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 la formazione di questo gusto dipenda da amminoacidi liberi e da additivi alimentari. Nel nostro dataset troviamo 20 amminoacidi liberi e 2 additivi alimentari.

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 alimentari, che, presumibilmente sono direttamente incidenti sul valore del glutammato, che da ora in poi assumeremo come indicatore del gusto umami.

1.0 – Schema del progetto e prime considerazioni

Come indicato nell'immagine in basso ho deciso di dividere il progetto in più fasi. La prima la fase, quella di preprocessing, è dedicata all'utilizzo di un **imputer KNN** per utilizzare una metodologia per scegliere dei valori per sostituire quelli mancanti.

1.0.1 Apprendimento non supervisionato

La seconda fase è quella di clustering: ho utilizzato un algoritmo di analisi dei segmenti partizionale, **k-means**, per poter dividere in segmenti i dati. Per scegliere il numero k di segmenti ho utilizzato il metodo di **elbow**.

1.0.2 Apprendimento supervisionato

La terza fase è in realtà fittizia, l'ho inclusa nel progetto per dimostrare che il problema considerato non è di classificazione ma di regressione.

La quarta fase è di predizione, in questa fase l'obiettivo è quello di predire il valore del glutammato (y) sulla base delle quantità di amminoacidi e additivi alimentari (features)

1.0.2 Valutazioni

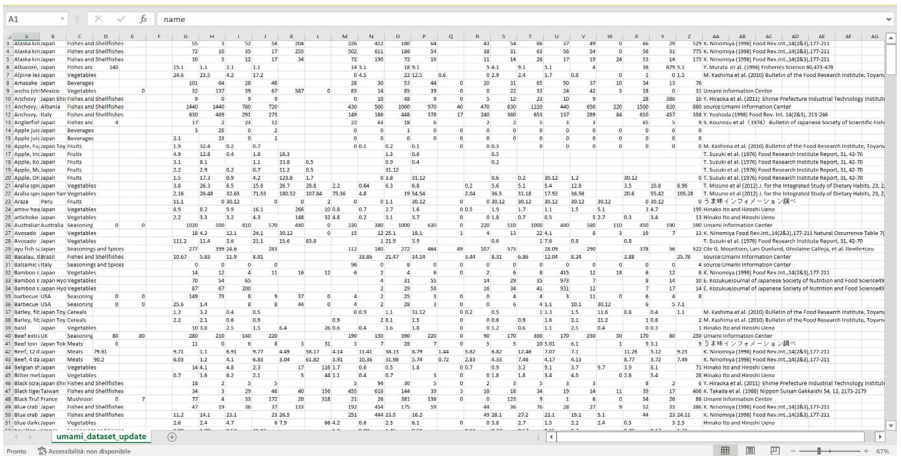
L'ultima fase è quella delle valutazioni, che saranno documentate in questa sede.

1.1 Struttura dei files e cartelle

Ho organizzato il progetto in tre cartelle principali:

1) **Data**: files per il data processing, utile per organizzare i dati in modo tale che gli stessi potessero essere più facilmente interpretabili dai modelli.
2) **Models**: files con i vari metodi di addestramento e di errori.
3) **Visualization**: Plot dei dati rappresentati dal dataset.

