Università degli studi di Salerno

Dipartimento di Informatica

***Corso di Laurea in Informatica***

******

***FONDAMENTI DI INTELLIGENZA ARTIFICIALE***

***“Invendum”***

**Studenti:**

##### Nome Matricola

Luigi De Chiara 0512109483

Alex Ferrara 0512106300

## 

**Docente:**

Fabio Palomba

*Anno Accademico: 2021/22*

INDICE

[1. Obiettivo del progetto 2](#_Toc68874152)

Invendum

# Introduzione

Invendum è una innovativa strategia di marketing realizzata sotto forma di plugin. Nasce dall’esigenza delle aziende di aumentare le vendite di prodotti meno gettonati, che sono sempre stati la causa delle principali problematiche economiche.

Invendum verrà utilizzato da piattaforme “e-commerce”.

# Introduzione-Agente:

L’obiettivo del progetto è quello di realizzare un’agente che sia in grado di suggerire all’utente prodotti, mediante acquisti e recensioni effettuati da altri utenti. Tale agente cercherà di aumentare le vendite dei prodotti meno acquistati, cosi da ridimensionare le quantità dei prodotti invenduti.

**2.2 Specifica PEAS**

|  |  |
| --- | --- |
| PEAS | |
| Performance | La capacità dell’agente di riuscire a indicare al cliente il giusto prodotto invenduto di una determinata categoria in base alla storicità dei sui ordini. |
| Environment | L’ambiente dove opera l’agente è rappresentato dalla pagina dell’articolo del sito.  L’ambiente è:   * **Completamente osservabile**: abbiamo una panoramica completa delle informazioni relative agli utenti e ai prodotti. * **Stocastico:** poiché l’ambiente non verrà condizionato dalle azioni dell’agente. * **Episodico**: poiché la funzione dell’agente è richiamata nel momento in cui viene visualizzato una scheda articolo. * **Dinamico**: poiché un utente potrebbe variare le categorie di prodotti acquistati. * **Discreto**: in quanto il numero di schede articolo è numericamente limitato. * **Singolo**: in quanto nella pagina dell’articolo le azioni avverranno da parte di un unico agente. |
| Actuators | Gli attuatori del nostro agente sono identificati dagli articoli acquistati dagli utenti. Avremo cosi un box “Articoli Suggeriti” all’interno della pagina dell’articolo. |
| Sensors | I sensori dell’agente consistono nell’accesso alla pagina dell’articolo. |

**2.3 Analisi del problema**

Il problema proposto avrebbe potuto essere risolto utilizzando un algoritmo che controllasse gli acquisti passati dell’utente.

Questa soluzione avrebbe prodotto risultati inefficienti:

* In quanto lo storico degli ordini, avrebbe consigliato solo prodotti non acquistati dagli utenti.
* Un classico algoritmo non sarebbe risultato dinamico alla scelta di un articolo da consigliare, in quanto banale e privo di informazioni specifiche.

Abbiamo deciso di utilizzare tecniche di **machine** learning per avere un’efficienza ed accuratezza maggiore del problema.

Analizzando tale problema abbiamo verificato che si trattasse di un problema di apprendimento non supervisionato, per questo motivo abbiamo deciso di applicare un algoritmo di **clustering** non avendo a disposizione etichette da assegnare ai dati.

**3 Raccolta, analisi ed elaborazione dei dati**

**3.1 Scelta del dataset**

In base ad un’analisi approfondita del nostro problema abbiamo deciso di prelevare un dataset disponibile online e di modellarlo in base alle nostre esigenze, poiché crearne uno da zero avrebbe causato alcune problematiche, come scarsità e inconsistenza dei dati.

