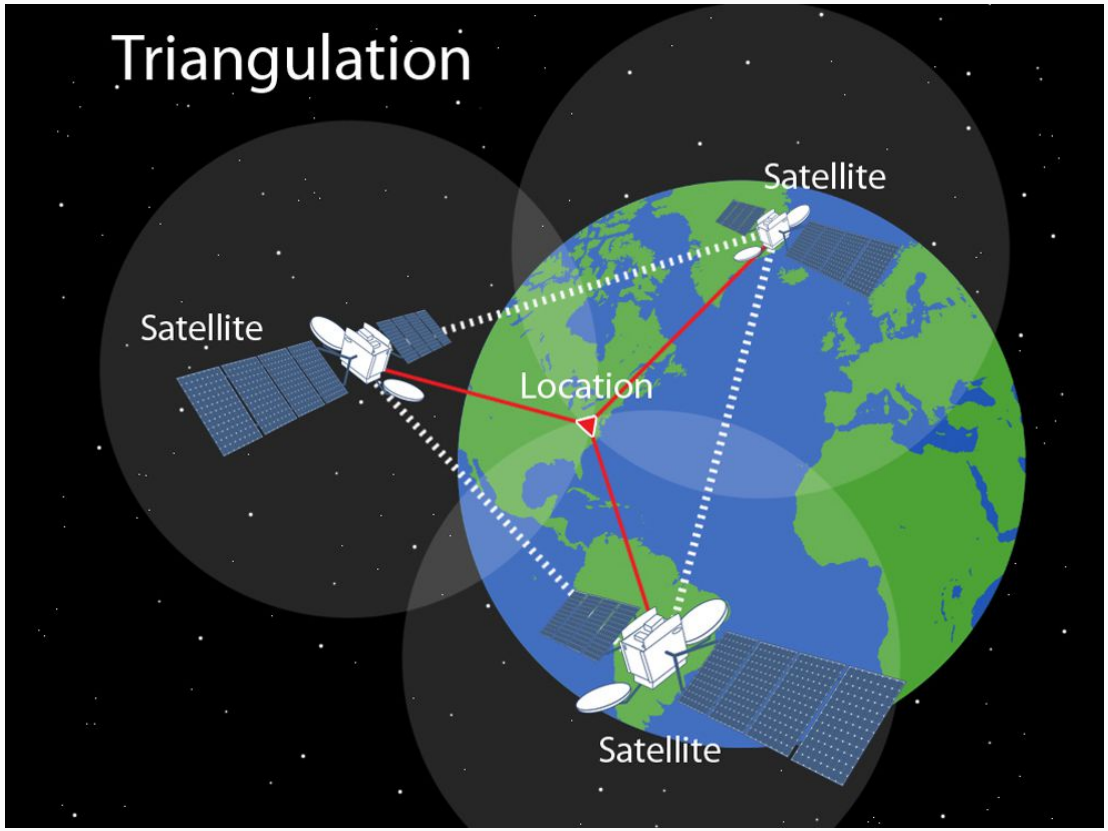
Sviluppo di un'applicazione Android per il posizionamento indoor

Michele De Vita

14 luglio 2017

Cos'è il posizionamento indoor?

- Una delle tecniche più usate per il posizionamento *outdoor*, la triangolazione GPS, non si può usare all'interno degli edifici a causa delle distorsioni provocate da muri e tetti.



- Perciò si ricorre a tecniche alternative per il posizionamento *outdoor*

Cos'è il posizionamento indoor?

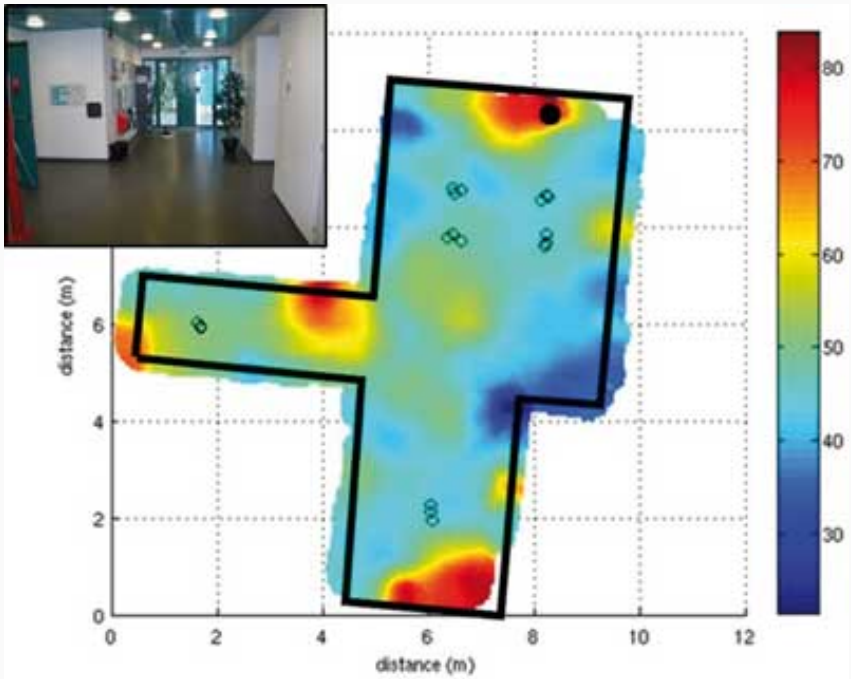
- Molte aziende stanno investendo in questo settore ed anche in ambito accademico escono pubblicazioni scientifiche.
- Adesso vediamo alcune tecniche di posizionamento *indoor*:
  - *Wi-Fi*
  - *BLE Beacons*
- Durante il tirocinio mi sono occupato del posizionamento *indoor* sfruttando la distorsione del campo magnetico terrestre.

Alcuni casi d'uso del posizionamento indoor

- Nel museo: l'applicazione ufficiale apre un *pop-up* in automatico che ci mostra informazioni aggiuntive sull'opera che stiamo visualizzando
- In aeroporto : ci potrebbe indicare la strada da percorrere per raggiungere il *gate*

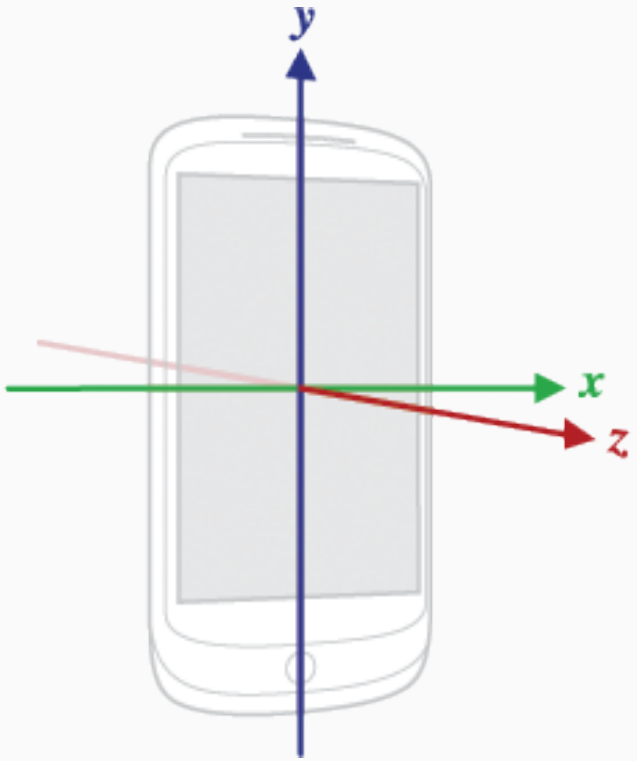
Onde magnetiche: cosa sono?

- È un vettore composto da un verso, direzione ed intensità
- L'unita di misura è il μT (micro Tesla)
- I valori del vettore variano in base ai materiali presenti intorno
- Durante la localizzazione *indoor* sfruttiamo le onde generate dalla Terra e la distorsione generata dagli oggetti all'interno dell'edificio



Smartphone ed onde magnetiche

- Su ogni *smartphone* è presente il magnetometro, un sensore capace di catturare il campo magnetico.
- Quando si registrano le onde magnetiche su *Android*, viene restituita una sequenza di 3 elementi rappresentante l'intensità del campo magnetico lungo i 3 assi mostrati qua sotto.