1.2 Struttura del dataset



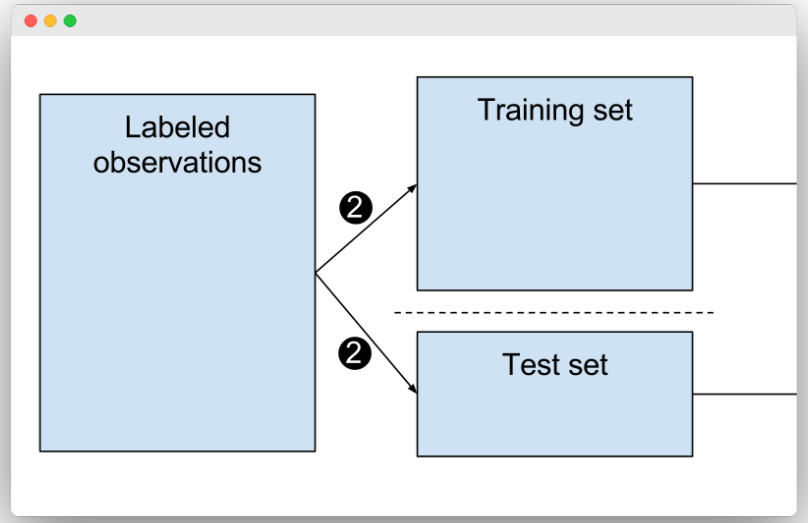
2.0 Metriche e splitting dei dati

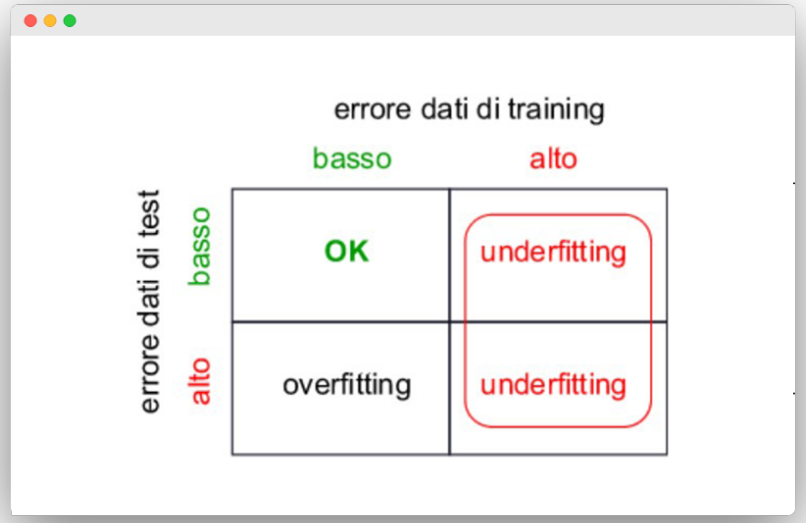
Prima di proseguire con le fasi del progetto, vorrei specificare alcune scelte. Dobbiamo differenziare le metriche utilizzate per il classificatore e le metriche utilizzate per i regressori. Per quanto riguarda il classificatore, ho utilizzato un metodo di una classe di *sklearn*, chiamato *classification report*, che si occupa di calcolare accuratezza, precisione, richiamo, f1 e supporto (con 5 run) e ne mostra tutti i risultati. Per i regressori ho utilizzato come metrica la **MAE**, realizzando una funzione che mi permettesse di effettuare 10 run per poi calcolare la media degli errori.

2.1 – Splitting dei dati

Ho diviso i dati osservati in dati di training e dati di test, questo per una procedura di valutazione che mi ha aiutato a capire se il modello avesse effettivamente imparato dai dati o se semplicemente avesse “imparato a memoria” il dataset, tanto da poter generalizzare.

Capire le differenze di prestazioni tra training set e test set ci dà informazioni sullo stato e ci fa capire se siamo in casi di **underfitting** o **overfitting**.





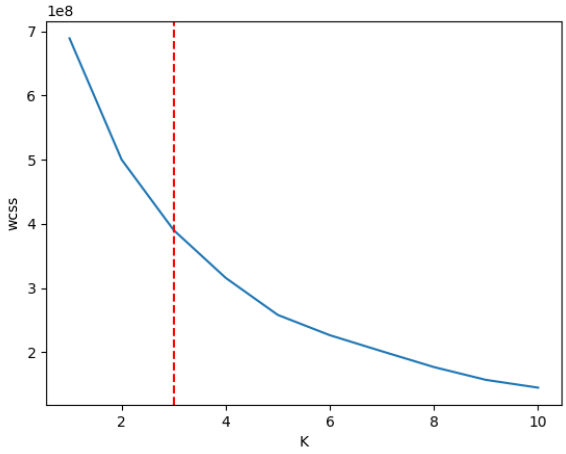
3.0 – Fase di preprocessing e imputer KNN

