



UNIVERSITÀ
DEGLI STUDI
FIRENZE

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

Tesi di Laurea

SVILUPPO DI UN'APPLICAZIONE
ANDROID PER IL POSIZIONAMENTO
INDOOR

DEVELOPEMENT OF AN INDOOR
POSITIONING ANDROID APPLICATION

MICHELE DE VITA

Relatore: *Andrea Ceccarelli*

Anno Accademico 2016-2017

INDICE

1	Introduzione	7
2	Fondamentali	9
2.1	Fase 1: Raccoglimento ed elaborazione dei dati	9
2.1.1	Onde Magnetiche	9
2.1.2	Estrazione del magnitudo	10
2.1.3	Raggruppamento	11
2.1.4	Estrazione degli attributi	11
2.1.5	Normalizzazione e standardizzazione	11
2.2	Fase 2: costruzione di un modello	12
2.2.1	IA ed Apprendimento	12
2.2.2	Tipi di apprendimento	13
2.2.3	Alcune nozioni sull'apprendimento automatico	15
2.2.4	K Nearest Neighbour	17
2.2.5	Naive bayes	19
2.2.6	Alberi di decisione	23
2.3	Fase 3: Ricerca della posizione	25
2.3.1	Ricerca di nuovi elementi nei classificatori	26
3	Struttura del software	29
3.1	Linguaggi e <i>framework</i>	29
3.2	Applicazione	29
3.3	UML del codice	31
3.4	Interfaccia grafica	32
3.5	Persistenza dei dati	35
3.6	Struttura del codice e design pattern	36
4	Test	37
4.1	Analisi dei dati	37
4.2	Piano dei test	37
4.3	Caratteristiche dei dati	38
4.4	Codice per l'analisi dati col <i>knn</i>	39
4.5	Classificatori a confronto	40
4.6	Analisi del Knn al variare dell'iper parametro k	42
4.7	Analisi del rumore durante la cattura dei dati	46
4.8	Un rimedio ingenuo al rumore	46
5	Miglioramenti e conclusioni	49
5.1	Possibili miglioramenti per la raccolta di dati	49

2 Indice

5.2 Conclusioni 49

ELENCO DELLE FIGURE

Figura 1	Assi x, y, z centrati sul cellulare	10
Figura 2	Un esempio di mappa del campo magnetico	10
Figura 3	Un esempio grafico di insieme d'addestramento ed una sua classificazione	13
Figura 4	Classificazione e regressione a confronto	14
Figura 5	Apprendimento supervisionato e non supervisionato a confronto	14
Figura 6	L'apprendimento con rinforzo e' molto adatto ai giochi, come per esempio pacman	15
Figura 7	Un diagramma di flusso che mostra le operazioni di verifica delle prestazioni	15
Figura 8	Effetti del sovradattamento	16
Figura 9	Rappresentazione grafica della Cross Validation	16
Figura 10	Esempio grafico dell'algoritmo KNN	18
Figura 11	Esempio di rete bayesiana	20
Figura 12	Esempio di rete bayesiana ingenua	20
Figura 13	Esempi di partite di tennis giocate in base alle condizioni meteorologiche e tabella di distribuzione delle probabilita'	22
Figura 14	Rappresentazione grafica di <i>Naive bayes</i> nell'esempio del tennis	23
Figura 15	Un albero e un albero di decisione a confronto. Nel secondo abbiamo come attributi <i>Smoker, Age, Diet</i> e come etichetta <i>Less Risk, More Risk</i> riferito alle malattie cardiache.	23
Figura 16	Tabella degli attributi meteorologici con valore associato un booleano che stabilisce se la partita e' stata giocata o no	25
Figura 17	Albero di decisione ricavato dalla tabella precedente	25
Figura 18	Diagramma delle classi dell'applicazione	32
Figura 19	Interfaccia grafica all'avvio dell'applicazione	33
Figura 20	Immagine dell'applicazione durante la raccolta dati	34

4 Elenco delle figure

Figura 21	Immagine dell'applicazione durante la ricerca della posizione	35
Figura 22	Una piccola raffigurazione delle stanze usate per le prove con sopra scritto la <i>label</i> assegnata	38
Figura 23	Grafico in 2 dimensioni della media e varianza di tutte le onde magnetiche. I colori dei punti rappresentano le etichette	39
Figura 24	Percentuale d'errore nei test dei classificatori	41
Figura 25	Percentuale d'errore nei test dei classificatori con la <i>cross validation</i>	41
Figura 26	Numero di errori nella predizione per etichetta	42
Figura 27	Percentuale di errore nella predizione per etichetta. Le barre blu rappresentano i risultati dei classificatori <i>cross-validati</i> mentre i rossi sono quelli senza.	42
Figura 28	Accuratezza del knn al variare di K	43
Figura 29	Classificazione binaria con il 1-nn	44
Figura 30	Classificazione binaria con il 20-nn	44
Figura 31	<i>Underfitting</i> ed <i>overfitting</i> nella regressione	45
Figura 32	Curve di validazione	45
Figura 33	Valori del magnetometro rispetto agli assi x ed y stando fermo	46
Figura 34	Percentuale d'errore al variare della grandezza dell'insieme di addestramento con KNN	47

"Puttana la madonna"
— *Inserire autore citazione*

INTRODUZIONE

La seguente tesi e' basata su un tirocinio esterno svolto con l'azienda KeepUp in cui e' stata sviluppata la base di un'applicazione Android col compito di localizzare all'interno degli edifici la posizione dello *smartphone* sfruttando le distorsioni del campo magnetico terrestre.

Un'applicazione del genere ha molti usi in luoghi chiusi aperti al pubblico: per esempio immaginiamoci di trovarci in un museo. In questo caso l'applicazione ufficiale del museo che supporta l'*indoor positioning* ci localizza all'interno di una mappa 2D dell'edificio. L'applicazione capisce quando ci avviciniamo ad un'opera d'arte e quando avviene, si apre un *pop-up* che ci fornisce informazioni aggiuntive su cio' che stiamo osservando. Oppure immaginiamoci di dover prendere l'aereo e di essere in ritardo in un aeroporto all'estero che non conosciamo affatto. Un navigatore *indoor* potrebbe far molto comodo a chi si trova in questa situazione perche' gli permetterebbe di raggiungere il proprio *gate* in pochissimo tempo senza conoscere la piantina dell'aeroporto.

Durante il primo capitolo, i fondamentali, approfondiremo il tipo di dato che dobbiamo trattare, le sue origini e la sua struttura per poi passare all'apprendimento automatico, usato per predire la posizione dell'utente all'interno dell'edificio elencandone i vari tipi e definendo alcuni termini generali per concludere con la descrizione approfondita di alcuni classificatori molto conosciuti nel mondo dell'AI.

Nel secondo capitolo parleremo della struttura del software realizzato durante il tirocinio, dal codice mobile Android a quello per la verifica dei risultati.

Nel terzo capitolo vedremo i risultati ottenuti durante il tirocinio tramite vari grafici che mostreranno punti di forza e debolezza della nostra ap-

plicazione.

Nel quarto capitolo discuteremo dei possibili miglioramenti futuri all'applicazione.

FONDAMENTALI

L'identificazione della posizione all'interno di un edificio si svolge in 3 fasi:

1. Scansione dell'ambiente
2. Costruzione di un modello
3. Ricerca della posizione

Nelle successive 3 sezioni discuteremo dei fondamenti teorici dietro ciascuna di queste fasi.

2.1 FASE 1: RACCOGLIMENTO ED ELABORAZIONE DEI DATI

Il raccoglimento dei dati avviene tramite il magnetometro del nostro *smartphone* da cui viene catturato diverse volte in un secondo il campo magnetico intorno ad esso. Ad ogni onda magnetica viene assegnata una *label*: un numero che identifica univocamente una parte dell'ambiente chiuso il cui uso verrà definito durante la fase di costruzione del modello.

2.1.1 *Onde Magnetiche*

Le onde magnetiche sono un vettore di 3 elementi, quindi in \mathbb{R}^3 che classifica. Il primo valore rappresenta la forza del campo magnetico lungo l'asse X, il secondo lungo Y ed il terzo lungo Z.

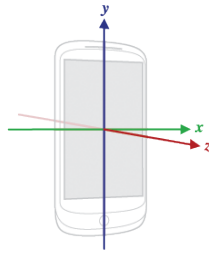


Figura 1: Assi x, y, z centrati sul cellulare

I tre valori sono espressi in μT (micro Tesla), unita' di misura della densita' di un flusso magnetico. Le onde magnetiche raccolte sono dati continui sia positivi che negativi. L'intensita' dell'onda deriva dalla distorsione del campo magnetico generata dagli oggetti statici intorno al punto e dipende anche dalla velocita' con cui ci muoviamo per cui, per semplicita', assumeremo d'ora in poi una velocita' costante di 3 piedi al secondo.

