UNIVERSITÀ DEGLI STUDI DI MILANO-BICOCCA
Scuola di Economia e Statistica
Corso di Laurea in
SCIENZE STATISTICHE ED ECONOMICHE

Algoritmi di clustering e loro applicazione all'interno di un recommender system

Relatore: Prof. Mirko Cesarini

Tesi di laurea di:
Luca Paoletti
Matricola 837161

Anno Accademico 2020-2021

alla mia famiglia e al mio gruppo di studio

Indice

1 Introduzione

Negli ultimi anni le applicazioni dell'intelligenza artificiale (AI) sono sempre più presenti nelle vite di ognuno di noi: si pensi alla domotica, alla robotica, alla sanità, oppure al mondo della finanza e, sopratutto, nel mondo dell'e-commerce. Quante volte è capito di essere indecisi su quale prodotto comprare all'interno di un sito online come può essere Amazon, e la nostra scelta finale si è basato sui prodotti consigliati da sistemi di Recommendation che negli ultimi anni sono sempre più spesso basati su algoritmi di Artificial Intelligence ed in particolare di Machine Learning.
Il machine learning è parte di questo mondo e ha l'obiettivo di permettere al computer di imparare delle nozioni senza la necessità di essere esplicitamente programmati dall'uomo. Come dice Sebastian Thrun, innovatore, educatore di imprenditori e informatico, CEO della Kitty Hawk Corporation, presidente e co-fondatore di Udacity: "Nessuno ne parla in questo modo, ma penso che l'intelligenza artificiale sia quasi una disciplina umanistica. È davvero un tentativo di comprendere l'intelligenza umana e la cognizione umana" [1]. In altre parole, la macchina impara come comportarsi all'interno di un particolare sistema, attraverso i comportamenti delle persone che la circondano. Non bisogna essere spaventati da queste scoperte, anzi, vedendo come queste macchine si comportano si può capire meglio anche il comportamento dell'uomo dal quale l'AI impara, può essere un modo per conoscere meglio se stessi.
Questo lavoro riassume quello che ho svolto durante la mia esperienza di stage presso un'azienda collocata nel mondo dell'intelligenza artificiale. In particolare verrà descritto il progetto sviluppato, nello specifico un sistema di raccomandazione indirizzato a una società che si occupa di vendita di prodotti di informatica.
L'obiettivo di questo progetto è stato affidarsi ad un algoritmo di Intelligenza Artificiale per poter soddisfare al meglio le necessità della clientela.
Verrà descritta in primis la componente più importante, i dati (3); si farà riferimento alla documentazione (listino [3.1] e sistema di fatturazione [3.2]) che sono stati offerti dall'azienda richiedente il progetto, e anche al loro utilizzo durante il progetto. Come si potrà constatare più avanti infatti, non tutte le informazioni presenti all'interno dei dati forniti (listino e sistema di fatturazione) saranno utili allo scopo del lavoro finale.
Seguirà una parte più teorica riferita alle metriche di clustering dei dati e alle tecniche di valutazione di quest'ultime. Anche il tema del raggruppamento dei dati è fondamentale per la tipologia di raccomandazione che si vuole proporre alla fine.
Infine verrà descritto nel modo più preciso possibile l'algoritmo sviluppato, attraverso l'utilizzo di Rstudio e Python, e l'implementazione di un sistema di raccomandazione.

Questo elaborato è organizzato nel modo seguente: nel Cap. 2 verrà descritto il progetto nel suo insieme, inclusa una breve panoramica sullo stato dell'arte dei Recommender Systems; nel Cap. 3 saranno descritti i dati a disposizione per il progetto; nel Cap. 4 saranno descritti gli algortimi di Clustering utilizzati per suddividere i prodotti in categorie: nel Cap. 5 sarà descritto il Recommender System sviluppato per questo lavoro: infine nel Cap. 6 saranno tratte le conclusioni e descritti i possibili sviluppi futuri.

2 Descrizione del progetto

Il progetto Componi & Configura (C&C) ha l'obiettivo di realizzare un configuratore intelligente e automatico dei computer assemblati e venduti da Nathan Instruments, una società che cura l'assistenza IT di aziende e studi professionali di diversi settori e affermandosi come consulente di realtà importanti come l'Università degli Studi di Milano e la società FC Internazionale.
Il configuratore potrà essere utilizzato sia all'interno del sistema di e-Commerce di Nathan Instruments, sia dai tecnici di Nathan che assistono i clienti durante le fasi di ordine e acquisto fisico. Ha la funzione di semplificare e guidare la massima personalizzazione del computer richiesto in termini di hardware, software e opzioni varie. Al tempo stesso è in grado di ridurre tempi e costi di produzione delle offerte stesse. Oggi la configurazione e vendita di prodotti hardware e software è principalmente svolta a partire da un'offerta a catalogo e viene effettuata da operatori umani tenendo conto delle caratteristiche tecniche e di compatibilità tra le componenti di base disponibili. Tale processo richiede una dettagliata e sempre aggiornata conoscenza della componentistica che non sempre è in possesso del cliente , quest'ultimo pertanto preferisce l'acquisto fisico al posto di utilizzare l'e-commerce. Il processo è complesso: dato l'elevato numero di possibili opzioni e varianti il numero delle configurazioni possibili è esponenziale e deve tener conto delle esigenze d'uso e dei vincoli di budget specificati dal cliente e delle esigenze connesse alle destinazioni d'uso (es. PC destinati all'office automation, all'elaborazione di grosse quantità di dati, preferenza per un particolare sistema operativo, ecc. ...). La raccolta delle esigenze, informazioni d'uso e specifiche tecniche sarà svolta in modo autonomo ed interattivo dal cliente che sarà guidato a fornire anche quelle informazioni utili al sistema per fornire dei suggerimenti. Ad esempio, l'uso principale dell'hardware e del software richiesto, il budget a disposizione, il carico di lavoro previsto, la quantità di dati, ecc
Si utilizzeranno le metodologie allo stato dell'arte in ambito intelligenza artificiale, quale il Machine Learning, necessarie ad acquisire in modo continuo e mantenere aggiornate la conoscenza su tutte le specifiche tecniche delle varie componenti, così da poter applicare in seguito un algoritmo di raccomandazione il più efficiente possibile.
Si utilizzeranno anche "metodologie di clustering", che permettono di raggruppare in modo omogeneo le configurazioni e componenti simili e quindi suggerire quelle che sono meno distanti dalle richieste dell'utente. Infatti la distanza ci permette di capire quanto due oggetti, siano diversi (o alternativamente simili); il clustering ci da la possibilità di raggruppare quelli che si somigliano. Questo farà sì che, dato l'input dell'utente, il sistema troverà quei prodotti che sono più simili alle specifiche indicate. L'archiviazione della conoscenza sulle specifiche tecniche dei prodotti hardware e

software a catalogo e le elaborazioni richieste dal progetto saranno implementate per mezzo di architetture in cloud-computing.Si sfrutteranno così servizi esterni affidabili in termini di sicurezza del dato, di performance ed efficienza per memorizzare ed aggiornare le componenti vendute e per mantenere anche uno storico delle operazioni di vendita concluse.
L'obiettivo del progetto è quello di creare un algoritmo in grado di consigliare all'utente, che sia nuovo o non, dei prodotti che, sulla base dei suoi acquisti passati o della similarità con altri utenti, possano soddisfare la sua necessità.

2.1 Stato dell'arte

I più grandi siti e-commerce offrono milioni di prodotti. Scegliere tra tutti questi oggetti diventa complicato per i consumatori, e, in risposta a questo problema, nascono i sistemi di raccomandazione.
Uno dei più antichi e anche più utilizzati sistema di raccomandazione è il così detto "collaborative filtering" (CF) [5]. In particolare questo algoritmo offre delle raccomandazioni basandosi sulle preferenze degli utenti. Attraverso il sistema collaborative filtering, verrà offerto al nuovo cliente un prodotto simile a quelli precedentemente comprati dal così detto "vicino", ovvero un secondo utente simile al nuovo. Questa similarità tra utenti verrà definita sulla base degli acquisti effettuati dai due, e di conseguenza i due saranno più "vicini" più i prodotti acquistati saranno simili.
Si suppone che un nuovo utente venga descritto o dalle preferenze espresse esplicitamente da quest'ultimo (es. star rating nelle vendite di amazon), oppure in mancanza di valutazioni esplicite (caso in questione), si usa lo storico degli acquisti (si assume che se un prodotto è stato acquistato allora piace al cliente).
Una seconda tipologia di RS è il content based (CB) (basato sul contenuto). Il contenuto di un elemento è costituito dalla sua descrizione, attributi, parole chiave e etichette. Questi dati vengono messi a confronto con il profilo utente che racchiude le preferenze dell'utente, costruite analizzando gli elementi visualizzati da un utente durante la navigazione. Anche il profilo delle preferenze è espresso tramite attributi, parole chiave e etichette. Mettendo a confronto il contenuto degli elementi e il profilo delle preferenze, il motore di raccomandazione suggerisce all'utente articoli che possono essere di suo interesse. L'approccio basato sul contenuto utilizza sistemi per creare delle stime probabili e affidabili.
A questo punto sorge un problema di dimensionalità. I dati a nostra disposizione sono insufficienti per poter far funzionare in maniera efficiente il sistema di raccomandazione: in particolare abbiamo una mancanza di dati relativi alle vendite che non ci permettono di racchiudere tutti i prodotti disponibili a catalogo. Per ovviare