A differenza di altri imputers di sklearn come *SimpleImputer*, che si limita a calcolare, ad esempio, la media degli elementi presenti nella colonna contenente i valori mancanti, KNN Imputer, fissato il numero di vicini, cerca di trovare le k righe che hanno i valori più vicini a quella con i valori mancanti. Da qui il termine di **analisi multivariata**, proprio perché per effettuare l'operazione non considera solo la feature con il valore mancante ma anche tutte le altre indicate, poiché potrebbero esistere correlazioni tra le stesse.

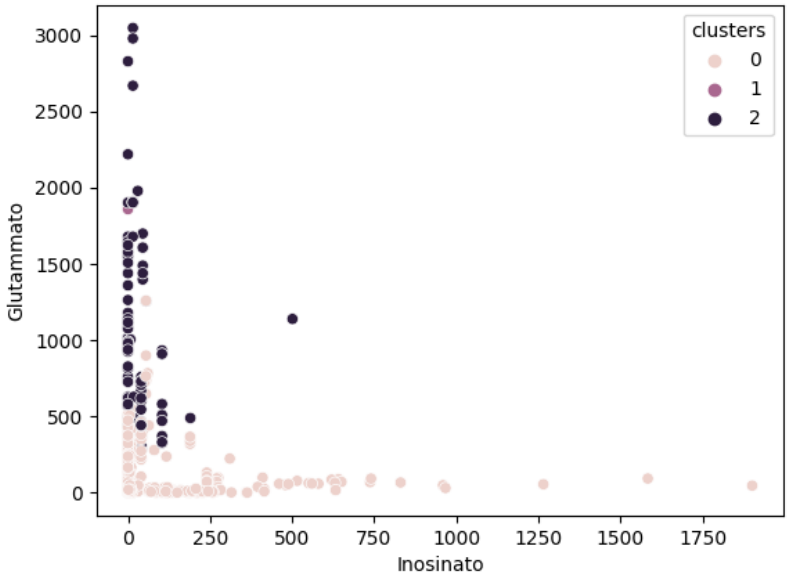
3.1 – Clustering

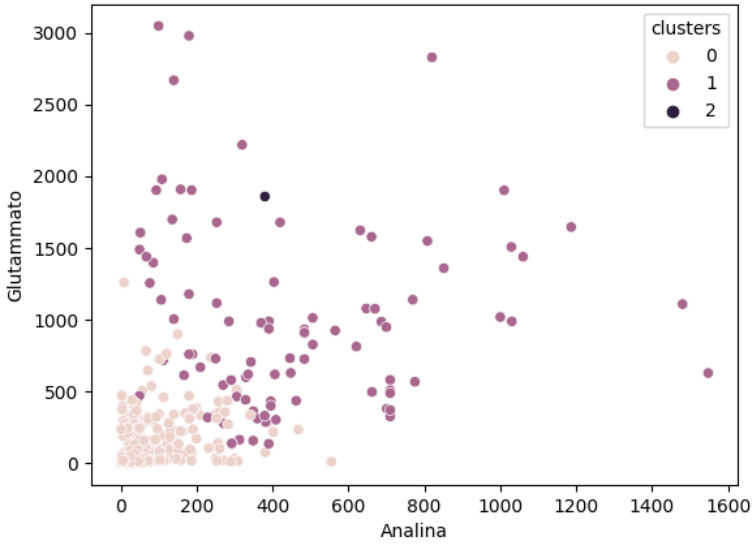
Ho segmentato gli esempi del dataset per poter raggruppare le varie tipologie di cibo sulla base delle caratteristiche del campione. Per segmentare questi dati ho utilizzato un algoritmo di apprendimento non supervisionato, k-means, che suddivide un insieme di oggetti in k gruppi. Si basa sui *centroidi*, che sono dei punti appartenenti allo spazio delle features e che media le distanze tra tutti i dati appartenenti al cluster associato. Come facciamo a scegliere il numero di k vicini?