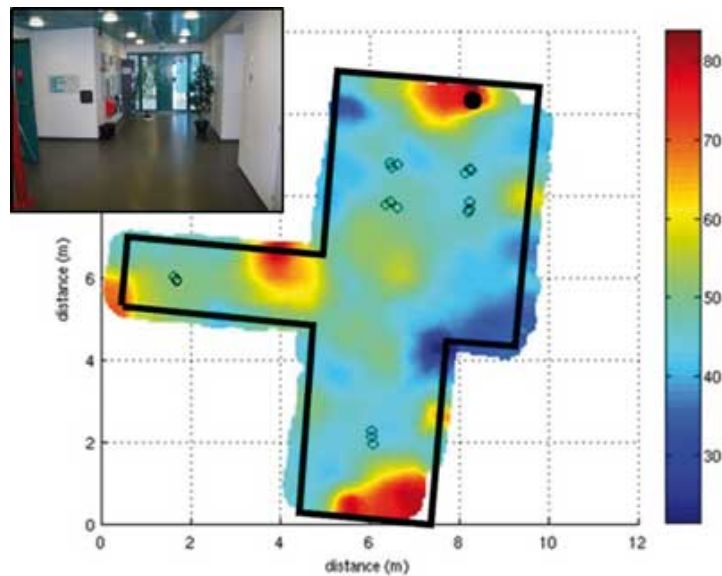


Figura 2: Mappa del campo magnetico all'universita' di Oulu: Discus Entrance hall

2.1.2 Estrazione del magnitudo

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

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

2.1.3 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.

2.1.4 Estrazione degli attributi

Per ogni *fingerprint*, l'estrazione degli attributi consiste nel creare nuovi attributi in funzione di quelli già esistenti. Una possibile applicazione è l'estrazione di variabili statistiche dagli attributi già presenti come per esempio:

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

2.1.5 Normalizzazione e standardizzazione

2 tecniche molto usate per poter confrontare i valori dei nostri attributi in una scala comune sono la normalizzazione e la standardizzazione. La prima centra porta la media dei valori di quell'attributo a 0 e la

deviazione standard ad 1 mentre la seconda porta tutti i valori in un intervallo $[0, 1]$

L'equazione della normalizzazione per il j -esimo attributo di tipo x_i e' la seguente:

$$x_{ij} = \frac{x_{ij} - \mu_i}{\sigma_i}$$

dove μ_i e' la media per gli attributi di tipo i e σ_i la deviazione standard. Adesso vediamo la standardizzazione:

$$x_{ij} = \frac{x_{ij} - x_{\min}}{x_{\max} - x_{\min}}$$

2.2 FASE 2: COSTRUZIONE DI UN MODELLO

Dopo aver raccolto ed elaborato le onde magnetiche, un algoritmo creera' un classificatore in grado di assegnare un'etichetta ai nuovi input ricevuti durante l'uso dell'utente finale. Sono stati utilizzati vari classificatori per cercare il piu' preciso fra tutti. Qui di seguito introdurremo la teoria dietro ad ogni classificatore utilizzato.

2.2.1 IA ed Apprendimento

La definizione secondo wikipedia ¹ di IA e' la seguente:

Definizioni specifiche possono essere date focalizzandosi o sui processi interni di ragionamento o sul comportamento esterno del sistema intelligente ed utilizzando come misura di efficacia o la somiglianza con il comportamento umano o con un comportamento ideale, detto razionale:

- Agire umanamente: il risultato dell'operazione compiuta dal sistema intelligente non e' distinguibile da quella svolta da un umano.
- Pensare umanamente: il processo che porta il sistema intelligente a risolvere un problema ricalca quello umano. Questo approccio è associato alle scienze cognitive.
- Pensare razionalmente: il processo che porta il sistema intelligente a risolvere un problema è un procedimento formale che si rifà alla logica.
- Agire razionalmente: il processo che porta il sistema

¹ https://it.wikipedia.org/wiki/Intelligenza_artificiale

L'apprendimento automatico e' una branca dell'intelligenza artificiale che permette al computer di apprendere da insiemi di dati per generare conoscenza con lo scopo di effettuare previsioni.

2.2.2 Tipi di apprendimento

Esistono 3 tipi di apprendimento:

- Apprendimento supervisionato: al calcolatore vengono forniti esempi del tipo (x, y) per poter apprendere. L'obiettivo della predizione per nuovi input sara' quello di calcolare la variabile dipendente. Y puo' essere sia discreta che continua: nel primo caso si parla di classificazione, nel secondo di regressione.

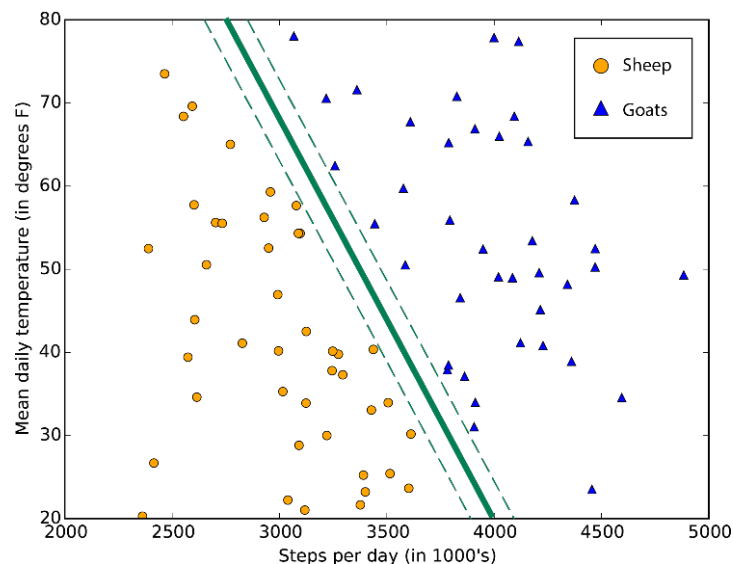


Figura 3: Un esempio grafico di insieme d'addestramento ed una sua classificazione

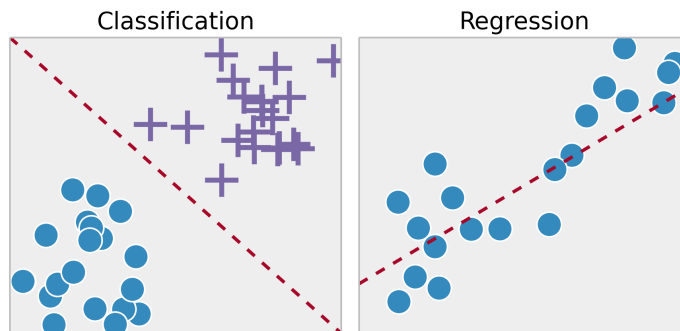


Figura 4: Classificazione e regressione a confronto

- **Apprendimento non supervisionato:** Gli esempi non contengono una variabile dipendente ma solo un insieme di attributi x . L'obiettivo è quello di inferire pattern nascosti dai dati non etichettati. Un'importante applicazione è il *clustering*: raggruppare i dati in base ad una similarità fra di essi.

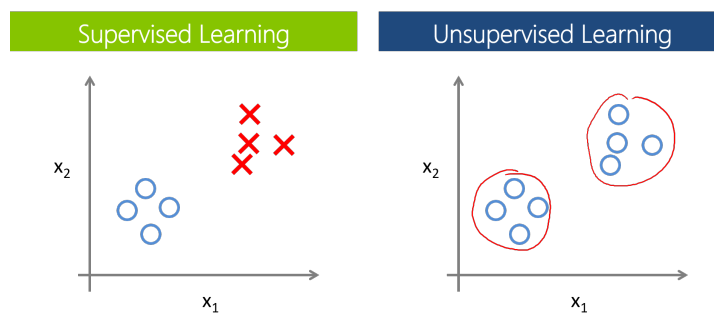


Figura 5: Apprendimento supervisionato e non supervisionato a confronto

- **Apprendimento con rinforzo:** per apprendere viene fornita una funzione ricompensa cioè una funzione che, data un'azione effettuata dall'agente, restituirà una ricompensa di tipo numerico.



Figura 6: L'apprendimento con rinforzo e' molto adatto ai giochi, come per esempio pacman

2.2.3 Alcune nozioni sull'apprendimento automatico

2.2.3.1 Verifica delle prestazioni

Per verificare la correttezza del classificatore viene diviso in 2 parti il *dataset* a nostra disposizione: il primo si chiama insieme di addestramento mentre il secondo insieme di test. Il nostro classificatore si allenera' sull'insieme di addestramento, cioe' imparera' dagli esempi come classificare i nuovi input e verifichera' l'efficacia dell'apprendimento sugli esempi in base ad una metrica. Ne esistono diverse ma noi useremo l'accuratezza, ovvero il numero di classificazioni corrette diviso il numero di esempi nell'insieme di test

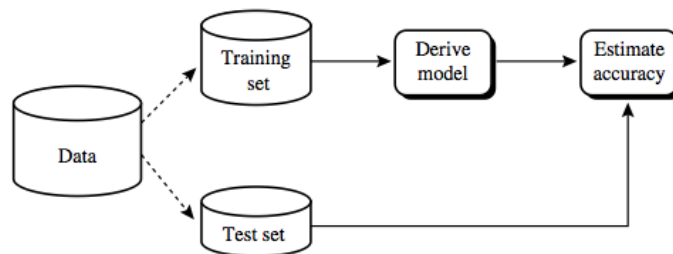


Figura 7: Un diagramma di flusso che mostra le operazioni di verifica delle prestazioni

2.2.3.2 Parametri ed iperparametri

Ogni modello e' composto da 2 tipi di valori che ne stabiliscono l'efficacia: i parametri, i quali sono determinati internamente dal modello in base al dataset mentre gli iperparametri sono stabiliti dall'utente prima dell'addestramento del modello e modificano profondamente l'efficacia di predizione. Un esempio che vedremo e' l'iperparametro k del knn.