a questo problema introduciamo delle tecniche di clustering che ci permettono di creare delle macrocategoria e, in maniera parziale, eliminare il problema di scarsità di dati per determinati prodotti.
Quando si ha a che fare con problemi di dimensionalità del dataset o delle informazioni, si cerca di classificare queste ultime così da ridurre la dimensionalità e lavorare in maniera più efficiente sui dati che si hanno a disposizione.
Per poter risolvere questi problemi e poter continuare con la definizione del RS viene introdotto e successivamente usato, il terzo ed ultimo sistema di raccomandazione, il sistema di raccomandazione ibrido: è ottenuto dalla combinazione tra l'approccio collaborativo e l'approccio basato sul contenuto. In questo caso, non si può constatare come il sistema crea delle raccomandazioni perché dipende dalla combinazione degli altri due approcci.
Nel nostro caso quest'ultimo sarà il più plausibile in quanto si avrà una raccomandazione sia per storico degli acquisti (CF) che per caratteristiche del prodotto (CB). In particolare si riuscirà a rafforzare le similarità tra utenti attraverso una similarità tra prodotti definita tramite un algoritmo di clustering basato sulle distanze vettoriali.
L'importanza di aggiungere la componente ibrida al sistema di raccomandazione collaborativo si basa sul fatto che, all'interno di ciascuna categoria utente (che definisce la similarità tra utenti), si hanno molteplici tipologie di oggetti che potrebbero essere estremamente diverse tra di loro. Una raccomandazione che si basa esclusivamente sulla descrizione dell'utente, basata quindi sulla categoria di quest'ultimi, darebbe dei risultati inefficienti e incompleti.
Nel caso qui descritto, gli utenti non creano problemi alla macchina che sviluppa la raccomandazione, in quanto la clientela non è così vasta da dover richiedere un ampio tempo di calcolo. Al contrario invece, le poche informazioni che si hanno relative agli utenti, potrebbero gravare sull'identificazione delle similarità, rendendole troppo vaghe e imprecise. Questo problema di scarsa informazione, viene colmato in parte dall'utilizzo degli algoritmi di clustering che trovano differenze sostanziali tra i diversi prodotti che si ritrovano all'interno di una categoria utente.
Nel capitolo successivo vengono introdotti i dati attraverso i quali verrà creato il sistema di raccomandazione: listino dei prodotti e sistema di fatturazione.

3 Dati

Dal desiderio di voler creare un sistema di raccomandazione per un e-commerce di computer, basato sulle componenti di quest'ultimi, nasce la necessità di avvalersi di alcuni documenti ufficiali facenti riferimento a queste macchine rese disponibili per la vendita. Per questo motivo sono stati raccolti i dati relativi al listino di prodotti ufficiali delle aziende fornitrici di Nathan, rappresentati i prodotti vendibili, e il sistema di fatturazione di Nathan, prodotti già venduti. Con queste due classi di prodotti si riescono a identificare i prodotti che possono essere raccomandati dal sistema di raccomandazione (prodotti vendibili) secondo delle precise preferenze degli utenti, identificabili dai prodotti già venduti. Attraverso questi documenti si potrà identificare e progettare un sistema di raccomandazione efficiente.
In particolare saranno prelevate dal listino informazioni sulle caratteristiche dei prodotti e delle componenti. Di conseguenza l'algoritmo riuscirà a trovare delle similarità (misura basata sulle distanze tra cluster) tra prodotti in modo tale da poter raccomandare un prodotto simile a quello già acquistato dall'utente. Quest'ultima informazione, invece, relativa agli acquisti passati degli utenti, la si può ritrovare all'interno del sistema di fatturazione che memorizza in maniera puntuale i diversi oggetti venduti ai clienti.
Come si può facilmente immaginare, le fonti dati come quelle sopra descritte, contengono molte informazioni futili al fine di creare un sistema di raccomandazione. Si decide quindi di andare a filtrare, per necessità, i dati presenti, raccogliendo esclusivamente le informazioni necessarie al funzionamento ottimale del processo di raccomandazione e descritte qua di seguito.

3.1 Listino

Per ogni fornitore di Nathan viene fornito un Listino contenente la lista di prodotti ai quali Nathan può accedere. Dopo una serie di filtri applicati al documento originale, con l'obiettivo di eliminare le variabili non utili all'analisi, il listino deve apparire come segue:

ProdCode	**Description**	**CatCode**	**Stock**	**Web price**
10U28EA	ÑHP DRAGONFLY I7-8586U 16/512 SVWP	NO	2	1469,00

Tabella 1: Listino dei Prodotti

In particolare avremo che:

- **ProdCode** identifica il prodotto e servirà a collegare il listino al sistema di fatturazione di Nathan;
- **Description** raccoglie le informazione rilevanti relative alle componenti del prodotto; (solitamente separate da un segno quale "-" oppure "/" oppure " ", varia al variare del fornitore). Attraverso questo campo raccogliamo le informazioni che ci interessano relative a **Processore**, **Ram** e **Archiviazione**;
- **CatCode** identifica la categoria di prodotto che, nel caso specifico richiesto da Nathan, comprende le voci Notebook(NO), All in One (IO) e Desktop (PC) ;
- **Stock**: descrive la quantità di prodotti disponibili in magazzino. Nel nostro caso, come richiesto da Nathan, renderemo questa variabile dicotomica; in particolare apparirà il valore 1, se il prodotto è disponibile, 0 se il prodotto non è disponibile;
- **WebPrice** il prezzo sul quale andremo a fare le nostre analisi successive.

3.2 Vendite

Nathan ha messo a disposizione per il progetto i dati del sistema di fatturazione, attraverso il quale potremo studiare le preferenze dei clienti rispetto ai prodotti venduti. Questo secondo dataset avrà la seguente forma:

UserId	**CatUtente**	**ProdCode**	**Mese**	**Anno**
1441	5	10U28EA	2	2018

Tabella 2: Vendite

In particolare avremo che:

- **UserId** identifica l'utente che fa un acquisto presso Nathan;
- **CatUtente** identifica la categoria utente di appartenenza fornita da Nathan;
- **ProdCode** identifica il prodotto e servirà a collegare il sistema di fatturazione di Nathan al listino dei prodotti vendibili;

- **Mese** rappresenta il mese in cui è stata effettuata la vendita;
- **Anno** rappresenta l'anno in cui è stata effettuata la vendita.

Nel capitolo 4 verranno descritte le metodologie di machine learning utilizzate per definire il sistema di raccomandazione, verranno introdotte quindi le metodologie di clustering e le diverse metriche di valutazione dei rispettivi metodi di classificazione.

4 Clustering

4.1 Metodologie di Clustering

Si decide di procedere con delle metodologie di clustering per raggruppare dei prodotti simili tra di loro, sulla base di caratteristiche specificate in seguito, per poter effettuare raccomandazioni il più efficienti e compatibili possibili con le richieste dell'utente. In particolare verranno utilizzate delle metriche in grado di evidenziare le distanze (in termini vettoriali) tra i diversi prodotti.
Per quanto riguarda la suddivisione dei prodotti si decide di procedere con due metodologie differenti (accogliendo anche la richiesta di Nathan di fornire in output al sistema di raccomandazione due prodotti consigliati):

- effettuare il clustering sulle componenti dei prodotti all'interno di ciascuna categoria utente, così da andare a ricercare le diverse tipologie di oggetti che si differenziano all'interno della stessa tipologia di utente;
- effettuare un doppio clustering sull'intera gamma di prodotti disponibili, con lo scopo di andare a ricercare le differenze sostanziali tra i vari oggetti ed andare a classificarli sulla base delle componenti di ognuno. In particolare l'approccio appena descritto vuole tentare di replicare i sottogruppi che nel primo metodo erano stati definiti attraverso le categorie utenti. In questo modo, attraverso un doppio clustering, verranno raccomandati dei prodotti ancora meglio classificati.

Entrambi i metodi seguono la metodologia k-means, basata sulla distanza vettoriale. Nonostante questa metodologia sia quella applicata di default, si studiano anche altre tipologie di clustering attraverso le quali si giunge a soluzioni simili tra di loro. In particolare le diverse metodologie analizzate sono:

4.1.1 K-Means

Definito $C_1, ..., C_K$ un insieme di gruppi contenenti le osservazioni $\{1, ..., n\}$, questo insieme di gruppi deve soddisfare le seguenti proprietà [2]:

- $C_1 \cup C_2 \cup ... \cup C_K = \{1, ..., n\}$
- $C_k \cap C_{k'} = 0$, con $k \neq k'$

In K-Means una buona suddivisione delle osservazioni in gruppo è quella per cui la **variazione intra-cluster**, $W(C_K)$, è minima, ovvero bisogna risolvere il seguente problema:

$$\min_{C_1,\dots,C_K} \sum_{k=1}^{K} W(C_k)$$

dove, attraverso la **distanza euclidea quadratica**, viene definita

$$W(C_k) = \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2$$

dove $|C_K|$ è il numero di osservazioni nel cluster K. Quindi, procedendo per punti, abbiamo che:

1. Si assegna casualmente un valore, da a 1 a K, ad ogni osservazione;

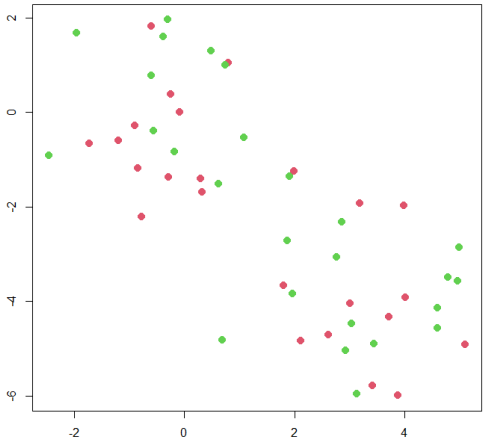

Figura 1: Si inizializza casualmente un valore da 1 a K (in questo caso K=2) ad ogni osservazione

2. Si calcola il valore del **centroide**. Il centroide del *k-esimo* cluster è il vettore di dimensione p che contiene le medie delle variabili per le osservazioni nel cluster *k-esimo*;

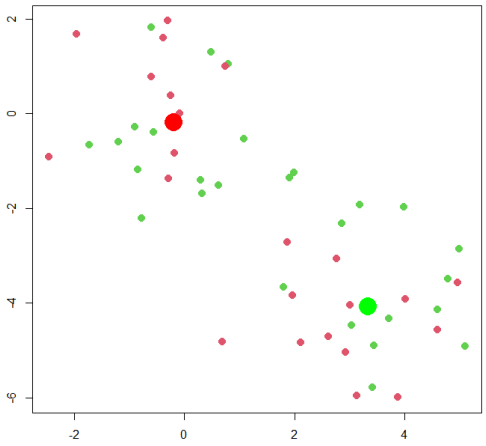

Figura 2: Vengono scelti i centroidi

3. Si assegna ogni osservazione al cluster per il quale il centroide risulta più vicino. La vicinanza è determinata dal valore della distanza Euclidea.

