

UmamiDetector (Luigi Daddario mat. 685195)

Lo scopo del progetto è quello di studiare la formazione del gusto nel cibo. Per fare questo mi avvalgo di un dataset trovato su kaggle, denominato “*Umami (savoriness) in food*”.

Assumiamo che il gusto che vogliamo studiare sia “umami”, e, attenendoci a quanto letto sulle info del dataset e grazie ad altre rilevanze scientifiche sappiamo che:

“Amino acids in foods have two types. The first type is amino acids that are joined to build proteins. The other type is free amino acids which are dispersed. While protein has no taste, free amino acids have a taste. Free glutamate is one of the representative umami taste substances. “

-Gonzalo Recio in Kaggle

Umami è uno dei cinque gusti fondamentali percepiti dalle cellule recetttrici specializzate presenti nel cavo orale umano. E’ stato descritto come “gradevole al palato” ed è caratteristico dei brodi e degli arrosti.

Da cosa dipende l’Umami? Come è stato anticipato prima, la formazione del gusto umami dipende da **amminoacidi liberi**. Nel nostro dataset ne troviamo 20. Il gusto umami inoltre dipende anche da due additivi alimentari, L’**inosinato** e il **Guanilato**. Come indicatore della quantità di gusto assumeremo che la molecola da considerare sia, invece, il Glutammato.

$$U(x) = fa + (IsG)$$

Aldilà dell’utilizzo corretto degli operatori dell’equazione definita sopra, parto con l’ipotesi che il gusto umami possa essere formato dall’interazione di Inosinato e Guanilato (s rappresenta l’operatore che definisce l’interazione), dagli amminoacidi liberi e da una f, che potrebbe essere una costante o un’altra funzione da scoprire.

Il progetto è stato realizzato utilizzando Python e scikit-learn, libreria per machine e deep learning. Il dataset di kaggle ha diversi problemi, primo tra tutti la mancanza di molti valori, soprattutto quelli relativi ai due additivi che, presumibilmente sono direttamente incidenti sul valore del glutammato, che abbiamo detto essere rappresentativo del gusto Umami, cioè quello che stiamo cercando.

1.0 - Schema del Progetto

Come indicato nell’immagine sotto ho deciso di dividere il progetto in più fasi, la prima, la fase di **preprocessing** è dedicata all’utilizzo di un Imputer KNN per utilizzare una metodologia per scegliere dei valori per sostituire i valori mancanti. La seconda fase è quella di **clustering**: ho utilizzato un modello k-means per poter segmentare le varie qualità di cibo. Ho immaginato che, se è vero che esiste una relazione tra amminoacidi, additivi, glutammato

e umami allora la segmentazione permetterà di far emergere i gruppi di cibi basati sulle varie qualità degli stessi, poichè ad una certa configurazione di amminoacidi e additivi (o solo additivi) corrisponderà un valore del glutammato più o meno grande. La terza parte è la parte di **classificazione**; Qui ho utilizzato un albero di Decisione per poter realizzare un modello che sia capace di predire la classe di cibi, definita durante la fase di clustering, sulla base della configurazione dei suoi componenti. La quarta fase è la fase di predizione. In questa fase ho utilizzato diversi regressori, per poi confrontare i risultati delle metriche applicate agli stessi. La quinta e sesta fase le ho chiamate Ensemble, perchè, in queste due fasi ho utilizzato un classificatore e un regressore che potessero lavorare anche in presenza di valori mancanti.

