FACERECOPY

RICONOSCIMENTO FACCIALE DI ATTORI

MEMBRI DEL GRUPPO

MATTEO MARIANO
MICHELE METTA
NICOLA NARGISO

DOCENTE DEL CORSO

NICOLA FANIZZI

INTRODUZIONE

FaceRecoPy è un programma sviluppato in Python per riconoscere il volto di determinati attori presenti all'interno di un Dataset e mostrare attori simili e informazioni sull'attore rilevato prelevate da un'ontologia online.

Ciò avviene principalmente grazie all'utilizzo della libreria *Face Recognition*.

PRINCIPALI LIBRERIE UTILIZZATE

- Per la scrittura del codice del programma sono state utilizzate le seguenti librerie:
- **face_recognition:** per riconoscere i volti. Mette a disposizione una serie di funzioni per la manipolazione dei dati relativi ai volti.
- **cv2:** libreria per elaborare dati relativi alle immagini, per leggerle, visualizzarle e salvarle.
- **numpy:** mette a disposizione array, matrici multi-dimensioali e tensori.
- **math:** libreria che mette a disposizione funzioni matematiche.
- **pickle:** implementa un algoritmo per serializzare un oggetto arbitrario di Python(trasformare l'oggetto in una serie di byte).

PRINCIPALI LIBRERIE UTILIZZATE

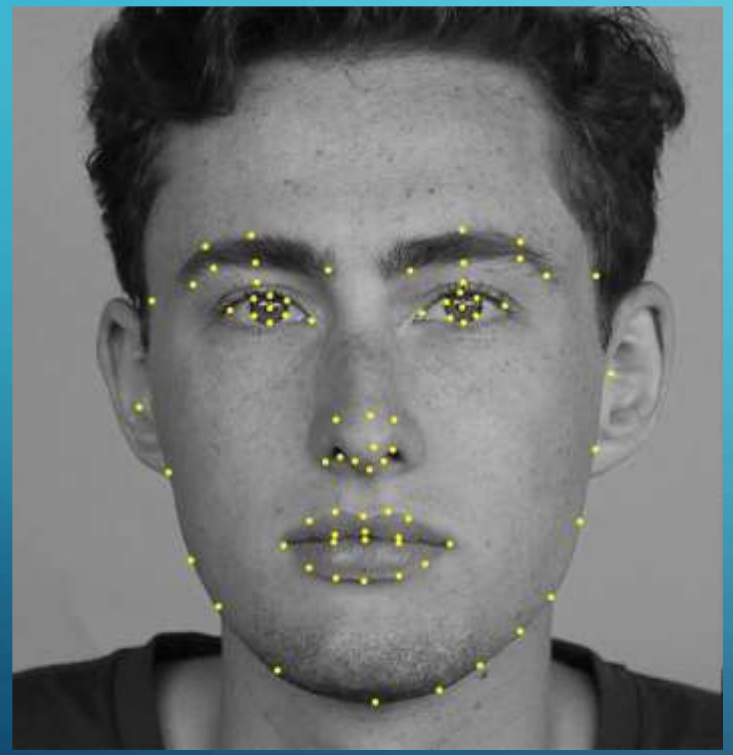
- **SPARQLWrapper:** permette di effettuare query in linguaggio SPARQL mettendo a disposizione metodi per rendere manipolabili i risultati.
- **pandas:** mette a disposizione dati strutturati e strumenti per l'analisi dei dati.
- **sklearn:** mette a disposizione funzioni per analisi dei dati e data mining. Usata per la classificazione e il clustering. In particolare sono stati usati:
- **matplotlib:** di cui sono stati usati i metodi di pyplot per creare dei grafici con i dati ottenuti.
- **xlswriter:** per ottenere una compatibilità con i file xls e scrivere su tali file per raccogliere le feature del dataset.

IL DATASET

Il Dataset è costituito da 52 attori tra cui sono presenti Johnny Depp, George Clooney, Aaron Taylor-Johnson, Al Pacino e molti altri.
Ogni immagine è stata procurata manualmente dai membri del gruppo fino a raggiungere una media di 50 foto per attore.

