



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

Nome candidato: *Titolo italiano*, Corso di Laurea in Informatica, © Anno  
Accademico 2014-2015

---

## INDICE

---



---

## ELENCO DELLE FIGURE

---

Figura 1	Raffigurazione grafica delle onde raccolte	8
Figura 2	Un esempio grafico di insieme d'addestramento ed una sua classificazione	10
Figura 3	Classificazione e regressione a confronto	10
Figura 4	Esempio grafico dell'algoritmo KNN	12
Figura 5	Esempio di rete bayesiana	13
Figura 6	Esempio di rete bayesiana ingenua	14
Figura 7	Esempi di partite di tennis giocate in base alle condizioni meteorologiche e tabella di distribuzione delle probabilita'	16
Figura 8	Tabella degli attributi meteorologici con valore associato un booleano che stabilisce se la partita e' stata giocata o no	18
Figura 9	Albero di decisione ricavato dalla tabella precedente	18
Figura 10	Interfaccia grafica all'avvio dell'applicazione	20
Figura 11	Una piccola raffigurazione delle stanze usate per le prove con sopra scritto la <i>label</i> assegnata	22



*"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 raccoglierà le onde magnetiche per ogni intervallo di tempo, le classificherà con una semplice label la quale rappresenterà 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. Estrazione degli attributi

## 8 Elenco delle figure

### *Raccolgimento 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  $\mu T$  (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

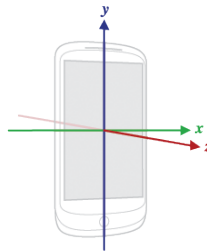


Figura 1: Raffigurazione grafica delle onde raccolte

### *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.

### *Estrazione degli attributi*

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 classificazione, cercare di inferire la *label* su cui ci troviamo. Gli algoritmi utilizzati per analizzare il problema sono i seguenti

1. *K Nearest Neighbour*
2. Bayes ingenuo
3. Alberi di decisione

## APPENDIMENTO SUPERVISIONATO

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

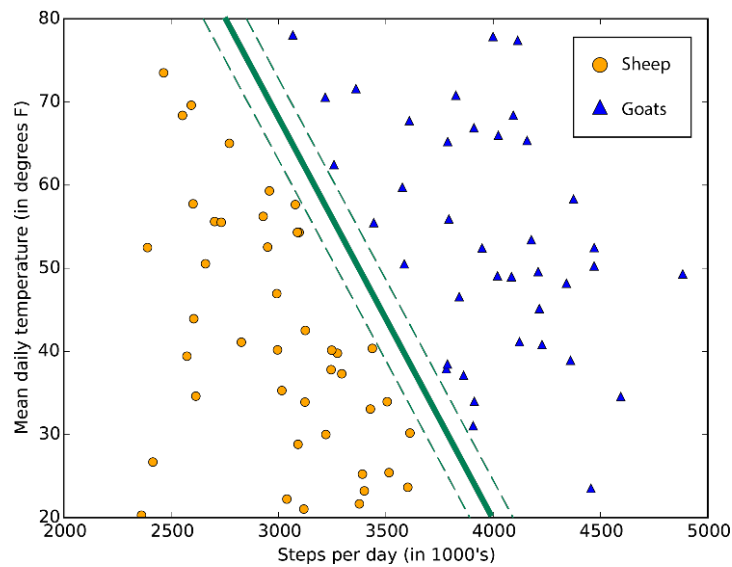


Figura 2: Un esempio grafico di insieme d'addestramento ed una sua classificazione

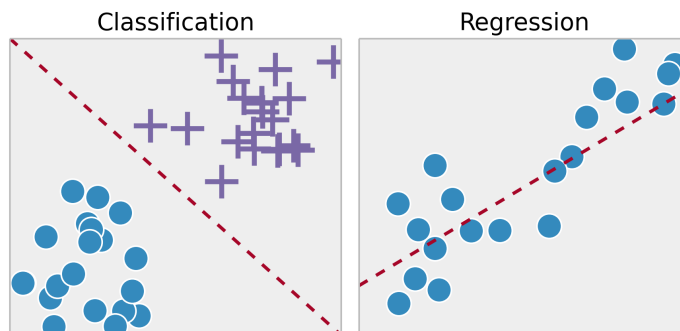


Figura 3: Classificazione e regressione a confronto

### *Verifica delle presatzioni*

Per verificare la correttezza del classificatore viene diviso in 2 parti il *dataset* a nostra disposizione: il primo si chiamera' insieme di addestramento mentre il secondo insieme di test. Il nostro classificatore si allenera' sull'insieme di addestramento ed effetuera' le predizioni sull'insieme di test, su cui verranno misurate le *performance* in base alle predizioni corrette.

