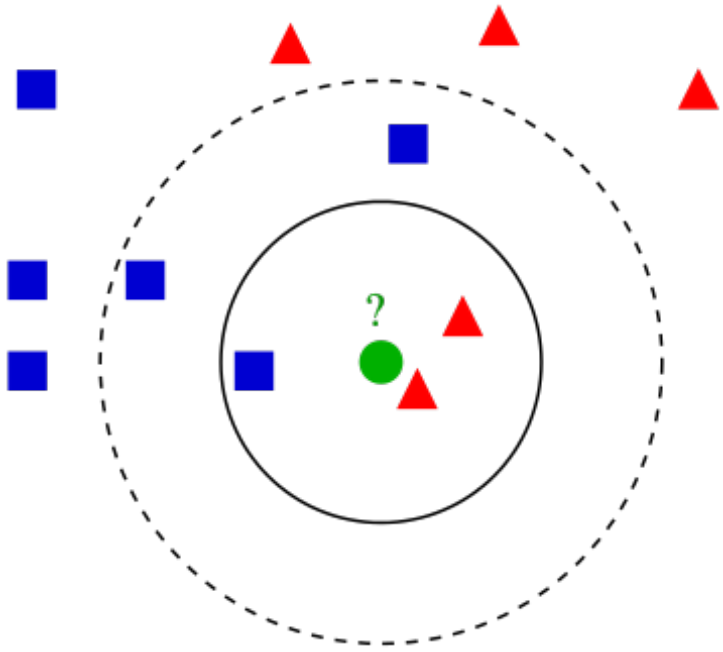


Figura 2: Esempio grafico dell'algoritmo KNN

Scelta del parametro k

La scelta del parametro dipende, ovviamente, dal tipo dei dati che abbiamo e dalla quantita', anche se in generale piu' e' grande k meno rumore viene generato da questo algoritmo. Un buon metodo per trovare il giusto valore e' l'uso di tecniche euristiche, come la *cross validation*. Un altra fonte di rumore di cui bisogna stare attenti e' la presenza di *features* insignificanti nella ricerca del vicino. Per porre rimedio possiamo, ad esempio, usare un algoritmo genetico per selezionare le *features* piu' significative

Cross validation

Tecnica per migliorare le performance di un classificatore, consiste nel suddividere l'intero *dataset* in n parti uguali (di solito 10), ognuna delle quali svolgera' per una volta il ruolo di *validation set* mentre il resto sara' il *training set*. Questa tecnica risolve vari problemi tra i quali l'*overfitting*.

NAIVE BAYES

Un altro tipo di apprendimento usato per classificare le *label* e' Naive bayes: un algoritmo di classificazione e regressione basato sulla statistica. Prima di spiegare in cosa consiste occorre spiegare un paio di concetti:

Teorema di bayes

: esso si fonda sul famoso teorema di bayes enunciato come segue:

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)}$$

Rete bayesiana

Una rete bayesiana e' un grafo diretto aciclico i cui nodi rappresentano le variabili casuali del sistema mentre gli archi rappresentato la condizione di dipendenza fra nodi. Ad ogni nodo e' associata una tabella di distribuzione delle probabilita' la cui complessita' e' proporzionale al numero di archi entranti.
Per esempio se il nodo con variabile casuale Leggere ha un arco verso

Istruito allora possiamo dire che Istruito e' condizionalmente dipendente da Leggere. Qui sotto potete vedere una raffigurazione grafica di una semplice rete bayesiana formata da nodi padre ed un figlio:

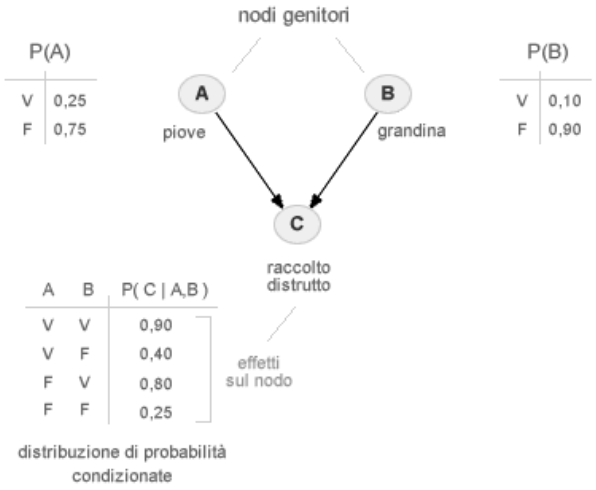


Figura 3: Esempio di rete bayesiana

Naive Bayes

Naive bayes e' una rete bayesiana in cui si assume l'indipendenza condizionale fra tutte le variabili casuali del sistema data la classe. Questa forte assunzione non mira a modellare esattamente la realta' ma fornisce delle buone performance sulla predizione di una classe.