In genere, delle 50 foto, 35 sono state destinate al Training Set, 15 al Test Set.
In totale si contano 5109 esempi nel Training Set e 1968 esempi nel Test Set.

FEATURE



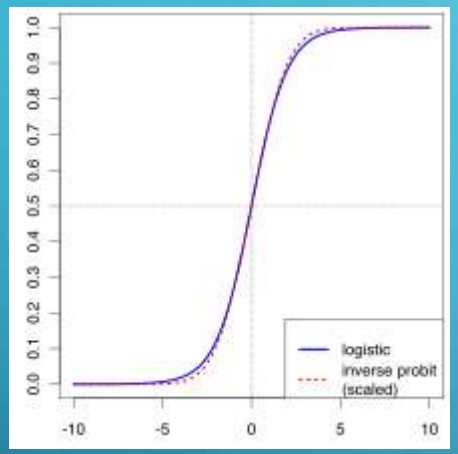
Per raccogliere le feature è stata usata la **Rete Neurale** pre-addestrata messa a disposizione dalla libreria Face Recognition (*Una Rete Neurale Multi-Layer Perceptor*).

Sono state fatte effettuare alla Rete Neurale 128 misurazioni per raccogliere le feature.

CLASSIFICAZIONE

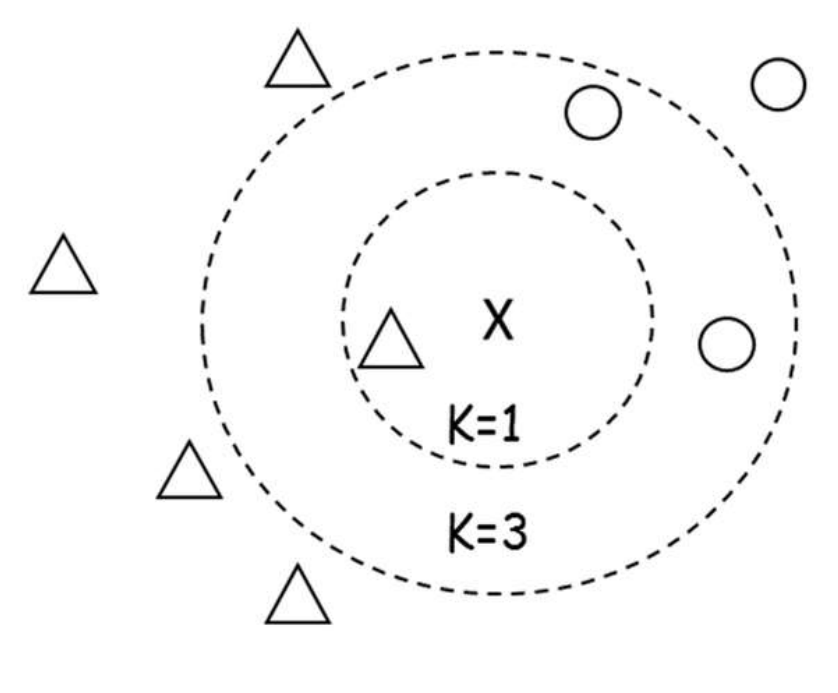
- Per scegliere un classificatore sono stati presi in considerazione:
- **Regressione Logistica**
- **Knn (K-nearest-neighbours)**
- **Gradient Boosting Classifier**
- **Support Vector Machine**
- **Albero di Decisione**
- **Random Forest**
- **Random Forest Regressor**
- **Rete Neurale (MLP Classifier)**
- **Classificatore Bayesiano**

