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

1

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.

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 gergali 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 applicazione.

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

2

FONDAMENTALI

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

1. Scansione dell'ambiente
2. Ricerca della posizione

2.1 SCANSIONE DELL'AMBIENTE

La scansione dell'ambiente consiste in un'analisi dell'ambiente per mezzo del magnetometro con lo scopo di registrare tutte le distorsioni del campo magnetico presenti in modo da permettere la ricerca della posizione.

2.2 RACCOGLIMENTO 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 verra' definito durante la fase di costruzione del modello.

2.3 ONDE MAGNETICHE

Le onde magnetiche sono un vettore di 3 elementi, quindi in 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.

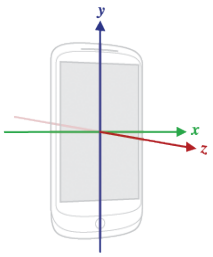

Figura 1: Assi x, y, z centrati sul cellulare

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.

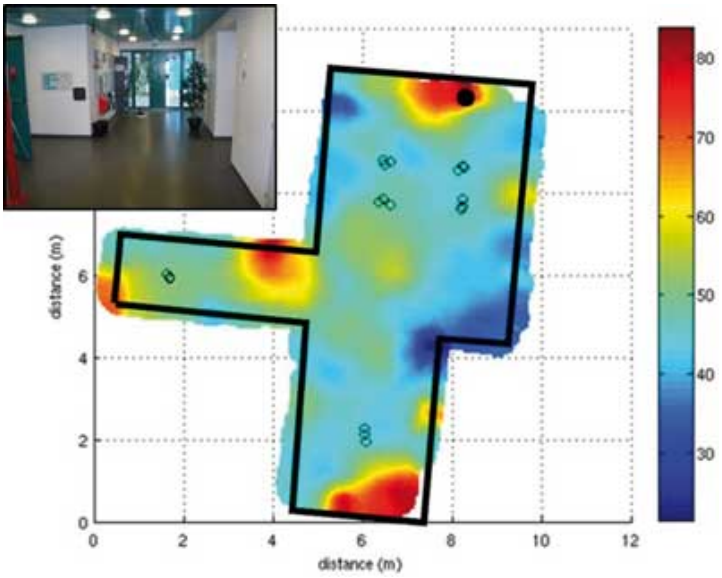

Figura 2: Mappa del campo magnetico all'universita' di Oulu: Discus Entrance hall

2.4 ELABORAZIONE DEI DATI

2.4.1 *Estrazione del magnitudo*

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

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

2.4.2 *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.

2.4.3 *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

2.5 FASE 2: ELABORAZIONE DEI DATI

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.

2.6 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 Ã¨ associato alle scienze cognitive.
- Pensare razionalmente: il processo che porta il sistema intelligente a risolvere un problema Ã¨ un procedimento formale che si rifÃ alla logica.
- Agire razionalmente: il processo che porta il sistema

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.

2.7 TIPI DI APPRENDIMENTO

Esistono 3 tipi di apprendimento:

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

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

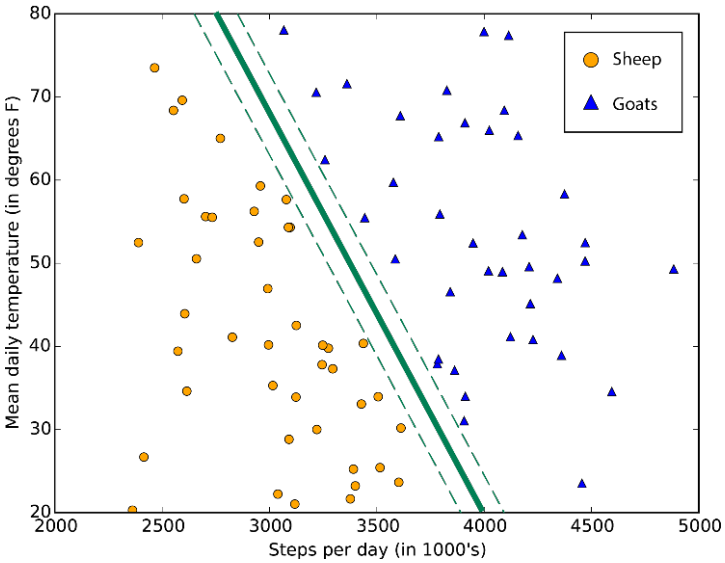

Figura 3: Un esempio grafico di insieme d'addestramento ed una sua classificazione