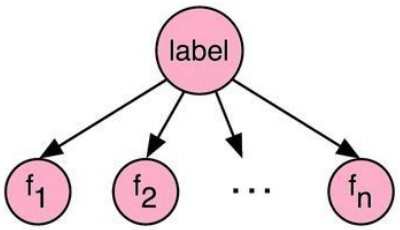


Figura 4: Esempio di rete bayesiana ingenua

Un piccolo esempio basato su naive bayes

Supponiamo di dover usare Naive Bayes per predire se giocare una partita di tennis o no in base alle condizioni meteorologiche. Dati gli esempi qui sotto a sinistra, possiamo ricavare la tabella di distribuzione delle probabilita' generale come qui di seguito:

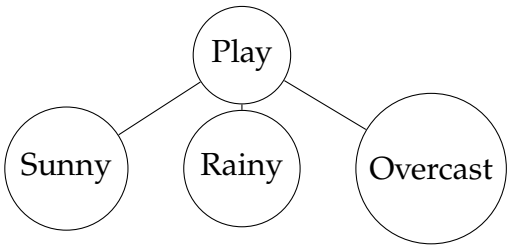
Weather	Play
Sunny	No
Overcast	Yes
Rainy	Yes
Sunny	Yes
Sunny	Yes
Overcast	Yes
Rainy	No
Rainy	No
Sunny	Yes
Rainy	Yes
Sunny	No
Overcast	Yes
Overcast	Yes
Rainy	No

Frequency Table		
Weather	No	Yes
Overcast		4
Rainy	3	2
Sunny	2	3
Grand Total	5	9

Likelihood table				
Weather	No	Yes		
Overcast		4	=4/14	0.29
Rainy	3	2	=5/14	0.36
Sunny	2	3	=5/14	0.36
All	5	9		
	=5/14	=9/14		
	0.36	0.64		

Figura 5: Esempi di partite di tennis giocate in base alle condizioni meteorologiche e tabella di distribuzione delle probabilita'

Ecco una rappresentazione grafica di *Naive Bayes*:



ALBERI DI DECISIONE

Gli alberi di decisione sono un altro tipo di apprendimento supervisionato, quindi dovremmo avere Da un punto di vista strutturale, l'albero di decisione e' un albero (inteso come struttura dati) dove i nodi interni sono gli attributi, i rami tutti i possibili valori assumibili dall'attributo (oppure un range nel caso continuo) e le foglie sono la predizione da scegliere
D'ora in poi parleremo, per semplicita', solo di classificazione quando non espresso chiaramente.

Costruzione di un albero di decisione

L'albero di decisione si potrebbe definire prendendo a caso un attributo ed iniziare a dividere gli esempi fino ad ottenere foglie, anche se cosi' otterremo un albero non molto utile in tutti i casi in cui non abbiamo gli esempi. Quindi quale albero scegliere fra tutti quelli possibili? In questo caso ci aiuta il rasoio di Occam, che ci dice di scegliere quello piu' piccolo fra tutti. Per generare l'albero piu' piccolo dovremmo scegliere gli attributi piu' significativi per generare l'albero. Cosa intendiamo per significativo? Intendiamo l'attributo che genera figli con meno varieta' di classi presenti all'interno di essi.
Un esempio: immaginiamo di avere 10 esempi con attributi A e B e come valore associato un booleano. se abbiamo l'attributo A che genera due figli con esempi aventi meta' valore vero e falso mentre se suddividiamo secondo B abbiamo figli con esempi esclusivamente veri oppure falsi. In questo caso indubbiamente l'attributo piu' significativo e' B.

Classificazione dei nuovi elementi

Quando dovremmo predire l'etichetta di un nuovo insieme di attributi x bastera' semplicemente scorrere l'albero di decisione dalla radice fino ad una foglia, che sara' l'etichetta da assegnare.

Un esempio di albero di decisione

Riprendiamo l'esempio della partita di tennis, questa volta con qualche attributo in piu'. La tabella e' la seguente:

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Figura 6: Tabella degli attributi meteorologici con valore associato un booleano che stabilisce se la partita e' stata giocata o no

da cui possiamo ricavare il seguente albero di decisione:

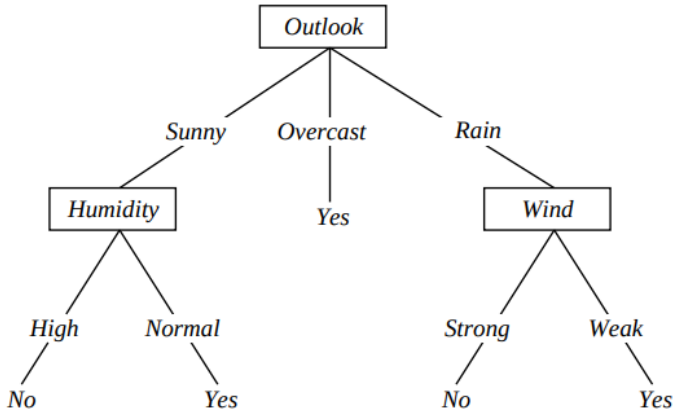


Figura 7: Albero di decisione ricavato dalla tabella precedente

STRUTTURA DEL SOFTWARE

LINGUAGGI E FRAMEWORK

L'applicazione ha come *target platform* Android percio' il linguaggio usato principalmente e' stato Java. Da notare che su Android e' presente nella versione 7, percio' non sono disponibili alcune funzionalita' come i metodi di default, *stream*, *lambda expression* ecc. Per rimediare soprattutto alla mancanza di quest'ultime, che migliorano la lettura e scrittura di alcune parti di codice, ho affiancato un altro linguaggio a Java: Kotlin. Senza fare un confronto tra i due linguaggi, Kotlin mi ha permesso di scrivere lambda expression compilando un bytecode perfettamente compatibile con Java 6.

INTERFACCIA GRAFICA

L'interfaccia e' stata realizzata ai fini di test pratici delle funzionalita' finali dell'applicazione quindi non ha una grande cura da un punto di vista estetico come vedremo piu' avanti. La parte alta contiene delle label raffiguranti i 3 valori catturati tramite il magnetometro e presi tramite le API di Android.
Nel mezzo ci sono dei pulsanti per:

- Iniziare/Terminare la scansione dell'ambiente.
- Incrementare la *label* che verra' assegnata alla prossima *fingerprint* registrata
- Iniziare/Terminare la ricerca.
- Serializzare tutti i dati registrati finora
- De-serializzare i dati salvati in un JSON.

Nella parte bassa invece c'e' una *textbox* contenente il log che verra' stampato durante l'esecuzione del programma.

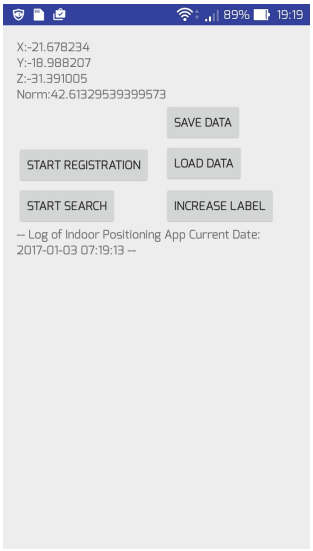


Figura 8: Interfaccia grafica all'avvio dell'applicazione

PERSISTENZA DEI DATI

L'applicazione consente anche di serializzare tutti i dati registrati fino a quel momento nel formato standard JSON tramite la libreria *gson* che fornisce delle funzioni per il linguaggio Java per la serializzazione/deserializzazione di oggetti. Il file viene salvato nella cartella dati dell'applicazione non visibile all'utente.

STRUTTURA DEL CODICE E DESIGN PATTERN

Nello sviluppo del software sono stati applicati vari *design pattern* visti durante i vari corsi e principi di programmazione. Fra questi ultimi abbiamo il *dependency inversion principle*, il *open closed principle*. Riguardo i *design pattern*, ho usato molto l'*observer*, il *template* e *factory*.

ANALISI DEI DATI

La predizione dei risultati e' stata implementata sia nel software *Android* sia sul computer. Nel codice mobile e' stato adoperato solamente il KNN per via della facilita' d'implementazione da zero, anche se non e' stato utilizzato per testare la precisione, ma per verificare il corretto funzionamento dell'applicazione. Invcece su computer, presi i dati serializzati dal software mobile, sono stati applicati tutti gli algoritmi di apprendimento

elencati precedentemente e gia' tutti implementati da librerie di terze parti per verificare la precisione dei dati. Il linguaggio scelto su computer e' *Python* per via del suo buon supporto all' apprendimento automatico.

BASE DEI TEST

Per testare l'effettivo funzionamento dell'applicazione ho usato alcune stanze di casa mia ed ho assegnato a ciascuna di esse una *label*. Qui di seguito una piccola piantina rappresentante le stanze utilizzate:

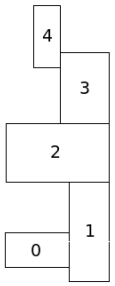


Figura 9: Una piccola raffigurazione delle stanze usate per le prove con sopra scritto la *label* assegnata

RISULTATI DEI TEST