
Dipartimento di ingegneria Informatica, Modellistica, Elettronica e Sistemistica (DIMES)
Corso di Laurea Magistrale in Ingegneria Informatica

**Relazione del PROGETTO MACHINE E DEEP LEARNING**

# Classificazione e Anomaly Detection Di:

- **immagini**
- **testi**

Studente:
Mattia Gatto, 216649

Docenti:
Fabrizio Angiulli
Fabio Fassetti

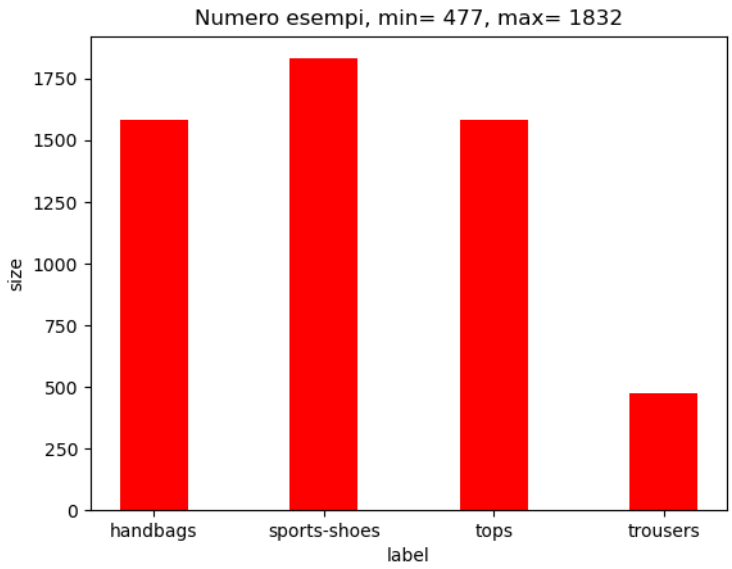
Esercitatore:
Luca Ferragina

# Sommario

# 1. Immagini

Nel realizzare un sistema di apprendimento automatico per la classificazione di immagini, mediante l'utilizzo del linguaggio di programmazione Python, ho preferito partire da un'analisi iniziale del dataset.
Durante questa fase di analisi ho valutato quale fosse l'insieme migliore di tecniche da definire per la fase di pre-processing, al fine di ottenere ottimi risultati in termini di accuratezza.
Il formato dati è una cartella che dispone di 5477 immagini a colori, di dimensione 80*60, in formato jpg.
Vi sono 4 tipi di immagini:

1. "handbags"
2. "sports-shoes"
3. "tops"
4. "trousers"




## 1.1. Preprocessing

In questa fase sono partito con l'utilizzo della libreria "opencv" al fine di riuscire a leggere le immagini e trasformarle in matrici.
Ogni elemento della matrice rappresenta l'intensità del pixel secondo un formato float che va da 0 a 1, in quanto ho preferito normalizzare le singole intensità che precedentemente spaziavano da 0 a 255.

Un ulteriore operazione che ho effettuato tramite la libreria "opencv" durante la lettura delle immagini è stata quella di trasformare quest'ultime in scala di grigi, in quanto mantenerle in formato RGB portava ad un aumento della spazialità, poiché il formato RGB ha 3 canali e di conseguenza per ogni pixel dell'immagine bisognava valutare la corrispondente cella in ognuna delle tre matrici che iniseme determinano il colore vero è proprio, il quale è composto da una certa quantità di red , da una certa quantità di green e da una certa quantità di blue.
Al contrario la scala di grigi dispone di un unico canale che va a determinare l'intensità del pixel in toni di grigio.
Per quanto riguarda le label ho sviluppato un piccolo algoritmo che legge il path delle singole immagini e in base a queste inserisce un indicatore relativo alla categoria dell'immagine all'interno di un array numpy.
Tale l'indicatore è un intero che vale :

- 0 per "handbags"
- 1 per "sports-shoes"
- 2 per "tops"
- 3 per "trousers"

Al termine delle seguenti operazioni ho definito un trainset e un testset sia per le label che per il dataset che verranno utilizzati per l'addestramento e la corrispondente valutazione dei modelli.

## 1.2. Classificazione

Nella fase di classificazione sono stati sviluppati modelli relativi alle tecniche richieste dalla traccia del progetto, e addestrati prima su un train delle label e del dataset di immagini e valutati successivamente su un testset per determinare l'accuratezza iniziale, e infine valutate secondo una tecnica di cross validation ossia Kfold, con valore di k=10.

Infatti per ogni tecnica di classificazione, al termine dell'addestramento, ho effettuato la valutazione dell'accuratezza su un testset di dataset e label al fine di produrre i relativi risultati in termini di accuratezza e loss "sparse_categorical_crossentropy", e successivamente ho lanciato una serie di comandi per la valutazione delle prestazioni sfruttando kfold validation che mi ha restituito come risultato un vettore "scores" contenente le 10 valutazioni di accuratezza secondo la tecnica usata, e da ciò ho preso la media di questi 10 valori e ho confrontato il risultato con il valore precedentemente ottenuto dalla valutazione con il test set.