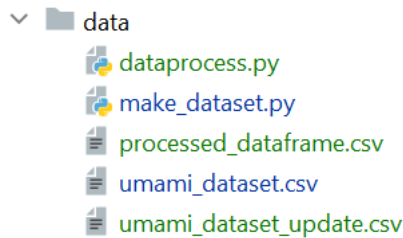


1.1 – Struttura dei files e cartelle

Ho organizzato il progetto principalmente in 3 cartelle principali:

- 1) **Data**
- 2) **Models**
- 3) **Visualization**

Data: Nella prima cartella ho collocato tutti i files che mi hanno permesso di modificare il dataset per poter utilizzare i modelli di classificazione e regressione. Per tutti i valori mancanti ho utilizzato un Imputer KNN e poi ho segmentato i dati per poter raggruppare le tipologie di cibo. In modo del tutto astratto ho immaginato che ogni cluster potesse rappresentare una certa qualità di cibo. (cluster 1: gruppo dei cibi con pochissimo umami, cluster 5: gruppo dei cibi con tanto umami)

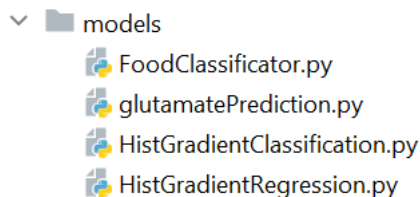


data contiene inoltre il dataset “umami_dataset.csv” e il dataset dopo che sia stato processato dall’ imputer e dopo che sia stata effettuata la clusterizzazione.

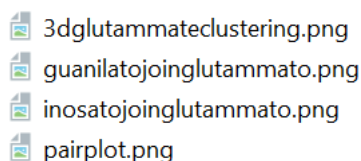
Models: Nella cartella models ho utilizzato “**FoodClassifier.py**” per i task di classificazione del cibo. Ho utilizzato come target i cluster creati precedentemente e come classificatore ho utilizzato un DecisionTree. Per quanto riguarda il file

“**glutamatePrediction.py**”, in questo file ho utilizzato diversi regressori, che poi ho confrontato. Nel powerpoint di presentazione spiego tutto nel dettaglio.

“**HistGradientClassification.py**” e “**HistGradientRegression.py**” sono rispettivamente due files in cui ho utilizzato il classificatore e il regressore con l’omonimo nome per poter lavorare con i dati raw, senza la necessità di applicare imputers.



Visualization: Nella cartella visualization vi sono semplicemente tutti i plot realizzati per l’analisi dei dati rappresentati dal dataset.



Possibili evoluzioni: L’evoluzione principale per quanto riguarda questo dataset è sicuramente relativa al miglioramento dello stesso. I risultati, soprattutto della predizione non sono molto confortanti (ma commentabili), anche se forniscono dei sintomi di giustificazione della tesi. Un dataset completo e senza troppi valori mancanti potrebbe aiutare sicuramente molto, inoltre, per ora, mi limito a dire che si potrebbe pensare di estendere la questione a tutti gli altri gusti e magari, costruito un modello completo, pensare a sistemi basati su conoscenza, capaci di ragionare vincolati ai risultati appresi durante il lavoro di machine learning.

1.2 – Struttura del dataset

Nell'immagine sopra invece abbiamo dei pairplots, che mi aiutano a capire come gli additivi e il glutammato abbiano qualcosa in comune e nonostante i problemi relativi alla mancanza di valori, posso già iniziare a capire se esiste o meno una relazione tra questi.