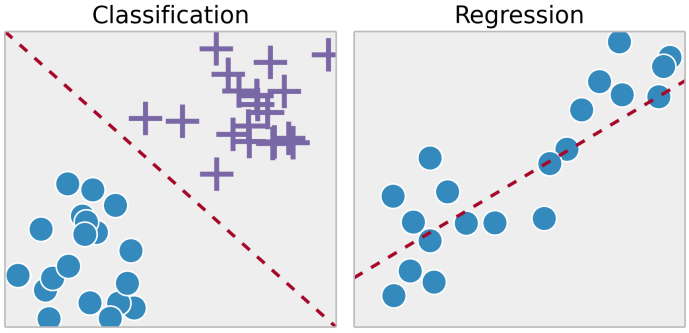

Figura 4: Classificazione e regressione a confronto

- Apprendimento non supervisionato: Gli esempi non contengono una variabile dipendente ma solo un insieme di attributi $\mathbf{x}$. L'obbietto e' quello di inferire pattern nascosti dai dati non etichettati. Un'importante applicazione e' il *clustering*: raggruppare i dati in base ad una similarita' fra di essi.

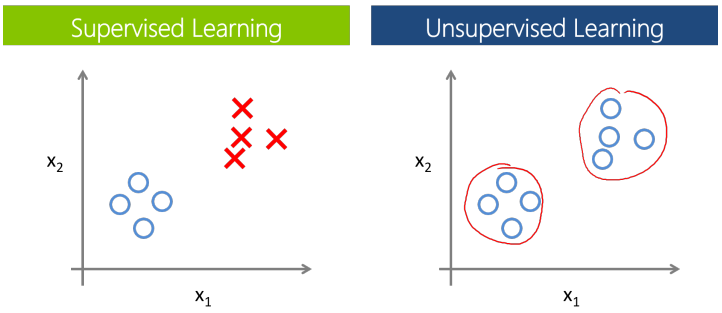

Figura 5: Apprendimento supervisiono e non supervisionato a confronto

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

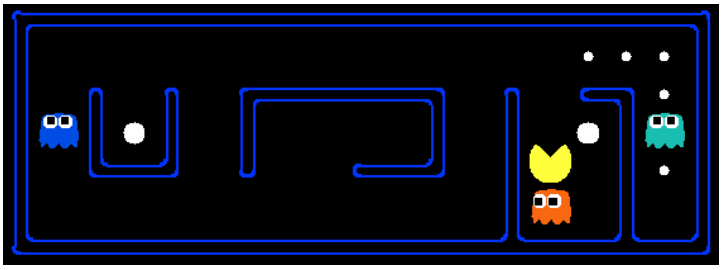

Figura 6: L'apprendimento con rinforzo e' molto adatto ai giochi, come per esempio pacman

2.8 ALCUNE NOZIONI SULL'APPRENDIMENTO AUTOMATICO

2.8.1 *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 ed effetuera' le predizioni sull'insieme di test, su cui verranno misurate le prestazioni in base al rapporto fra predizioni corrette e totali.

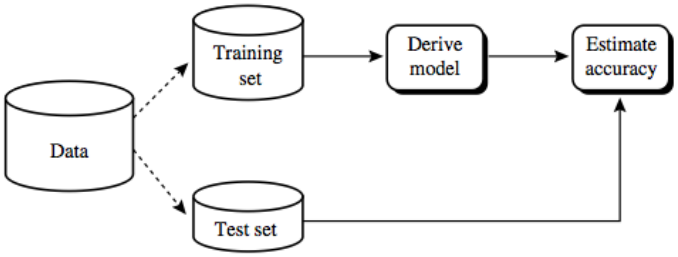

Figura 7: Un diagramma di flusso che mostra le operazioni di verifica delle prestazioni