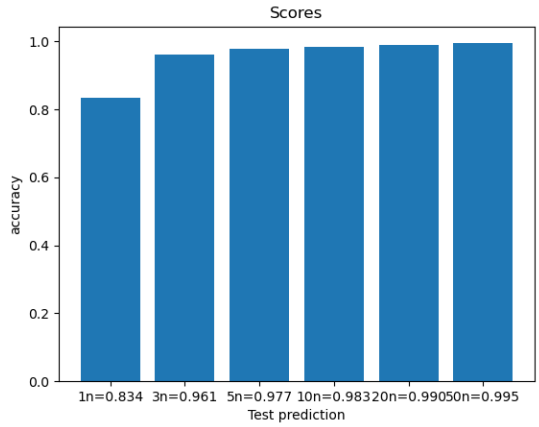
Al termine dell'esecuzione di ogni modello vengono plottati i risultati al fine di effettuare confronti.

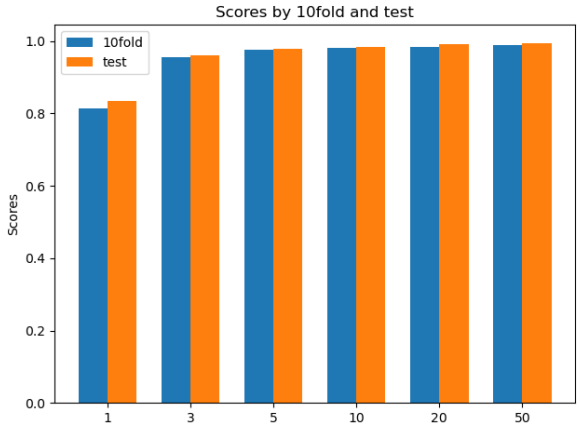
### 1.2.1. AdaBoost

Inizialmente ho portato il trainset e il testset in un formato "2D" in modo da semplificare e dare in pasto all'algoritmo i dati presenti nel dataset che precedentemente erano in formato matriciale.
Nella prima fase di addestramento e valutazione con il test set ho definito una lista di parametri da dare ai diversi classificatori di tipo Adaboost, i quali successivamente vengono dati in pasto all'algoritmo OneVsRestClassifier di sklearn utilizzato per effettuare la multi-classificazione, dal momento che abbiamo a disposizione 4 elementi da classificare e non 2.

Al termine di questa operazione che verrà effettuata anche per l'algoritmo di SVM e delle Stime di densità, ho lanciato l'addestramento per ognuno dei parametri.

I risultati ottenuti sono questi:



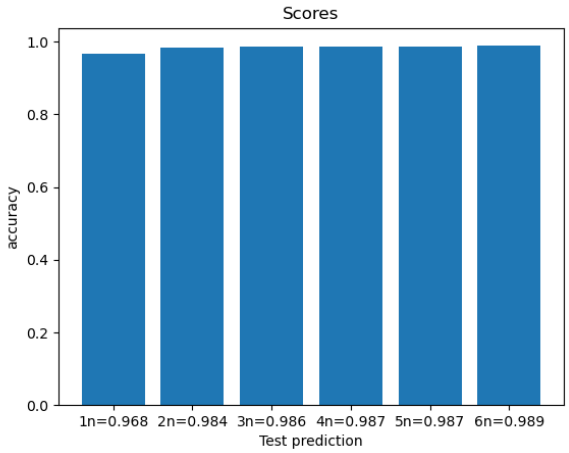




Dai plot si può notare come in presenza di un numero elevato di semispazi l'accuratezza tende a migliorare e dal confronto con i risultati della cross validation possiamo confermare tutto ciò.

### 1.2.2. SVM

Come operazioni preliminari sono state effettuate le medesime per Adabost, e qui ho effettuato una analisi dell'algoritmo di support vector machine sfruttando, un kernel di tipo polinomiale, poiché ho riscontrato notevoli aumenti in termini di accuratezza rispetto al classico lineare.
Effettuate le operazioni preliminari, ho lanciato l'addestramento per ognuno dei parametri esposti nella lista che indica il grado del polinomio da assegnare al classificatore, che varia tra 1 e 8, poiché ho voluto far notare come tende a migliore l'accuratezza dal grado 1 al 4 all'incirca, e poi via via, tende ad arrestarsi e rimanere costante.