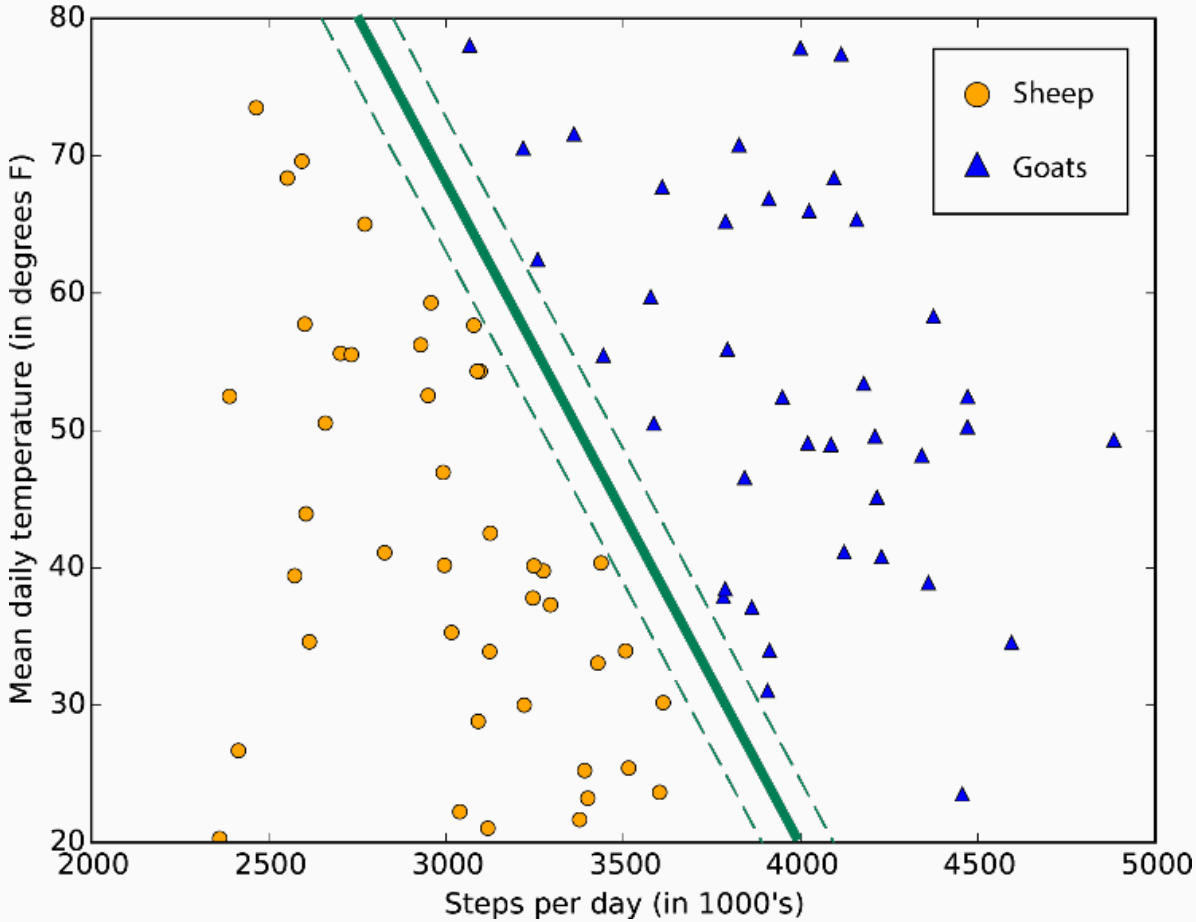
Elaborazione e classificazione dei dati

- Una volta raccolti i dati, vanno elaborati per rendere più efficace la classificazione.
- Ci sono molte tecniche che possiamo usare, più efficaci o meno in base al tipo di dato trattato.
- Le tecniche più usate possiamo riassumerle in queste 3 macro categorie:
  - Scalatura dei valori
  - Estrazione di attributi
  - Selezione di attributi

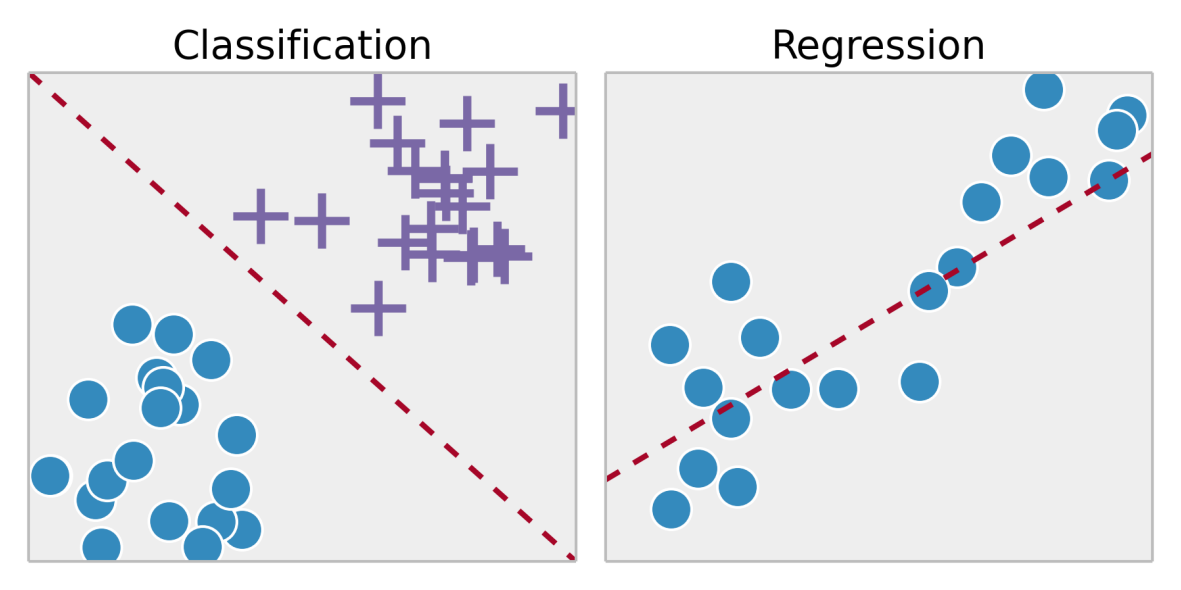
Apprendimento automatico

- Branca dell'IA che conferisce al calcolatore l'abilità di apprendere senza essere esplicitamente programmati.
- Tipi di apprendimento:
  - Supervisionato
  - Non supervisionato
  - Rinforzo
- Durante il tirocinio si è utilizzato l'apprendimento supervisionato con output un valore discreto (classificazione)

Un esempio di classificazione



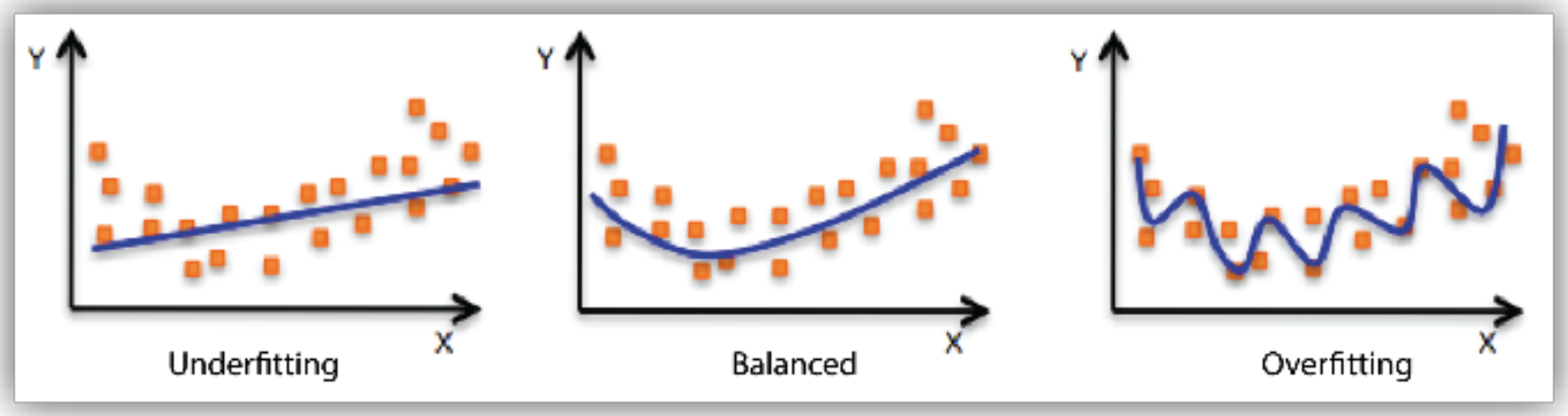
Classificazione e regressione a confronto



Alcune nozioni sull'apprendimento automatico

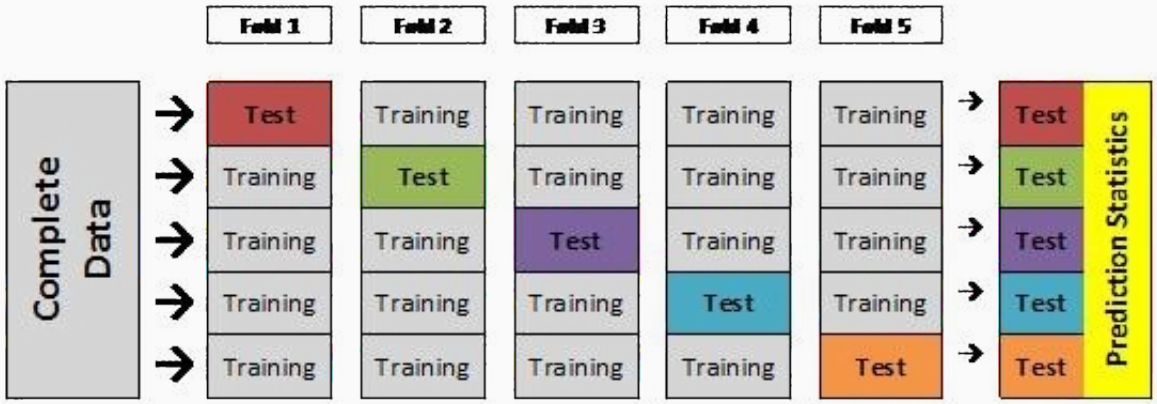
- Rumore: dati contenenti valori anomali od errori.
- *Overfitting*: il modello si è adattato troppo ai nostri dati.
- *Underfitting*: problema opposto al precedente: il modello ha imparato troppo poco.
- Parametri: valori del modello stabiliti internamente.
- Iper-parametri: valori stabiliti dal suo creatore.
- Insieme di addestramento: sottoinsieme del *dataset* di partenza su cui il modello apprende.
- Insieme di validazione: sottoinsieme usato per cercare i migliori iper-parametri.
- Insieme di test: sottoinsieme usato per verificare l'efficacia del modello.