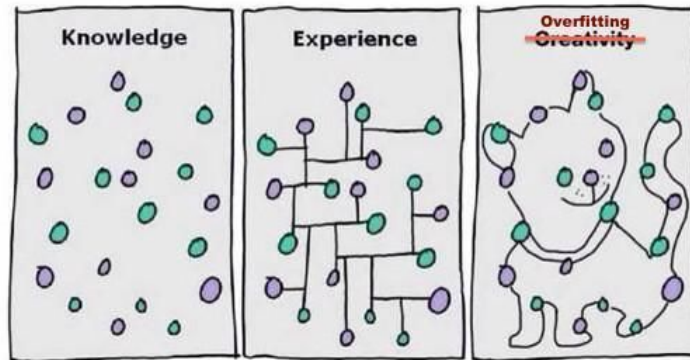


Figura 8: Effetti del sovradattamento

2.2.3.3 Insieme di validazione

Per poter stimare il miglior valore da assegnare ai iperparametri del nostro modello, e' opportuno creare un altro insieme dal nostro dataset di partenza: l'insieme di validazione. Notiamo che non e' possibile utilizzare l'insieme di test per questo scopo perche' incapperemo in sovradattamento.

2.2.3.4 Cross validation

Un'alternativa all'insieme di validazione, specie se abbiamo un dataset piccolo, potrebbe essere la *cross validation*. Inizialmente si suddivide l'intero dataset in n parti (di solito 10). A turno una singola parte svolgerà il ruolo di insieme di validazione mentre tutto il resto sarà l'insieme di addestramento. Per valutare le prestazioni verrà fatta una media dell'accuratezza nelle predizioni in modo da avere un risultato più preciso rispetto al singolo insieme di validazione. Questa tecnica e' anche utile per evitare sovradattamento sui dati.

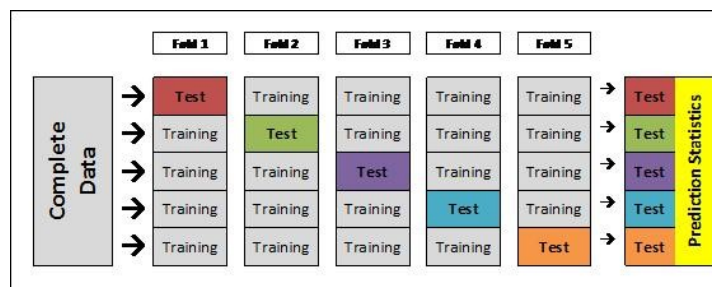


Figura 9: Rappresentazione grafica della Cross Validation

2.2.3.5 Rumore e sovradattamento

Un problema molto comune durante l'addestramento dei classificatori è il rumore: durante la discriminazione degli esempi, ci potrebbe capitare di avere vari esempi con attributi identici ma con etichetta differente. Una probabile causa potrebbe essere la presenza di errori nei dati mentre una possibile soluzione è il voto di maggioranza fra gli esempi rimanenti.

Un'altro problema è il sovradattamento: la costruzione di un classificatore consistente con tutti gli esempi porta ad inaccuratezza nei casi reali d'uso. Una possibile causa è l'utilizzo di attributi irrilevanti nella classificazione. Supponiamo di voler predire l'esito del lancio di un dado e fra gli attributi di avere ora, giorno, mese ed anno; ecco un esempio lampante di sovradattamento. Nei casi reali tuttavia non sono così evidenti gli attributi insignificanti e, per esempio, una tecnica utilizzata negli alberi di decisione è la potatura.

2.2.4 *K Nearest Neighbour*

Uno degli algoritmi più semplici di apprendimento automatico, è il *k-nearest-neighbours*. Il KNN viene definito un algoritmo pigro (*lazy*) perché non ha bisogno di apprendere dall'insieme di addestramento per poter creare un classificatore ma può usare direttamente i dati forniti per classificare i nuovi esempi. Questo vantaggio ha un prezzo da pagare: durante la predizione abbiamo una complessità di tempo proporzionale alla grandezza del dataset.

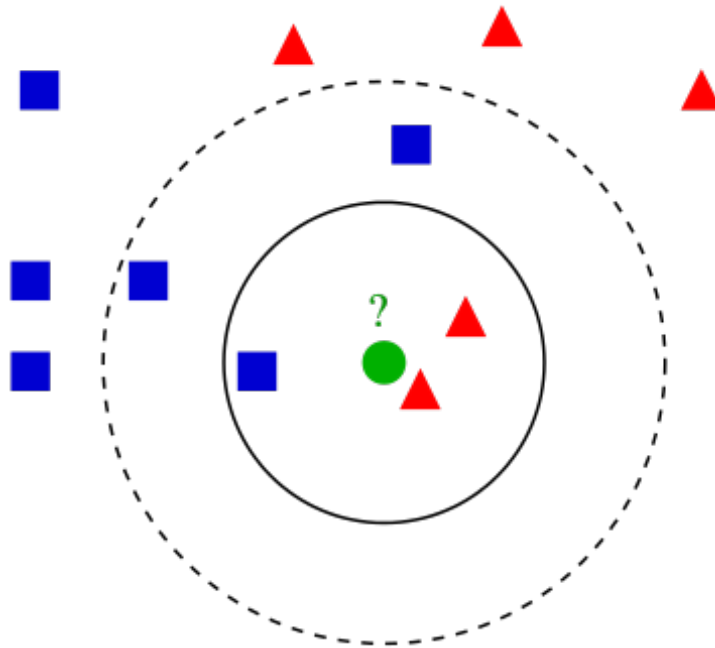


Figura 10: Esempio grafico dell'algoritmo KNN

2.2.4.1 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.

2.2.4.2 Curse of dimensionality

Un problema di cui soffrono vari modelli fra cui knn è la maledizione della dimensionalità ovvero quando abbiamo dataset con elevata dimensionalità come 50 attributi, modelli come knn diventano molto imprecisi. Nel nostro classificatore la motivazione è semplice: con tanti attributi la distanza tra punti incorrelati fra loro potrebbe diventare bassa soprattutto se presente del rumore nei dati oppure attributi insignificanti. Nel nostro problema non soffriamo della maledizione della dimensionalità perché abbiamo solo 3 attributi.

2.2.5 *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:

2.2.5.1 *Teorema di bayes*

Il teorema di bayes ci fornisce una relazione fra probabilita' condizionate molto utile nel calcolo probabilistico ma anche per l'apprendimento automatico. L'equazione e' la seguente:

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

2.2.5.2 *Rete bayesiana*

Una rete bayesiana e' un grafo diretto aciclico i cui nodi sono 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:

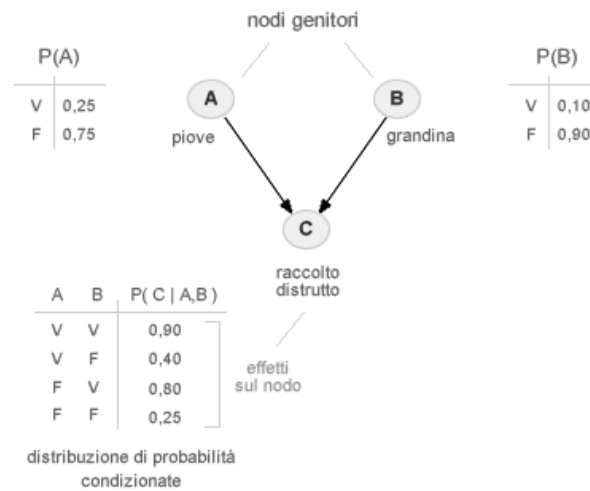


Figura 11: Esempio di rete bayesiana

2.2.5.3 Naïve Bayes

Naïve 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 nonostante cio' fornisce delle buone performance sulla predizione di una classe.

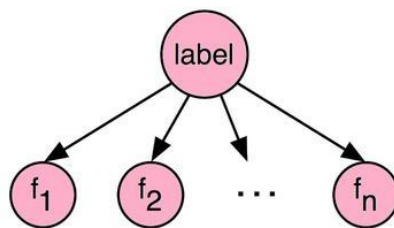


Figura 12: Esempio di rete bayesiana ingenua

2.2.5.4 Modelli generativi e discriminativi

Una distinzione fra classificatori e' possibile farla nel modo in cui viene calcolata l'espressione $P(y_j|x)$: nel caso di modelli discriminativi (ad esempio *knn* e alberi di decisione), l'obiettivo e' discriminare, cioe' suddividere i dati originale per poter assegnare un'etichetta ai nuovi

input mentre nei modelli generativi) l'obiettivo e' di generare una distribuzione congiunta di probabilita' per $P(\mathbf{x}, y_j)$ oppure da $P(\mathbf{x}|y_j)$ e $P(y_j)$ per poter trovare l'etichetta piu' probabile. Fra questi ultimi ricade *Naive Bayes*.

2.2.5.5 Calcolo della probabilita' a priori

Per calcolare la probabilita' a priori esistono varie tecniche: l'equiprobabilita' $\left(\frac{1}{|y|}\right)$, rapporto fra esempi di classe j e il totale degli esempi dall'insieme di addestramento, modelli ad eventi oppure un modello non parametrico dall'insieme di addestramento.