REGRESSIONE LOGISTICA



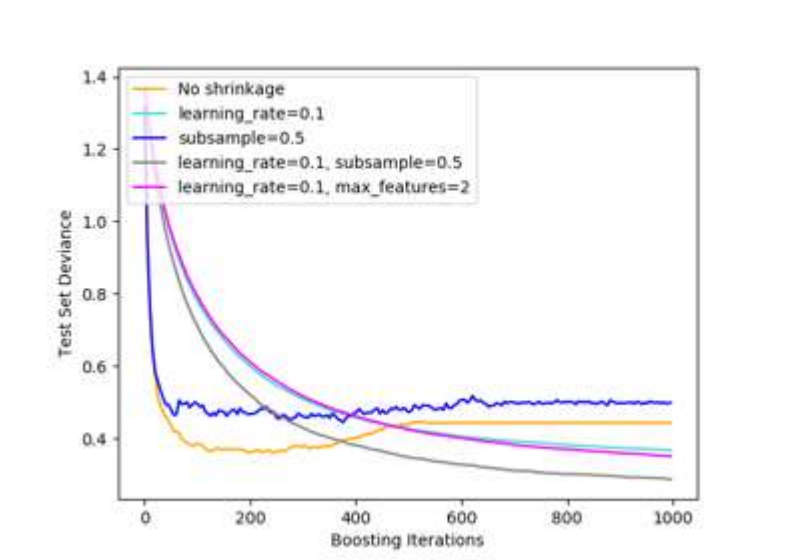
E’ un modello di classificazione in cui si determinano i pesi di una funzione lineare il cui valore di predizione verrà successivamente appiattito da una funzione lineare appiattita. Funzioni di questo tipo sono ad esempio la Sigmoide, la funzione a scalino, la RLU ecc..

K NEAREST-NEIGHBOUR



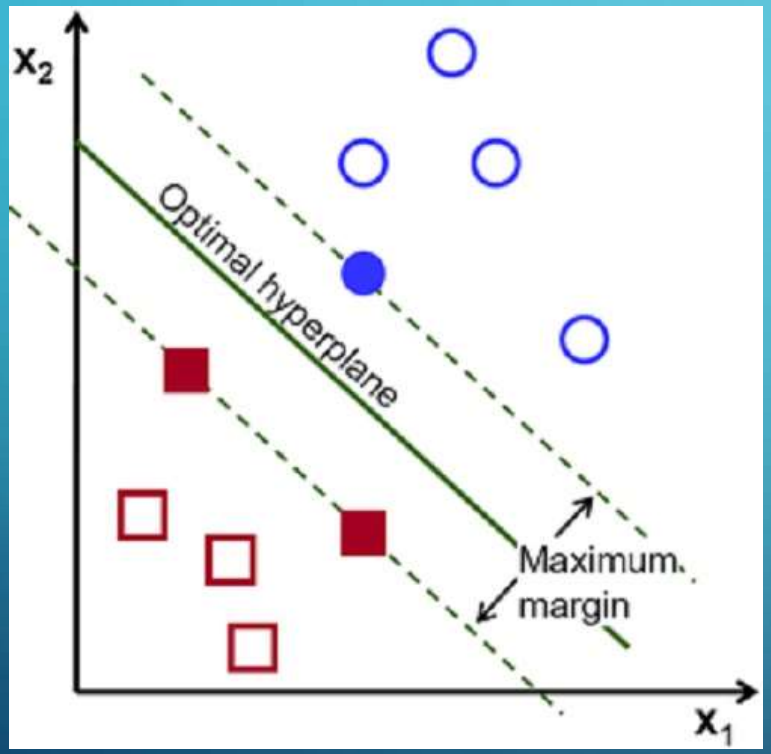
E’ un classificatore che, dato un certo training set e un nuovo esempio da classificare, si comporta nella seguente maniera:individua i k esempi di training piu’ simili a quello da classificare e a questi assegna un peso maggiore per decidere a quale classe il nuovo esempio appartiene. La predizione può essere fatta attraverso l’utilizzo della moda, media o interpolazione dei valori obiettivo dei k esempi più simili. La similarità tra gli esempi viene individuata tramite una metrica (ad es. la distanza Euclidea), prima di fare ciò però è utile molte volte scalare i dati, nel nostro caso non è stato necessario poichè non c’era molta differenza tra la scala di valori delle nostre features.