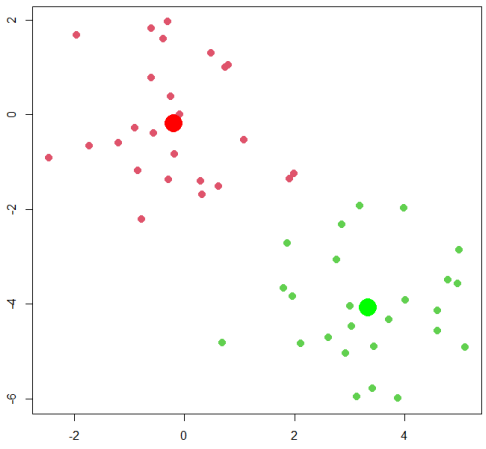

Figura 3: Vengono assegnati i punti al cluster più vicino

4. Si ripetono i punti 2. e 3. finché non si raggiunge la convergenza.

4.1.2 Hierarchical clustering

In questo tipo di metodo i dati vengono associati ad una struttura ad albero in modo tale che le foglie dell'albero corrispondano alle osservazioni e i nodi a sottoinsiemi di osservazioni. La stessa natura di albero introduce una gerarchia nei sottoinsiemi associati ai rami.
Esistono due ampie famiglie di metodi gerarchici:

- **Metodi Agglomerativi [AGNES]**: partendo da uno stato iniziale di n gruppi, in cui ogni osservazione fa gruppo a sè, si procede effettuando successive fusioni di gruppi con bassa dissimilarità tra loro, fino a che $k = 1$, cioè tutte le osservazioni appartengono ad uno stesso gruppo;

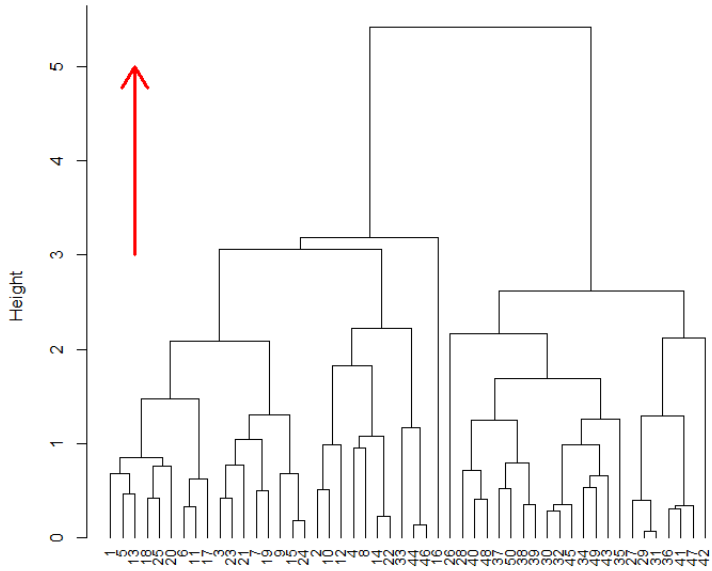

Figura 4: Dendogramma del clustering gerarchico Agglomerativo

- **Metodi Divisivi [DIANA]**: partendo da uno stato iniziale con $k = 1$ gruppi, ossia ad un unico gruppo, si procede per suddivisioni successive fino ad arrivare a n gruppi.

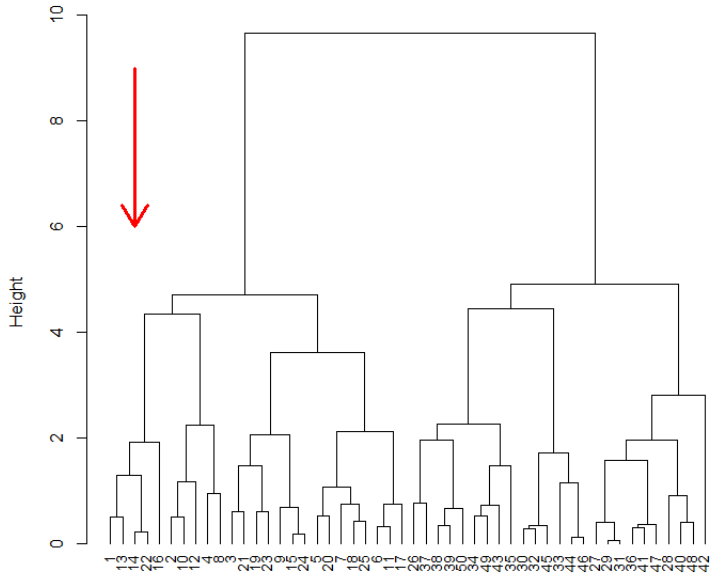

Figura 5: Dendogramma del clustering gerarchico divisivo

Entrambi i procedimenti operano su una matrice di dissimilarità. La caratteristica principale di questo tipo di metodi è che una volta che due gruppi sono stati uniti non saranno più separati in seguito e, in modo analogo, una volta che due gruppi vengono separati non faranno più parte dello stesso cluster. Inoltre applicando un algoritmo di questo tipo, si usa lo stesso albero per tutti i valori di k, facendo riferimento ogni volta a un livello diverso dell'albero stesso. Si tratta quindi di una struttura rigida. I metodi per la valutazione delle distanze tra i gruppi ai fini delle aggregazioni o divisioni successive sono:

- **Metodo del legame singolo** [**Single Linkage**]: la distanza tra gruppi è definita come la minima tra tutte le distanze che si possono calcolare tra ciascuna unità i in un gruppo e ciascuna unità di j di un altro gruppo;

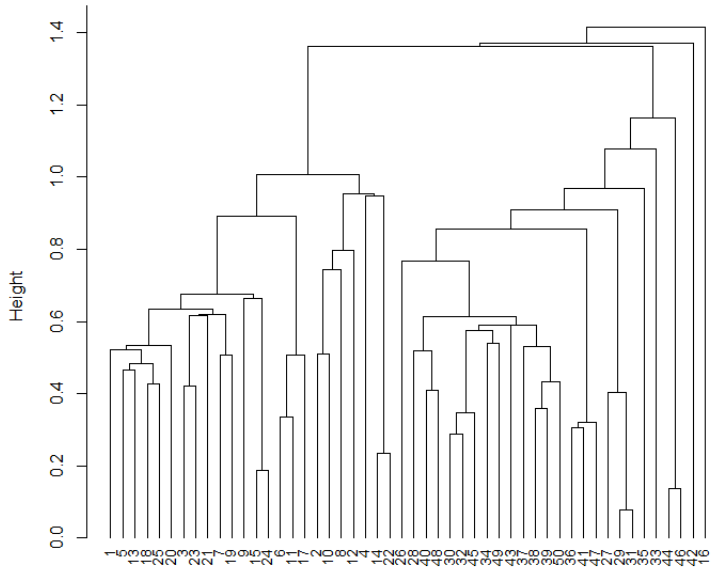

Figura 6: AGNES con Single Linkage

- **Metodo del legame completo [Complete Linkage]**: la distanza tra gruppi è definita come la massima tra tutte le distanze che si possono calcolare tra ciascuna unità i in un gruppo e ciascuna unità di j di un altro gruppo;

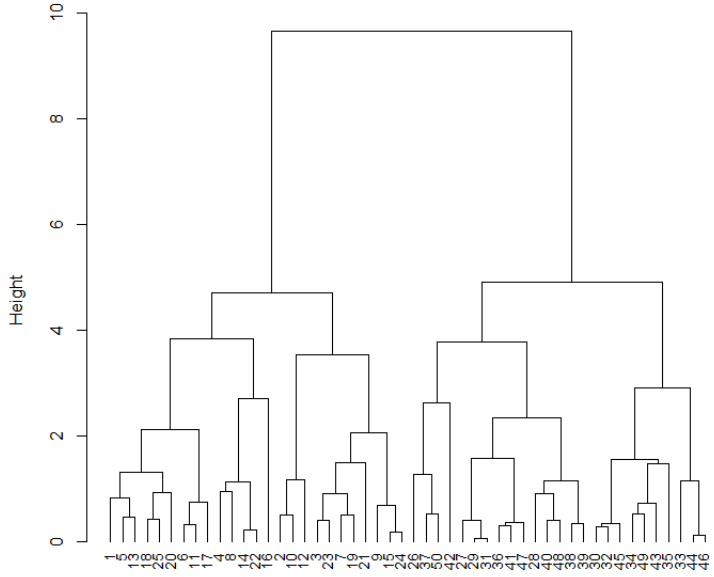

Figura 7: AGNES con Complete Linkage

- **Metodo del legame medio** [**Average Linkage**]: la distanza tra i gruppi è definita come la media tra tutte le distanze che si possono calcolare tra due gruppi;

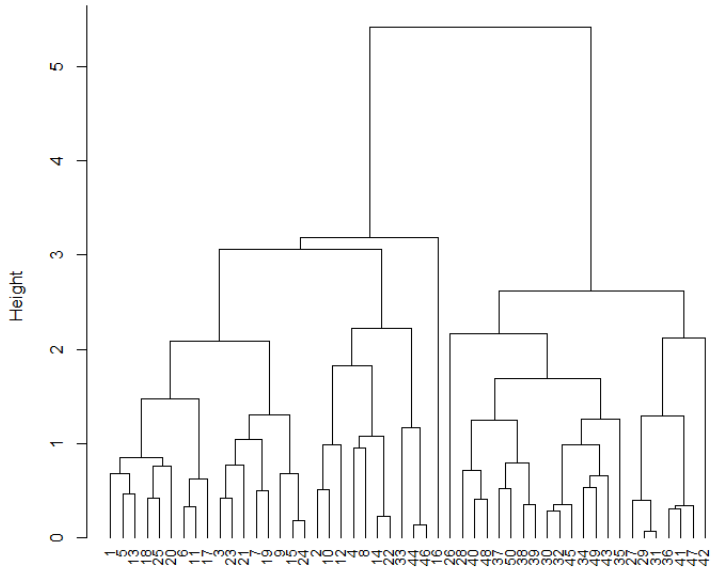

Figura 8: AGNES con Average Linkage

- **Metodo del centroide** [**Centroid Method**]: la distanza tra i gruppi è definita dalla distanza tra i loro centroidi, che sono i vettori dei valori medi delle p variabili nei gruppi;
- **Metodo di Ward** [**Ward Method**]: si basa sulla scomposizione della devianza totale nella somma delle devianze entro i cluster e tra i cluster. Nel passare da $k+1$ a k gruppi (aggregazione) la devianza entro i cluster aumenta, mentre la devianza tra i cluster diminuisce. Ad ogni passo, il metodo di Ward aggrega tra loro i gruppi per cui vi è il minor incremento della devianza entro i gruppi.