2.2.5.6 Modelli ad eventi

I modelli ad eventi McCallum et al. [1] sono distribuzioni di probabilita' che considerano gli attributi come probabilita' di eventi. Ci sono 3 modelli usati con *Naive Bayes* che 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 e' 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} e' la probabilita' per y_j che x_i sia vero. Un esempio canonico e' 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 probabilita' di trovarlo. Dobbiamo notar bene che Bernoulli a differenza della multinomiale valuta nella produttoria anche la probabilita' che l'evento i non avvenga.

- 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 e' la probabilita' che i occorra. Gli attributi $\mathbf{x} = (x_1, x_2 \dots x_n)$ contano quante volte l'evento i e' avvenuto. La probabilita' condizionata $P(\mathbf{x}|y_i)$ e' stimata come

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

- 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 μ_j e la varianza σ_j^2 . Supponiamo di aver raccolto un insieme di valori v allora la probabilit  sara':

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

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

2.2.5.7 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
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
Rainy	3	2
Sunny	2	3
All	5	9
	=5/14	=9/14
	0.36	0.64

			=4/14	0.29
			=5/14	0.36
			=5/14	0.36

Figura 13: Esempi di partite di tennis giocate in base alle condizioni meteorologiche e tabella di distribuzione delle probabilit 

Ecco una rappresentazione grafica di *Naive Bayes*:

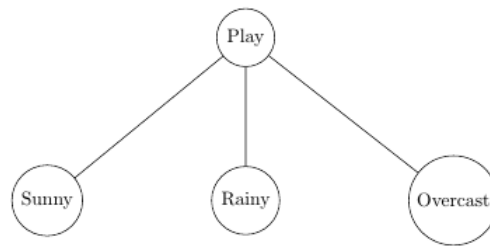


Figura 14: Rappresentazione grafica di *Naive bayes* nell'esempio del tennis

2.2.6 Alberi di decisione

Da un punto di vista strutturale, l'albero di decisione è 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.

2.2.6.1 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.

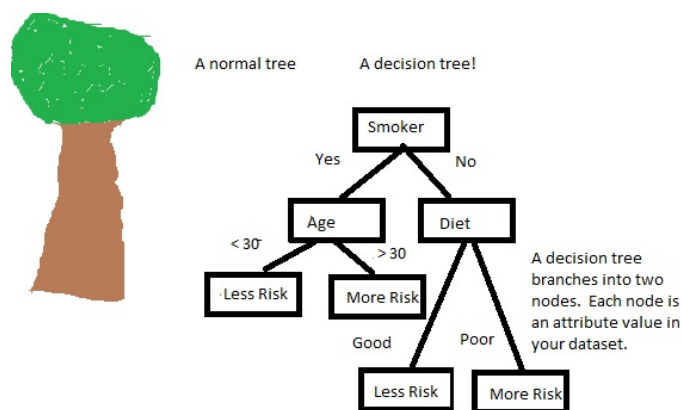


Figura 15: Un albero e un albero di decisione a confronto. Nel secondo abbiamo come attributi *Smoker*, *Age*, *Diet* e come etichetta *Less Risk*, *More Risk* riferito alle malattie cardiache.

2.2.6.2 Costruzione di un albero di decisione

La costruzione dell'albero e' molto semplice: basta prendere gli attributi ed iniziare a suddividere a caso fino ad ottenere nodi con esempi di un solo tipo di etichetta. In questo modo pero' otterremmo un albero non molto utile in tutti i casi diversi dagli esempi. Quindi quale albero scegliere fra tutti quelli possibili? In questo caso ci aiuta un principio filosofico, il rasoio di Occam, che ci consiglia 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, cioe' che discrimina meglio degli altri. Un esempio: immaginiamo di avere 10 esempi con attributi A e B e come etichetta 0, 1. Ipotizziamo che suddividendo tramite l'attributo A avremmo due figli con esempi aventi meta' valore 0 e meta' 1 mentre se suddividiamo secondo B abbiamo nodi figli con esempi esclusivamente 1 oppure 0. In questo caso indubbiamente l'attributo piu' significativo e' B. Esistono vari criteri per misurare l'attributo piu' significativo su cui suddividere il nodo che in termini tecnici vengono chiamate misure d'impurita'. Una misura e' l'indice di Gini il quale corrisponde alla seguente formula:

$$\sum_j p(j|n)(1 - p(j|n))$$

dove $p(j|n)$ e' la percentuale di esempi con classe j nel nodo n

2.2.6.3 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 16: 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:

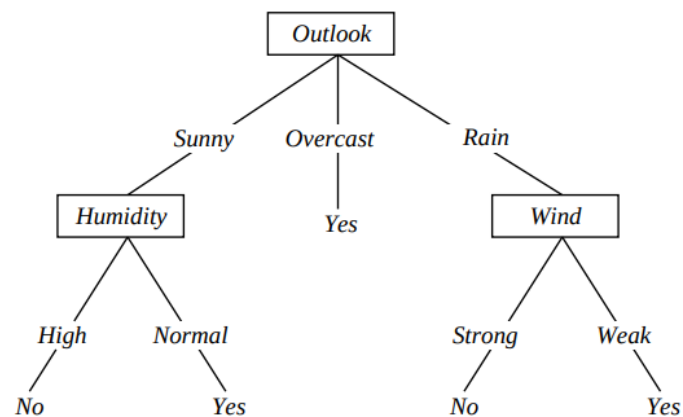


Figura 17: Albero di decisione ricavato dalla tabella precedente

2.3 FASE 3: RICERCA DELLA POSIZIONE

Dopo la costruzione del modello esso viene adoperato per predire la posizione corrente dell'utente che esegue l'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 quelli descritti in precedenza quindi

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

2.3.1 Ricerca di nuovi elementi nei classificatori

Adesso vedremo come avviene la ricerca della posizione (quindi l'etichetta) nel Knn, naive bayes, alberi di decisione.

2.3.1.1 *K nearest neighbours*

Preso un nuovo insieme di attributi $\bar{x} = \bar{x}_1 \bar{x}_2 \dots \bar{x}_n$ saranno presi i primi k elementi che minimizzeranno la distanza euclidea tra \bar{x} e tutti gli altri esempi già conosciuti precedentemente dal dataset d .

$$\forall x \in d \quad \min_k \sqrt{(\bar{x}_1 - x_1)^2 + \dots (\bar{x}_n - x_n)^2}$$

Una volta presi questi k elementi verra' trovata l'etichetta piu' frequente tra di essi e questa sara' l'etichetta di \bar{x}

2.3.1.2 *Naive bayes*

Preso un insieme di attributi x con valori $x_1 x_2 x_3 \dots x_n$ e tutti i valori possibili della variabile dipendente $y = y_1 y_2 y_3 \dots y_m$, per classificare l'insieme applicheremo il teorema di bayes con un'approssimazione molto usata nel campo scientifico chiamata MAP (Massima ipotesi a posteriori)

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

i cui valori della precedente equazione si possono ricavare dalla tabella di distribuzione delle probabilita'.

2.3.1.3 *Alberi di decisione*

Per ottenere la nuova etichetta di un insieme di attributi \bar{x} e' sufficiente scorrere l'albero dalla radice fino ad uno dei nodi che sara' la nostra etichetta. Per scorrere l'albero e' sufficiente scegliere il ramo giusto in base al valore dell'attributo \bar{x} . Per fare un esempio pratico, in un certo nodo interno potremmo andare a sinistra se avessimo il valore di x_3 ed altrimenti a destra.

STRUTTURA DEL SOFTWARE

3.1 LINGUAGGI E *framework*

L'applicazione ha come *target platform* Android perciò il linguaggio usato principalmente è stato Java. Da notare che su Android è presente nella versione 7, perciò non sono disponibili alcune funzionalità di Java 8 come i metodi di default, *stream*, *lambda expression* ecc. Sono state usate delle librerie esterne fra i quali *Gson* che consente la serializzazione/deserializzazione dei dati, *lombok* che fornisce delle annotazioni che abbreviano il codice mantenendo una buona espressività, varie librerie di supporto Android ed una libreria *Apache* che fornisce molte utilità matematiche. Mosso dalla curiosità, ho provato ed alla fine utilizzato nel progetto *Kotlin*, un linguaggio che compila in *bytecode* per la JVM 100% interoperabile con Java 6 (esuccessivi) che, fra le varie cose, fornisce le funzionalità sopracitate mancanti a Java 7.

3.2 APPLICAZIONE

L'applicazione ha lo scopo di catturare le onde magnetiche presenti all'interno di un edificio tramite i sensori presenti sullo smartphone. Per poter ricevere le onde magnetiche dal magnetometro è necessario dare un'istanza *SensorListener* alla libreria Android. Il codice che implementa l'interfaccia *SensorListener* è il seguente:

```
public class MagneticWavesListener implements SensorEventListener {  
  
    private Collector<MagneticWave> collector;  
    private long currentTime;  
    private long recordingRate = -1;  
  
    public MagneticWavesListener(Collector<MagneticWave> collector)
```

```

    {
        this.collector = collector;
        this.currentTime = SystemClock.elapsedRealtime();
    }

    public MagneticWavesListener(Collector<MagneticWave> collector,
        long recordingRate)
    {
        this.collector = collector;
        this.recordingRate = recordingRate;
        this.currentTime = SystemClock.elapsedRealtime();
    }

    @Override
    public void onSensorChanged(SensorEvent event)
    {
        if(recordingRate == -1 || abs(SystemClock.elapsedRealtime() -
            currentTime) > recordingRate )
        {
            float[] waveValues = event.values;
            MagneticWave mw = extractWave(waveValues);
            collector.collect(mw);
            currentTime = SystemClock.elapsedRealtime();
        }
    }

    private MagneticWave extractWave(float[] waveValues)
    {
        Double x = (double) waveValues[0];
        Double y = (double) waveValues[1];
        Double z = (double) waveValues[2];
        return new MagneticWave(x,y,z);
    }
}

```

Sono state omesse alcune parti per non allungare ulteriormente il codice e concentrarci sulla parte essenziale della classe. Il metodo *onSensorChanged* viene chiamato da Android quando rileva un cambiamento nel campo magnetico. La condizione dell'*if* ci permette di registrare le onde magnetiche ad intervalli regolari a patto che *recordingRate* sia diverso da -1 (che rappresenta la disattivazione della registrazione di onde ad intervalli regolari).