GRADIENT BOOSTING CLASSIFIER



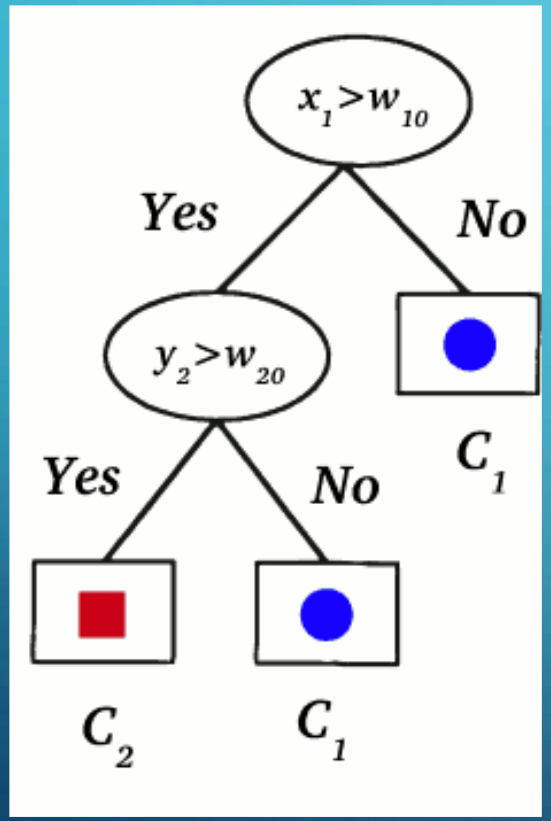
Applicazione dell'algoritmo di *Boosting*, ovvero della costruzione di uno Strong Learner partendo da più Weak Learner. Generalmente vengono usati alberi decisionali. Lo scopo è quello di minimizzare una funzione di costo.

SUPPORT VECTOR MACHINE



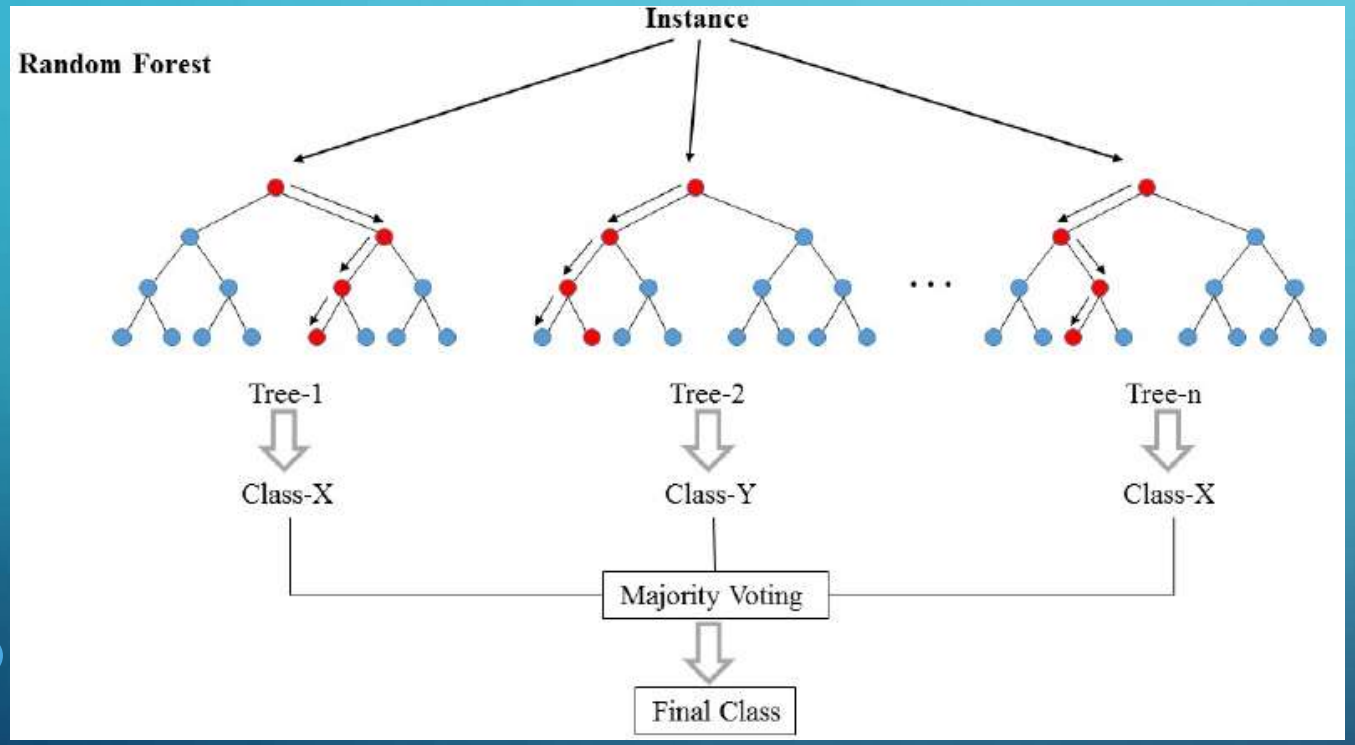
Classificatore che si basa su una superficie di decisione che separa valore negativi da valori positivi. Con le feature esempio si crea un iperpiano di massimo margine per trovare il supporto alla decisione. Il margine è la distanza tra la superficie di decisione e gli esempi, minore è la distanza, maggiore è il supporto alla decisione.