2.8.2 *Rumore e sovradattamento*

Un problema molto comune durante l'addestramento dei classificatori e' il rumore: durante la discriminazione degli esempi, ci ritroviamo ad un punto in cui gli attributi rimanenti sono identici ma con etichetta differente. Una probabile causa potrebbe essere la presenza di errori nei dati mentre una possibile soluzione consisterebbe nel voto di maggioranza. Un'altro problema e' il sovradattamento: la costruzione di un classificatore consistente con tutti gli esempi a causa dell'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 sovradattemento. Nei casi reali tuttavia non sono cosi' evidenti gli attributi insignificanti e, per esempio, una tecnica utilizzata negli alberi di decisione e' la potatura.

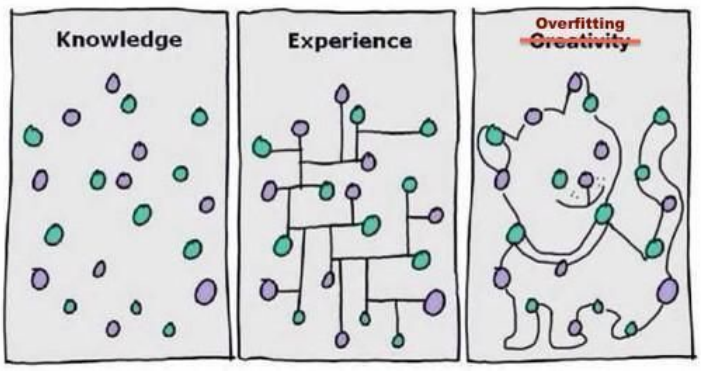

Figura 8: Effetti del sovradattamento

2.8.3 *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.

2.8.4 *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 svolgera' il ruolo di insieme di validazione mentre tutto il resto sara' l'insieme di addestramento. Per valutare le prestazioni verra' fatta una media dell'accuratezza nelle predizioni in modo da avere un risultato piu' preciso rispetto al singolo insieme di validazione. Questa tecnica e' anche utile per evitare sovradattamento sui dati.

Figura 9: Rappresentazione grafica della Cross Validation

2.9 K NEAREST NEIGHTBOUR

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

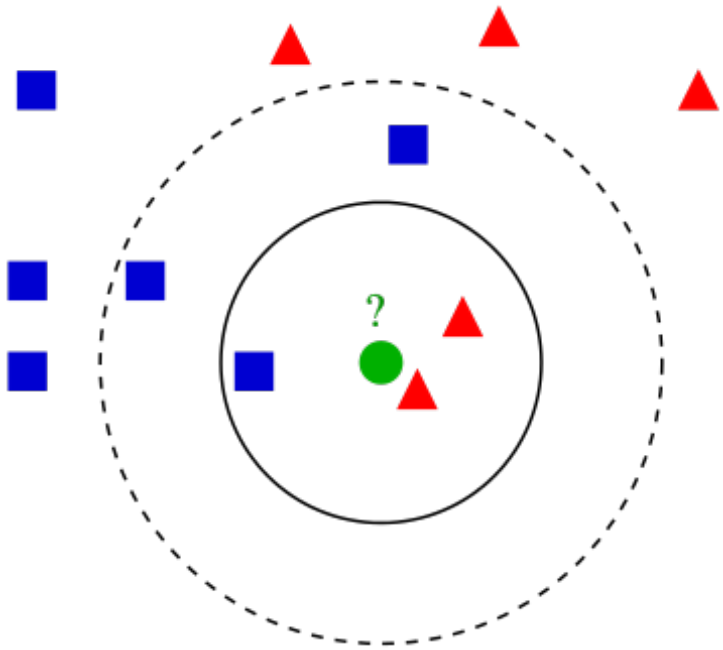

Figura 10: Esempio grafico dell'algoritmo KNN

2.9.1 *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

2.10 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:

2.10.1 *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:

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

2.10.2 *Rete bayesiana*

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

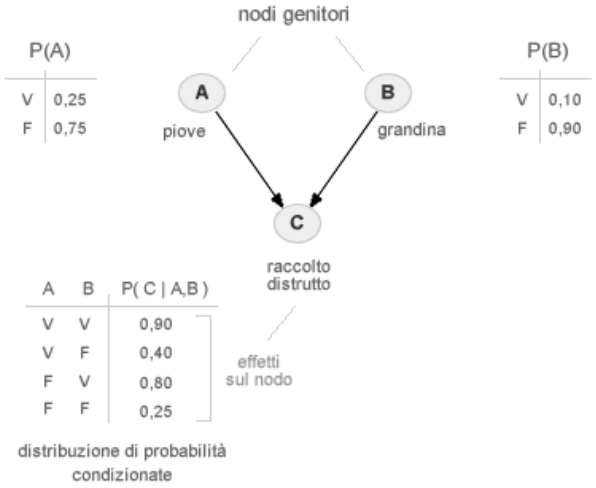

Figura 11: Esempio di rete bayesiana

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

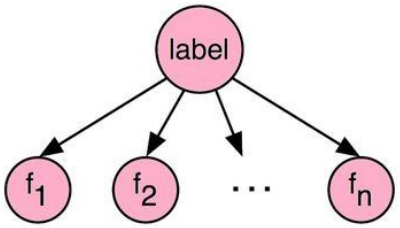

Figura 12: Esempio di rete bayesiana ingenua

2.10.4 *Predizione di risultati*

Preso un insieme di attributi $\mathbf{x}$ con valori $x_1x_2x_3...x_n$ e tutti i valori possibili della variabile dipendente $\mathbf{y} = y_1y_2y_3...y_m$, per classificare l'insieme applicheremo il teorema di bayes con un'approssimazione molto usata nel campo scientifico chiamata MAP (Massima ipotesi a posteriori)

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

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

2.10.5 *Modelli generativi e discriminativi*

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

input mentre nei modelli generativi) l'obbiettivo 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*.

2.10.6 *Calcolo della probabilita' a priori*

Per calcolare la probabilita' a priori esistono varie tecniche: l'equiprobabilita' $\left(\frac{1}{|\mathbf{y}|}\right)$, 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.

2.10.7 *Modelli ad eventi*

I modelli ad eventi 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 $x_i \in \mathbf{x} = (x_1, x_2, ..., x_n)$ vale 1 se l'evento i e' avvenuto, 0 altrimenti. Nel caso bernoulliano possiamo valutare $P(\mathbf{x}|y_j)$ come

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

  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 $(p_1...p_n)$ dove p_i e' la probabilita' che i occorra. Gli attributi $\mathbf{x} = (x_1, x_2...x_n)$ contano quante volte l'evento i e' avvenuto. La probabilita' condizionata $P(\mathbf{x}|y_j)$ 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 probabilita' 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.

2.10.8 *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 13: Esempi di partite di tennis giocate in base alle condizioni meteorologiche e tabella di distribuzione delle probabilita'

Ecco una rappresentazione grafica di *Naive Bayes*:

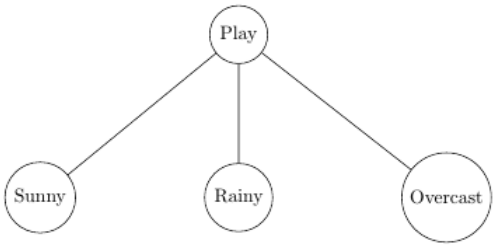

Figura 14: Rappresentazione grafica di *Naive bayes* nell'esempio del tennis

2.11 ALBERI DI DECISIONE

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.

2.11.1 *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.