Questo codice viene utilizzato sia durante la scansione dell'ambiente, cioè' quando viene costruita per la prima volta la mappa delle distorsioni magnetiche all'interno dell'edificio, sia quando

Ad ogni onda magnetica registrata viene assegnata un'etichetta, ovvero un numero intero rappresentante una zona della mappa che stiamo scansionando. Visto che in un secondo vengono raccolte molte onde magnetiche, quando ne abbiamo un numero abbastanza grande si creano delle *fingerprint* che identificano un singolo punto della zona. Di solito un'etichetta identifica un'area (in genere una stanza) rispetto alla *fingerprint* che rappresenta un singolo punto.

Appena i dati sono stati raccolti, inizia la fase di preprocessamento dei dati. In questo caso non e' stata fatta una normalizzazione/standardizzazione dei valori degli attributi perche', tramite test empirici si e' verificato che, portava ad una accuratezza piu' bassa. Da ogni *fingerprint* sono state estratte nuove caratteristiche di natura statistica, tra le quali: Media, Varianza, Deviazione standard, Mediana, Media troncata, Coefficiente di variazione, Massimo, Minimo, 1°, 5°, 95°, 99° percentile, 1°, 2°, 3° quartile. Dopo non abbiamo una fase di addestramento del modello perche' nel codice su dispositivo Android e' stato usato il *knn* e quindi c'e' direttamente la fase di ricerca della posizione, od in termini piu' tecnici per quanto riguarda l'apprendimento automatico, la predizione dell'etichetta.

Adesso vediamo alcune statistiche sul codice Java scritto: il numero di righe scritto e' pari a 1443 linee di codice in 43 file quindi 43 classi od interfacce mentre con kotlin sono state scritte 742 linee di codice in 16 file.

3.3 UML DEL CODICE

Vediamo adesso un diagramma delle classi del codice *Android* sviluppato. Sono state omesse alcune classi dall'UML perche' futili al fine di comprendere la struttura del programma.

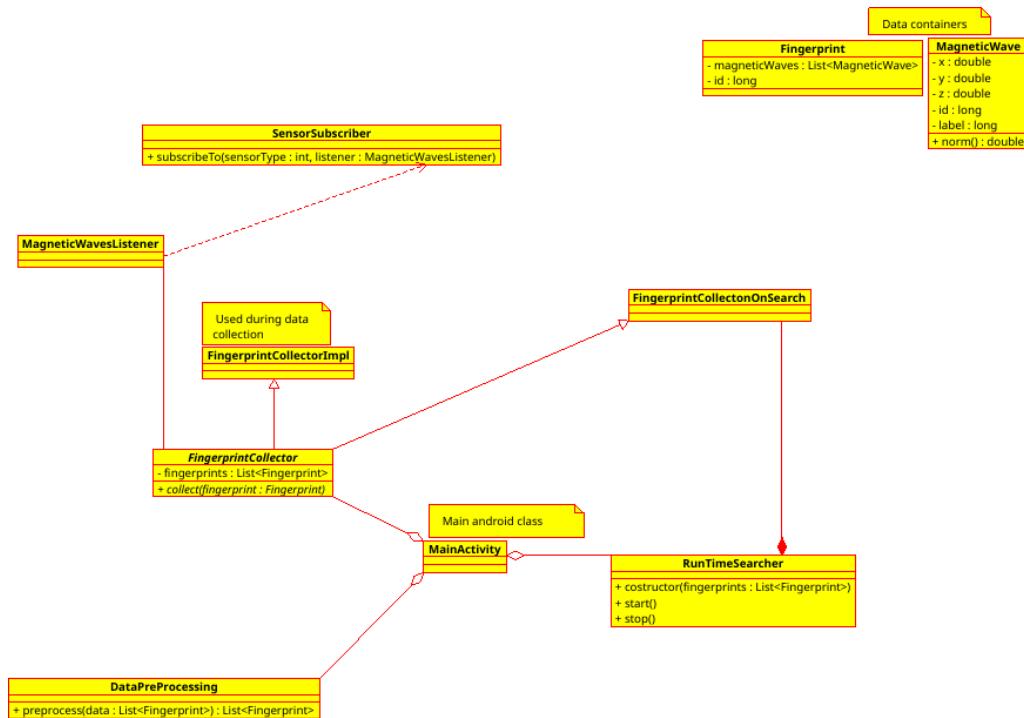


Figura 18: Diagramma delle classi dell'applicazione

3.4 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.