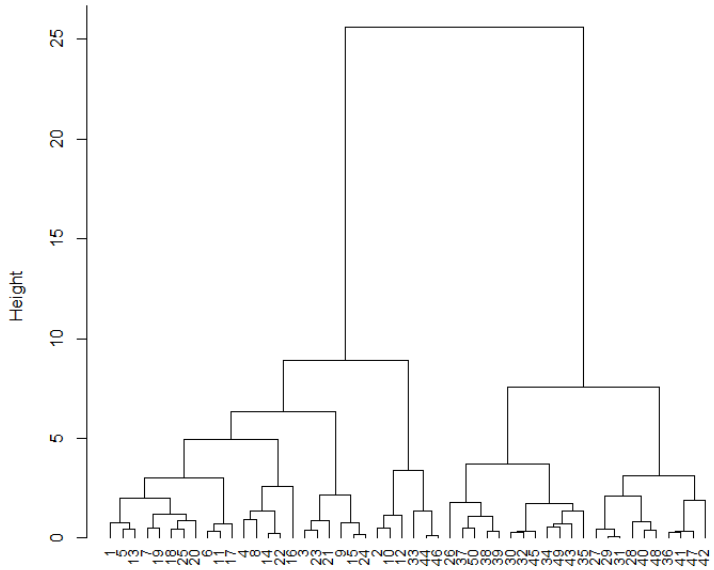

Figura 9: AGNES con Ward Linkage

Due esempi di metodi gerarchici sono AGNES e DIANA (Kaufman e Rousseeuw, 1990)[4].

4.1.3 Mini Batch K-Means

Man mano che i set di dati crescono, aumenta anche il tempo di calcolo. Il completamento del clustering K-Means di base può richiedere molto tempo se seguito su set di dati di grandi dimensioni e, di conseguenza, sono state apportate modifiche al clustering K-Means per consentire di ridurre i costi di memoria occupata e temporali dell'algoritmo.
Il clustering Mini-Batch K-means è una variante del clustering K-means in cui la dimensione del set di dati considerato ad ogni interazione è limitata. Il clustering K-mean normale opera sull'intero set di dati contemporaneamente, mentre il clustering K-means per mini-batch suddivide il set di dati in sottoinsiemi. I mini-batch vengono campionati in modo casuale dall'intero set di dati e per ogni nuova iterazione viene selezionato un nuovo campione casuale e utilizzato per aggiornare la posizione dei centroidi.
Nel clustering K-Means per minbatch, i cluster vengono aggiornati con una combinazione dei valori del mini-batch e della velocità di apprendimento. La velocità di apprendimento diminuisce durante le iterazioni ed è l'inverso del numero di punti dati inseriti in un cluster specifico. L'effetto della riduzione del tasso di apprendimento è che l'impatto dei nuovi dati si riduce e si ottiene la convergenza quando, dopo diverse iterazioni, non ci sono cambiamenti nei cluster.
I risultati degli studi sull'efficacia del clustering mini-batch suggeriscono che può ridurre il tempo di calcolo con un leggero compromesso nella qualità del cluster.

4.2 Evaluation

Trattandosi di algoritmi non supervisionati, la componente di valutazione non è così immediata come per altre tipologie di clustering supervisionato, per i quali l'accuratezza di un algoritmo è data dalla differenza della predizione rispetto al valore osservato. La scelta del k-ottimo (K^*) è una procedura fondamentale per ottenere la miglior suddivisione di prodotti in entrambe le metodologie seguite; in particolare l'obiettivo è quello di massimizzare la distanza fra cluster (BSS) e minimizzare la distanza infra-cluster (WSS).
Per fare ciò proseguiamo con l'analisi di due sistemi di valutazione del cluster attraverso le quali sarà poi possibile andare a scegliere il k^* [3].

4.2.1 Silhouette

Ogni cluster è rappresentato da una *silhouette*, la quale indica l'appartenenza di un punto specifico al cluster di riferimento. In particolare più il valore sarà prossimo a 1,

più il punto è stato "ben" classificato, e viceversa più il punto si allontana dall'unità. L'intero clustering è rappresentato da tutte le silhouettes in un unico diagramma che illustra la qualità del clustering.
Processo:

1. considerare ogni osservazione i del dataset, assumendo A come il cluster di i;
2. calcolare la distanza media a_i di i con tutte le altre altre osservazioni in A

$$a_i = \frac{1}{N_A} \sum_{j \in A, j \neq i} d(i, j)$$

3. considerare ogni cluster C differente da A e definire la distanza media $d(i, C)$ di i con le osservazioni in C

$$d(i, C) = \frac{1}{N_C} \sum_{c \in C} d(i, c)$$

4. calcolare $d(i, C)$ $\forall C \neq A$, e selezionare quello con $b_i = \min\ d(i, C)$, $C \neq A$

La silhouette di i è definita come:

$$sil_i = \frac{b_i - a_i}{\max(a_i, b_i)}$$

Mentre quella che si è utilizzata nel progetto corrente è la così detta **Silhouette Media**:

$$sil_{av} = \sum_i \frac{sil_i}{N}$$

Il numero di cluster k può essere determinato scegliendo il valore di k che porta al **valore medio di silhouette più alto**.

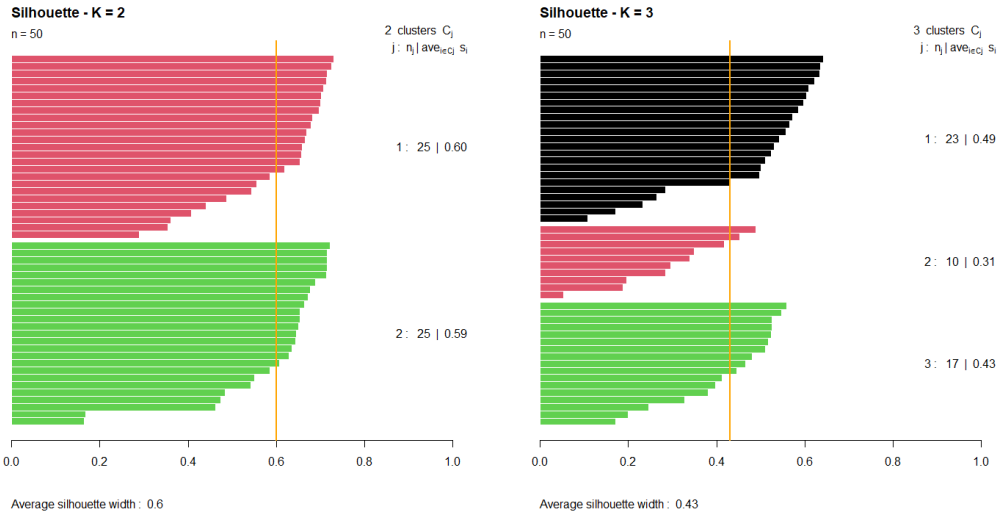

Figura 10: confronto tra le silhouette per due diversi k

4.2.2 Indice di Calinski-Harabasz

Per ogni numero di cluster $k \geq 2$, l'indice di Calisnki Harabasz è definito come

$$ch(k) = \frac{tr(B_k)/(k-1)}{tr(W_k)/(n-k)}$$

dove n e k sono rispettivamente il numero totale di osservazioni e il numero di cluster. $tr(B_k)$ è la traccia della matrice dei cluster tra gruppi, mentre $tr(W_k)$ è la traccia della matrice dei cluster entro i gruppi.
Il numero di cluster ottimo si ottiene in corrispondenza del k per cui vi sono grandi dissimilarità tra i cluster e grandi similarità entro i cluster: la soluzione è quindi il valore di k che massimizza $ch(k)$.