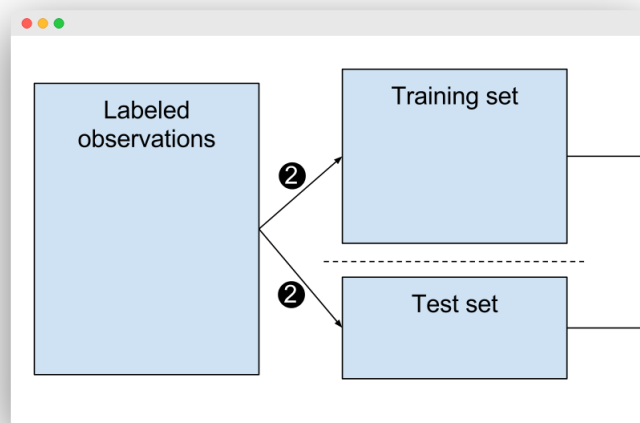
2.0 – Metriche e splitting dei dati

Prima di proseguire con le fasi del progetto, è bene specificare alcune scelte fatte in fase di utilizzo dei modelli. Dobbiamo differenziare le metriche utilizzate per il classificatore e le metriche utilizzare per i regressori. Per quanto riguarda il classificatore, ho utilizzato un metodo di una classe di sklearn, chiamato classification report, che si occupa di calcolare accuratezza, precisione, richiamo, f1 e supporto e ne mostra tutti i risultati. Per la classificazione ho utilizzato anche la matrice di confusione per capire l'errata classificazione dei dati.

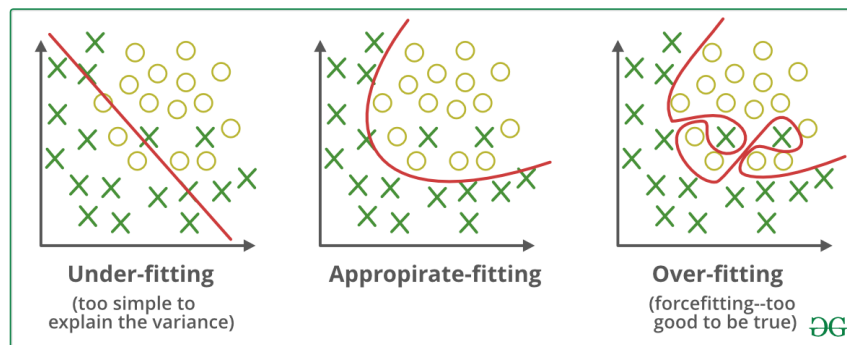
2.1 – Splitting dei dati

Quando si addestra un modello è procedura comune dividere i dati osservati in dati di training e dati di test. Perché? **Per una procedura di validazione** che ci aiuta a capire se il modello ha effettivamente imparato dai dati o se semplicemente ha "imparato a memoria" il dataset. (tanto da generalizzare)

Capire le differenze di prestazioni tra questi due dataset ci dà informazioni sullo stato, ci fa capire se siamo in **underfitting** o **overfitting**.



		errore dati di training	
		basso	alto
errore dati di test	basso	OK	underfitting
	alto	overfitting	underfitting



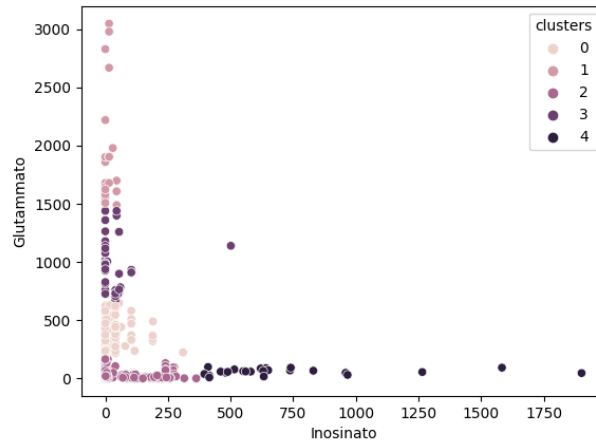
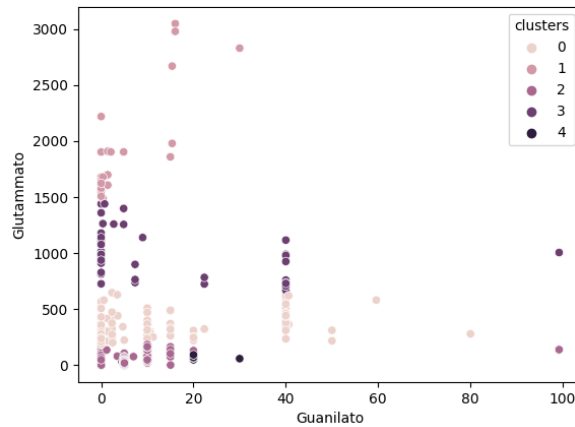
In queste tre immagini si capisce come dividere il dataset in dati di training e in dati di test ci possa aiutare a capire in quale stato ci troviamo e quale sia lo stato di apprendimento del modello.