### *Cross validation*

Per migliorare le performance di un classificatore si suddivide l'intero *dataset* in  $n$  parti uguali (di solito 10), ognuna delle quali svolgera' per una volta il ruolo di insieme di test mentre il resto sara' l'insieme di addestramento. Questa tecnica risolve vari problemi tra i quali l'*overfitting*.

### K NEAREST NEIGHTBOUR

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).

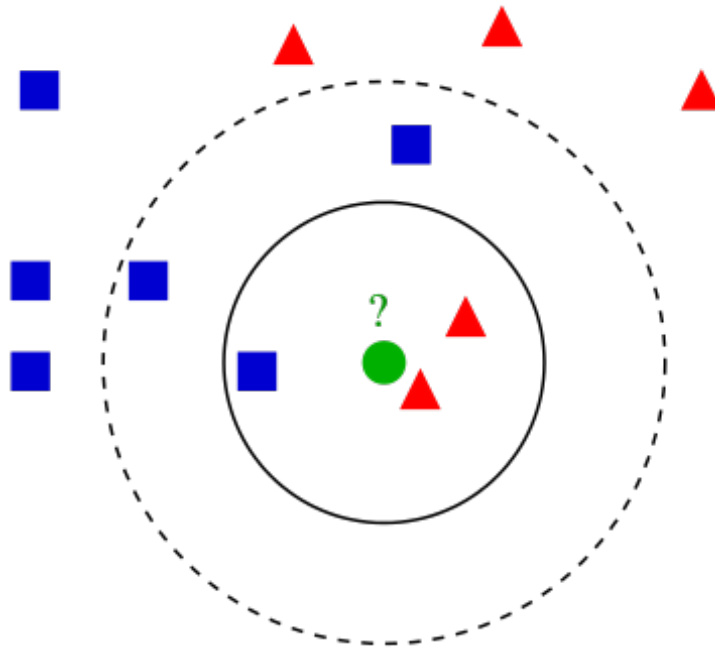


Figura 4: Esempio grafico dell'algoritmo KNN

### Scelta del parametro $k$

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

### NAIVE BAYES

Un altro tipo di apprendimento usato per classificare le *label* è 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 rappresentano 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:

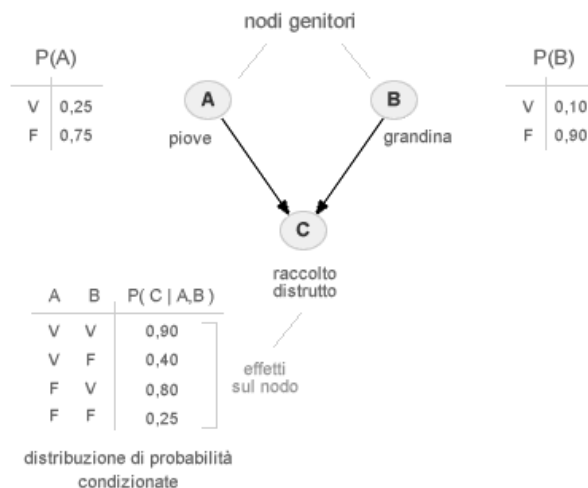


Figura 5: Esempio di rete bayesiana

### 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.

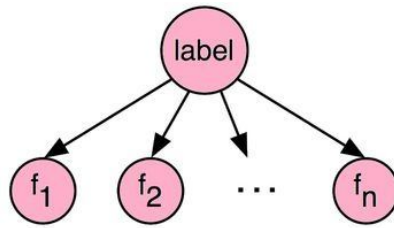


Figura 6: Esempio di rete bayesiana ingenua

*Costruzione di una rete bayesiana ingenua*

Preso l'insieme di esempi, ricavare la tabella di distribuzione delle probabilit  al variare di tutti i valori che la variabile dipendente puo' assumere. Successivamente costruire un grafo che ha come nodo radice la variabile dipendente e come figli tutte le variabili indipendenti.