I risultati ottenuti sono questi:

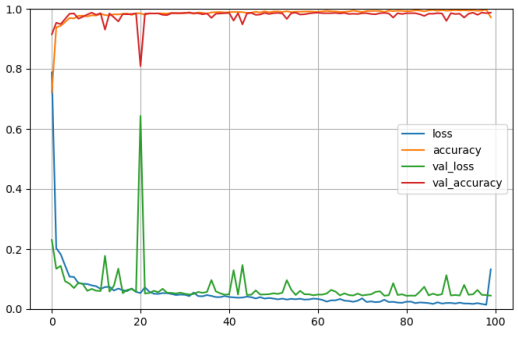


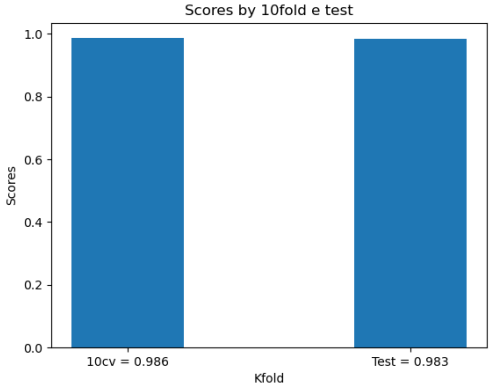


### 1.2.3. Reti Neurali

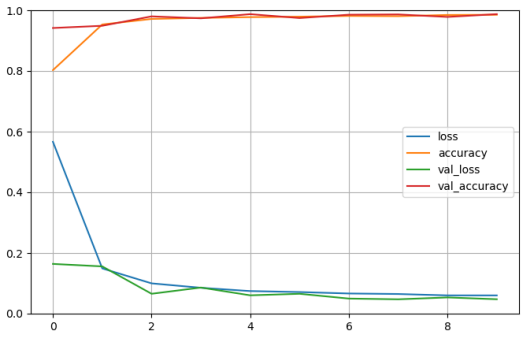
Inizialmente facciamo delle operazioni preliminari, al fine di definire un trainset e un testset, e successivamente ho definito un modello di tipo denso e un modello di tipo convoluzionale, al fine di vedere le differenze che portavano in termini di validation:

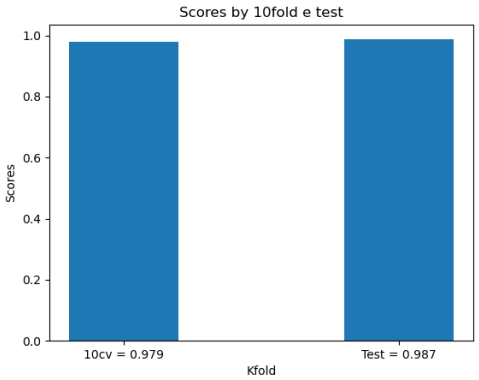
- Nel modello denso ho impostato prima l'input nel flattern che successivamente ho mandato in altri 2 livelli densi di cui il primo di 300 neuroni e funzione di attivazione "relu", e il secondo di 200 neuroni e anche esso attivato con la funzione "relu", e alla fine ho passato il risultato all'ultimo livello per l'uscita in output che questa volta sfrutta una funzione di attivazione softmax.
  Per la compilazione del modello ho usato un'ottimizzazione di discesa stocastica del gradiente, ho addestrato il modello su 100 epoche, valutandolo in seguito.
  I risultati ottenuti sono questi:



- Nel modello convoluzionale ho impostato prima l'input da dare al modello, definendo la dimensione dei miei dati in ingresso e successivamente ho mandato il risultato in altri 2 livelli densi convoluzionali di cui il primo di 32 neuroni e funzione di attivazione "relu", e il secondo di 16 neuroni e anche esso attivato con la funzione "relu", che operano con un filtro 3*3 nella definizione dell'immagine, e alla fine ho passato il tutto all'ultimo livello per l'uscita in output che questa volta sfrutta una funzione di attivazione softmax.
  Per la compilazione del modello ho preferito una ottimizzazione sfruttando "adam", e successivamente ho addestrato il modello su 10 epoche, valutandolo in seguito.
  I risultati ottenuti sono questi:



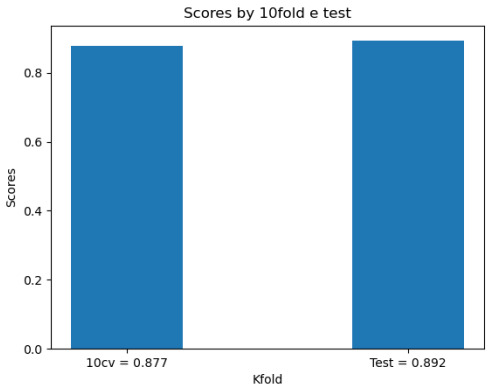




Si può notare come la rete densa sia migliore rispetto alla convoluzionale in termini di accuracy media definita dalla kfold cross validation.