Io ho utilizzato il metodo di **elbow**. Molto semplicemente ho iterato k-means per diversi valori di k ([1,10]) e ad ogni iterazione si calcola la somma delle distanze al quadrato tra ogni centroide ed i punti del cluster.



Plottando i k valori e i valori della somma delle distanze al quadrato ho ottenuto il grafico sopra. La linea tratteggiata inoltre, evidenzia il cosiddetto “gomito”, che leggendo il grafico da destra a sinistra mi ha permesso di capire che i cluster ottimali sono 3.





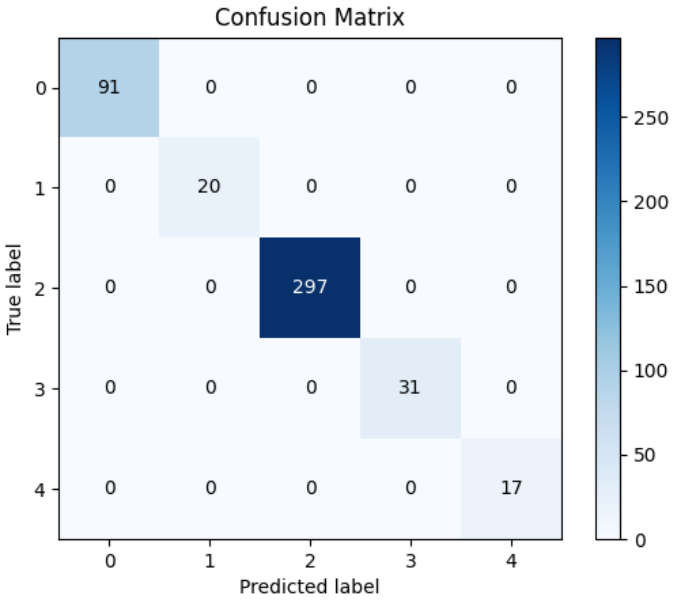
Abbiamo detto che il valore del glutammato dipende da 20 amminoacidi liberi e da due additivi alimentari: l'inosinato e il guanilato. Osservando la distribuzione del glutammato rispetto all'inosinato ci rendiamo conto che la distribuzione non ci dice granché sulla loro relazione e anzi ci porta completamente fuori strada. Non è possibile sicuramente evidenziare una distribuzione proporzionale e in realtà neanche una relazione tra di essi.

Leggermente differente è la situazione prendendo in considerazione *l'analina*, che è uno dei 20 amminoacidi liberi che abbiamo a disposizione. Qui abbiamo diversi **outliers**, tuttavia si riesce ad evidenziare una separazione netta, soprattutto tra i cluster 0 e 1, che sembrano distribuirsi diversamente all'**aumentare** dei valori di x e y.

4.0 – Classificazione

In realtà la classificazione ha poco senso a monte, questo perché è evidente che sia difficile determinare delle classi, non sappiamo nulla a riguardo, ipotizziamo solo che ci siano relazioni numeriche tra il glutammato, amminoacidi e additivi, ma di fatto non abbiamo nessuna certezza. Pur ipotizzando che i gruppi determinati da k-means possano differenziare le classi di cibo nel senso di diverse qualità, è chiaro che, classificarli sulla base degli stessi, in questa sede, non ha senso, perché chiaramente seguirà i **pattern** evidenziati precedentemente. Infatti, i risultati sono i seguenti:

	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



I risultati della matrice di confusione e del classification report sono perfetti, situazione molto improbabile anche considerando la natura del dataset. Questo è probabilmente sintomo del fatto che il problema analizzato non è un problema di classificazione ma bensì di regressione.