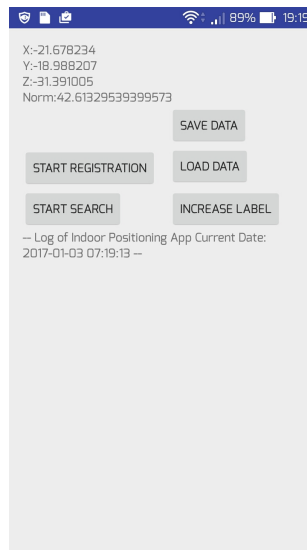


Figura 19: Interfaccia grafica all'avvio dell'applicazione

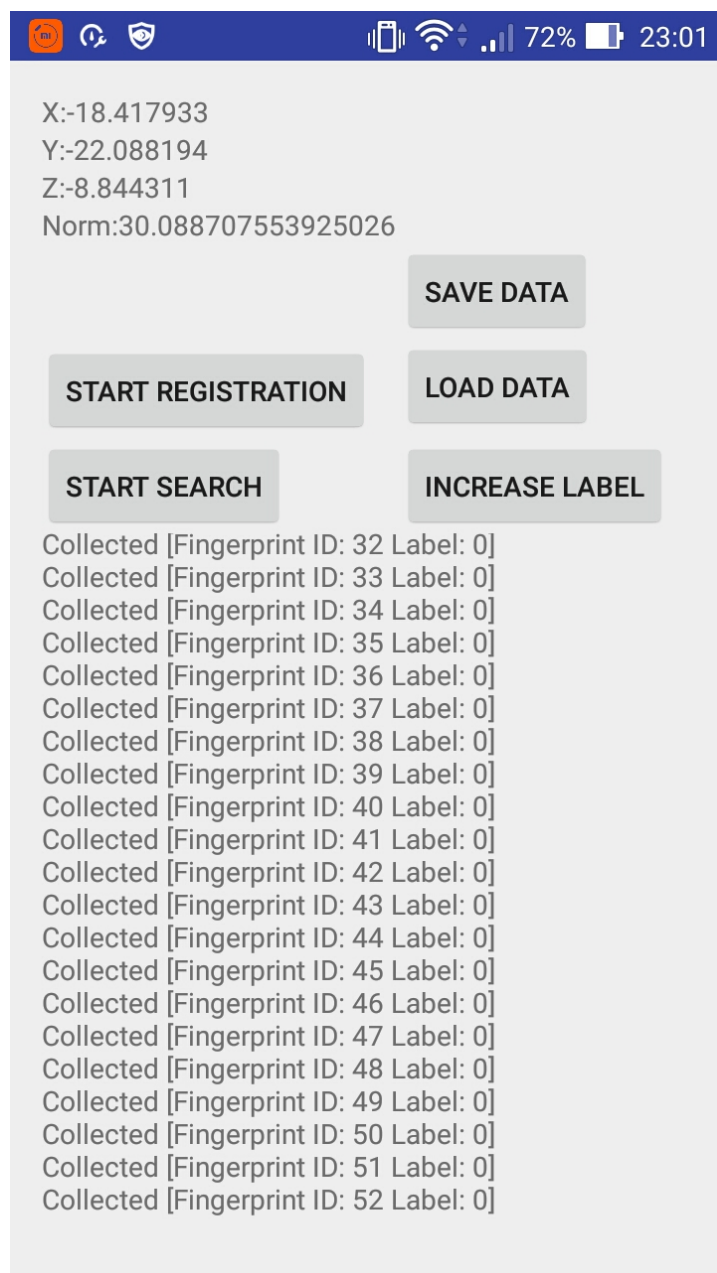


Figura 20: Immagine dell'applicazione durante la raccolta dati

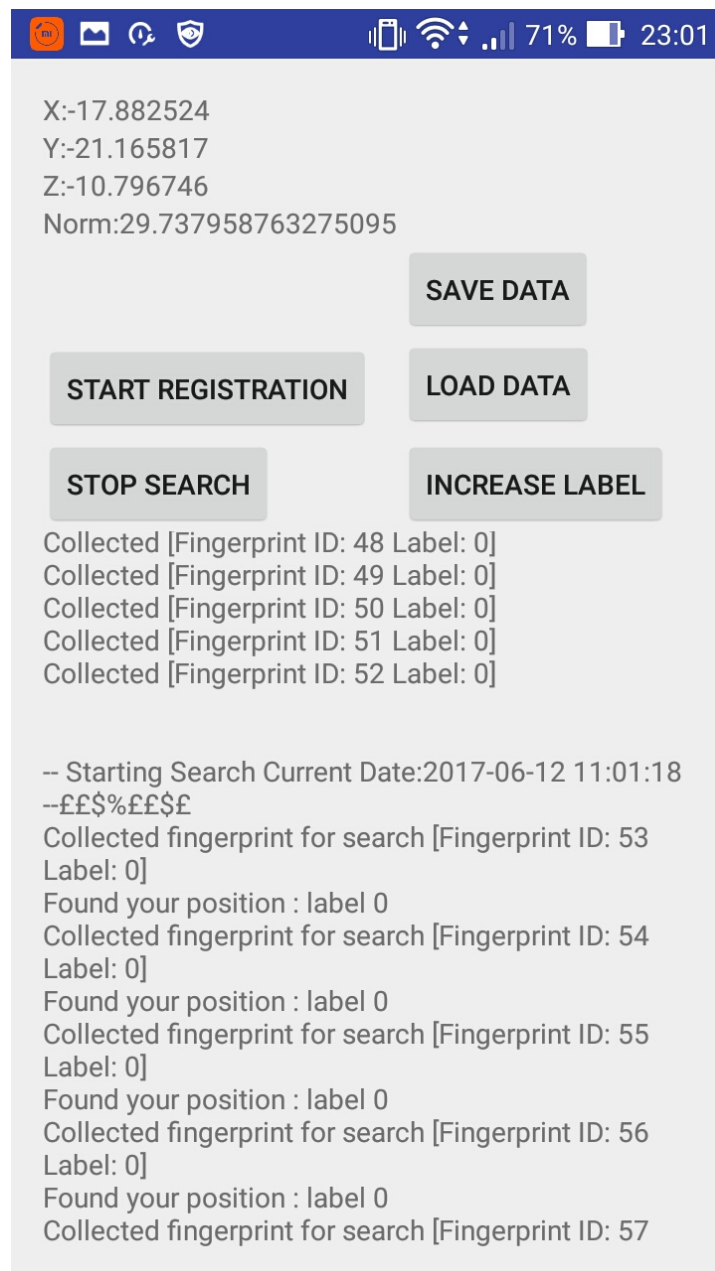


Figura 21: Immagine dell'applicazione durante la ricerca della posizione

3.5 PERSISTENZA DEI DATI

Innanzitutto spieghiamo che cos'è la serializzazione: è una procedura che consente di salvare su supporti di memorizzazione (hard disk, chiavetta usb) dei dati della nostra applicazione mentre la deserializzazione

e' il procedimento inverso. 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.

3.6 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*, *open closed principle*. Riguardo i *design pattern*, sono stati usati frequentemente l'*observer*, il *template* e *factory*.

TEST

4.1 ANALISI DEI DATI

La predizione dei risultati è stata implementata sia nel software *Android* sia sul computer. Nel codice eseguito su dispositivo mobile è stato adottato solamente il KNN per via della facilità d'implementazione 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 è *Python* per via del suo buon supporto all'apprendimento automatico tramite la libreria *sklearn*.

4.2 PIANO 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:

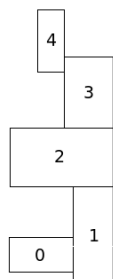


Figura 22: Una piccola raffigurazione delle stanze usate per le prove con sopra scritto la *label* assegnata

Sono stati raccolti circa 18000 campioni di onde magnetiche. La suddivisione fra addestramento e test è 70/30. La misura delle performance è l'errore sui test quindi $1 - \frac{\text{\#predizioni azzecate}}{\text{\#predizioni totali}}$

4.3 CARATTERISTICHE DEI DATI

Dai grafici qui di seguito possiamo notare che c'è sovrapposizione fra i dati, quindi è presente del rumore nei dati.

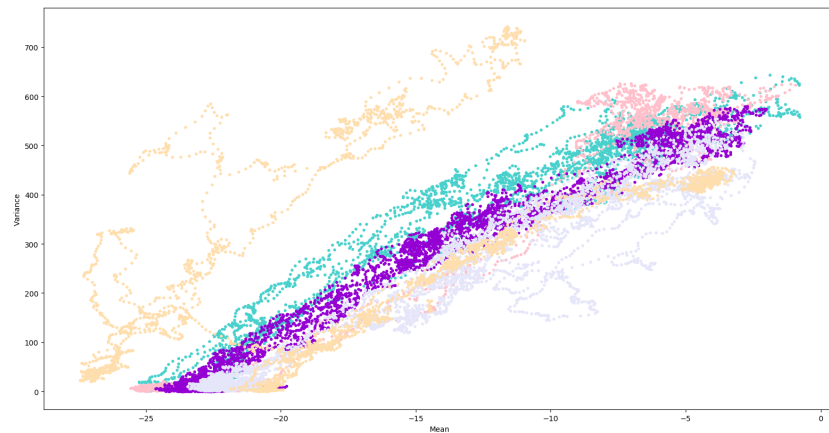


Figura 23: Grafico in 2 dimensioni della media e varianza di tutte le onde magnetiche. I colori dei punti rappresentano le etichette

La causa del rumore sono i sensori che offrono una misurazione non precisa. Per risolvere, almeno in parte questo problema, è stato usato il *filtro di Kalman* durante il preprocessing dei dati per pulire il rumore sui dati.

4.4 CODICE PER L'ANALISI DATI COL *knn*

```
#before each hyper param there is 'estimator__' because OneVsRest
  classifier
h_params_knn = {'estimator__n_neighbors': range(5,61,2)}
knn = KNeighborsClassifier()
#Incapsulation of knn model into OneVsOneClassifier is necessary to
  multiclass prediction
multiclass_knn = OneVsOneClassifier(knn)
grid_src_knn = GridSearchCV(multiclass_knn, h_params_knn, cv=5,
  scoring='accuracy')

grid_src_knn.fit(X_train, y_train)

#Now grid_src_knn.best_params_ contains the n_neighbors to
  maximize the accuracy

best_params_knn = {k[len('estimator__')]:v for k,v in
  grid_src_knn.best_params_.items() }
```

```

knn_val = KNeighborsClassifier(**best_params_knn)
knn = KNeighborsClassifier()
multiclass_knn = OneVsRestClassifier(knn)
multiclass_knn_val = OneVsRestClassifier(knn_val)
multiclass_knn.fit(X_train, y_train)
multiclass_knn_val.fit(X_train, y_train)

predictions_knn = multiclass_knn.predict(X_test)
predictions_knn_val = multiclass_knn_val.predict(X_test)

accuracy_knn = np.sum(predictions_knn == y_test) / len(y_test)
accuracy_knn_val = np.sum(predictions_knn_val == y_test) /
    len(y_test)

```

Nel codice è stato preso come esempio il *knn*, ma si può fare la stessa cosa con tutti gli altri classificatori. E' stata saltata la fase di pre-elaborazione dei dati perché poco importante nel nostro caso studio mentre ci concentriamo di più sulla validazione degli iperparametri e la predizione di risultati per poi terminare con il calcolo dell'accuratezza. Da notare che la validazione è stata fatta tramite *cross validation* tramite la tecnica della *grid search*, che consiste semplicemente nel provare tutti i valori all'interno di *h_params_knn* e selezionare quello con l'accuratezza migliore. Il parametro *cv=5* significa che il dataset d'addestramento è stato suddiviso in 5 parti. Dopo la validazione è stato preso il *k* migliore e sono stati confrontati i risultati fra un *knn* con i migliori iperparametri e quelli di *default*. I risultati di tale prova sono più avanti.

4.5 CLASSIFICATORI A CONFRONTO

Qui di seguito vediamo i risultati ottenuti da ciascun classificatore con un istogramma:

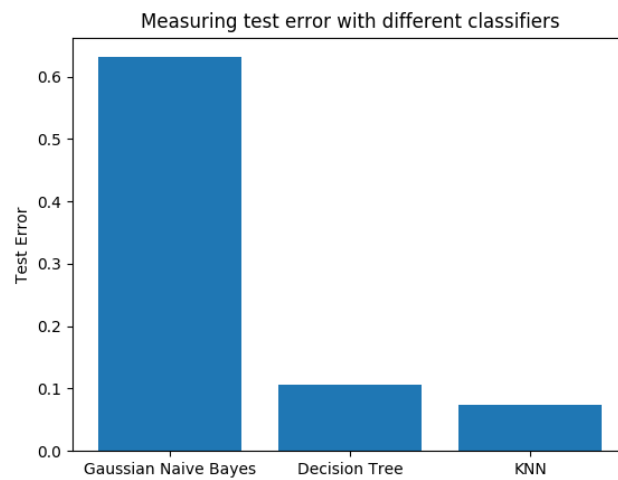


Figura 24: Percentuale d'errore nei test dei classificatori

come possiamo notare *Gaussian Naive Bayes* è totalmente inadatto alla classificazione di onde magnetiche mentre gli alberi di decisione e *K Nearest Neighbours* si comportano molto bene, con risultati leggermente migliori in quest'ultimo. A questo punto qualcuno potrebbe però pensare che gli ultimi 2 modelli si sono sovradattati agli esempi (*overfitting*) ed avrebbe ragione, perché applicando la *cross validation* abbiamo risultati diversi da quelli precedenti.