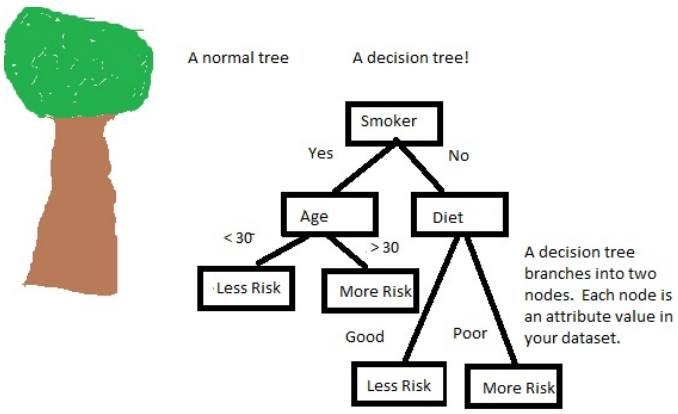

Figura 15: Un albero e un albero di decisione a confronto. Nel secondo abbiamo come attributi *Smoker, Age, Diet* e come etichetta *Less Risk, More Risk* riferito alle malattie cardiache.

2.11.2 *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 (sovradattamento). 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.

2.11.3 *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:

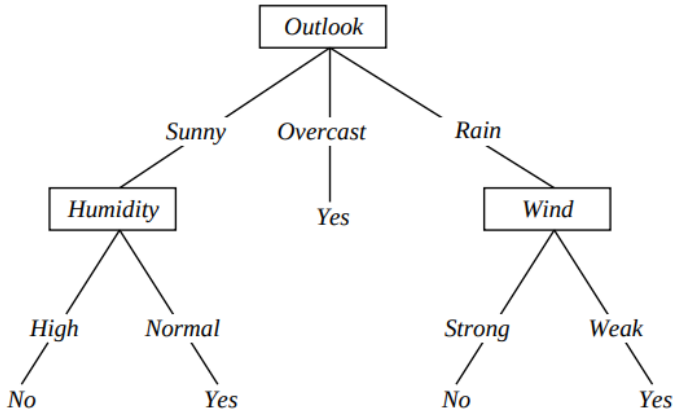

Figura 17: Albero di decisione ricavato dalla tabella precedente

2.12 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 Neightbour*
2. Bayes ingenuo
3. Alberi di decisione

3

STRUTTURA DEL SOFTWARE

3.1 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' 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 espressivita', varie librerie di supporto Android ed una libreria *Apache* che fornisce molte utilita' matematiche. Mosso dalla curiosita', 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 funzionalita' sopracitate mancanti a Java 7.

3.2 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.

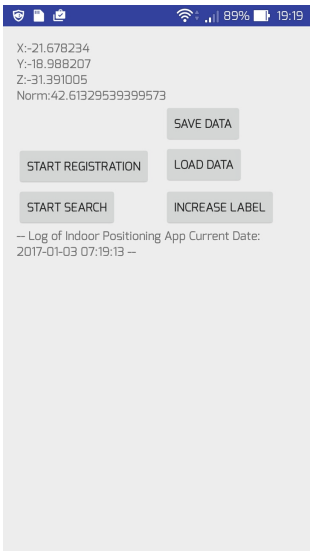

Figura 18: Interfaccia grafica all'avvio dell'applicazione

3.3 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.

3.4 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*.

3.5 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 e non e' 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 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 tramite la libreria *sklearn*.

3.6 CODICE USATO PER L'ANALISI

```
def train_model(dataset: list, model, cross_validation=True):
    """

    :param dataset: List of tuples (label, features).
    :param model: a model instance
    :param cross_validation: flag for cross validation
    :return: the model and the test error predictions. In case of
        cross validation the mean of all the errors
    """
    y, X = zip(*dataset)
    X_train, X_test, y_train, y_test = train_test_split(X, y,
        test_size=0.3)
    model = model.fit(X_train, y_train)
    test = list(zip(X_test, y_test))
    return model, test_model(model, test, cross_validation)


add = lambda a, b: a + b


def test_model(model: {'predict'}, test: list, cross_validation):

    if cross_validation:
        X_test, y_test = zip(*test)
        scores = cross_val_score(model, X_test, y_test, cv=10,
            scoring='accuracy')
```

```
        return 1 - scores.mean()
    else:
        predictions = [(model.predict([X_i])[0], y_i) for X_i, y_i in
            test]
        wrong_predictions = reduce(add, [1 for prediction, real_label
            in predictions if prediction != real_label], 0)
        return wrong_predictions / float(len(test))
```