Overfitting ed underfitting



Cross-validation

- Invece di avere un insieme di addestramento e validazione, si divide l'insieme d'addestramento in n parti. A turno ciascuna parte svolge il ruolo di insieme di validazione mentre il resto d'addestramento.
- Molto utile per dataset piccoli



Valutazione delle prestazioni e metriche

- Per valutare la potenza del classificatore si effettuano predizioni sull'insieme di test e si utilizza una metrica per quantificarla.
- Durante il tirocinio l'obbiettivo è stato quello di massimizzare l'accuratezza (predizioni corrette su grandezza dell'insieme di test)
- Un'alta accuratezza non è sempre sinonimo di un buon modello.

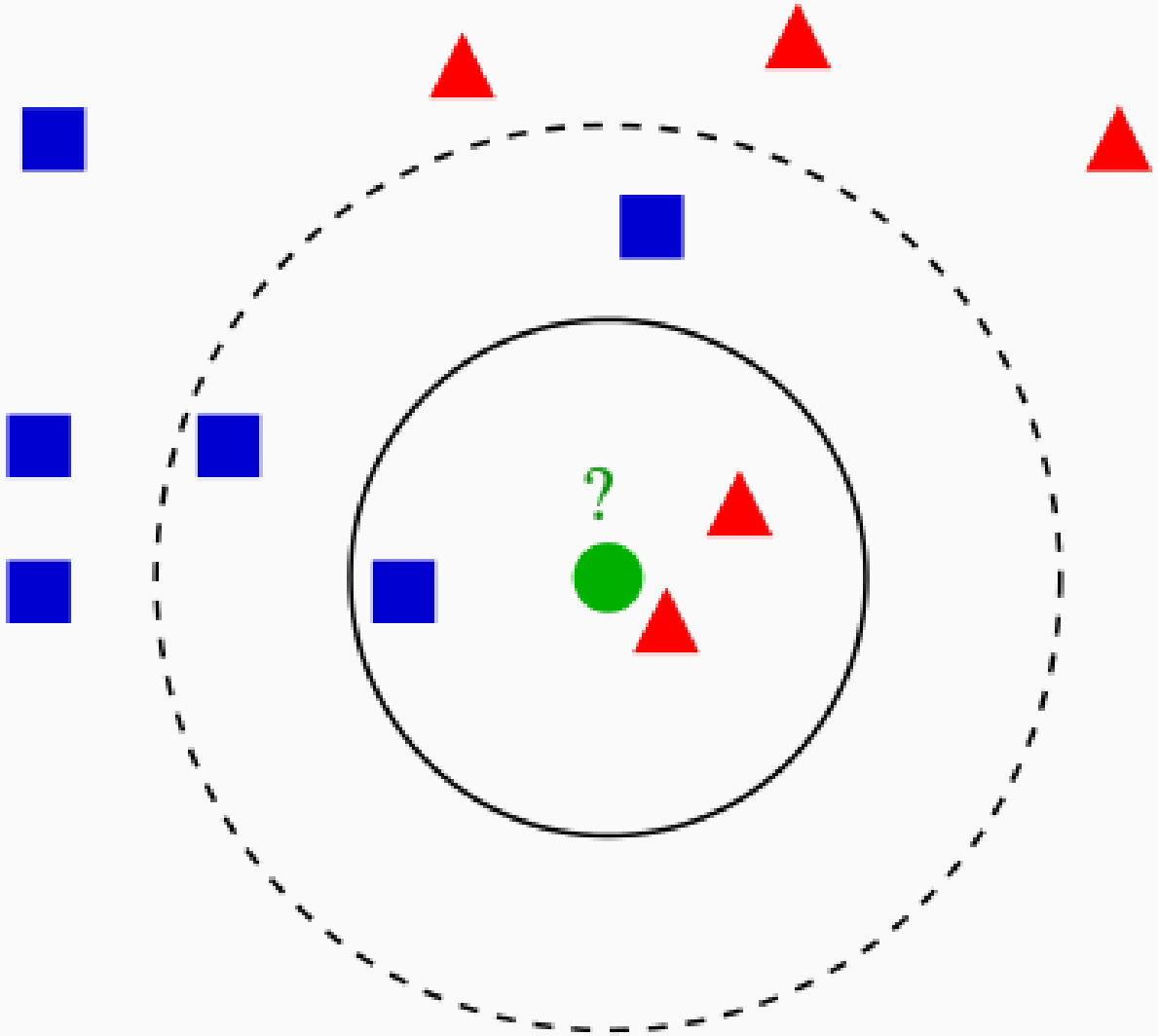
Classificatori usati per la predizione della posizione

- Knn
- Alberi di decisioni

Knn

- Non ha una fase di addestramento
- La predizione si svolge nel seguente modo:
  1. Trova i k elementi che minimizzano la distanza dalla nuova istanza.
  2. La classe più frequente tra i vicini diventa quella assegnata ai nuovi attributi.
  3. L'unico iper-parametro è k.
- La distanza si può definire in vari modi: useremo la distanza euclidea

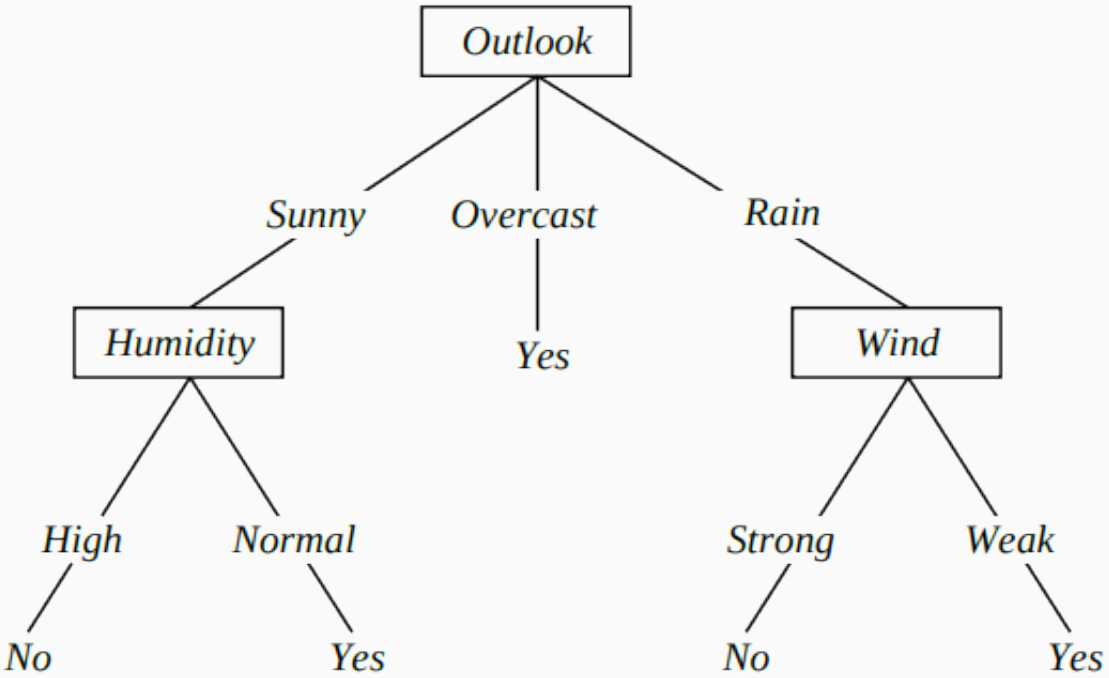
Knn



Alberi di decisione

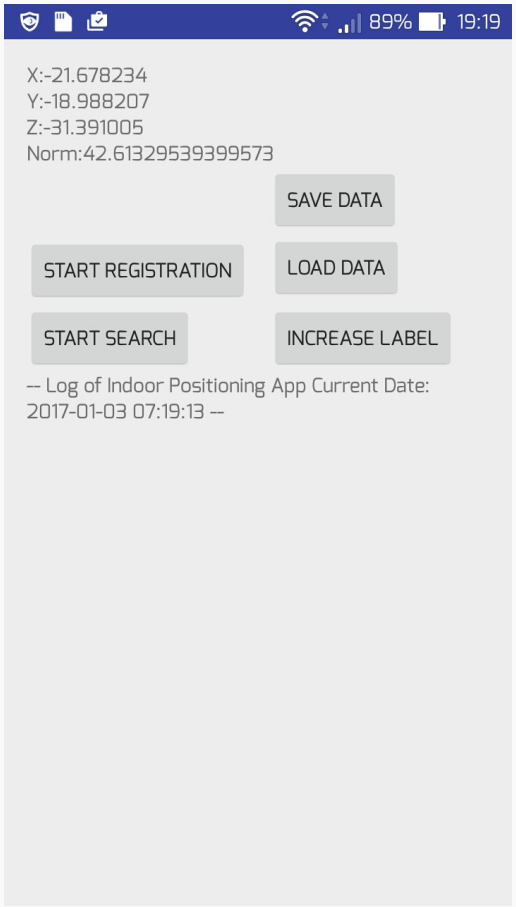
- Costruzione dell'albero (Addestramento): dagli esempi vengono costruiti i nodi interni in base all'attributo che li suddivide meglio (minimizza il decremento d'impurità). La costruzione di un albero si ferma in base a vari criteri, impostabili tramite iper-parametri.
- Predizione: scorrere l'albero in base ai valori presenti nell'istanza da classificare, per poi arrivare in una foglia, che diventerà la nuova classe.