### 1.2.4. Stima di densità

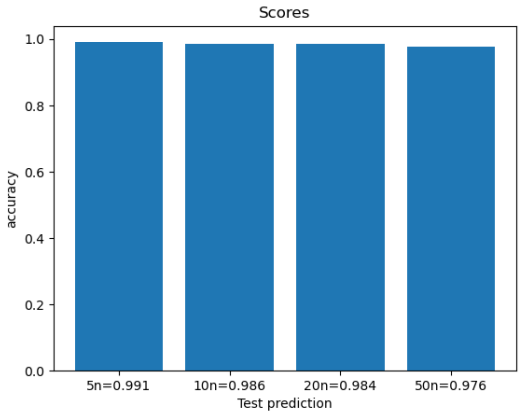
Come stime di densità ho preferito utilizzare sia un gaussian naive bayes, che un nearest neabor differenziandolo sui valori degli iperparametri:

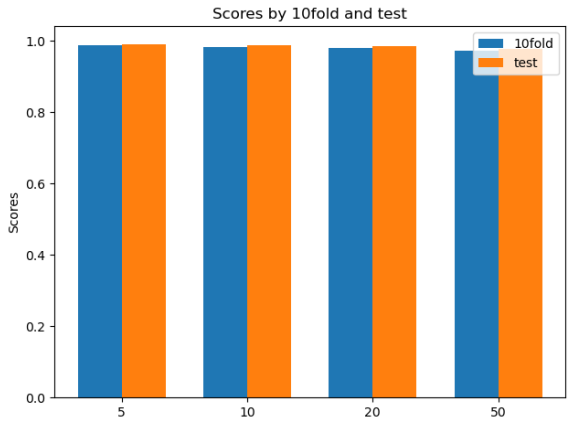
- Nel modello gaussian naive bayes ho impostato inizialmente una distribuzione gaussiana e definito un modello che successivamente ho dato all' OneVsRestClassifier per poi addestrarlo sui corrispettivi trainset definiti, valutandolo in seguito.
  I risultati ottenuti sono questi:




- Come operazioni preliminari sono state effettuate le medesime per Adabost, e qui ho effettuato una analisi dell'algoritmo di nearest neighbor, dove dopo avere eseguito operazioni preliminari ho lanciato l'addestramento per ognuno dei parametri esposti nella lista che indica il numero dei vicini da assegnare al classificatore, poiché ho voluto far notare come tende a migliorare l'accuratezza spaziando nella definizione del numero di vicini e poi via via tende ad arrestarsi e rimanere

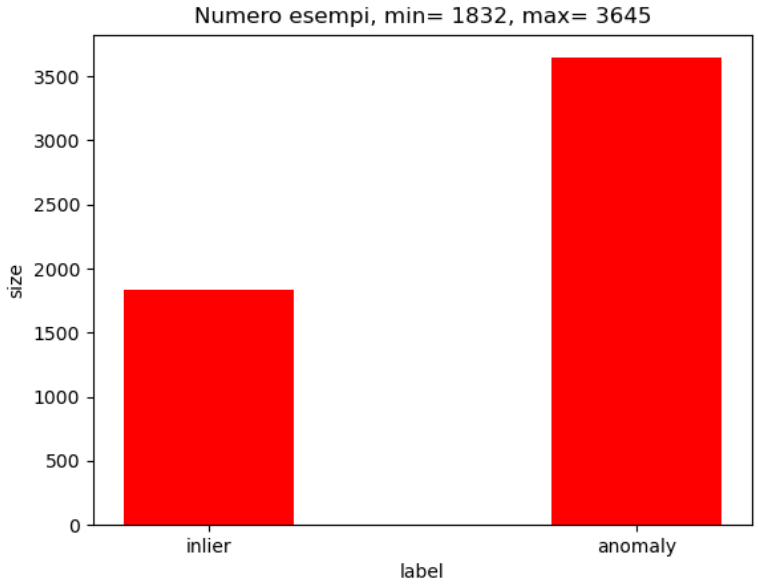
costante, e in seguito ho effettuato le opportune valutazioni.
I risultati ottenuti sono questi:



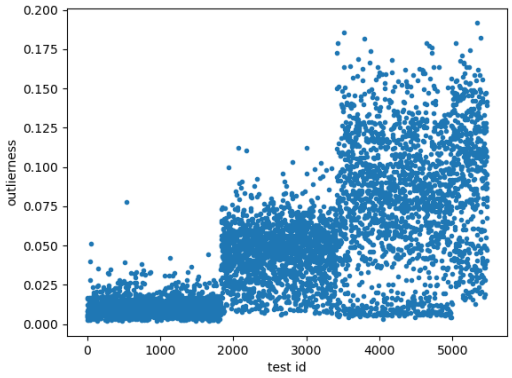

## 1.3. Anomaly Detection

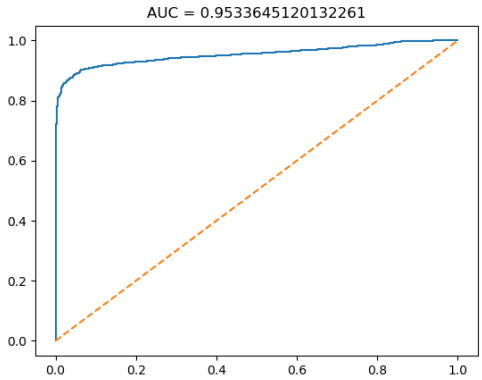
In questa fase sono dovuto ritornare ad una fase di preprocessing al fine di determinare un'insieme di label composti da 0 in presenta del valore massimo in termini di numero di immagini che rappresenterà il valore inlier e 1 tutti gli altri valori identificati come outlier, e al termine delle seguenti operazioni ho definito un trainset sia per le label che per il dataset.