Visto che ogni classificatore condivide i metodi per l'addestramento e la predizione con tutti gli altri, ho creato una funzione generica che dato in input il dataset, un'instanza del classificatore ed un flag per la *cross validation*, ritorni la percentuale d'errore sull'insieme di test.
Un semplice uso della funzione precedente e':

```
decision_tree, test_error = train_model(dataset, DecisionTreeClassifier())
print("Test error with decision tree = %d"% test_error)
```

4

TEST

4.1 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:

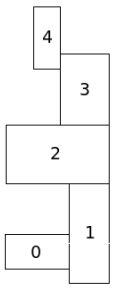

Figura 19: Una piccola raffigurazione delle stanze usate per le prove con sopra scritto la *label* assegnata

Sono stati raccolti circa 18000 campioni di onde magnetiche. La suddivisione fra addestramento e test e' 70/30.

4.2 CARATTERISTICHE

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

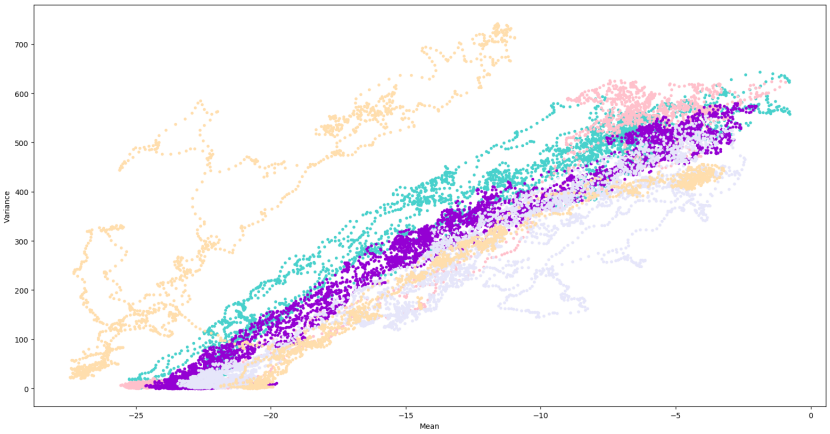

Figura 20: Grafico in 2 dimensioni della media e varianza di tutte le onde magnetiche. I colori dei punti rappresentano le etichette

La causa del rumore sono i sensori che offrono una misurazione non precisa. Un possibile riparo al problema e' il *filtro di Kalman*

4.3 CLASSIFICATORI A CONFRONTO

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

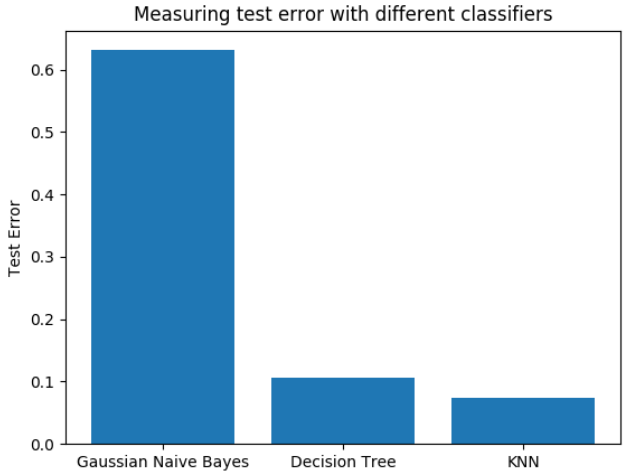

Figura 21: Percentuale d'errore nei test dei classificatori

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

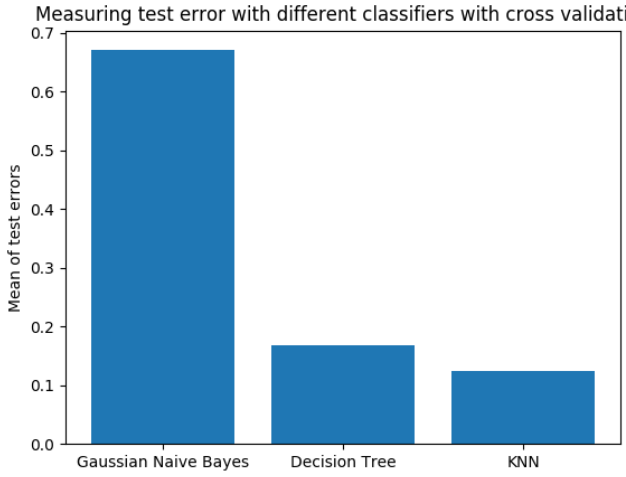

Figura 22: Percentuale d'errore nei test dei classificatori con la *cross validation*