**3.2 Analisi del dataset**

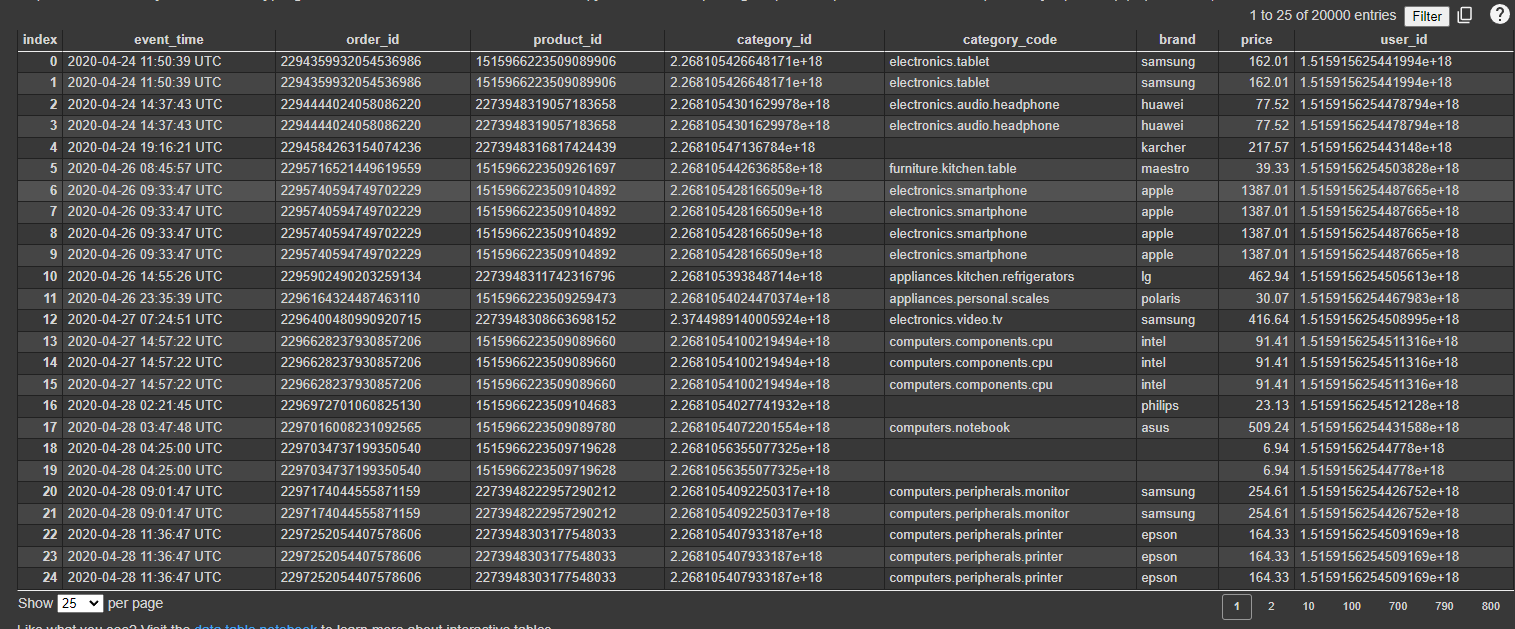
Il dataset in questione è stato accuratamente scelto e scaricato dalla piattaforma <https://www.kaggle.com/datasets>, tale dataset riporta dettagli sulle transazioni effettuate dai clienti.

Abbiamo deciso di sceglierlo pur non avendo moltissimi features, poiché per lo scopo del nostro progetto erano irrilevanti.

**3.3 Formattazione dei dati**

Il primo passo di formattazione dei dati è stato quello di effettuare:

* un taglio verticale per eliminare la colonna “event\_time” non utile alla fine della predizione.
* Un taglio orizzontale con “df.dropna(inplace=True)” in modo da eliminare tutte le righe con campi NULL.