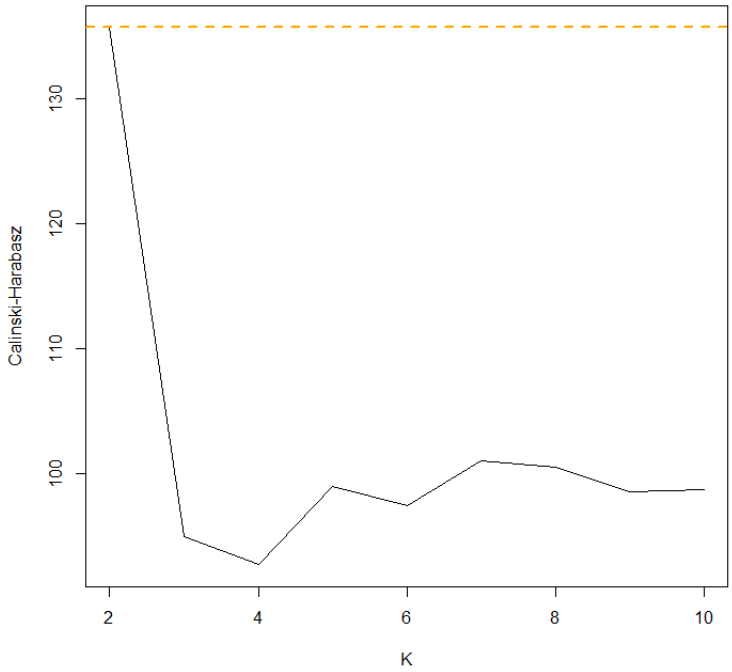

Figura 11: confronto dei $ch(k)$ per ogni k

Entrambi i metodi elencati danno risultati simili in termini di k^*, dovendo però scegliere una misura soltanto di valutazione, si decide di applicare la Silhouette media come metrica di default.
Infine, nel capitolo 5, viene definito il sistema di raccomandazione. Verrà, inizialmente, introdotto il processo di simulazione dei dati tramite R, per poi passare alla parte di algoritmo. Tutto il capitolo sarà accompagnato dai diversi script (Python e R). L'ultima parte di questo capitolo, prima di passare alla conclusione, fa riferimento ai risultati ottenuti.

5 Recommender System

L'obiettivo del progetto è quello di creare un algoritmo in grado di consigliare all'utente, che sia nuovo o non, dei prodotti che, sulla base dei suoi acquisti passati o della similarità con altri utenti, possano soddisfare la sua necessità [6].
L'algoritmo che viene qui definito, è un algoritmo che si basa principalmente sulle similarità tra prodotti. In particolare attraverso dei dati simulati, come verrà spiegato nel paragrafo dedicato (5.2), secondo precisi pattern di configurazioni di computer, si riesce a consigliare due prodotti: il primo che verrà estratto sulla base delle similarità che un utente (appartenete a una precisa categoria utente) ha con gli altri utenti appartenenti alla stessa categoria, e di conseguenza il prodotto rispecchierà le esigenze del cliente che viene descritto da quella categoria utente; e una seconda raccomandazione, frutto di una doppia suddivisione dei prodotti, sempre basata sulla similarità che c'è tra quest'ultimi, che sarà il più simile possibile ai prodotti precedentemente acquistati dal clienti che effettua la richiesta. Come si può facilmente intuire, questa seconda raccomandazione diventerà futile nel momento in cui il cliente è nuovo e non ha mai acquistati prodotti presso Nathan. In questo specifico caso, i prodotti estratti, saranno entrambi provenienti dalla prima metodologia di ricerca.
Tutte le raccomandazione seguono inoltre altri criteri, quali prezzo inferiore, disponibilità da magazzino e anno di pubblicazione del prodotto.

5.1 Dati in Input

Nella situazione in cui ci si trova, i dati a disposizioni sono troppo pochi e di difficile interpretazione. In particolare non esiste un identificativo in grado di collegare i due set di dati, Listino e Sistema di Fatturazione, e di conseguenza non si hanno dei dati a sufficienza per poter creare un sistema efficiente per la raccomandazione dei prodotti venduti. Per porre rimedio a questa mancanza, si è deciso di simulare dei dati che seguissero nel modo più simile possibile i dati reali di Listino e di Vendita.
Nel dominio delle vendite di computer i dati rilevanti riguardano le componenti di quest'ultimi e, siccome questi devono essere esclusivamente dati fittizi sui quali testare un modello, si decide di simulare solo tre componenti, in particolare:

- Processore
- Ram
- Archiviazione

è importante specificare come queste tre componenti non siano state estratte casualmente, bensì i sono quelle che, attraverso il catalogo dei prodotti fornito dal cliente, possono essere estratte e utilizzate. Infatti nel catalogo dei prodotti non sono presenti tutte le caratteristiche del prodotto, ma solo quelle più importanti: nel caso nostro queste caratteristiche sono processore, ram e archiviazione.
Oltre a queste variabili, si decide di simulare, sempre con il criterio di poter applicare in seguito l'algoritmo su dati reali, le seguenti variabili:

- Id_Utente
- Id_Prodotto
- Prezzo di vendita
- Disponibilità (prodotto disponibile Sì/No)
- Mese (di vendita)
- Anno (di vendita)
- Anno (di produzione del prodotto).

5.2 Processo di simulazione dei dati tramite R

La parte di simulazione del dataset si basa su conoscenze personali e un breve sondaggio effettuato tramite google moduli, attraverso il quale si è riusciti a definire le probabilità di presenza di una determinata componente all'interno di uno specifico computer. Per questo motivo la simulazione si basa su diversi pattern di configurazioni, ognuna con una probabilità di estrazione differente.

```
1
2 #PROCESSORE
3
4 processore=c(paste("i",c(3,5,7,9), sep=""), paste("R", c(3,5,7,9),
    sep=""))
5
6 #intel 60%, AMD 40%
7
8 probs=c(0.2*0.6,0.4*0.6,0.25*0.6,0.15*0.6, 0.2*0.4,0.4*0.4,0.25*
    0.4,0.15*0.4)
9
10 proc=sample(processore, prob = probs, size=3000, replace=T)
11 product_id=sample(1:10000, size=3000, replace=FALSE)
12 disponibilita=sample(c(0,1), size=3000, replace=T)
```

```r
dataset=data.frame("User_id"=0, "categoria_utente"=0, product_id, "processore"=proc, "ram"=0, "archiviazione"=0, "prezzo"=0.0)
dataset

# RAM

RAM1=c(2,4,8)
prob_RAM1=c(0.2,0.6,0.2)
dataset[which(dataset$processore == "i3" | dataset$processore == "R3"),"ram"]=sample(size=length(proc[which(proc=="i3"|proc=="R3")]), RAM1, prob = prob_RAM1, replace=T)

RAM2=c(4,8,16)
prob_RAM2=c(0.2,0.6,0.2)
dataset[which(dataset$processore == "i5" | dataset$processore == "R5"),"ram"]=sample(size=length(proc[which(proc=="i5"|proc=="R5")]), RAM2, prob = prob_RAM2, replace=T)

RAM3=c(8,16,32,64)
prob_RAM3=c(0.1,0.6,0.2,0.1)
dataset[which(dataset$processore == "i7" | dataset$processore == "R7"),"ram"]=sample(size=length(proc[which(proc=="i7"|proc=="R7")]), RAM3, prob = prob_RAM3, replace=T)

RAM4=c(16,32,64,128)
prob_RAM4=c(0.2,0.3, 0.3,0.2)
dataset[which(dataset$processore == "i9" | dataset$processore == "R9"),"ram"]=sample(size=length(proc[which(proc=="i9"|proc=="R9")]), RAM4, prob = prob_RAM4, replace=T)

## HARD DISK

HDD1=c(128,256,512)
prob_HDD1=c(0.2,0.6,0.2)
dataset[which(dataset$processore == "i3" | dataset$processore == "R3"),"archiviazione"]=sample(size=length(proc[which(proc=="i3"|proc=="R3")]), HDD1, prob = prob_HDD1, replace=T)


HDD2=c(256,512, 1024)
prob_HDD2=c(0.2,0.6,0.2)
dataset[which(dataset$processore == "i5" | dataset$processore == "R5"),"archiviazione"]=sample(size=length(proc[which(proc=="i5"|proc=="R5")]), HDD2, prob = prob_HDD2, replace=T)


```

```
HDD3=c(512,1024,2048,3072)
prob_HDD3=c(0.1,0.6,0.2,0.1)
dataset[which(dataset$processore == "i7" | dataset$processore == "R7
    "),"archiviazione"]=sample(size=length(proc[which(proc=="i7"|proc
    =="R7")]), HDD3, prob = prob_HDD3, replace=T)


HDD4=c(1024,2048,3072,10000)
prob_HDD4=c(0.15,0.6,0.2,0.05)
dataset[which(dataset$processore == "i9" | dataset$processore == "R9
    "),"archiviazione"]=sample(size=length(proc[which(proc=="i9"|proc
    =="R9")]), HDD4, prob = prob_HDD4, replace=T)

# prezzi

media_prezzi1=500; sd_prezzi1=50
dataset[which(dataset$processore == "i3" | dataset$processore == "R3
    "),"prezzo"]=round(rnorm(n=length(proc[which(proc=="i3"|proc=="R3
    ")]), mean = media_prezzi1, sd=sd_prezzi1),2)

media_prezzi2=900; sd_prezzi2=80
dataset[which(dataset$processore == "i5" | dataset$processore == "R5
    "),"prezzo"]=round(rnorm(n=length(proc[which(proc=="i5"|proc=="R5
    ")]), mean = media_prezzi2, sd=sd_prezzi2),2)

media_prezzi3=1400; sd_prezzi3=120
dataset[which(dataset$processore == "i7" | dataset$processore == "R7
    "),"prezzo"]=round(rnorm(n=length(proc[which(proc=="i7"|proc=="R7
    ")]), mean = media_prezzi3, sd=sd_prezzi3),2)

media_prezzi4=2100; sd_prezzi4=300
dataset[which(dataset$processore == "i9" | dataset$processore == "R9
    "),"prezzo"]=round(rnorm(n=length(proc[which(proc=="i9"|proc=="R9
    ")]), mean = media_prezzi4, sd=sd_prezzi4),2)
```