Nella fase di Anomaly Detection ho sviluppato un modello del tipo autoencode basato su rete neurale densa.
Nel modello denso definito, abbiamo prima l'encoder che prende l'input sfruttando keras e successivamente lo manda in altri 2 livelli densi, uno con 256 e il successivo con 128 neuroni sfruttando come funzione di attivazione la "relu", e in seguito ho definito il decoder che prende l'encoder, lo manda ad un livello denso con 128 neuroni che sfrutta come funzione di attivazione sempre la "relu" e alla fine passa il risultato all'ultimo livello denso per l'uscita in output con una dimensione pari alla stessa dell'input, sfruttando questa volta la funzione sigmodea come attivatore. Per la compilazione del modello ho usato un'ottimizzazione di tipo "adam" con loss="mse", e subito dopo ho addestrato il modello su 10 epoche con un batch di 32, valutandolo in seguito nella definizione dell' andamento sulla curva di ROC e il corrispettivo

valore di AUC.
Per la definizione dei numeri relativi ai neuroni nei livelli densi ho impostato questi particolari valori in quanto attraverso esperimenti empirici ho notato come il valore di AUC tendeva a diminuire con l'aumento dei neuroni nel livello denso, e allo stesso tempo anche per quanto riguarda il numero di epoche e il corrispettivo batch ho notato come su questi particolari iperparametri ottenevo valori migliori rispetto all'utilizzo di altri di valore maggiori.
I risultati ottenuti sono questi:



# 2. Dati Sequenziali testuali

Nel realizzare un sistema di apprendimento automatico per la classificazione di dati sequenziali ti tipo testuale, mediante l'utilizzo del linguaggio di programmazione Python, ho preferito partire da un'analisi iniziale del dataset. Durante questa fase di analisi ho valutato quale fosse l'insieme migliore di tecniche da definire per la fase di pre-processing, al fine di ottenere ottimi risultati in termini di accuratezza.
Il formato dati è un file excel che dispone di 10262 recensioni, ognuna delle quali ha una valutazione da 1 a 5.

## 2.1. Preprocessing

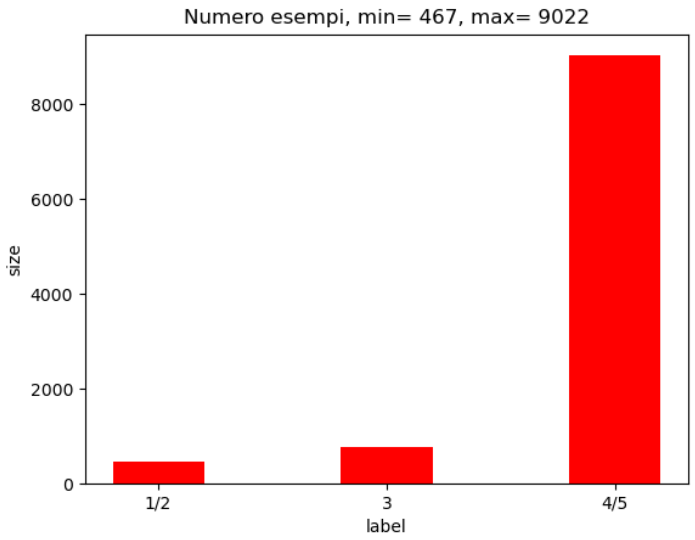
In questa fase sono partito con l'utilizzo della libreria "nltk" al fine di riuscire elaborare ogni frase, infatti, per ognuna di esse vado ad effettuare:

1. Tokenizatione: ossia un processo di divisione in token della frase;
2. Processo di rimozione della punteggiatura poichè non utile;
3. Stopping: processo di eliminazione delle stopwords dalla frase;
4. Stemming: ossia un processo di riduzione della forma flessa di una parola alla sua forma radice, detta "tema".
5. Feature_extraction: un processo che pesa le stringhe per definire una forma di confronto tra le varie valutazioni, e nello specifico ho preferito la tecnica di TfidfVectorizer che tende a normalizzare i valori nell'intervallo [0,1].

Al termine di questo processo il mio dataset sarà l'insieme delle stringhe che hanno effettuato il processamento iniziale e sono state inserite in una matrice sparsa che è stata ridefinita a numpy matrix tramite la funzione "todense()".

Per quanto riguarda le label dal momento che risultavano già precedentemente fissate tra 1 e 5, ho preferito suddividerle nuovamente in un nuovo formato, dove:

- 0 indica le label 1 e 2 poiché così identifichiamo una valutazione di tipo pessima.
- 1 indica la label 3 poiché così identifichiamo una valutazione di tipo normale.
- 2 indica le label 4 e 5 poiché così identifichiamo una valutazione di tipo ottima.




Al termine delle seguenti operazioni ho definito un trainset e un testset sia per le label che per il dataset che verranno utilizzati per l'addestramento e la corrispondente valutazione dei modelli.