Il secondo passo è stato quello di ridurre la dimensione del nostro dataset applicando l’agente solo su un sottogruppo di categorie, poiché gestire tutte le categorie con i relativi record sarebbe stato impensabile, pertanto si è optato di considerare circa trenta categorie:

"SELECT  category\_code,count(category\_code) from categoria where category\_code like 'computers.%'or category\_code like 'electronics.%' group by category\_code"

df=pd.read\_csv(r'/datasetcompleto.csv',sep=',',index\_col=['order\_id'],usecols=['order\_id','product\_id','category\_id','category\_code','brand','price','user\_id'])

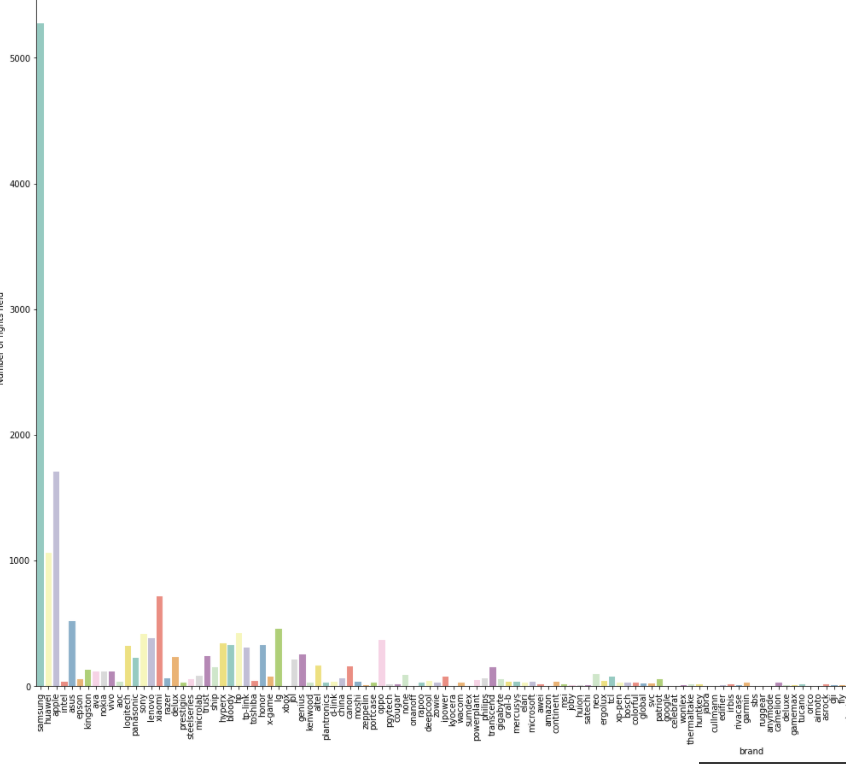
q="select \* from df where category\_code like 'computers.%' or  category\_code like 'electronics.%'"

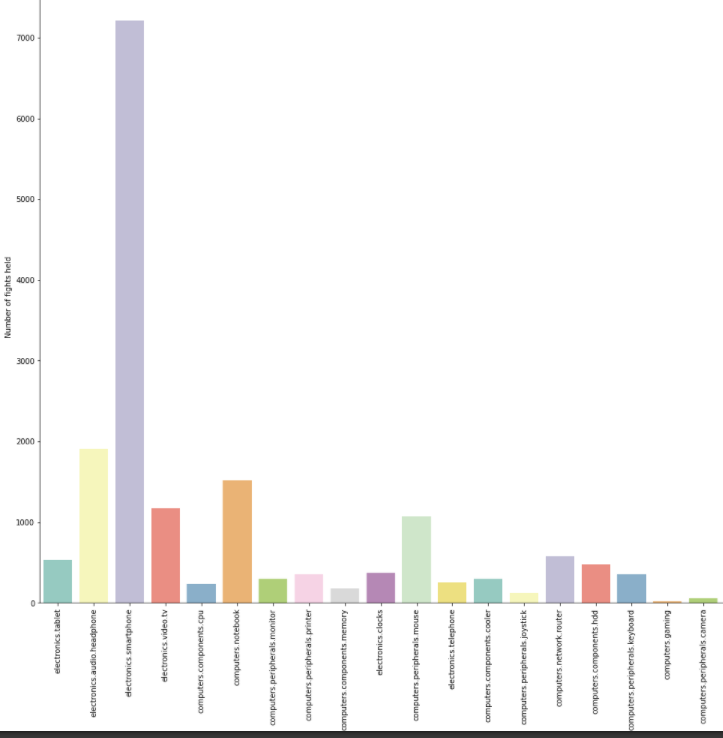
df=sqldf(q,globals())