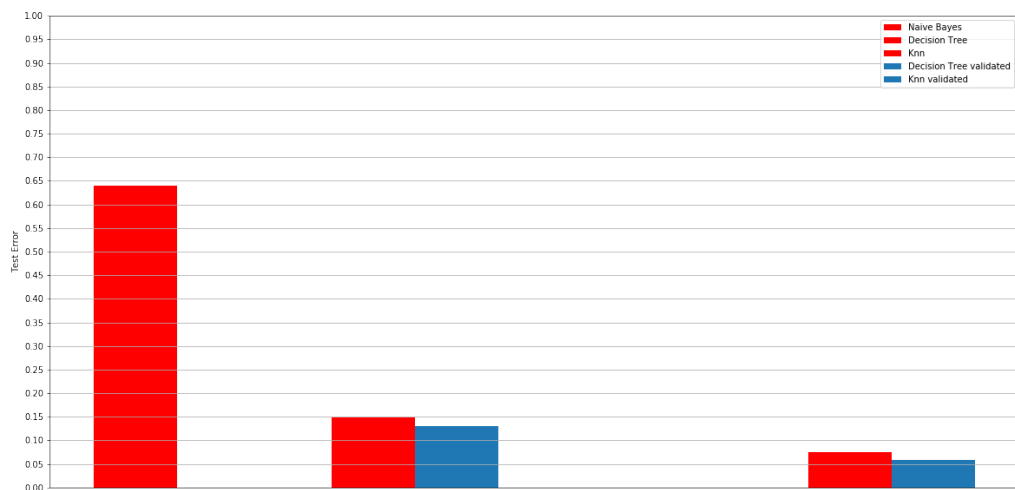


Figura 25: Percentuale d'errore nei test dei classificatori con la *cross validation*

Adesso visualizziamo gli errori per etichetta:

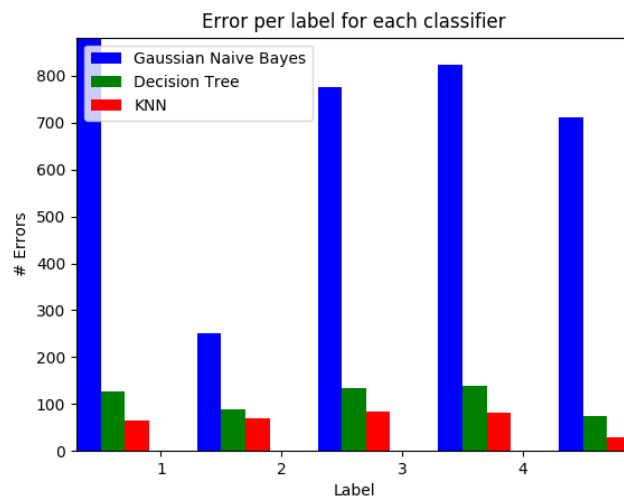


Figura 26: Numero di errori nella predizione per etichetta

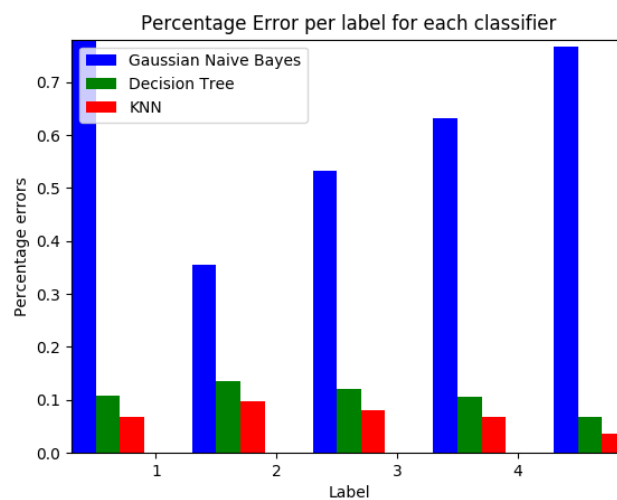


Figura 27: Percentuale di errore nella predizione per etichetta. Le barre blu rappresentano i risultati dei classificatori *cross-validati* mentre i rossi sono quelli senza.

4.6 ANALISI DEL KNN AL VARIARE DELL'IPER PARAMETRO K

Un grafico interessante da analizzare è l'accuratezza del *Knn* al variare di *K*. L'accuratezza è intesa come la percentuale di predizioni azzeccate sull'insieme di test, inoltre i risultati sono stati *cross validati*.

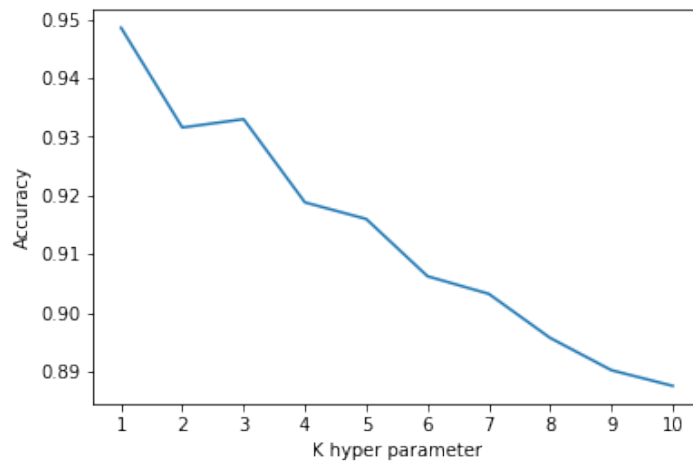


Figura 28: Accuratezza del knn al variare di K

In base a questo grafico potremmo supporre di prendere *Knn* con $k = 1$ ma sarebbe un grave errore che porterebbe a pessimi risultati nel mondo reale. Come mai? Innanzitutto spieghiamo ciò che succede col 1-nn: per ogni nuovo punto la sua etichetta sarà quella del vicino più prossimo. I pessimi risultati nel mondo reale sono causati dall'*overfitting*, cioè che il nostro modello ha imparato troppo bene (a memoria) i nostri dati oppure, detta in termini statistici, vuol dire che il modello si è adattato troppo al campione ed ha perso la "generalità" necessaria per effettuare predizioni corrette sulla popolazione. Invece se impostiamo un k troppo alto avremo lo stesso risultato l'*underfitting* ma per il motivo contrario, cioè il nostro modello ha imparato troppo poco dai dati. Per notare veramente gli effetti dell'*overfitting*, adesso vedremo su un piano cartesiano le sue conseguenze prendendo come esempio una classificazione binaria su due attributi dove ogni punto sarà colorato di blu e rosso in base alla sua etichetta. Nel grafico è stata tracciata una linea nera di separazione fra le due aree rosse e blu per indicare che i punti in quella area rossa saranno classificati come rossi dal knn e viceversa per i blu.

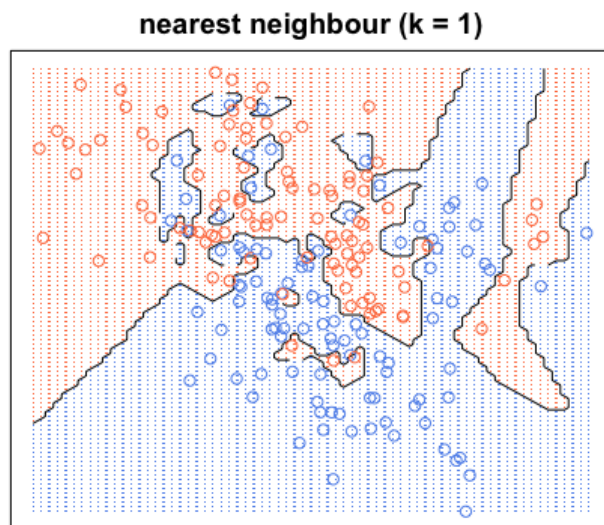


Figura 29: Classificazione binaria con il 1-nn

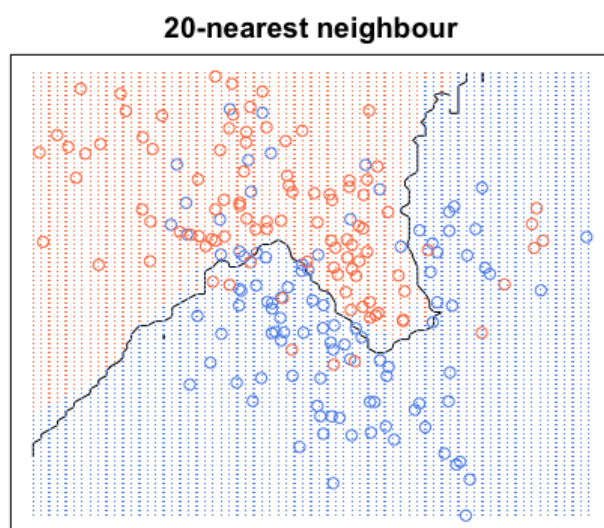


Figura 30: Classificazione binaria con il 20-nn

Come vediamo dal grafico, l'*overfitting* ($k=1$) porta a linee di separazione tra rossi e blu molto più frastagliate mentre un normale adattamento ($k=20$) porta a una linea più smussata e ci possiamo immaginare che l'*underfitting* porti ad una linea troppo smussata, che nel grafico precedente potrebbe essere una semplice bisettrice.