ALBERO DI DECISIONE



Classificatore basato su un albero binario. Ogni nodo presenta una condizione, ogni arco dell'albero è etichettato con un valore di verità (*True*/*False*), a nodo foglia risulterà la classe di appartenenza dell'input inserito nell'albero.

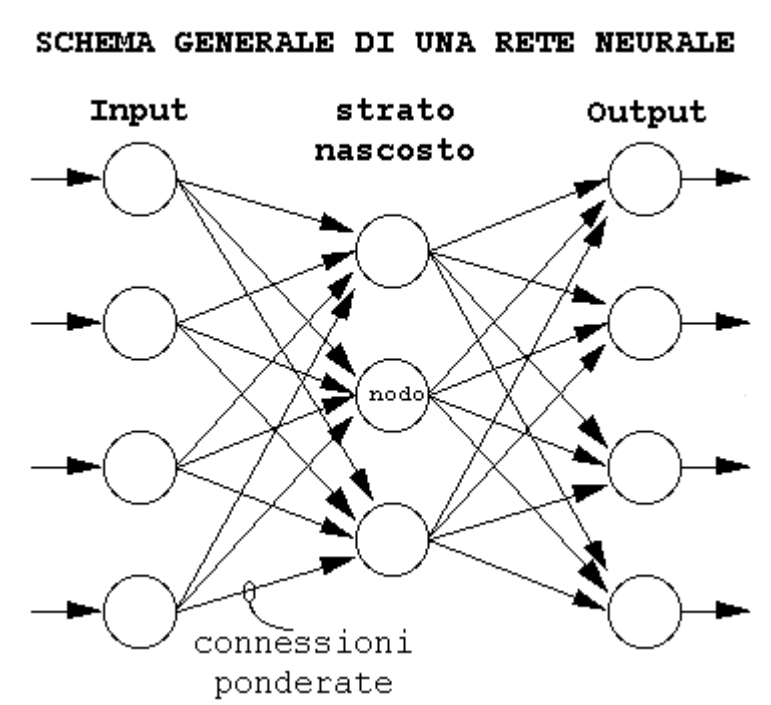
RANDOM FOREST



Si addestrano n alberi di decisione e l'input viene filtrato attraverso di essi. Per definire la classe di appartenenza si calcolerà la media delle decisioni prese dagli alberi di decisione oppure ogni albero voterà la classe di appartenenza.