Listing 1: Simualzione componenti

Come si può notare dal Listing 1, per ogni componente (prezzo compreso) si è assegnata una diversa probabilità ad ogni differente combinazione di pattern. Questa scelta si basa sul fatto che, un errore banale consiste nell'assegnare dei prezzi o delle particolari componenti, a configurazioni che in realtà non esistono (es. computer da €2.000 con 2 GB di Ram ,256 GB di Archiviazione e un processore i3). Questa tipologia di errore avrebbe gravato decisamente sul clustering dei prodotti , un item così mal strutturato, può essere definito *outlier*, estremo.
Per quanto riguarda la simulazione della disponibilità a listino del prodotto, si è

deciso di applicare una probabilità uguale sia allo 0 (non disponibile) che al valore 1 disponibile). Nonostante ciò, come si può facilmente dedurre, nel momento in cui un prodotto viene inserito a listino, si ha una probabilità maggiore che quest'ultimo sia disponibile, ma, per testare l'algoritmo, si è deciso di applicare una probabilità equa per entrambi i valori.
Infine, per quanto riguarda la simulazione dei prezzi, si è deciso di estrarre da una normale di media e standard deviation variabile, sulla base della fascia di prezzo, un numero casuale che rappresentasse il prezzo. Anche in questo caso ci si è avvalsi del sondaggio effettuato, nel quale veniva richiesto di indicare una fascia di prezzo al quale appartenesse il proprio computer (la fascia di prezzo con percentuale di risposta maggiore, sarà stata estratte più volte dalla normale con media e standard deviation rappresentativi di quella fascia).
Ora che la parte di simulazione dei dati è stata completata, si va a riempire i due dataset, Listino e Vendite, con i dati appena generati. Questa procedura viene ancora svolta in Rstudio così da mantenere separate le due componenti di questo progetto vale a dire, simulazione dei dati e RecSys.

```
dataset["disp"]=disponibilita
dataset ## genero casualmente 3000 item
## estraggo casualmente 1500 item
## gli altri saranno quelli non acquistati
listino=dataset[c("product_id", "processore", "ram", "archiviazione"
   , "prezzo", "disp")]
anno=sample(c(2018,2019), size=3000, replace=T)
mese=sample(c(1:12), size=3000, replace=T)
listino["mese"]=mese
listino["anno"]=anno
listino

acquisti=sample(1:3000, size=2500, replace=T)
data=dataset[acquisti,]
data
user_cat=data.frame("User_id"=1:4000, "categoria_utente"=sample
   (1:10, size=4000, replace = T))
user_cat
idx=sample(1:4000, size=2500, replace=T)
user_cat[idx,]
data["User_id"]=user_cat[idx,1]
data["categoria_utente"]=user_cat[idx,2]
data

mesi=sample(1:12, size=2500, replace=T)
anni=sample(2018:2020, size=2500, replace = T)
```

```
25 data["mese"]=mesi
26 data["anno"]=anni
```

Listing 2: Generazione dei dataset

Con questo ultima parte di script viene ultimato il processo di simulazione di dati. In particolare vengono generati casualmente degli user id, che possono fare riferimento a più acquisti; ad ogni user id viene assegnata una categoria utente, che lo caratterizza a prescindere dai differenti acquisti di PC che un singolo utente potrebbe fare; e infine vengono generati casualmente gli anni e i mesi sia per quanto riguarda le vendite (mese e anno di vendita di un singolo prodotto) che per quanto riguarda il periodo di pubblicazione del prodotto sul listino.

I dataset finali avranno quindi la seguente forma, sempre compatibile con i dati originali:

Index	product_id	processore	ram	archiviazione	prezzo	disp	Anno
0	1883	i9	16GB	2048GB_HDD	1977.11	1	2018
1	537	i3	4GB	128GB_HDD	578.14	1	2018
2	6651	i5	8GB	256GB_HDD	1025.07	1	2018
3	4140	R3	4GB	256GB_HDD	439.54	0	2018
4	8087	i9	8GB	2048GB_HDD	1977.96	1	2018
5	932	i5	16GB	256GB_HDD	904.24	0	2018
6	373	R5	8GB	512GB_HDD	993.69	0	2018
7	1515	i3	2GB	256GB_HDD	414.01	1	2018
8	9889	i3	4GB	512GB_HDD	569.43	1	2018
9	9028	i5	8GB	1024GB_HDD	855.72	0	2018
10	3493	i7	8GB	2048GB_HDD	1289.01	1	2018
11	8777	R5	8GB	512GB_HDD	780.29	0	2018

Figura 12: Listino dei prodotti (simulato)

Index	User_id	categoria_utente	product_id	processore	ram	archiviazione	prezzo	mese	anno
0	1441	9	1883	i9	16GB	2048GB_HDD	1977.11	5	2018
1	1530	7	537	i3	4GB	128GB_HDD	578.14	2	2018
2	2234	10	6651	i5	8GB	256GB_HDD	1025.07	6	2019
3	2507	8	4140	R3	4GB	256GB_HDD	439.54	3	2018
4	1661	1	8087	i9	8GB	2048GB_HDD	1977.96	5	2019
5	50	1	932	i5	16GB	256GB_HDD	904.24	4	2018
6	1669	6	373	R5	8GB	512GB_HDD	993.69	1	2019
7	1478	3	1515	i3	2GB	256GB_HDD	414.01	7	2019
8	1741	10	9889	i3	4GB	512GB_HDD	569.43	5	2018
9	421	7	9028	i5	8GB	1024GB_HDD	855.72	2	2019
10	74	6	3493	i7	8GB	2048GB_HDD	1289.01	3	2020
11	3817	3	8777	R5	8GB	512GB_HDD	780.29	4	2020

Figura 13: Vendite dei prodotti (simulato)

Questi due dataset seguono lo schema di costruzione dei dataset di Listino (1) e Vendite originali (2), in questo modo si è sicuri che l'algoritmo implementato possa soddisfare gli attributi di questi due set di dati simulati.
Come si può facilmente notare dalle foto sopra riportate, per applicare un algoritmo di clustering su di un set di dati, bisogna ricodificare e rendere numeriche le variabili su quale verrà applicato l'algoritmo di classificazione. Per fare questa procedura applicheremo dei numeri da 1 a n alle variabili *Ram*, *Processore* e *Archiviazione* ed utilizzeremo le variabili ricodificate da questo momento in poi nell'algoritmo.

5.3 Algoritmo

5.3.1 Import e Ricodifica delle variabili

Come primo step si importa il dataset (simulato), dal quale in seguito, vengono estratti i due sotto dataset Listino e Vendite. E' importante sottolineare una profonda differenza tra questi due dataset, in particolare il fatto che in "Listino" i prodotti non sono mai ripetuti, mentre in "Vendite" i prodotti si possono ripetere visto che si tratta di un sistema di fatturazione societario. Inoltre, sempre in "Listino", un id utente può apparire diverse volte, sulla base dei numeri di acquisti che l'utente effettua (una riga per ogni acquisto singolo effettuato).
Successivamente, dopo aver creato i due dataset, si procede con la ricodifica delle variabili che fanno riferimento alle caratteristiche dei computer. Si decide di attribuire

un valore da 0 a n (diverso per ogni variabile da ricodificare), dove n rappresenta la lunghezza dei valori univoci delle singole componenti. Di seguito si riporta lo script di questa prima parte di algoritmo:

```
import pandas as pd
import numpy as np
import sklearn
from sklearn.datasets import make_classification
from sklearn.cluster import KMeans
from sklearn.cluster import AgglomerativeClustering
from sklearn.cluster import MiniBatchKMeans
from sklearn import metrics


#import dataset
data_orig=pd.read_csv(r"C:\Users\Luca\Downloads\dataframe_finale.csv")
data_orig=data_orig.drop("Unnamed: 0", axis=1)

#dataset vendite
vendite=data_orig.iloc[:, [0, 1, 2, 3, 4, 5, 6, 8, 9]]
#listino
listino=pd.read_csv(r"C:\Users\Luca\Downloads\listino_finale.csv")
listino=listino.drop("Unnamed: 0", axis=1)
listino=listino.drop("mese", axis=1)


def mat(matrice_iniz, sparse=False):
    if sparse==True:
        prodotti=np.unique(matrice_iniz.iloc[:,2]).tolist()
        df=pd.pivot_table(matrice_iniz.iloc[:,range(2,6)], index="product_id", aggfunc="first")
        tmp1=matrice_iniz.iloc[:,range(2,6)]
        tmp2=tmp1.drop_duplicates()
        tmp2.index=tmp2.product_id
        tmp2=tmp2.iloc[:,[1,2,3]]
        prodotti=tmp2.index
        features=tmp2.values.tolist()
        feat_prod=pd.DataFrame((prodotti,features), index=["product_id","feature"]).T
        feat_prod.index=feat_prod.product_id
        feat_prod=feat_prod.feature

        from sklearn.preprocessing import MultiLabelBinarizer
        mlb = MultiLabelBinarizer()
```

```
39          dataclust = pd.DataFrame(mlb.fit_transform(feat_prod),
    columns=mlb.classes_, index=feat_prod.index)
40          return df, dataclust, features, prodotti, feat_prod
41      elif sparse==False:
42          processore=np.unique(matrice_iniz.iloc[:,1]).tolist()
43          processore_ricod=np.arange(len(processore)).tolist()
44          ram=np.unique(matrice_iniz.iloc[:,2]).tolist()
45          ram_ricod=np.arange(len(ram)).tolist()
46          archiviazione=np.unique(matrice_iniz.iloc[:,3]).tolist()
47          archiviazione_ricod=np.arange(len(archiviazione))
48          df=pd.pivot_table(matrice_iniz.iloc[:,range(0, 5)], index="
    product_id", aggfunc="first")
49          df["archiviazione_ricod"]=df["archiviazione"].replace(
    archiviazione, archiviazione_ricod)
50          df["processore_ricod"]=df["processore"].replace(processore,
    processore_ricod)
51          df["ram_ricod"]=df["ram"].replace(ram, ram_ricod)
52          return df
53      else :
54          return print("sparse must be True or False")
55
56  data_cc=mat(listino)
57
58  #recod ram
59  data=vendite
60  ram_recod=data.ram.replace(np.unique(data.ram), np.arange(0, len(np.
    unique(data.ram))))
61  data["ram_recod"]=ram_recod
62  #recod processore
63  processore_recod=data.processore.replace(np.unique(data.processore),
     np.arange(0, len(np.unique(data.processore))))
64  data["processore_recod"]=processore_recod
65  #recod archiviazione
66  archiviazione_recod=data.archiviazione.replace(np.unique(data.
    archiviazione), np.arange(0, len(np.unique(data.archiviazione))))
67  data["archiviazione_recod"]=archiviazione_recod
```

Listing 3: Import e Ricodifica variabili

Dalla riga 1 alla riga 8 del Listing 3 vengono importate le librerie che serviranno in seguito durante lo sviluppo, in particolare si sottolinea l'importanza della libreria *pandas* per quanto riguarda la gestione del dataset e la libreria *sci-kit learn*, e tutte le relative funzioni, per quanto riguarda la componente di clustering dell'algoritmo. Dalla riga 11 alla riga 20 del Listing 3 vengono importati i due dataset simulati (5.2). Vengono eliminate le colonne non utilizzate che si generano automaticamente durante l'import dei dataset.