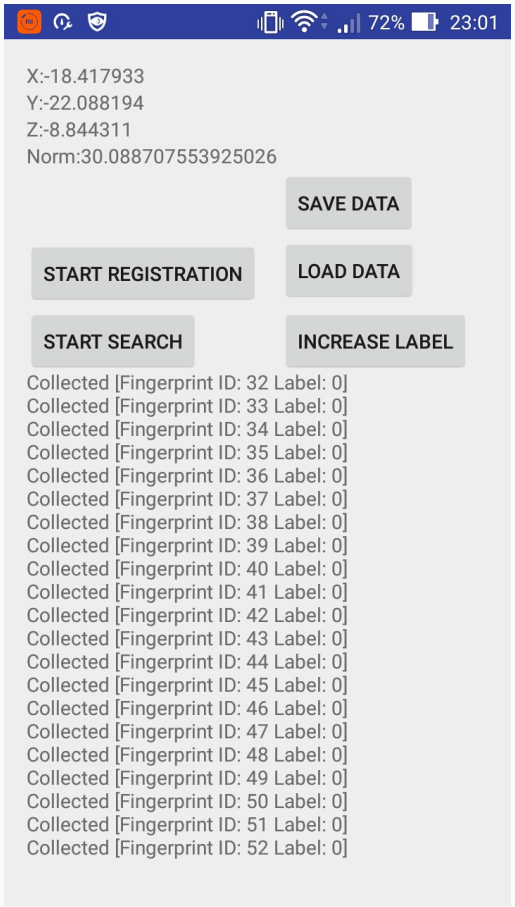
Alberi di decisione

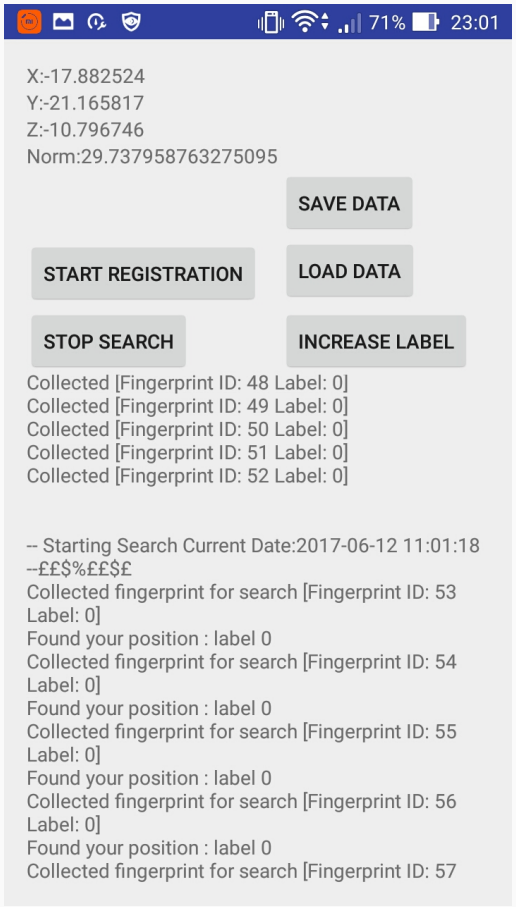


Applicazione

- Sviluppata su *Android* API 24 (Lollipop) mantenendo una retrocompatabilità con le versioni precedenti