3.0 – Fase di preprocessing e imputer KNN

A differenza di altri imputer di sklearn come SimpleImputer, che si limite a calcolare, ad esempio la media degli elementi presenti nella colonna contenente i valori mancanti, KNN, fissato il numero dei vicini, cerca di trovare le n righe che hanno i valori più vicini a quella con i valori mancanti. Da qui il termine: analisi multivariata, proprio perchè per effettuare l'operazione non considera solo una ma più features, poichè potrebbero esistere correlazioni.

3.1 – Clustering

Ho deciso di segmentare gli esempi del dataset per poter raggruppare le varie tipologie di cibo, sulla base della quantità di Umami contenuta negli stessi.



Ovviamente nei grafici precedenti abbiamo una nuvola di punti che ci dice poco sulla struttura del dataset, questo tuttavia potrebbe essere dovuto dal fatto che per i due additivi alimentari, il numero di valori mancanti è davvero elevato e rende inconsistente la struttura. Vediamo però come si distribuiscono i cluster rispetto alle quantità e tutto sommato si evidenzia una certa regolarità.

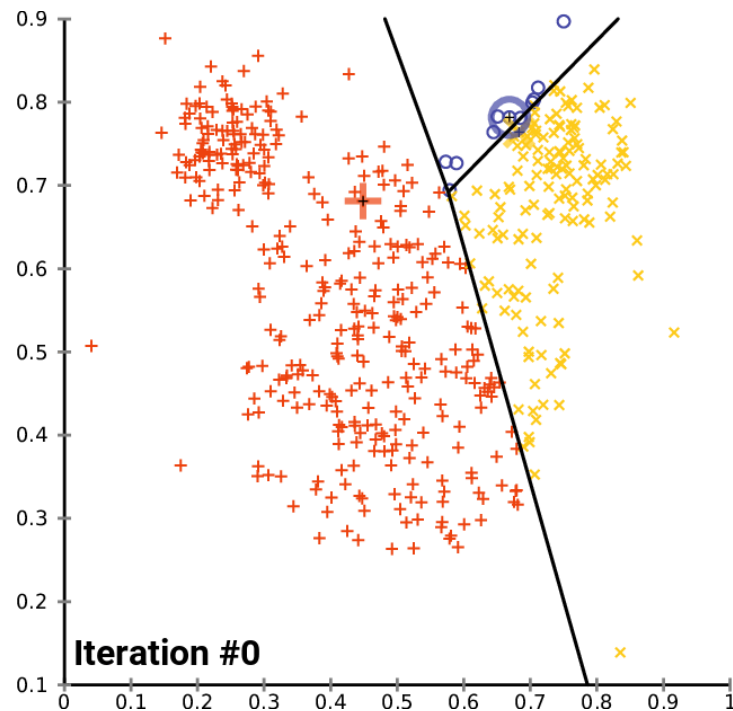
3.1.1 – Clustering K-means: come funziona?

Il clustering, utilizzando come modello K-means ha un funzionamento che schematizzo sotto:

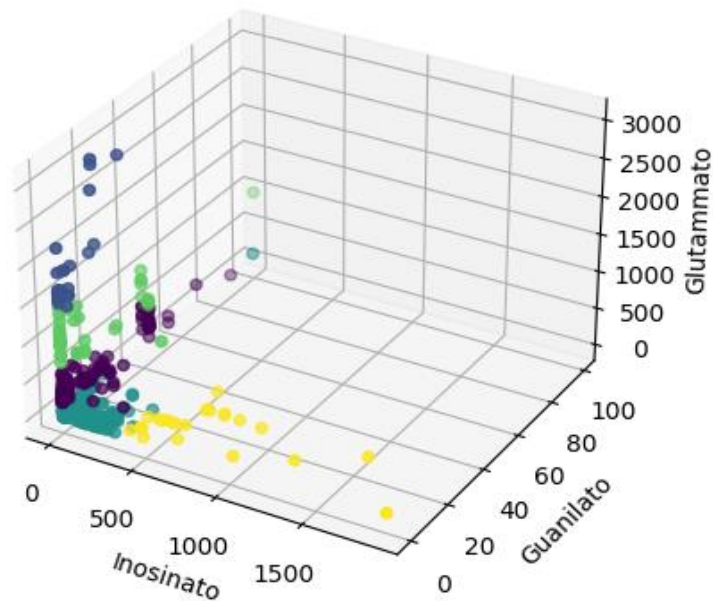
1. Input: K , *punti* $x_1 \dots x_n$
2. Imposta i centroidi $c_1 \dots c_k$

3. Ripeti finché non raggiunge la convergenza

Per ogni punto x_i deve cercare il **centroide** c_j più vicino, calcolandone la distanza, assegna quindi il punto x_i con il cluster j . Per ogni cluster $j = 1..k$ devo definire il nuovo centroide c_j come la media di tutti i punti x_i assegnati al cluster j nello step precedente.



Raggiungimento della convergenza ad ogni iterazione (immagine animata)



Valori del glutamato rispetto a Inosinato e Guanilato, utilizzando come hue del plot i cluster.

4.0 – Fase di Classificazione

Ho deciso di classificare il cibo sulla base della quantità di Umami contenuta nello stesso. In particolare ho addestrato un classificatore (DecisionTree) che predice il cluster corrispondente utilizzando come features gli amminoacidi liberi e gli additivi.

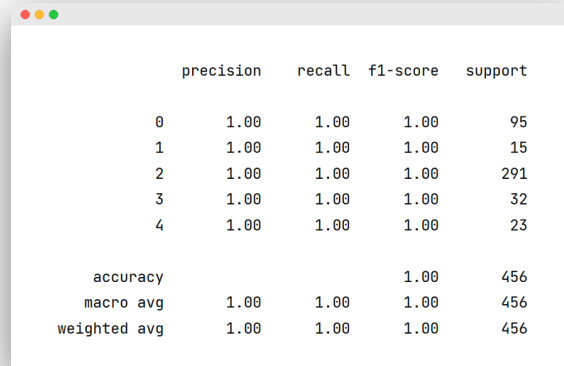
4.1 – DecisionTree: come funziona?

Un albero di decisione è un albero binario dove ogni nodo interno viene associato ad una domanda o condizione, imposta ad una feature, che ha il compito di splittare i dati fino a raggiungere foglie che indicano la categoria associata alla decisione.

4.2 Metriche decision tree

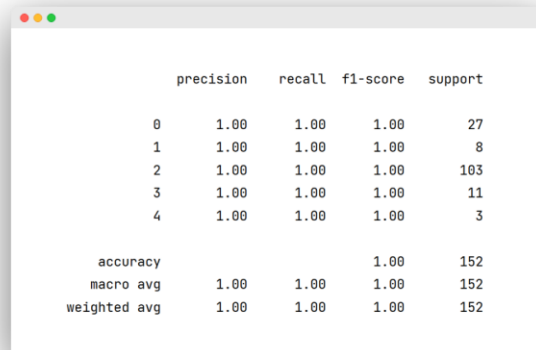
Una buona condizione divide gli esempi di classi eterogenee in sottoinsiemi abbastanza omogenei per evitare di avere troppa varianza in ogni strato. Per fare in modo che tutto questo sia lecito è necessario fornire una metrica corretta.