df.to\_csv('/datasetModificato.csv')

Effettuando un’analisi accurata dei dati, ci siamo resi conto che la categoria “smartphone” avente come brand “Samsung” destabilizzava l’insieme dei dati poiché aveva valori troppo alti, pertanto si è optato di effettuare un taglio orizzontale.

Di seguito vengono riportate le analisi delle categorie e dei brand.





**4 Algoritmo di clustering**

Dopo aver formattato i dati, ci siamo posti il problema della scelta dell’algoritmo di Clustering da utilizzare. L’indecisione dovuta alle nostre conoscenze era quella di provare sia con:

* DBscan: adatto a dataset densi
* K-Means

**4.1** **DBscan (**Density-Based Spatial Clustering of Applications with Noise)

Il numero di cluster non è necessario fornirlo in quanto sarà l’algoritmo stesso a scoprire quanti cluster esistono in base a due iperparametri che verranno forniti:

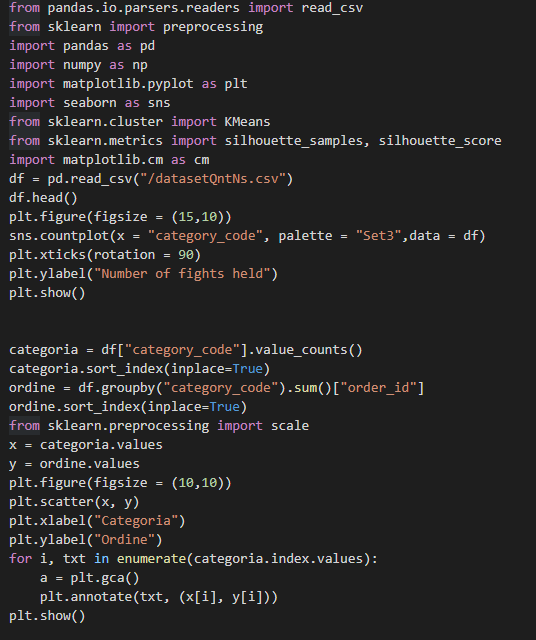
* epsilon o eps (ε): la distanza minima che devono avere due dati in ingresso per essere considerati “vicini”
* minSamples: numero minimo di dati in ingresso, “vicini” tra loro, richiesto per formare un cluster.