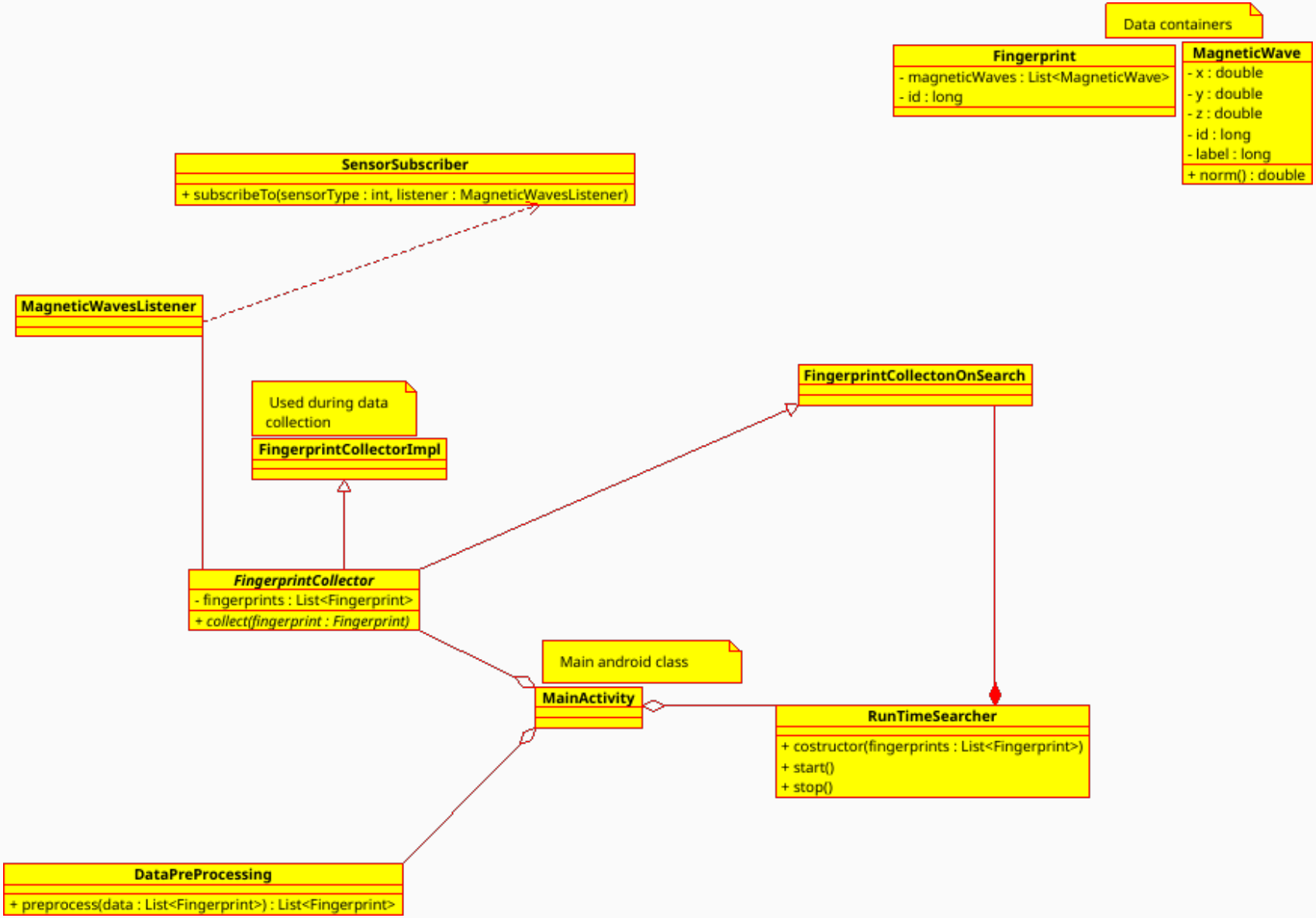




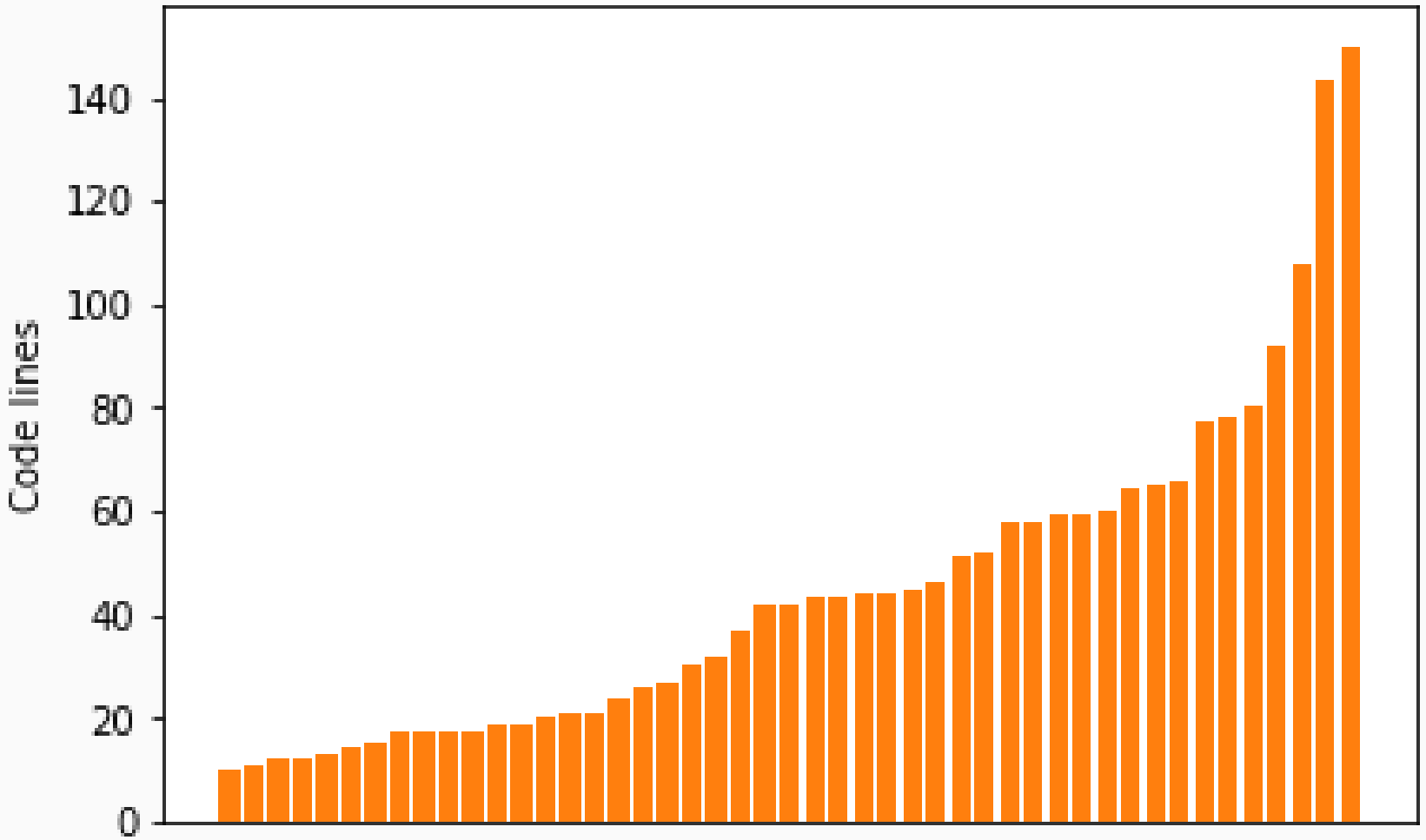
Descrizione dell'applicazione in fasi

1. Dichiarazione ed istanziazione di una sottoclasse dell'interfaccia *SensorListener* la quale ha lo scopo di ricevere l'intensità delle onde magnetiche lungo gli assi visti precedentemente.
2. Superata una certa soglia di onde magnetiche ricevute, esse vengono incapsulate dentro una *fingerprint*.
3. Dopo che l'utente dà l'input di fermare la raccolta delle onde parte in automatico la fase di *preprocessing* dei dati. Per ogni *fingerprint* vengono estratte attributi di natura statistica e non viene effettuata una scalatura sui dati poichè provoca un decadimento dell'accuratezza significativa.
4. Quando l'utente preme sul pulsante "*Start search*" inizia la raccolta di una *fingerprint* per classificarla in base ai dati raccolti nei punti 1-2. Il classificatore usato in questa fase è il knn per via della sua facilità d'implementazione.

UML e codice


Data containers
Fingerprint
- magneticWaves : List<MagneticWave>
- id : long
MagneticWave
- x : double
- y : double
- z : double
- id : long
- label : long
+ norm() : double
SensorSubscriber
+ subscribeTo(sensorType : int, listener : MagneticWavesListener)
MagneticWavesListener
Used during data collection
FingerprintCollectorImpl
FingerprintCollectonOnSearch
FingerprintCollector
- fingerprints : List<Fingerprint>
+ collect(fingerprint : Fingerprint)
Main android class
MainActivity
RunTimeSearcher
+ costructor(fingerprints : List<Fingerprint>)
+ start()
+ stop()
DataPreProcessing
+ preprocess(data : List<Fingerprint>) : List<Fingerprint>

Linee di codice per file

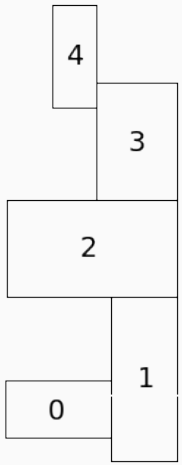


Librerie e linguaggi usati per i test

- Nei test, per via delle ottime librerie votate all'analisi dei dati, è stato usato *Python*
- Le librerie in questione sono *NumPy*, *scikit-learn*, *Pandas*, *matplotlib*

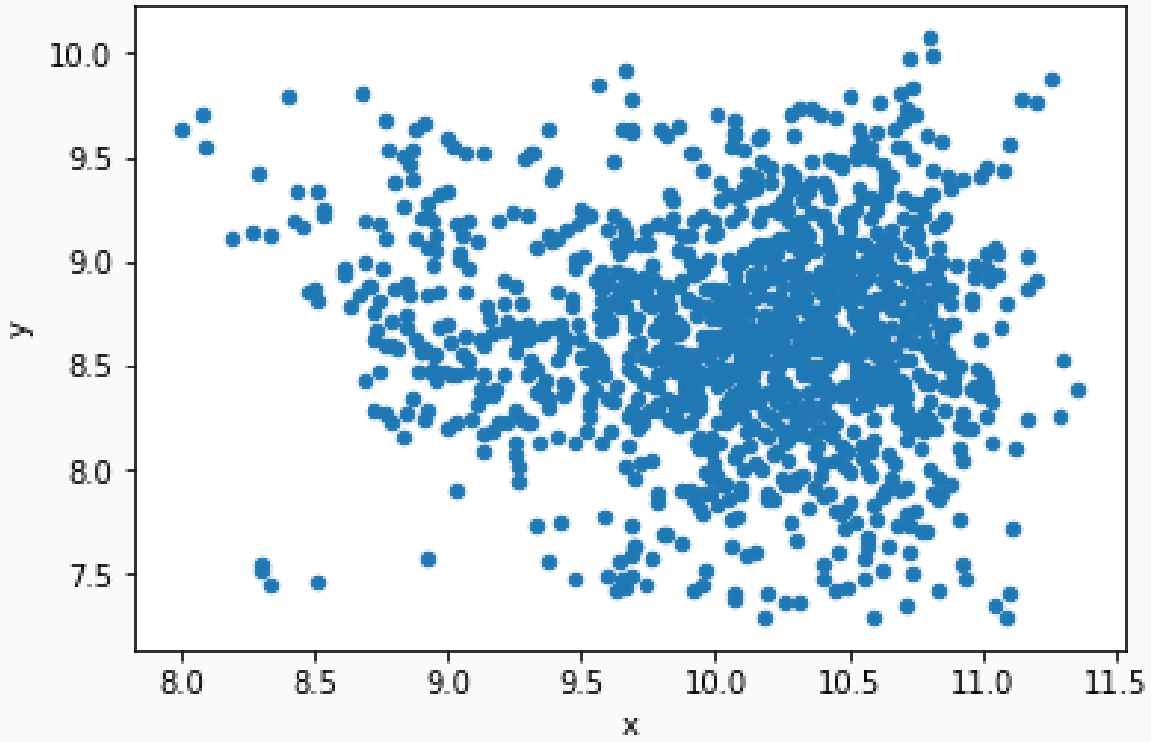
Piano dei test

- Raccolte circa 18000 onde magnetiche
- Metrica usata: errore sull'insieme di test $\left(1 - \frac{y_{true} = y_{pred}}{|y|}\right)$
- Qui sotto vediamo una piantina delle stanze scansionate con l'etichetta in sovrimpressione