Alla riga 23 del Listing 3 viene definita la prima funzione, *mat*, utilizzata per ricodificare le variabili che fanno riferimento alle componenti dei computer. Viene fornito in input il listino e il parametro "sparse" di tipo True o False col quale si può specificare se si voglia generare una matrice di tipo sparso o non. In questo caso si utilizzerà sempre la matrice di tipo non sparso e per questo motivo viene impostato il valore predefinito "False" al parametro "sparse". L'output di questa funzione servirà in seguito per il clustering applicato all'intero listino di computer (4.1)
Infine, dalla riga 58 fino alla riga 67 del Listing 3, vengono ricodificate le variabili riguardanti le componenti dei computer che verranno utilizzate in seguito per la prima metodologia di clustering (4.1).
Finita questa parte di script si sono definite tutti i dati che serviranno da ora in poi per applicare la classificazione dei vari prodotti.

5.3.2 Clustering

Di seguito verranno riportati gli script di tre funzioni riguardanti la parte di clustering dell'algoritmo. L'obiettivo è quello di trovare dei gruppi di computer che si diversificano tra di loro per le loro componenti. La prima funzione farà riferimento alla tipologia di clustering in grado di trovare delle similarità tra utenti, attraverso la categoria utente, mentre la seconda permetterà di trovare delle similarità tra prodotti all'interno dell'intero listino fornito dal fornitore, così da andare ad aggiungere dei prodotti nuovi o mai venduti (funzionalità che nella prima funzione non era permessa). Per entrambi i metodi si utilizzeranno delle metriche di valutazione del clustering, per scegliere il numero di gruppi ottimali in cui dividere i prodotti.

```
1  def cluster (data, kmax):
2      df=pd.DataFrame()
3      for i in range(len(np.unique(data["categoria_utente"]))):
4          tmp=data.copy()
5          tmp=tmp[tmp.categoria_utente==i+1].iloc[:,[1, 2, 6, 9, 10,
       11]]
6          #elimino le righe doppie (causa product_id uguale)
7          tmp=tmp.drop_duplicates()
8          #prendo solo le colonne che servono per il cluster
9          #creo un df per la valutazione del K*
10         evaluation=pd.DataFrame(np.zeros((kmax-1,2)), columns=["
       silhouette","k"])
11         #cluster per diversi valori di k (kmax)
12         for x in range(2,kmax+1):
13             model=KMeans(n_clusters=x).fit(tmp.iloc[:, 2:])
14             labels = model.labels_
```

```
            evaluation.iloc[x-2,0]=metrics.silhouette_score(tmp.iloc
    [:, 2:], labels, metric = 'euclidean')
            evaluation.iloc[x-2,1]=x
            #estrazione k* sulla base della silhouette
            max_sil_ind=int(evaluation.iloc[:,0].idxmax())
            k_opt=evaluation.iloc[max_sil_ind,1]
            clust=KMeans(n_clusters=int(k_opt))
            clust.fit(tmp.iloc[:, 2:])
            #estraggo i labels
            index_labels=clust.labels_
            #inserisco la colonna k e product_id in tmp_cl
            tmp["k"]=index_labels
        df=df.append(tmp)

    return df

df=cluster(data, 5)
```

Listing 4: Cluster User-Item

In questa prima funzione vengono definite i k gruppi, attraverso l'algoritmo k-means (4.1), di ogni categoria utente. In particolare si va a vedere, per ogni categoria utente, quale può essere la miglior suddivisione di prodotti sulla base delle diverse componenti e del prezzo di quest'ultimi. La funzione *cluster* prende in input il dataset delle vendite e un numero massimo di gruppi per ogni categoria (kmax). Attraverso la silhouette (4.2.1) si trova il migliore K (numero di gruppi) in grado di dividere in cluster il dataset preso in input; in particolare il numero ottimo di gruppi sarà tale da minimizzare la distanza infra-cluster (punti all'interno del gruppo saranno il più "vicino" possibile) e massimizzare la distanza fra cluster (i cluster tra di loro saranno il più "lontano" possibile). Il punto forte di questa metodologia è il fatto che si tiene conto dell'unico dato relativo all'utente, categoria utente, per trovare delle similarità anche tra utenti e non solo tra item. Il difetto però è che, applicando un sistema di clustering a un set di dati che non comprendono tutti i prodotti disponibili, bensì, tutti i prodotti già venduti, non verranno mai classificati e di conseguenza raccomandati i prodotti che non sono già stati acquistati in passato dagli utenti. Naturalmente nel momento in cui un prodotto viene inserito nel sistema di fatturazione, di conseguenza è stato acquistato, anch'esso verrà classificato dall'algoritmo e avrà la possibilità di essere raccomandato all'utente che effettua la ricerca. In breve, una delle due raccomandazioni che l'algoritmo produce, si basa esclusivamente su di un dataset contenente i prodotti già venduti (sistema di fatturazione, tabella 2), mentre una seconda raccomandazione viene estratta da un set di dati contenenti tutti i prodotti disponibili (listino, tabella 1), già venduti e non.

Per ovviare a questa problematica vengono definite le seguenti due funzioni che invece fanno riferimento al listino, e di conseguenza, integrano la funzione precedente che perdeva delle informazioni importanti relative a determinati prodotti, in particolare non poteva classificare prodotti non acquistati in precedenza in quanto non presenti all'interno del sistema di fatturazione che la funzione prendeva come dato in input.

```
def kcluster(sparse_matrix, kmax=100, method="kmeans", sil=False, rs=42):
    index_labels=["something went wrong"]
    k_opt=0
    kmax=int(kmax)
    if sil==True:
        if method.lower()=="kmeans":
            # creo una matrice vuota
            tmp=pd.DataFrame(np.zeros((kmax-1,2)), columns=["silhouette","k"])
            # creo un dataframe con 2 colonne indice di calinski e k
            for i in range(2,kmax+1):
                model=KMeans(n_clusters=i, random_state=rs).fit(sparse_matrix)
                labels = model.labels_
                tmp.iloc[i-2,0]=metrics.silhouette_score(sparse_matrix, labels, metric = 'euclidean')
                tmp.iloc[i-2,1]=i
            # creo il modello kmeans con il migliore k rispetto all'indice di calinski
            max_sil_ind=int(tmp.iloc[:,0].idxmax())
            k_opt=tmp.iloc[max_sil_ind,1]
            clust=KMeans(n_clusters=int(k_opt))
            clust.fit(sparse_matrix)
            index_labels=clust.labels_


        elif method.lower()=="hierarchical":
            # creo una matrice vuota
            tmp=pd.DataFrame(np.zeros((kmax-1,2)), columns=["silhouette","k"])
            # creo un dataframe con 2 colonne indice di calinski e k
            for i in range(2,kmax+1):
                model=AgglomerativeClustering(n_clusters=i, random_state=rs).fit(sparse_matrix)
                labels = model.labels_
                tmp.iloc[i-2,0]=metrics.silhouette_score(sparse_matrix, labels, metric = 'euclidean')
```

```
                tmp.iloc[i-2,1]=i
                # creo il modello kmeans con il migliore k rispetto
    all'indice di calinski
            max_sil_ind=int(tmp.iloc[:,0].idxmax())
            k_opt=tmp.iloc[max_sil_ind,1]
            clust=AgglomerativeClustering(n_clusters=int(k_opt))
            clust.fit(sparse_matrix)
            index_labels=clust.labels_

        elif method.lower()=="minibatch":
            # creo una matrice vuota
            tmp=pd.DataFrame(np.zeros((kmax-1,2)), columns=["
    silhouette","k"])
            # creo un dataframe con 2 colonne indice di calinski e k
            for i in range(2,kmax+1):
                model=MiniBatchKMeans(n_clusters=i, random_state=rs)
    .fit(sparse_matrix)
                labels = model.labels_
                tmp.iloc[i-2,0]=metrics.silhouette_score(
    sparse_matrix, labels, metric = 'euclidean')
                tmp.iloc[i-2,1]=i
            # creo il modello kmeans con il migliore k rispetto all'
    indice di calinski
            max_sil_ind=int(tmp.iloc[:,0].idxmax())
            k_opt=tmp.iloc[max_sil_ind,1]
            clust=MiniBatchKMeans(n_clusters=int(k_opt))
            clust.fit(sparse_matrix)
            index_labels=clust.labels_

        else:
            print("Warning!\n","The choosen method : ", method,"\n
    Is not a valid method, choose a valid method","\n The available
    methods are: ","\n kmeans","\n hierarchical","\n minibatch")


     elif sil==False:
        if method.lower()=="kmeans":
            # creo una matrice vuota
            tmp=pd.DataFrame(np.zeros((kmax-1,2)), columns=["
    calinski_index","k"])
            # creo un dataframe con 2 colonne indice di calinski e k
            for i in range(2,kmax+1):
                model=KMeans(n_clusters=i, random_state=rs)
                model.fit(sparse_matrix)
                tmp.iloc[i-2,0]=metrics.calinski_harabasz_score(
    sparse_matrix, model.labels_)
```

```
68                  tmp.iloc[i-2,1]=i
69              # creo il modello kmeans con il migliore k rispetto all'
    indice di calinski
70              max_cal_ind=int(tmp.iloc[:,0].idxmax())
71              k_opt=tmp.iloc[max_cal_ind,1]
72              clust=KMeans(n_clusters=int(k_opt))
73              clust.fit(sparse_matrix)
74              index_labels=clust.labels_
75
76          elif method.lower()=="hierarchical":
77              # creo una matrice vuota
78              tmp=pd.DataFrame(np.zeros((kmax-1,2)), columns=["
    calinski_index","k"])
79              # creo un dataframe con 2 colonne indice di calinski e k
80              for i in range(2,kmax+1):
81                  model=AgglomerativeClustering(n_clusters=i,
    random_state=rs)
82                  model.fit(sparse_matrix)
83                  tmp.iloc[i-2,0]=metrics.calinski_harabasz_score(
    sparse_matrix, model.labels_)
84                  tmp.iloc[i-2,1]=i
85              # creo il modello kmeans con il migliore k rispetto all'
    indice di calinski
86              max_cal_ind=int(tmp.iloc[:,0].idxmax())
87              k_opt=tmp.iloc[max_cal_ind,1]
88              clust=AgglomerativeClustering(n_clusters=int(k_opt))
89              clust.fit(sparse_matrix)
90              index_labels=clust.labels_
91
92          elif method.lower()=="minibatch":
93              # creo una matrice vuota
94              tmp=pd.DataFrame(np.zeros((kmax-1,2)), columns=["
    calinski_index","k"])
95              # creo un dataframe con 2 colonne indice di calinski e k
96              for i in range(2,kmax+1):
97                  model=MiniBatchKMeans(n_clusters=i, random_state=rs)
98                  model.fit(sparse_matrix)
99                  tmp.iloc[i-2,0]=metrics.calinski_harabasz_score(
    sparse_matrix, model.labels_)
100                 tmp.iloc[i-2,1]=i
101             # creo il modello kmeans con il migliore k rispetto all'
    indice di calinski
102             max_cal_ind=int(tmp.iloc[:,0].idxmax())
103             k_opt=tmp.iloc[max_cal_ind,1]
104             clust=MiniBatchKMeans(n_clusters=int(k_opt))
105             clust.fit(sparse_matrix)
```

```
106             index_labels=clust.labels_
107
108         else:
109             print("Warning!\n","The choosen method : ", method,"\n
    Is not a valid method, choose a valid method","\n The available
    methods are: ","\n kmeans","\n hierarchical","\n minibatch")
110
111     else:
112         print("The choosen sil:",sil," is not valid \n","sil must be
     True or False")
113
114
115
116     return index_labels, int(k_opt), tmp
117
118
119 def clustofclust(item_df, kmax1=10, kmax2=7, method="kmeans", rs=42,
     sil=True):
120     clust1=kcluster(item_df.iloc[:,[1, 4, 5, 6]], kmax=kmax1, method
    =method, rs=rs, sil=sil)
121     df=pd.DataFrame(np.zeros((len(item_df),3)), index=item_df.index,
     columns=["item_id", "clust1", "clust2"])
122     df.item_id=df.index
123     df.clust1=clust1[0]
124     for i in range(len(np.unique(clust1[0]).tolist())):
125         tmp=kcluster(item_df.loc[df.iloc[:,1]==i].iloc[:,[1, 4, 5,
    6]], kmax=kmax2, sil=sil, method=method, rs=rs)
126         df.loc[df.iloc[:,1]==i, "clust2"]=tmp[0].tolist()
127     df.clust2=[int(i) for i in df.clust2]
128     return df
129
130 df_cc=clustofclust(data_cc)
```