5.0 – Regressione

La fase di regressione è probabilmente la fase più importante del progetto, perché da qui ho cercato di capire se effettivamente le ipotesi potessero essere giustificate o meno. Ho utilizzato tre regressori differenti, per poterne confrontare le performances e per poterne evidenziare le differenze. Ho utilizzato: KNN, Regressione Lineare e un modello neurale. In realtà però ho addestrato i modelli due volte, una scalando i dati e una senza scalare gli stessi. Per quanto riguarda KNN,

come metodologia per scegliere il numero di vicini K, ho semplicemente calcolato $k = sqrt(n)$ dove n è il numero di cluster.

Perché ho scalato i dati?

I domini delle features considerati sono molto diversi, ad esempio il dominio del glutammato è significativamente più esteso rispetto a quello del guanilato, un additivo alimentare. E chiaramente questo non è il solo caso, anzi i domini sono sempre molto diversi.

Ho scalato i dati con **MinMaxScaler**, che in maniera molto semplice scala il valore affinché lo stesso possa essere compreso tra 0 e 1. Portando tutti i valori in questo range, modelli che lavorano sulla distanza, come KNN, ne dovrebbero risentire e dovrebbero migliorare le performances.

Dopo aver usato i metodi di sklearn per poter addestrare i tre modelli, dopo 10 run su dati di training e dati di test per ognuno i risultati sono i seguenti:

```
KNN train (10 run): 54.843470394736855 KNN test (10 run): 111.84597039473684
Linear train (10 run): 109.95911963972864 Linear test (10 run): 128.28981304751144
Neural networks train (10 run): 87.20243845593333 Neural networks test (10 run): 111.55928125084526
```

MAE su trainingset non scalato

```
KNN train (10 run): 92.94380482456141 KNN test (10 run): 77.32411184210525
Linear train (10 run): 150.5847449910461 Linear test (10 run): 130.14717092675818
Neural networks train (10 run): 208.83430463398093 Neural networks test (10 run): 160.03568171493228
```

MAE su trainingset scalato

l'errore sui dati di test dopo aver scalato i dati, per quanto riguarda il KNN, è diminuito significativamente. L'errore è comunque alto, ma questo potrebbe essere stato causato da diversi problemi, probabilmente il motivo principale è che gran parte dei dati sono stati imputati dall'imputer; quindi, hanno un valore molto basso dal punto di vista statistico e possono aver portato fuori strada il modello.

Ma in che caso ci troviamo?

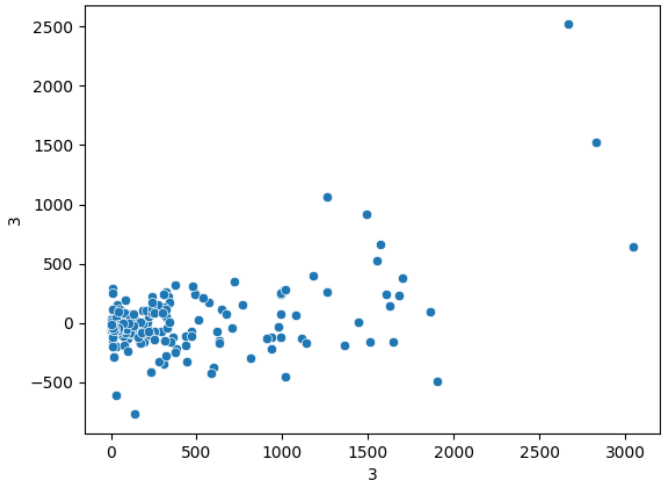
È difficile stabilire se ci troviamo in **underfitting** o in **overfitting**, probabilmente non scalando i dati abbiamo un caso di **overfitting**, perché è vero che entrambi gli errori sono alti, ma sul training sono quasi al 50% inferiori. Dopo aver scalato i dati invece, l'errore sui dati di training tendenzialmente aumenta mentre sui dati di test diminuisce; forse in questo caso ci troviamo in **underfitting**, ed effettivamente avrebbe anche più senso che il modello possa non riuscire a capire correttamente come si comportano i dati, vista la scarsa quantità.