## 2.2. Classificazione

Nella fase di classificazione sono stati sviluppati modelli relativi alle tecniche richieste dalla traccia del progetto, addestrati prima su un train delle label e del dataset di testi, poi valutati successivamente su un testset per determinare l'accuratezza iniziale, e infine valutate secondo una tecnica di cross validation ossia Kfold con valore di k=10.

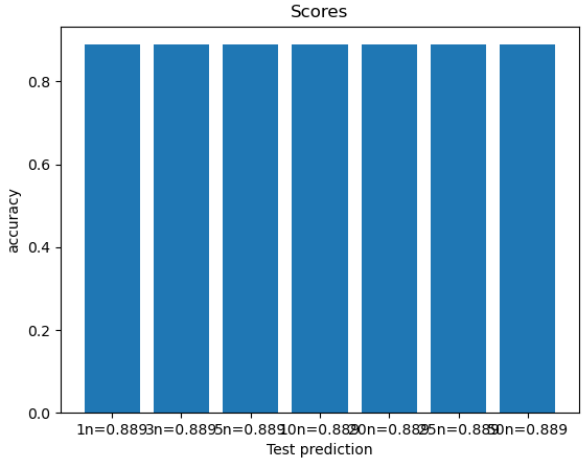
Al termine dell'esecuzione di ogni modello vengono plottati i risultati al fine di effettuare confronti.

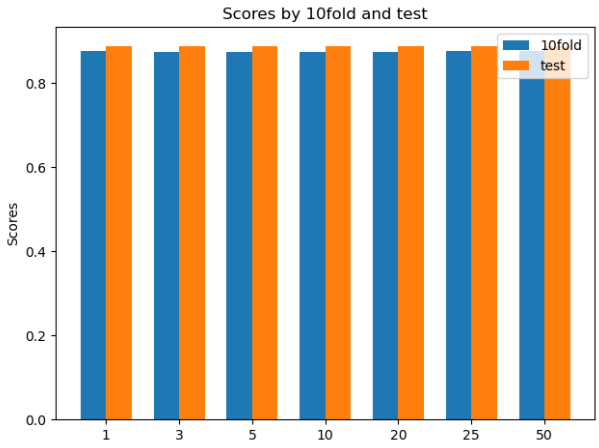
### 2.2.1. AdaBoost

Inizialmente prima della fase di addestramento e valutazione con il test set ho definito una lista di parametri da dare ai diversi classificatori di tipo Adaboost i quali successivamente vengono dati in pasto all'algoritmo OneVsRestClassifier di sklearn utilizzato per effettuare la multi classificazione, dal momento che abbiamo a disposizione 3 elementi da classificare e non 2.

Al termine di questa operazione che verrà effettuata anche per l'algoritmo di SVM e delle Stime di densità, ho lanciato l'addestramento per ognuno dei parametri.

I risultati ottenuti sono questi:



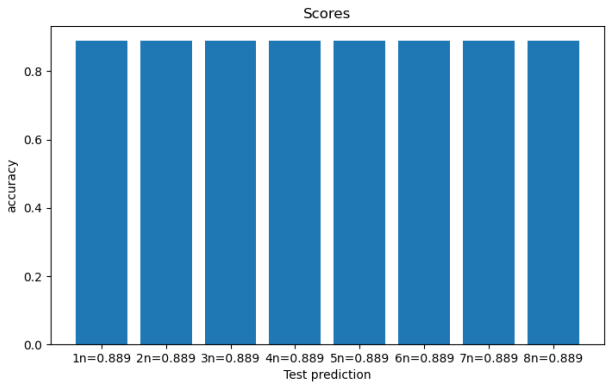




### 2.2.2. SVM

Come operazioni preliminari sono state effettuate le medesime per Adabost, e qui ho effettuato una analisi dell'algoritmo di support vector machine sfruttando, un kernel di tipo polinomiale, poiché ho riscontrato notevoli aumenti in termini di accuratezza rispetto al classico lineare.

Effettuate le operazioni preliminari ho lanciato l'addestramento per ognuno dei parametri esposti nella lista che indica il grado del polinomio da assegnare al classificatore che varia tra 1 e 8, poiché ho voluto far notare come tende a migliore l'accuratezza dal grado 1 al 4 all'incirca, e poi via via, tende ad arrestarsi e rimanere costante.

I risultati ottenuti sono questi:

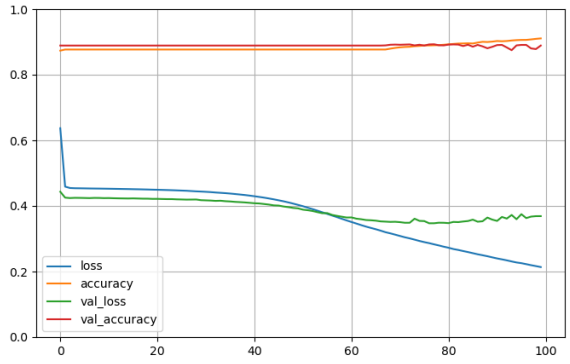


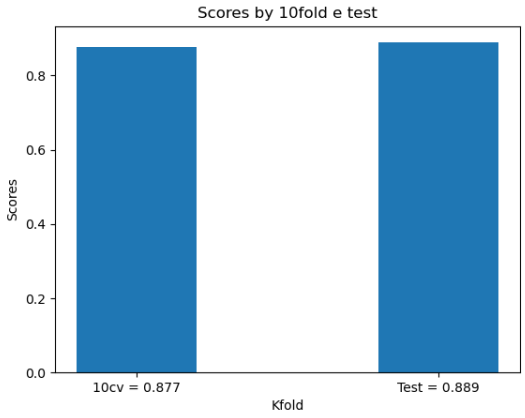


### 2.2.3. Reti Neurali

Inizialmente ho definito un modello di tipo denso al quale ho impostato prima l'input nel flattern, successivamente ho mandato il tutto in altri 2 livelli densi rispettivamente da 16 e 32 neuroni con funzioni di attivazione "relu" e alla fine ho passato il risultato all'ultimo livello con 3 neuroni e e funzione di attivazione sigmodea per l'uscita in output.
Nella valutazione della rete neurale a causa delle dimensioni elevate purtroppo ho dovuto ridurre la dimensione del dataset attraverso una fase di preprocessing che effettua le identiche operazioni fatte inizialmente, ma questa volta nella fase di feature extraction con la tecnica TfidfVectorizer ho impostato un parametro di max_df al fine di eliminare qualche elemento che disponeva di un alta frequenza e quindi poco rilevante nel confronto delle valutazioni.
Per la compilazione del modello ho usato un'ottimizzazione di discesa stocastica del gradiente, ho addestrato il modello su 100 epoche con un batch di size pari a 32, valutandolo in seguito.

I risultati ottenuti sono questi:

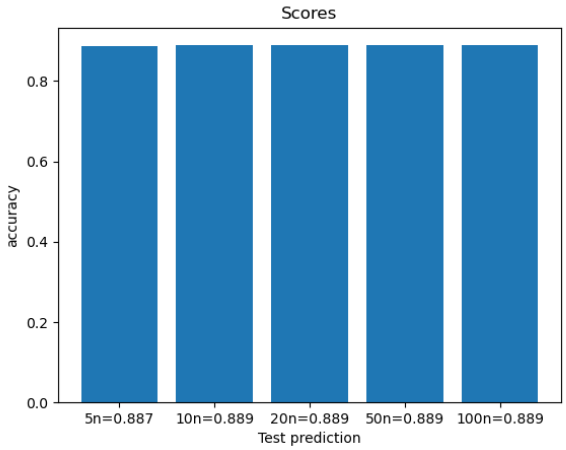



### 2.2.4. Stima di densità

Come stime di densità ho preferito utilizzare un nearest neabor differenziandolo sui valori degli iperparametri. Come operazioni preliminari sono state effettuate le medesime per Adabost, e qui ho effettuato una analisi dell'algoritmo di nearest neighbor, dove l'addestramento per ognuno dei parametri esposti nella lista che indica il numero dei vicini da assegnare al classificatore, poiché ho voluto far notare come tende a migliorare l'accuratezza spaziando nella definizione del numero di vicini e poi via via tende ad arrestarsi e rimanere costante, e in seguito ho effettuato le valutazione opportune.

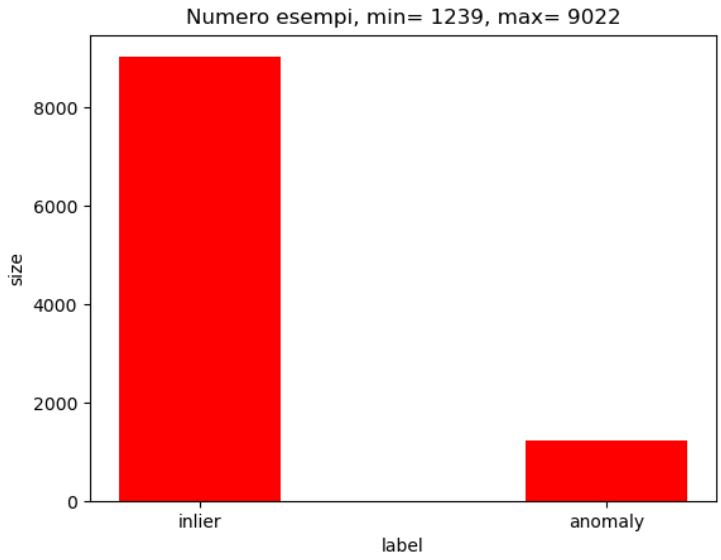
I risultati ottenuti sono questi:





## 2.3. Anomaly Detection

In questa fase sono dovuto ritornare ad una fase di preprocessing al fine di determinare un insieme di label composti da 0 in presenta del valore massimo in termini di numero di recensioni di un particolare tipo di valutazione, che rappresenterà il valore inlier, e 1 per tutti gli altri valori identificati come outlier.