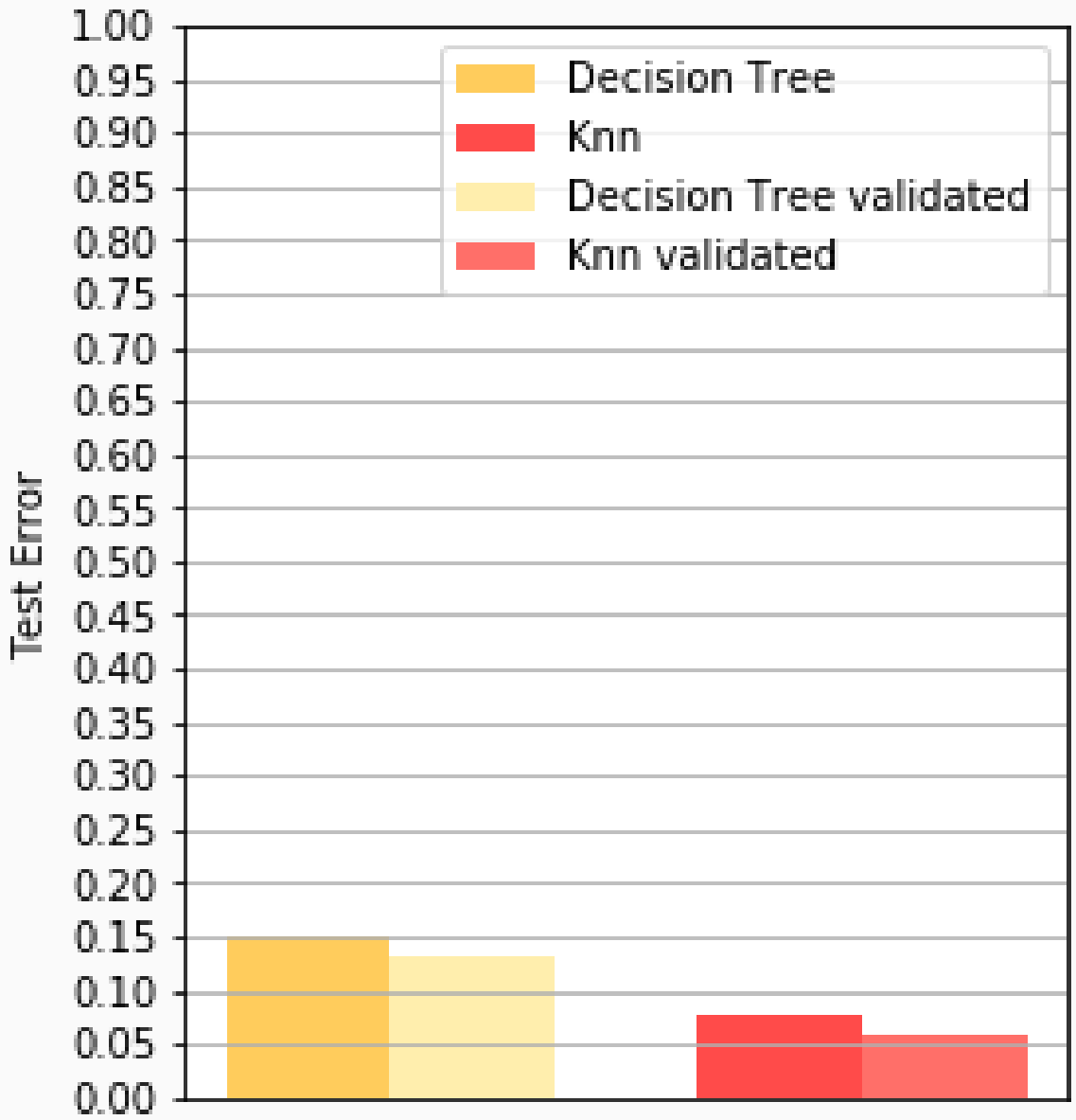


Analisi del rumore sui dati

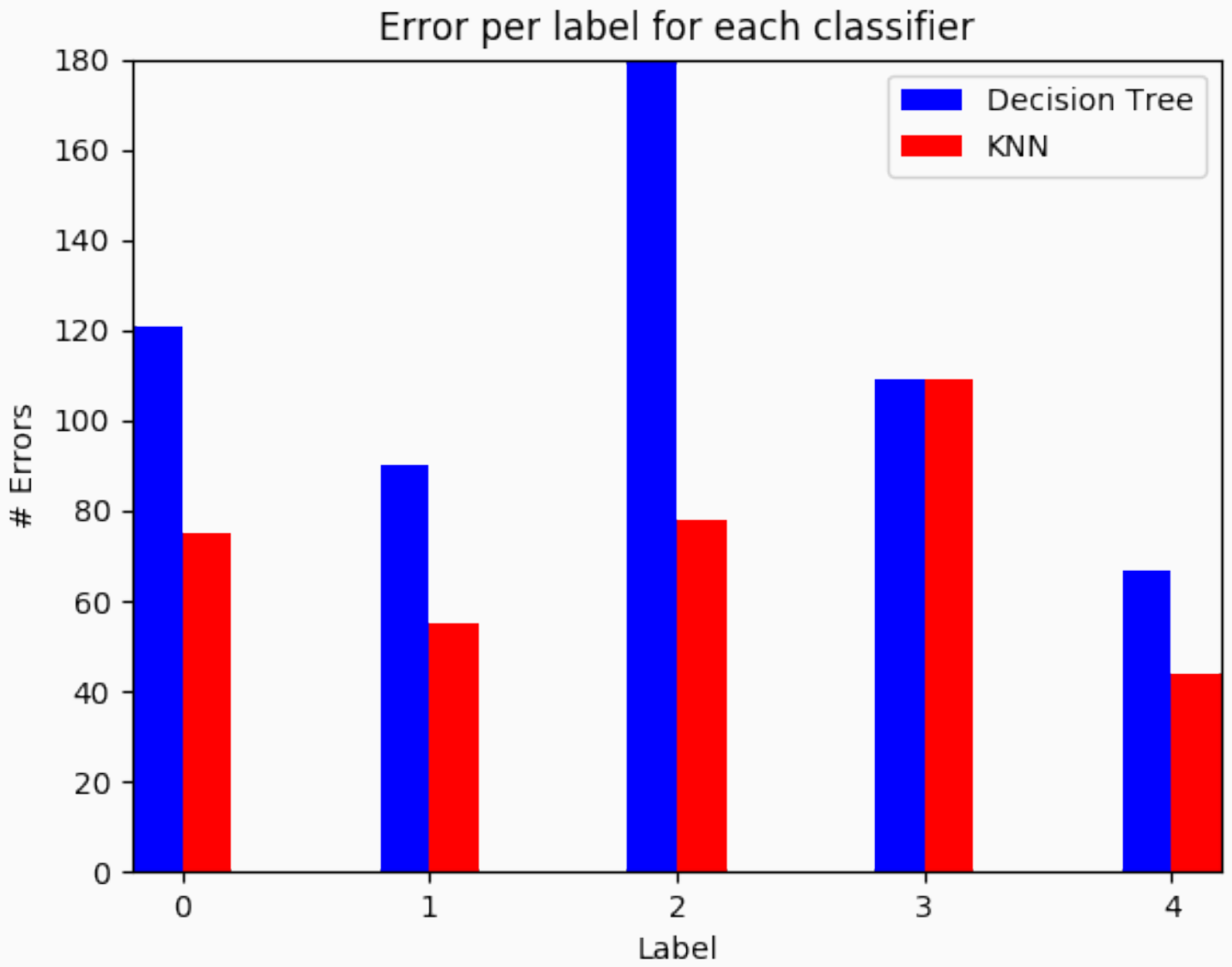
- Per analizzare il rumore sui dati mi sono posizionato fermo per 30 secondi registrando le onde magnetiche.