*Predizione di risultati*

Preso un insieme di attributi  $x$  con valori  $x_1x_2x_3...x_n$  e tutti i valori della variabile dipendente  $y = y_1y_2y_3...y_m$ , per classificare l'insieme bastera' applicare il teorema di bayes nel seguente modo:

$$\begin{aligned} \text{etichetta} = y_j \text{ in } \max_i P(y_j | X_1 = x_1, X_2 = x_2 \dots X_N = x_n) \\ = \max_i \frac{P(X_1 = x_1, X_2 = x_2 \dots X_N = x_n | y_j) P(y_j)}{P(X_1 = x_1, X_2 = x_2 \dots X_N = x_n)} \end{aligned}$$

i cui valori della precedente equazione si possono ricavare dalla tabella di distribuzione delle probabilit .

*Modelli generativi e discriminativi*

L'espressione  $P(y_j | x)$  puo' essere calcolata in modi differenti in base al modello trattato: nel caso di modelli discriminativi, l'espressione viene ricavata direttamente dai dati sorgente mentre nei modelli generativi l'obiettivo e' di costruire una distribuzione congiunta di probabilit  per  $P(x, y_j)$  direttamente oppure ricavando prima  $P(x | y_j)$  e  $P(y_j)$ . La maggior parte di tutti gli algoritmi di classificazione rientrano nell'insieme dei modelli discriminativi mentre *Bayes ingenuo* e' uno dei pochi generativi.



Per questo motivo, non esiste un unico *Bayes ingenuo* ma diversi, che si diversificano in base al tipo di attributi che andiamo a trattare. Essi sono:

- Bernoulli: gli attributi sono di tipo *booleano* e l'attributo  $x_i \in \mathbf{x} = (x_1, x_2, \dots, x_n)$  vale 1 se l'evento  $i$  è avvenuto, o altrimenti. Nel caso bernoulliano possiamo valutare  $P(\mathbf{x}|y_j)$  come

$$P(\mathbf{x}|y_j) = \prod_{i=1}^n P_{ij}^{x_i} (1 - P_{ij})^{(1-x_i)}$$

Dove  $P_{ij}$  è la probabilità per  $y_j$  che  $x_i$  sia vero. Un esempio canonico è quello della classificazione dei documenti, dove  $x_i$  rappresenta la presenza del termine  $w_j$  nei documenti di classe  $y_j$  e di conseguenza  $P_{ij}$  la probabilità di trovarlo. Da notare bene che Bernoulli non è una semplice multinomiale che conta 0 ed 1 ma considera anche la probabilità che  $w_j$  non sia presente.

- Multinomiale: Nella multinomiale gli esempi rappresentano la frequenza con il quale gli eventi sono stati generati dalla multinomiale  $(p_1 \dots p_n)$  dove  $p_i$  è la probabilità che  $i$  occorra. Gli attributi  $\mathbf{x} = (x_1, x_2 \dots x_n)$  contano quante volte l'evento  $i$  è avvenuto. La probabilità condizionata  $P(\mathbf{x}|y_j)$  è stimata come

$$P(\mathbf{x}|y_j) = \frac{(\sum_i x_i)!}{\prod_i x_i!} \prod_i p_{ji}^{x_i}$$

- Gaussiana: viene utilizzata con dati continui e si assume che siano distribuiti in base alla Gaussiana. Per ogni classe  $y_j$ , viene ricavata la media  $\mu_j$  e la varianza  $\sigma_j^2$ . Supponiamo di aver raccolto un insieme di valori  $v$  allora la probabilità sarà:

$$P(x = v|y_j) = \frac{1}{\sqrt{2\pi\sigma_j^2}} e^{-\frac{(v - \mu_j)^2}{2\sigma_j^2}}$$

Un'altra possibile opzione per trattare valori continui è la discretizzazione da cui possiamo utilizzare Bernoulli/Multinomiale anche se bisogna stare attenti a non perdere informazioni discriminanti.