Adesso visualizziamo gli errori per etichetta:

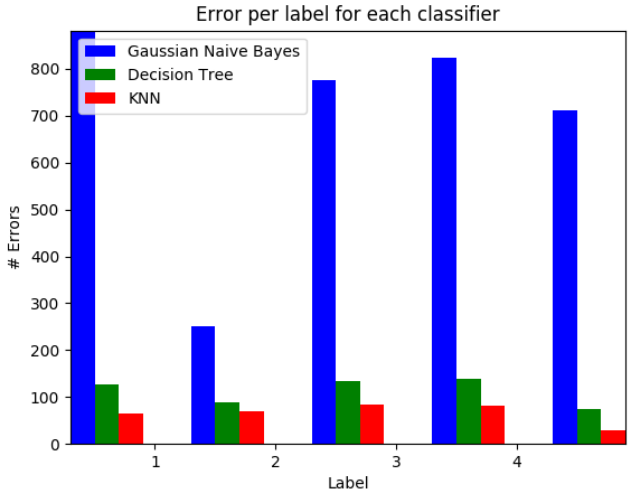

Figura 23: Numero di errori nella predizione per etichetta

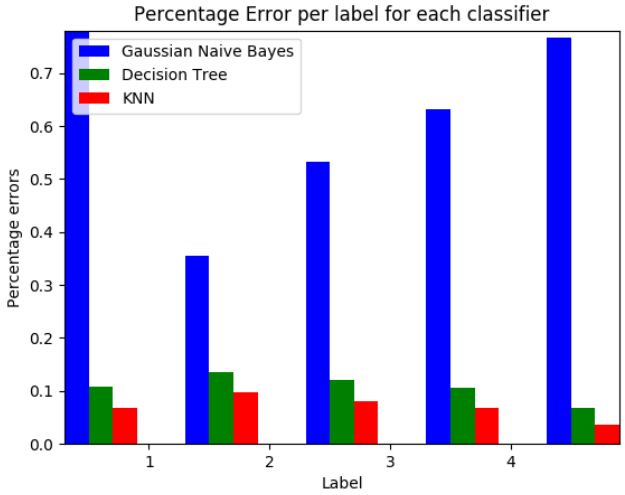

Figura 24: Percentuale di errore nella predizione per etichetta

4.4 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 e' vero

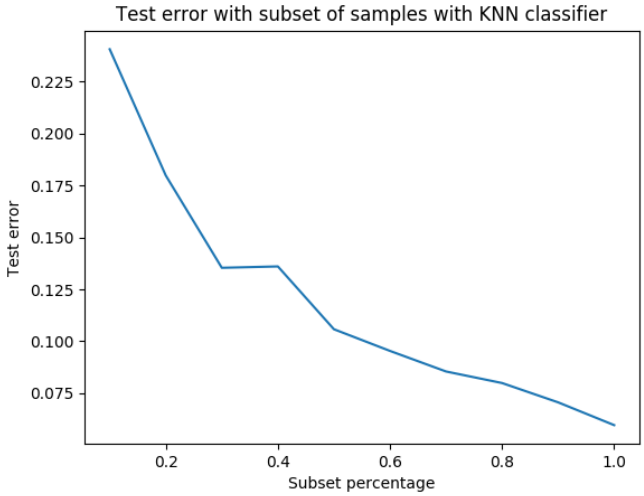

Figura 25: Percentuale d'errore al variare della grandezza dell'insieme di addestramento con KNN

5

MIGLIORAMENTI

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 rifermento 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 i propri attributi 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.
- 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*(Link a qualche paper)