**4.2 K-Means**

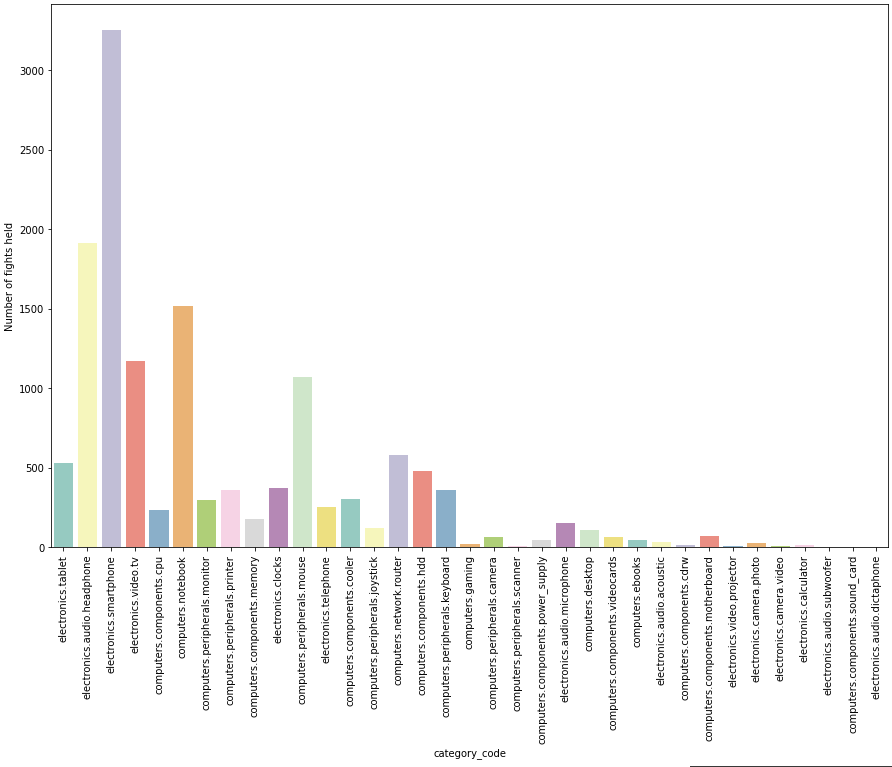
un algoritmo di analisi dei gruppi partizionale che permette di suddividere un insieme di oggetti in k gruppi sulla base dei loro attributi.

Poiché dovevamo lavorare su dati non estremamente densi, ci siamo affidati all’algoritmo K-Means.

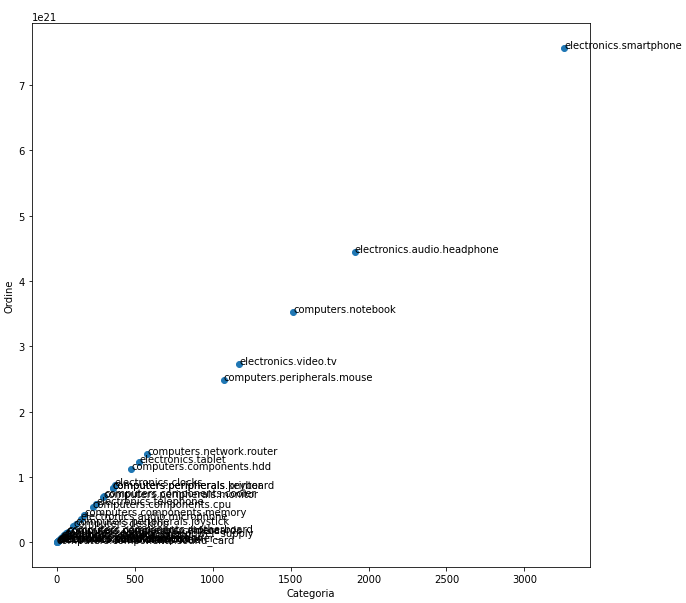
Il primo passo è stato quello di caricare il dataset appena modificato, abbiamo riutilizzato i grafi per verificare che i dati fossero perfettamente bilanciati (dati non troppo distanti fra loro). Abbiamo messo in correlazione le features della category\_code e dell’order\_id per effettuare la predizione delle quantità vendute di ogni singola categoria.



**Analisi delle categorie:**

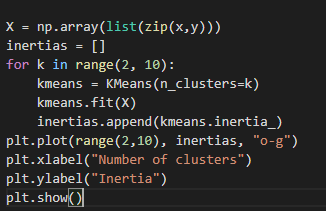


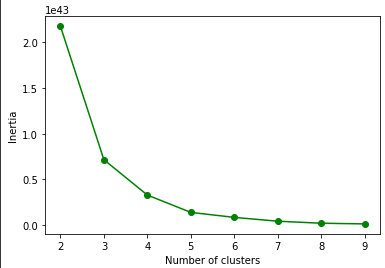
**Grafico a dispersione** basato sul numero di acquisti e sugli ordini effettuati dai clienti.