Considero X come un sottoinsieme di campioni di un particolare insieme di addestramento formato da m possibili classi. X è una variabile aleatoria. Posso quindi associare ad ogni x_i la distribuzione di probabilità $p(x_i) = p_i$. P_i = frequenza relativa della classe i all'interno di X . Dopo una piccola digressione teorica, ma comunque poi commentabile, adesso vediamo i risultati della classificazione, generati dal report utilizzato.



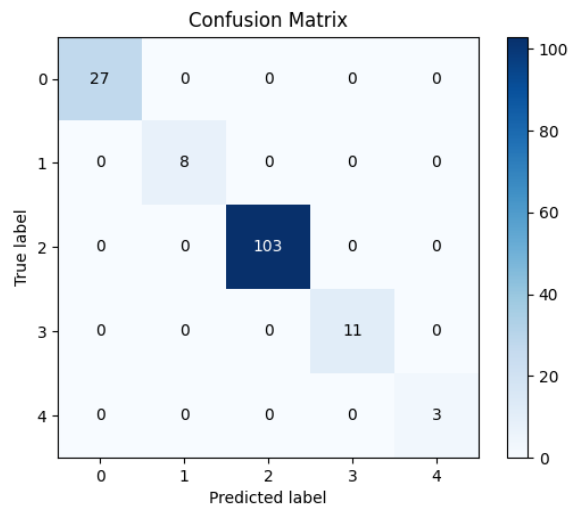
	precision	recall	f1-score	support
0	1.00	1.00	1.00	95
1	1.00	1.00	1.00	15
2	1.00	1.00	1.00	291
3	1.00	1.00	1.00	32
4	1.00	1.00	1.00	23
accuracy			1.00	456
macro avg	1.00	1.00	1.00	456
weighted avg	1.00	1.00	1.00	456

Dati di training

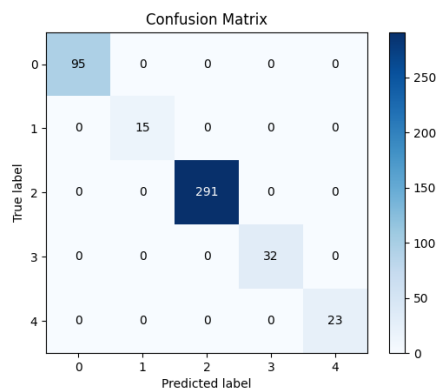


	precision	recall	f1-score	support
0	1.00	1.00	1.00	27
1	1.00	1.00	1.00	8
2	1.00	1.00	1.00	103
3	1.00	1.00	1.00	11
4	1.00	1.00	1.00	3
accuracy			1.00	152
macro avg	1.00	1.00	1.00	152
weighted avg	1.00	1.00	1.00	152

Dati di test



Matrice di confusione: train



Matrice di confusione: test

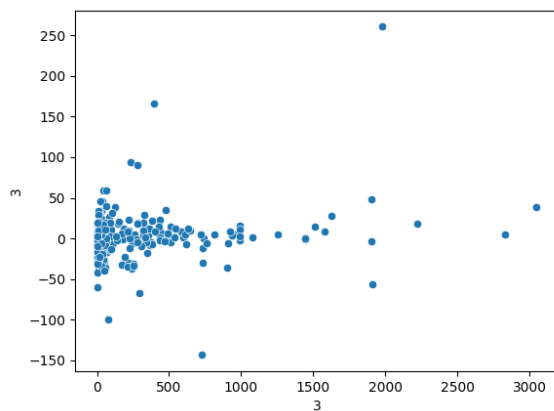
Come si evince dal report e dalle matrici di confusione, è evidente che nella classificazione non abbiamo problemi e anzi i risultati ottenuti sono ottimi. Il problema però è che il dataset sul quale lavora il modello non è preciso, questo anche perchè abbiamo utilizzato un Imputer per i valori mancanti, quindi è probabile che questa grande accuratezza sia dovuta dal fatto che il modello costruito è fatto sì di valori precisi, ma in ogni caso processati e probabilmene non veritieri.