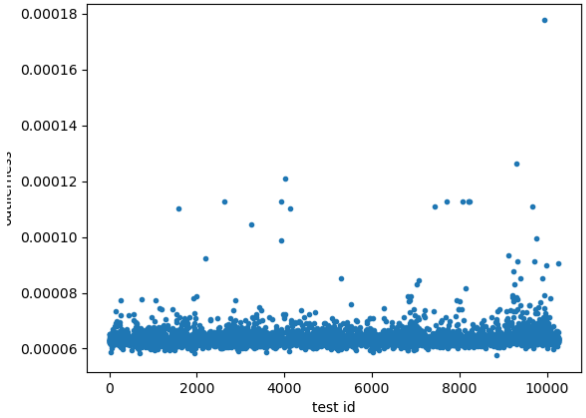

Al termine delle seguenti operazioni ho definito un trainset sia per le label che per il dataset.

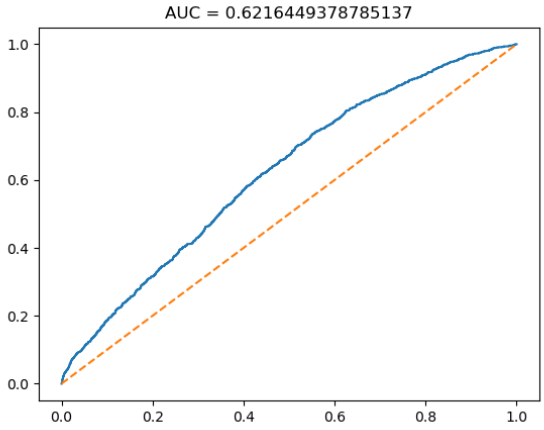
Nella fase di Anomaly Detection ho sviluppato un modello del tipo autoencode basato su rete neurale densa.

Nel modello denso ho definito prima l'encoder che prende l'input sfruttando keras, e successivamente ho mandato in altri 2 livelli densi, uno con 64 e il successivo con 32 neuroni sfruttando come funzione di attivazione la "relu", e in seguito ho definito il decoder che prende l'encoder, lo manda ad un livello denso con 32 neuroni che sfrutta come funzione di attivazione sempre la "relu" e alla fine passa il risultato all'ultimo livello denso per l'uscita in output con una dimensione pari alla stessa dell'input sfruttando questa volta la funzione sigmodea come attivatore.
Per la compilazione del modello ho utilizzato un'ottimizzazione di tipo "adam" con loss="mse", e subito dopo ho addestrato il modello su 10 epoche con un batch di 32, valutandolo in seguito nella definizione dell' andamento sulla curva di ROC e il corrispettivo valore di AUC.
Per la definizione dei numeri relativi ai neuroni nei livelli densi ho impostato questi particolari valori in

quanto attraverso esperimenti empirici ho notato come il valore di AUC tendeva a diminuire con l'aumento dei neuroni nel livello denso, e allo stesso tempo anche per quanto riguarda il numero di epoche e il corrispettivo batch ho notato come su questi particolari iperparametri ottenevo valori migliori rispetto all'utilizzo di altri di valore maggiore o minore.

I risultati ottenuti sono questi:



# 3. Funzionamento script di esecuzione

Lo script viene fornito tramite il modulo python “MAIN.py”.Nel lancio dello script tramite il comando python o python3 e ovviamente il nome dello script python ossia MAIN.py, indichiamo come:

1. Il primo parametro deve essere uno fra i seguenti:
    - “-i” o “--image” per le immagini;
    - "-t” o “--text” per i testi;
    - "-a_i” o “--all_i” per lanciare tutti gli script di immagini, ci vorrà un pò!
    - "-a_t” o “--all_t” per lanciare tutti gli script di testo, ci vorrà un pò!
    - "-h” o “--help” per informazioni sui comandi!
2. Il secondo parametro passato è il path del relativo percorso di testo o immagini.

Infine, viene richiesto, sempre se come primo parametro è stato inserito “-i” o “-t”, l'inserimento del modello da eseguire, e di questo basta o scrivere il nome dell'algoritmo (non importa se minuscolo o maiuscolo), oppure il numero corrispondente ad esso nella lista!

Ecco due esempi di esecuzione:

- Eseguendo: **python MAIN.py -i immagini-3** :

```
Hai scelto immage!

Che algoritmo vuoi lanciare?
Classificazione:
        * (1) AdaBoost
        * (2) SVM
        * Reti neurali:
                + (3) Neural network
                + (4) Convolutional Neural network
        * Stime di Densità:
                + (5) Gaussian Naive Bayes
                + (6) Nearest Neighbor

Anomaly Detection:
        * (7) Density autoencoder
```

- Eseguendo**: python MAIN.py -t test-3.xlsx :**

```
Hai scelto text!

Che algoritmo vuoi lanciare?
Classificazione:
        * (1) AdaBoost
        * (2) SVM
        * Reti neurali:
                + (3) Neural network
        * Stime di Densità:
                + (4) Nearest Neighbor

Anomaly Detection:
        * (5) Density autoencoder
```