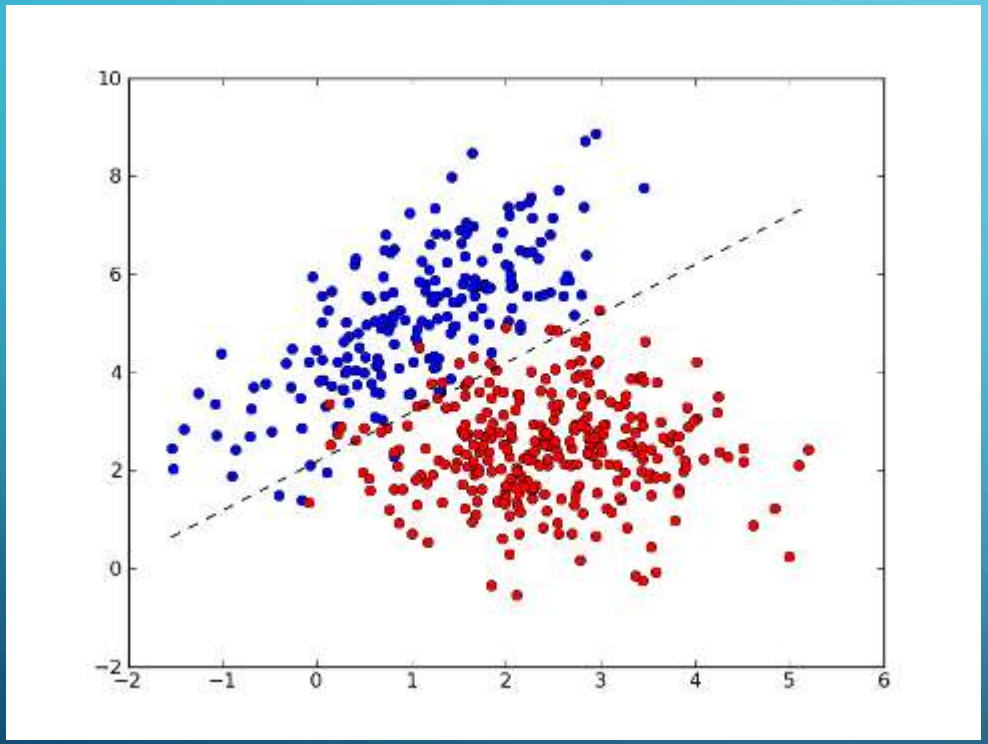
Il Random Forest Regressor è un Random Forest usato per prevedere output a valori reali che variano e non richiede output previsti in un set fisso.

RETE NEURALE (MLP)



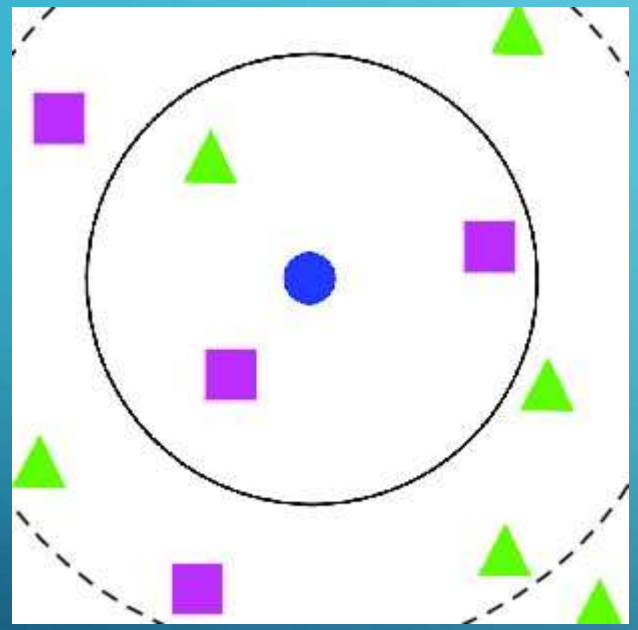
Le Reti Neurali servono principalmente per l'apprendimento supervisionato studiando relazioni tra variabili. Sono pertanto in grado di definire nuove feature. Sono costituite da più *layer nascosti*. L'input attraversa i layer della rete neurale per essere elaborato. Ogni layer è costruito in funzione del precedente e ne esistono di 3 tipologie: *layer di input* con un'unità per feature di input, *layer completo lineare* dove ogni output corrisponde al risultato di una funzione lineare, *layer di attivazione* dove ogni output ha il suo corrispondente valore di input.