**Punto di gomito**

Utilizziamo il punto di gomito per valutare il miglior numero di cluster per l’algoritmo k-means, pertanto andremo a creare un grafo, che ci permetterà di visionare i valori candidati del parametro k rispetto alla somma degli errori quadratici ottenuti dall’algoritmo che genererà i k cluster.





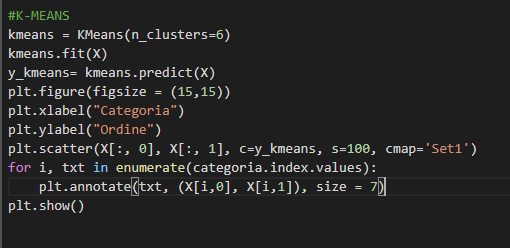
Dal punto di gomito si evince che il numero ideale di cluster da utilizzare è pari a 6.

Dopo aver assegnato tale valore a k(n\_clusters) possiamo implementare l’algoritmo precedentemente scelto.

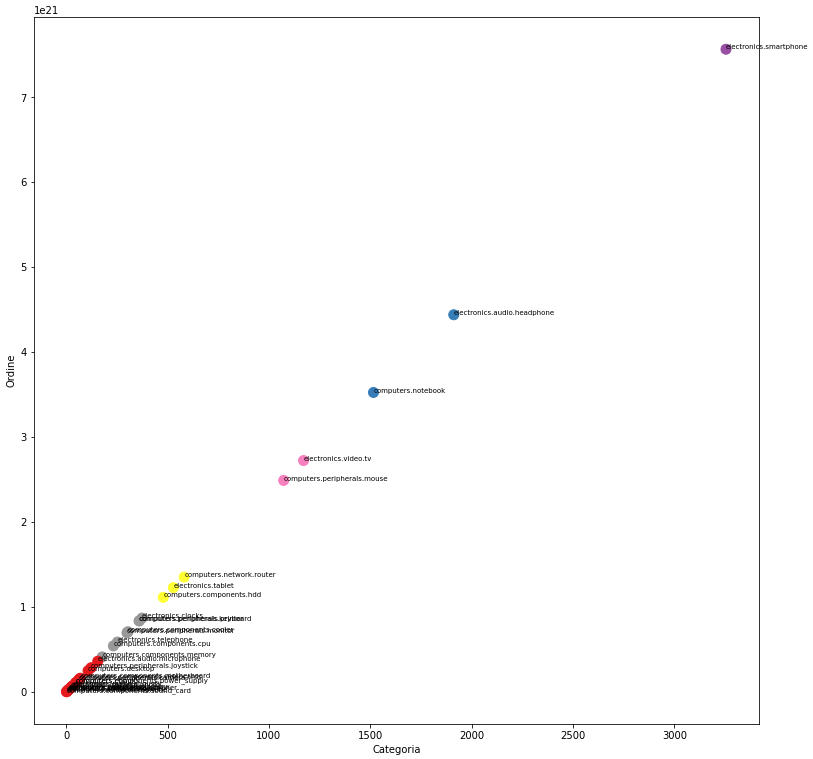
Mediante le funzioni:

* .fit()
* .predict()

Siamo riusciti ad addestrare il modello e ad effettuare previsioni sul set di dati.



Con .scatter() e .annotate() implementate mediante la libreria “*matplotlib”*, siamo riusciti a creare grafi a dispersione con le annotazioni delle rispettive categorie.



**SILHOUTTE SCORE**

Per poter misurare la consistenza dei cluster, ovvero misurare quanto simili sono gli elementi che compongono ogni singolo cluster, abbiamo utilizzato il “*coefficiente di forma”*.

Implementando il seguente algoritmo avremo come risultato un grafico il quale ripoterà la bontà del cluster variando da un range di valori [-1,+1]