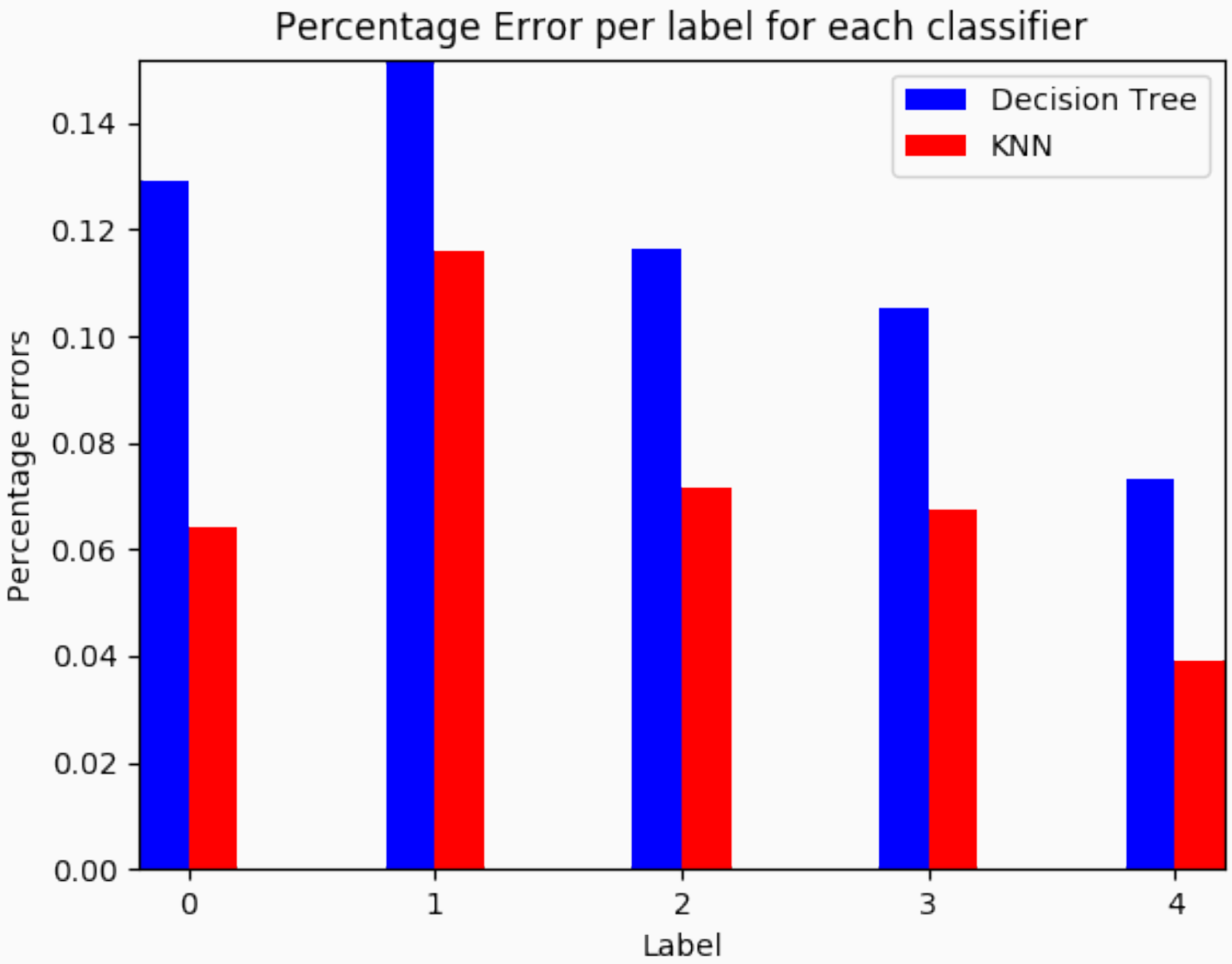
Errore sui test dei classificatori



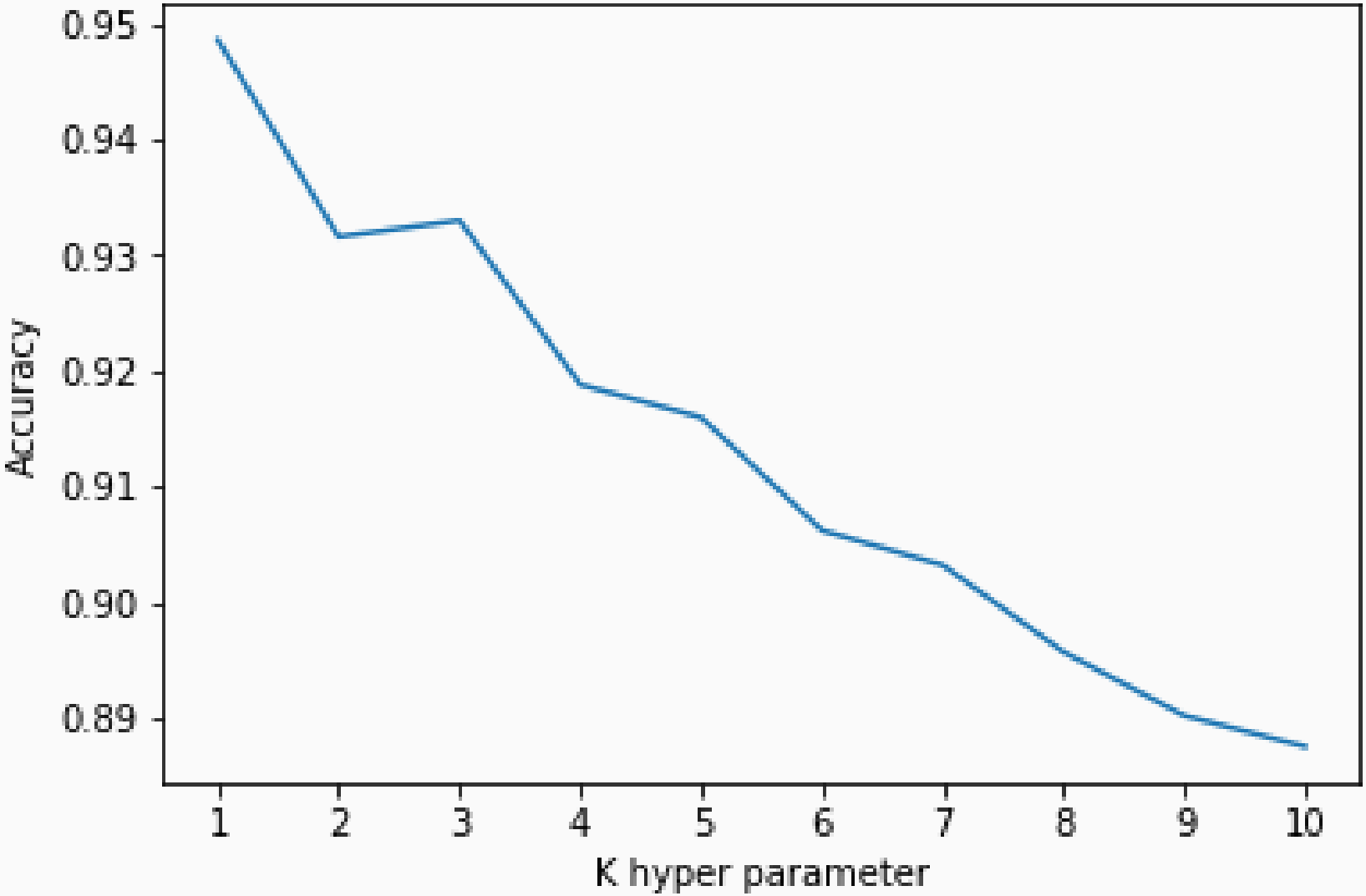
Errore sui test per etichetta



Errore sui test per etichetta in percentuale



Analisi del Knn al variare dell'iper-parametro k

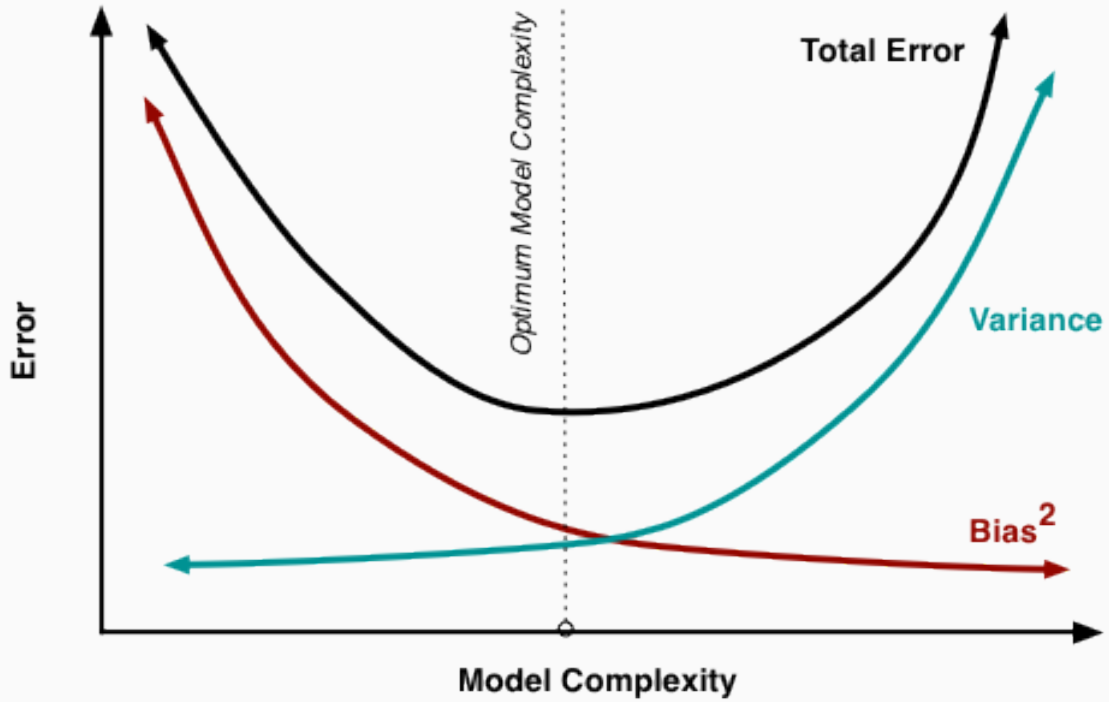


Analisi del Knn al variare dell'iper-parametro k

- Se volessimo la massima accuratezza sceglieremmo $k = 1$ anche se sicuramente faremmo *overfitting* sul modello
- Al contrario, scegliendo un k troppo alto otterremmo *underfitting*
- L'esperienza con il modello aiuta molto nel scegliere gli intervalli di iper-parametri giusti da validare.

Analisi del Knn al variare dell'iper-parametro k

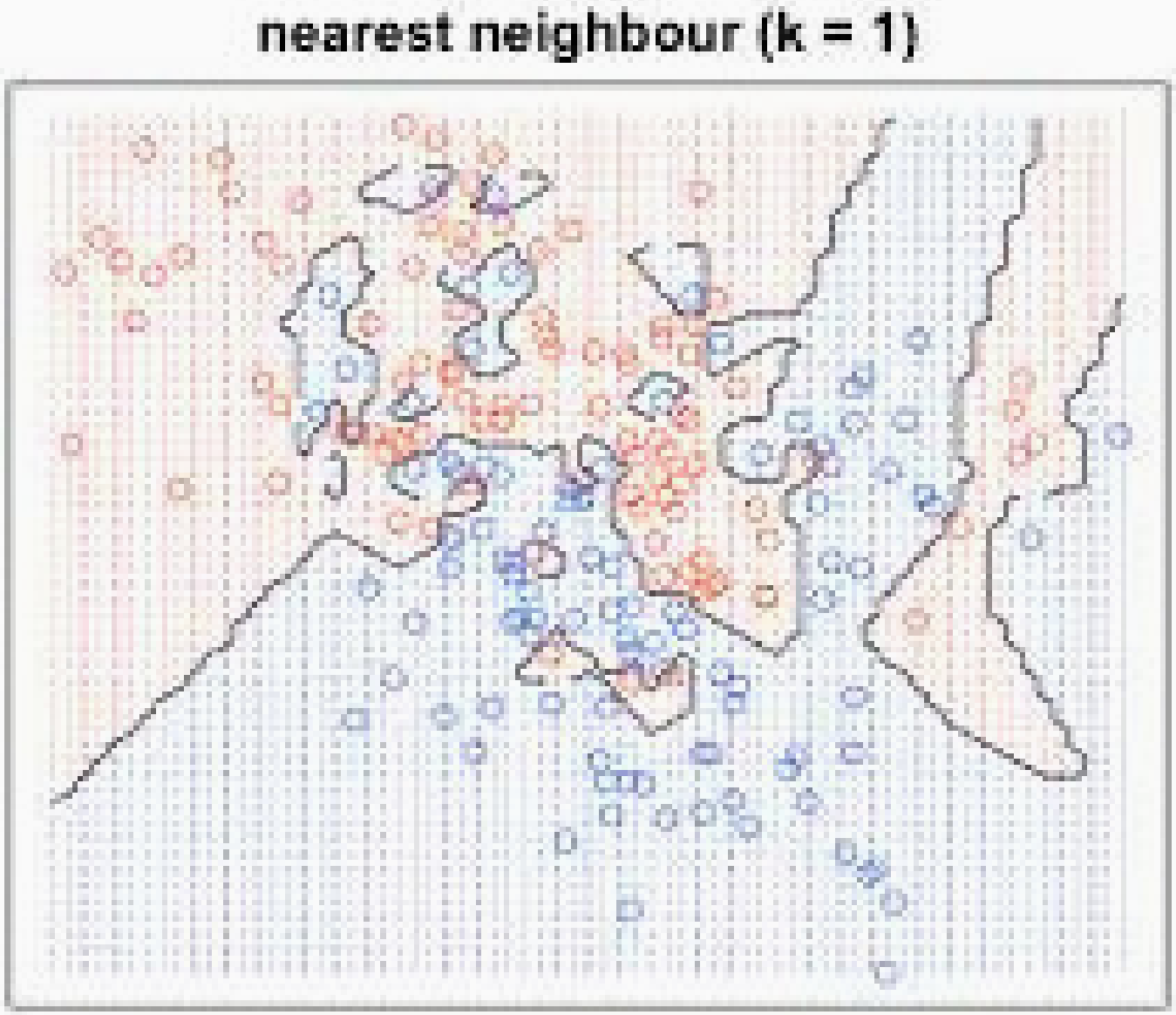
- Un altro strumento per capire se ci stiamo adattando male ai dati sono le curve di validazione