CLASSIFICATORE BAYESIANO



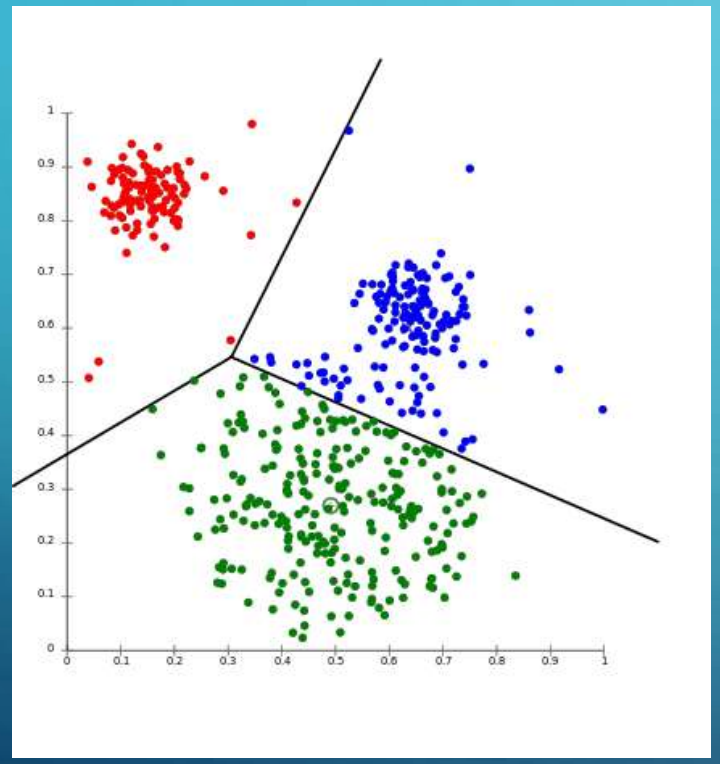
Classificatore probabilistico di apprendimento supervisionato che raggruppa gli esempi in classi poiché hanno valori comuni delle feature e impara la dipendenza delle feature dalla classe.

CLASSIFICATORE - SCELTA



Tenendo conto del *dominio*, del *test score*, del *training time*, il classificatore scelto è stato il **K Nearest-Neighbour.**

RITROVAMENTO DI ATTORI SIMILI



Per mostrare gli attori simili all'attore riconosciuto sono stati calcolati gli attori simili presenti nel Dataset attraverso un algoritmo di hard clustering, il **Kmeans**. Tale algoritmo permette di trovare cluster(insiemi di Dati Simili) senza aver bisogno di feature obiettivo.

INTERAZIONE CON L'ONTOLOGIA

Per mostrare le informazioni di ogni attore riconosciuto è stata usata l'ontologia ***DBpedia***. Attraverso query in linguaggio ***SPARQL*** è stato estratto il valore contenuto in **dbo:abstract** in lingua inglese per ottenere le informazioni sul personaggio riconosciuto

QUERY IN SPARQL

```
PREFIX dbo: <http://dbpedia.org/ontology/>
        SELECT ?abstract
        WHERE {
            <http://dbpedia.org/resource/""" + attore + """> dbo:abstract ?abstract .
            FILTER langMatches( lang(?abstract), "en")
        }
```

PREFIX dichiara prefissi e namespace. <*http://dbpedia.org/ontology/*> è l'**URI.**

SELECT seleziona le variabili da prendere in considerazione nel risultato

WHERE definisce il criterio di selezione.

In questo caso dovendo trovare come risorsa un attore definito è stato scelto di usare l'URI stesso della risorsa in DBpedia.

PYTHON E SPARQL

```
sparql = SPARQLWrapper("http://dbpedia.org/sparql")
sparql.setReturnFormat(JSON)
sparql.setQuery(query)
results = sparql.query().convert()
```

Si inizializza la variabile sparql e si seleziona come formato di output il formato **JSON** e settando la query da effettuare.

Nella variabile **results** sarà presente un result set con il risultato della query, poi esplorato per stampare a video i dati richiesti nella query.

CODICE SORGENTE