5.0 – Regressione

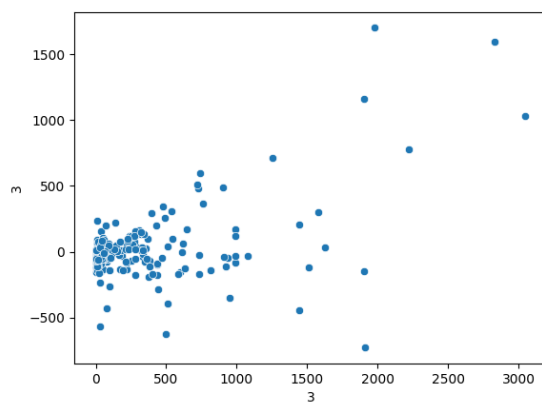
Ho deciso di predire il valore del glutammato utilizzando come features tutti i valori di tutti gli amminoacidi liberi e degli additivi alimentari. Ho utilizzato 4 regressori differenti, per poterli confrontare e per poter confrontare il pattern rappresentato (o meno) dagli errori. Come regressori ho utilizzato un regressore KNN, BayesianRidge, Regressione Lineare e MLPRegressor, che è un modello neurale.

5.1 – Regressione: metriche e risultati

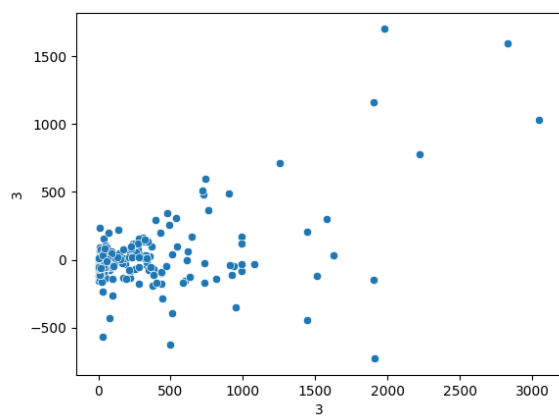
Per i regressori ho utilizzato una metrica, la **MAE**. Mean Absolute Error o L1 Loss confronta i dati di training con i dati predetti. In particolare calcola la distanza tra il valore da predire e il valore predetto (in quanto il valore della distanza è chiaramente non negativo)



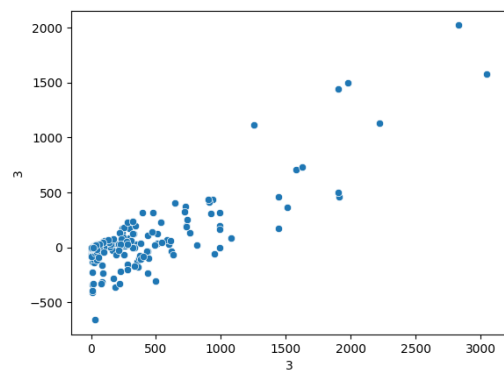
MAE KNN



MAE BayesiaRidge



MAE LinearRegression



MAE: MLPRegressor

```

Train KNN 104.2693947368421 Test KNN 130.7199769736842
Train Linear Regression 109.06877619625084 Test Linear Regression 138.93799701549597
Train Bayesian 107.4466751548129 Test Bayesian 128.32195222946436
Train Neural Network 15.06878953116414 Test Neural Network 189.8140492814566