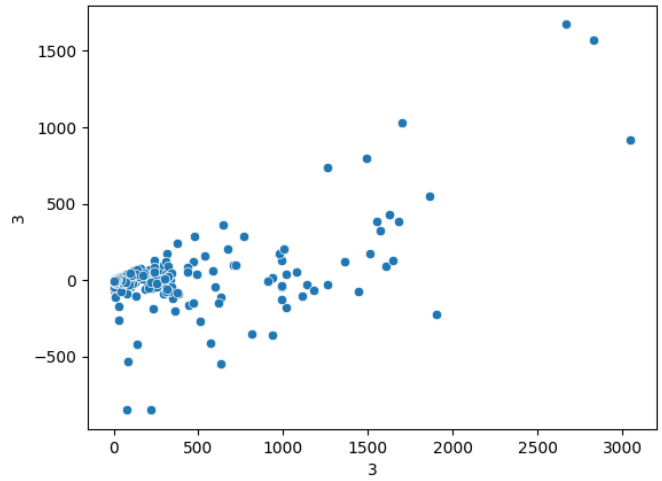
Ho pensato anche di tentare con la tecnica della **k-fold cross validation**, dividendo il dataset in k porzioni per poi iterativamente utilizzarne x per il train e y per il test, ma come già anticipato abbiamo pochi dati presumibilmente validi su cui lavorare.

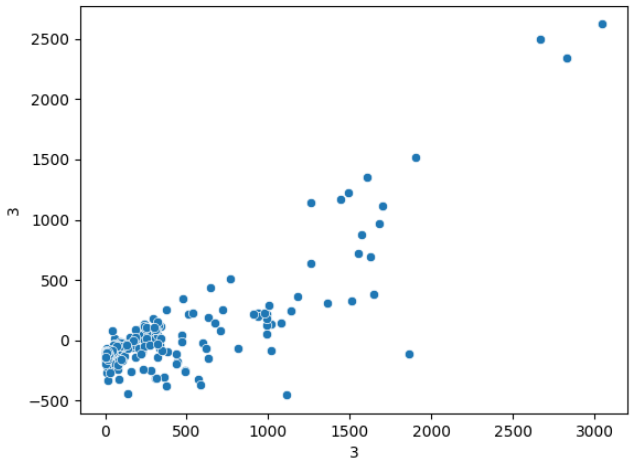
5.0 – Valutazione

In realtà parte della valutazione è stata fatta nei paragrafi precedenti. Vorrei specificare che per la stessa ho preso in considerazione KNN perché è l'algoritmo che mi ha permesso meglio di effettuare considerazioni e che ha dato sintomi di miglioramento con le operazioni fatte.

Inoltre, analizzando i grafici degli errori gli stessi sono in un certo senso significativi, perché tranne nei casi di *outliers*, troviamo una sorta di ***"pattern"*** nella distribuzione: questi punti sono posizionati con un criterio. Questo potrebbe essere sintomo di una possibile evoluzione della tesi, perché se gli errori si distribuiscono in un certo modo, preciso, evidentemente i valori assegnati sul dataset non sono del tutto casuali.







Ho inoltre effettuato delle predizioni con dei dati ottenendo per ognuna il valore del glutammato corrispondente, utilizzando i modelli addestrati con i dati scalati. Alterando in modo **casuale** i valori degli additivi e dei vari amminoacidi, il valore del glutammato viene modificato: nel caso degli additivi in realtà, a bassi valori degli stessi, corrispondono alti valori di glutammato, ad alti invece, corrispondono valori del glutammato significativamente più bassi, ma non nulli. Aumentando le concentrazioni degli amminoacidi invece, in generale aumenta sempre anche quella del glutammato.

Visti gli errori elevati però in realtà questi test hanno poco senso, poiché sono sicuramente molto lontani dalla realtà. E' evidente la relazione tra questi elementi, ma non è possibile con un numero di dati cosi poco elevato, cercare di dimostrare formalmente la tesi, che comunque ha dato dei piccoli risultati.