Per evitare problemi di *overfitting* ed *underfitting* ci sono alcuni strumenti per capirlo anche se, il miglior strumento e' l'esperienza sul campo con il modello utilizzato la quale ti fa capire in anticipo se stai sbagliando o no perche' una delle cause principali sono gli iperparametri impostati male.

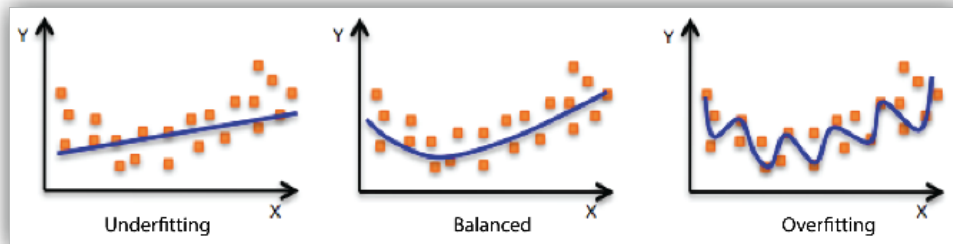


Figura 31: *Underfitting* ed *overfitting* nella regressione

Uno strumento per capire se c'è *overfitting/underfitting* sono le curve di validazione. Esse mostrano al variare della complessita' del modello l'errore sia sull'insieme di validazione che sull'insieme di addestramento.

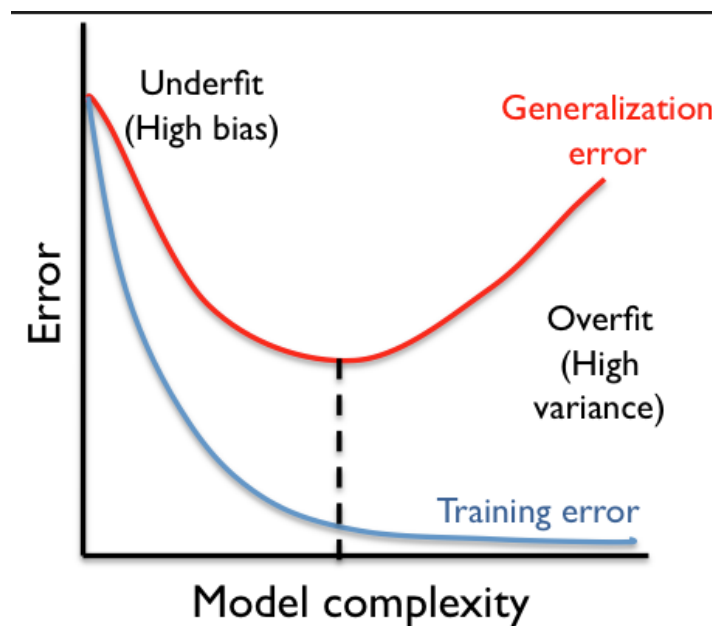


Figura 32: Curve di validazione

Come si vede dal grafico dove c'è *overfitting* si ha un'alta varianza. Cosa vuol dire? Che stiamo modellando anche il rumore generato dai

dati e cio' porta a cattive prestazioni quando avremo nuove etichette da predire. Quindi si verifica *overfitting* quando c'e' un alto errore di predizione sull'insieme di test e basso nell'insieme di addestramento. Si ha *underfitting* quando c'e' *High bias*, cioe' quando si ha un alto errore sulle predizioni delle etichette sia sull'insieme di test che di addestramento. Il punto ideale dove ci dovremo posizionare noi e' nel mezzo dove l'errore sull'insieme di test e di addestramento sono entrambi bassi.

4.7 ANALISI DEL RUMORE DURANTE LA CATTURA DEI DATI

Per analizzare la correttezza dei nostri dati l'approccio usato è stato quello di posizionarsi fermo in un punto e verificare tramite grafico che tutti i punti catturati dal magnetometro fossero concentrati in una piccola area del grafico. Purtroppo non è stato così e ce lo dimostrano i grafici dove è stato messo nell'asse X il valore X catturato e sull'asse Y il valore Y.

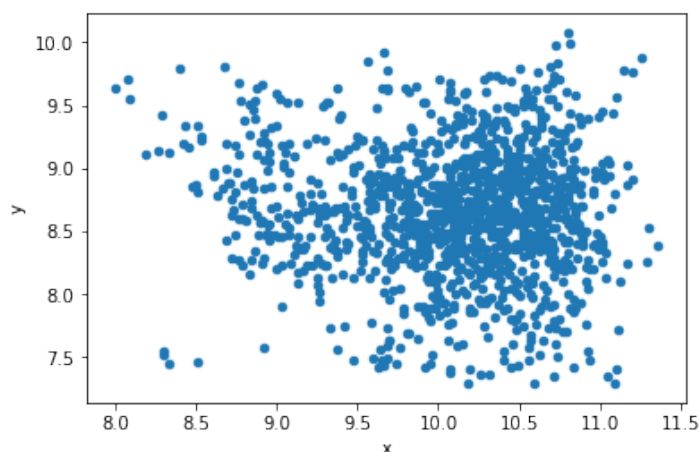


Figura 33: Valori del magnetometro rispetto agli assi x ed y stando fermo

4.8 UN RIMEDIO INGENUO AL RUMORE

Un approccio ingenuo per risolvere il problema al rumore potrebbe essere quello di prendere meno dati per etichetta. Il seguente grafico pero' ci mostra che cio' non è vero

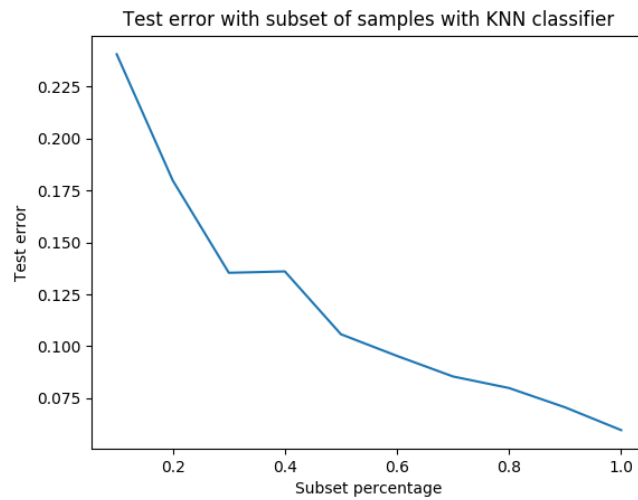


Figura 34: Percentuale d'errore al variare della grandezza dell'insieme di addestramento con KNN

Notiamo che questo grafico dimostra che all'aumentare del dataset otteniamo risultati piu' accurati poiche' ci stiamo avvicinando alla media e varianza della popolazione.

MIGLIORAMENTI E CONCLUSIONI

5.1 POSSIBILI MIGLIORAMENTI PER LA RACCOLTA DI DATI

- L'architettura *client-server* e' necessaria per avere un'applicazione che riesca a gestire con fluidita' la ricerca all'interno di grandi edifici. In riferimento alla memoria del telefono poiche' nella maggior parte dei casi possiede un quantitativo di *RAM* e memoria permanente molto limitati, ma anche rispetto al processore perche' la ricerca della posizione, assumendo di usare il *KNN*, ha bisogno di confrontare gli attributi delle nuove onde magnetiche con tutti gli altri e va da se che piu' e' grande il dataset, piu' potenza di calcolo ci vorra' per ottenere una risposta in tempi umani.
- Durante la raccolta sarebbe opportuno applicare il *filtro di Kalman* per ridurre il rumore causato dall'imprecisione dei sensori. E' stata usata in fase di test ma non nel codice che gira su dispositivi *mobile*
- Per migliorare la precisione sarebbe opportuno appoggiarsi anche ad altri sensori presenti sullo *smartphone*. Prendiamo come esempio il Wi-Fi, se il dispositivo e' connesso alla rete dell'edificio potrebbe sfruttare la potenza di segnale per avere una precisione maggiore; sfruttare l'accelerometro per stimare la velocita' del telefono e quindi standardizzare tutte le rilevazioni effettuate col magnetometro. Questa tecnica viene chiamata *sensor fusion*[\[2\]](#)

5.2 CONCLUSIONI

In conclusione abbiamo una base di applicazione Android che, tramite l'uso di tecniche di *machine learning* abbastanza semplici nella loro complessita' rispetto ad altre, ma che offrono comunque un buon grado di accuratezza anche se bisogna stare attenti a non cadere nella trappola

dell'*overfitting*. C'e' ancora da migliorare, come abbiamo visto nella sezione precedente, in tutti gli altri campi dell'applicazione, dall'interfaccia grafica alle risorse dedicate all'applicazione fino alle tecniche di raccolta dati.

BIBLIOGRAFIA

- [1] Andrew McCallum, Kamal Nigam, et al. A comparison of event models for naive bayes text classification. In *AAAI-98 workshop on learning for text categorization*, volume 752, pages 41–48. Madison, WI, 1998. (Cited on page 21.)
- [2] Ubejd Shala and Angel Rodriguez. Indoor positioning using sensor-fusion in android devices, 2011. (Cited on page 49.)