```

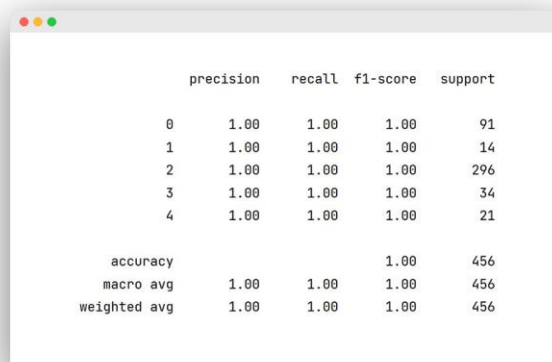
Risultati regressori

I risultati dei regressori non sono certamente confortanti, perchè come vediamo sono molto alti. Di interesse specifico però è il risultato del modello neurale in cui vediamo una differenza significativa tra train, che ha un mae molto basso e test, che ha un mae molto alto, questo potrebbe essere sintomo di **overfitting**. In realtà però le nuvole di punti rappresentate dai grafici relativi gli errori dei vari regressori sono in un certo senso significativi, perchè tranne in alcuni casi, troviamo un minimo di *'pattern'* nel grafico, questi punti sono in qualche modo posizionati con un criterio. Questo potrebbe essere sintomo di una possibile evoluzione della tesi, perchè se gli errori si distribuiscono con un certo pattern, evidentemente i valori assegnati sul dataset non solo casuali.

6.0 – HistGradientBoostingClassifier/Regressor

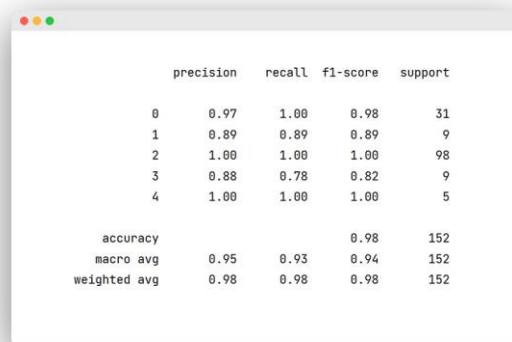
Vista la natura del dataset e visto il lavoro svolto per i valori mancanti, ho deciso di testare dei modelli che potessero lavorare anche con valori mancanti, questo per poter confrontare i risultati con il dataset processato e le applicazione dei classificatori e regressori.

HistGradientBoostingClassifier/Regressor sono dei modelli capaci di effettuare predizioni anche in caso di valori mancanti. Durante l'addestramento, il modello capisce se gli esempi con valori mancanti appartengono al figlio sinistro o destro dell'albero di decisione, in base al gain potenziale. Se durante l'addestramento non sono stati rilevati valori mancanti per una determinata feature, i campioni con valori mancanti vengono mappati sul figlio che ha il maggior numero di esempi.



	precision	recall	f1-score	support
0	1.00	1.00	1.00	91
1	1.00	1.00	1.00	14
2	1.00	1.00	1.00	296
3	1.00	1.00	1.00	34
4	1.00	1.00	1.00	21
accuracy			1.00	456
macro avg	1.00	1.00	1.00	456
weighted avg	1.00	1.00	1.00	456

Report train (classificazione)



	precision	recall	f1-score	support
0	0.97	1.00	0.98	31
1	0.89	0.89	0.89	9
2	1.00	1.00	1.00	98
3	0.88	0.78	0.82	9
4	1.00	1.00	1.00	5
accuracy			0.98	152
macro avg	0.95	0.93	0.94	152
weighted avg	0.98	0.98	0.98	152

Report Test (Regressione)

7.0 – Possibili evoluzioni del progetto e considerazioni

L'evoluzione principale per quanto riguarda questo dataset è sicuramente relativa al miglioramento dello stesso (in termini di valori mancanti). Ho scelto volutamente di non effettuare scaling sui dati, questo semplicemente per non applicare troppe modifiche al dataset che comunque non risulta preciso. I risultati, soprattutto della predizione non sono molto confortanti, anche se forniscono dei sintomi di giustificazione della tesi. Un dataset completo e senza troppi valori mancanti potrebbe aiutare sicuramente molto. Inoltre, per ora, mi limito a dire che si potrebbe pensare di estendere la questione a tutti gli altri gusti e magari, costruito un modello completo, pensare a sistemi basati su conoscenza, capaci di ragionare vincolati ai risultati appresi durante il lavoro di machine learning.