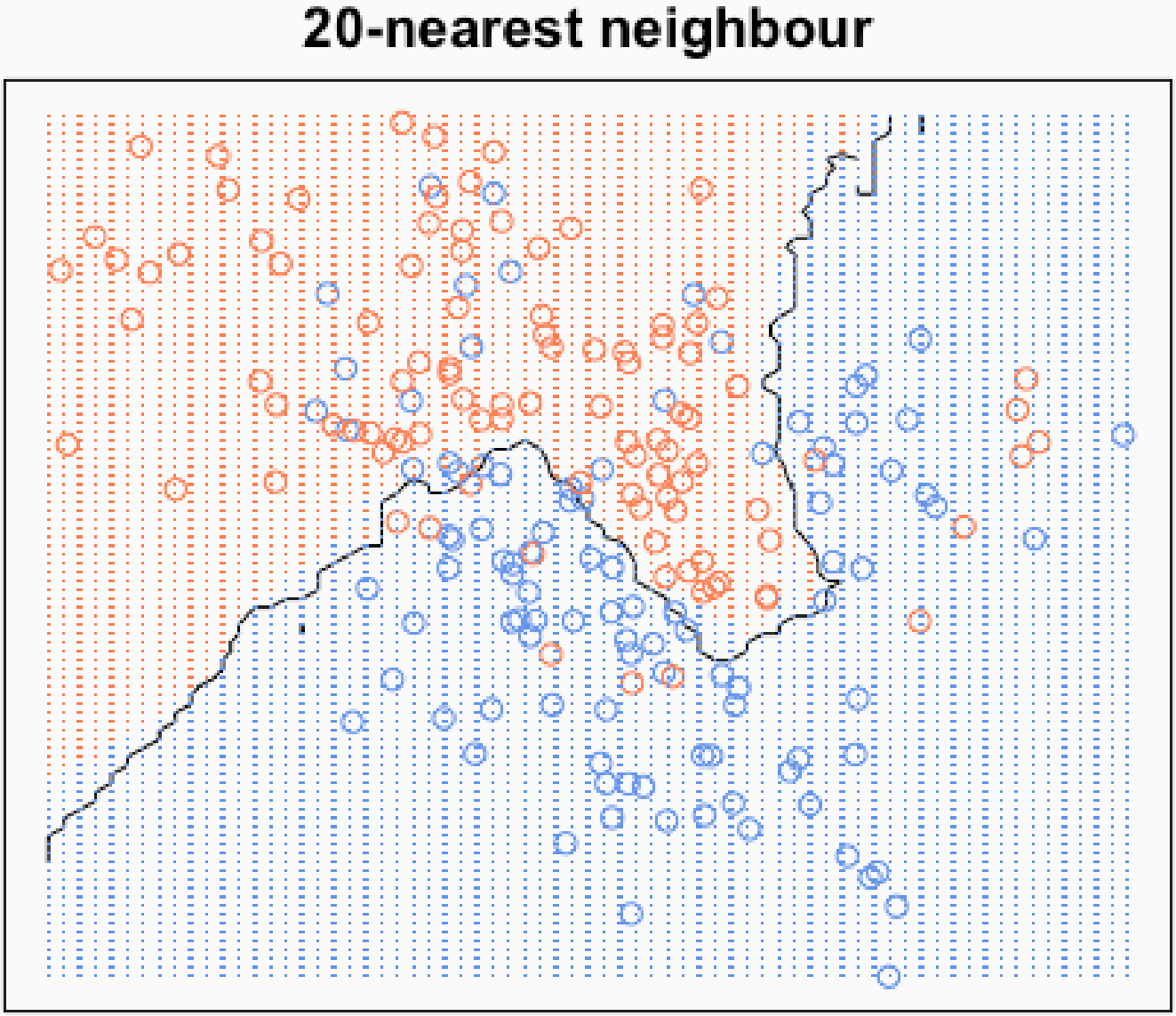
Analisi del Knn al variare dell'iper-parametro k

- L'errore complessivo del modello non è dato solo dalle predizioni sbagliate sull'insieme di test (*bias*) ma anche dalla varianza, cioè che stiamo modellando anche il rumore generato dai dati.
- In generale nel knn all'aumentare di k diminuisce l'errore causato dalla varianza ma aumenta il *bias*.

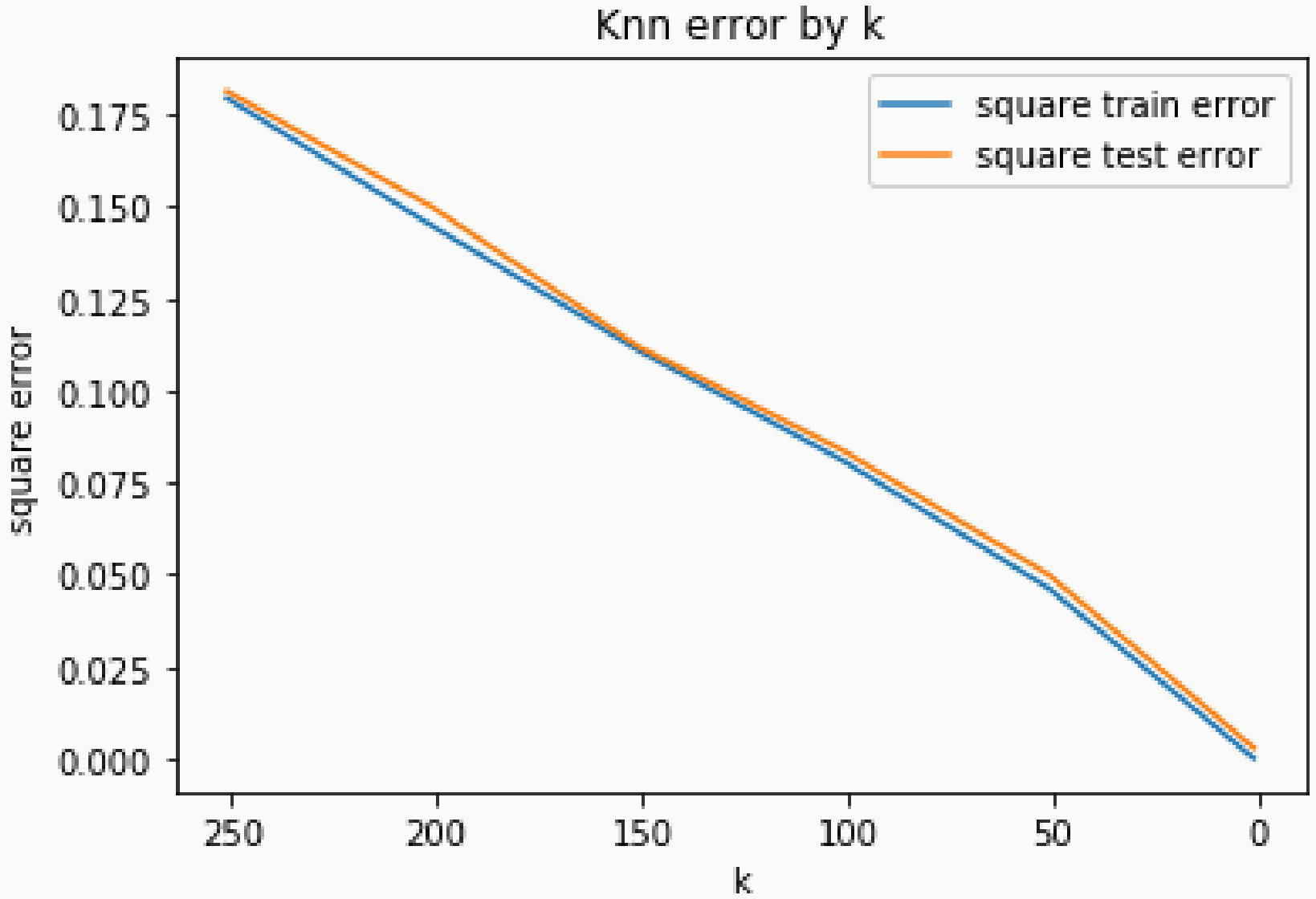
Analisi del Knn al variare dell'iper-parametro k



Analisi del Knn al variare dell'iper-parametro k



Analisi del Knn al variare dell'iper-parametro k



Miglioramenti

- Architettura client-server.
- Sfruttare altri sensori per identificare la posizione dell'utente.
- Sperimentare tecniche di estrazione e selezione di attributi e verificare come varia l'accuratezza.
- Provare altri classificatori più avanzati come svm o reti neurali
- Invece di usare un solo classificatore provare l'*ensemble learning*
- Interfaccia grafica *user-friendly*

Conclusioni

In conclusione abbiamo una base di applicazione *Android* che raccoglie onde magnetiche dal magnetometro, le elabora e predice la posizione dei nuovi input tramite il *knn*, un classificatore. Su computer sono stati effettuati test riguardanti le onde magnetiche tramite l'esposizione di grafici. Abbiamo dimostrato che nei dati è presente rumore, per poi passare ad un confronto di accuratezza fra i classificatori presenti in cui ha ottenuto un minore errore sull'insieme di test il *knn*. Ma è oro tutto ciò che luccica? Visti i buoni risultati, abbiamo approfondito i risultati d'accuratezza al variare di k per scoprire di avere una funzione monotona decrescente. Abbiamo notato che se impostiamo un valore di k troppo basso incorriamo in *overfitting* mentre in *underfitting* se troppo alto.