### 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 probabilit  generale come qui di seguito:

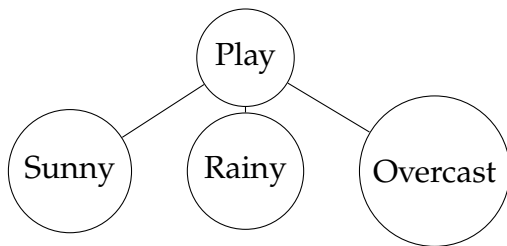
Weather	Play
Sunny	No
Overcast	Yes
Rainy	Yes
Sunny	Yes
Sunny	Yes
Overcast	Yes
Overcast	Yes
Rainy	No
Rainy	No
Sunny	Yes
Rainy	Yes
Sunny	No
Overcast	Yes
Overcast	Yes
Rainy	No

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 7: Esempi di partite di tennis giocate in base alle condizioni meteorologiche e tabella di distribuzione delle probabilit 

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   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 semplicit , 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 così 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 più piccolo fra tutti. Per generare l'albero più piccolo dovremmo scegliere gli attributi più significativi per generare l'albero. Cosa intendiamo per significativo? Intendiamo l'attributo che genera figli con meno varietà 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 più significativo è B.

*Classificazione dei nuovi elementi*

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

*Un esempio di albero di decisione*

Riprendiamo l'esempio della partita di tennis, questa volta con qualche attributo in più. La tabella è 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 8: 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:

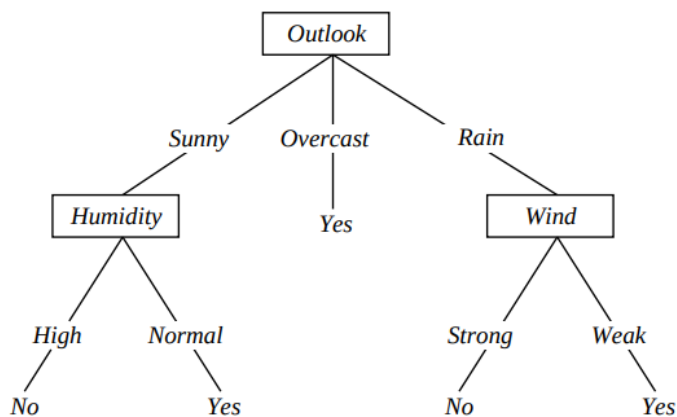


Figura 9: Albero di decisione ricavato dalla tabella precedente

---

## STRUTTURA DEL SOFTWARE

---

### LINGUAGGI E FRAMEWORK

L'applicazione ha come *target platform* Android perciò il linguaggio usato principalmente e' stato Java. Da notare che su Android e' presente nella versione 7, perciò 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.

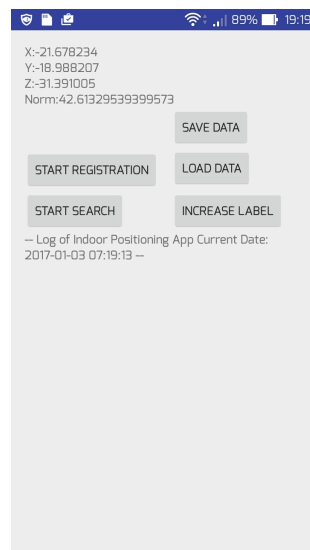


Figura 10: Interfaccia grafica all'avvio dell'applicazione

#### 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/de-serializzazione 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 è stata implementata sia nel software *Android* sia sul computer. Nel codice mobile è stato adoperato solamente il KNN per via della facilità d'implementazione da zero e non è stato utilizzato per testare la precisione, ma per verificare il corretto funzionamento dell'applicazione. Invece su computer, presi i dati serializzati dal software mobile, sono stati applicati tutti gli algoritmi di apprendimento elencati

precedentemente e già' 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:

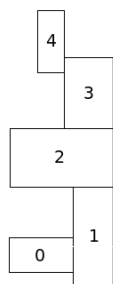


Figura 11: Una piccola raffigurazione delle stanze usate per le prove con sopra scritto la *label* assegnata

#### RISULTATI DEI TEST