Listing 5: Cluster Listino

Come specificato prima, queste due funzioni fanno riferimento al clustering basato esclusivamente sulle caratteristiche dei prodotti di listino, compresi quei prodotti che non compaiono nel sistema di fatturazioni perché non ancora venduti.
La prima funzione che viene definita è *klucster* attraverso la quale viene definito il metodo di clustering che si preferisce tra i tre implementati (K-means, Hierarchical, e Mini Batch K-means). Naturalmente viene impostato un metodo predefinito, K-means, che però può essere modificato dal programmatore che decide di avviare la funzione di clustering. Come sempre, trattandosi di metodi di classificazione non supervisionati, la scelta del miglior K (numero ottimo di gruppi) viene scelto attraverso la Silhouette media, che rientra tra gli input della funzione. L'input *sil*, appunto,

è un valore True o False nel caso si voglia utilizzare la Silhouette come metrica di valutazione (True, nonché valore predefinito) oppure l'indice di Calinski-Harabasz (False). Tra gli altri input della funzione viene impostato infine un k-max (100), ovvero il numero massimo di gruppi in cui può essere suddiviso il set di dati, anch'esso modificabile.
L'obiettivo di questa funzione è quello di definire un metodo di clustering che successivamente, nella funzione $clustofclust$, verrà implementato.
Passando dunque alla funzione appena citata, si può dire che quest'ultima ha l'obiettivo di applicare un doppio clustering sul set di dati per avere una suddivisione netta all'interno della gamma di prodotti vendibili dalla società. In particolare si andrà a suddividere in gruppi l'output della funzione *mat* che modificava il listino di vendita andando a ricodificare le variabili Ram, Processore e Archiviazione. Anche in questo caso il processo di classificazione viene applicato sulle caratteristiche del computer ricodificate e sul prezzo di vendita.
E' importante precisare come questo processo di clustering dei dati venga applicato una volta al giorno, in orario serale, per non rallentare la raccomandazione finale, che invece avverrà durante il giorno nei confronti dell'utente. Infatti, come si può immaginare facilmente, la componente di classificazione è la più complessa a livello computazionale.

5.3.3 Processo di raccomandazione

Una volta definita la componente statistica di clustering dell'algoritmo, si passa alla raccomandazione finale. In questa parte di script vengono definite due funzioni: la prima adibita alla raccomandazione del prodotto prelevato dal doppio clustering applicato sull'intera gamma di prodotti; la seconda invece ha il compito di raccomandare il prodotto estratto dal processo di clustering basato sulla similarità tra utenti (categoria utente) e similarità tra prodotti (componenti dei PC e prezzo). Come si potrà notare dallo script, la prima funzione viene richiamata nella seconda per avere un output unico e non due raccomandazione provenienti da due funzioni diverse.
Di seguito vengono riportate le due funzioni citate.

```
1 def racc1(data, df1, items):
2         b=pd.DataFrame(columns=["item_id", "clust1", "clust2"])
3         for i in range(len(items)):
4             b = b.append(df1.loc[df1.iloc[:,0]==items[i],:],
    ignore_index=True)
5         b=b.reset_index(drop=True)
6         b=b.drop_duplicates(subset=["clust1","clust2"], keep='last')
    .reset_index(drop=True)
```

```
        out1=[]
        for i in range(b.shape[0]):
            tmp1=df1.loc[df1.iloc[:,1]==b.iloc[i,:][1]]
            tmp2=tmp1.loc[tmp1.iloc[:,2]==b.iloc[i,:][2]]
            tmp3=tmp2.iloc[:,0].tolist()
            tmp5=[x for x in tmp3 if x not in items]
            tmp6=[x for x in tmp5 if data.loc[data.iloc[:, 0]==x].
    iloc[:, 5].values==1]
            out1.append(tmp6)
        racc_list=[]
        racc=[]
        for i in range(len(out1)):
            tmp4=pd.DataFrame()
            for x in range(len(out1[i])):
                tmp4=tmp4.append(data[data.product_id==out1[i][x]].
    iloc[0, 1:7])
            idx_max=tmp4.Anno.idxmax()
            racc_list=data.loc[idx_max, :].tolist()
            item_racc=racc_list[0]
            racc.append(item_racc)
        index_racc=[]
        for i in range(len(racc)):
            index_racc.append(data.loc[data.iloc[:, 0]==racc[i]].
    index.tolist()[0])
        prezzo_item=data.loc[index_racc, "prezzo"].tolist()
        racc_fin=[x for _, x in sorted(zip(prezzo_item, racc), key=
    lambda pair: pair[0])][0]
        return racc_fin
```

Listing 6: Raccomandazione da *clustofclust*

Tramite questa prima funzione *racc*1 si vuole estrarre una singola raccomandazione (come richiesto dalla società che ha commissionato il lavoro) attraverso dei filtri ben precisi che saranno spiegati di seguito.
La prima informazione necessaria per estrarre una raccomandazione è l'id utente (se l'utente è già presente nel sistema di fatturazione), infatti attraverso questo dato si può risalire allo storico degli acquisti della persona definita dall'id e di conseguenza riuscire a identificare il cluster di appartenenza del/i prodotto/i acquistato/i dall'utente, ed andare a inserire quest'ultimi nella lista *items* (input della funzione *racc*1). Una volta fornita la seguente informazione viene estratta una lista di prodotti appartenenti tutti al/i cluster identificante/i i PC già acquistati. Attraverso questo filtro si cerca di definire una preferenza dell'utente nei confronti di determinati prodotti rispetto ad altri. Naturalmente, per evitare di consigliare prodotti già acquistati, questi oggetti vengono eliminati dalla lista creata.

Lo step successivo è quello di filtrare la lista per prodotti disponibili da listino, infatti non avrebbe senso proporre degli oggetti che non possono essere comprati per mancanza di disponibilità.
Un altro filtro molto importante richiesto e applicato, è il prezzo di acquisto. Dovendo estrarre da questa funzione un solo prodotto da raccomandare, si decide di estrarre il prodotto con prezzo inferiore. Naturalmente un filtro di questa natura può essere cambiato in ogni momento, andando a modificare la parte di script a cui fa riferimento.
Come ultimo passo si decide di raccomandare il prodotto più recente in termini di pubblicazione. La fonte di questa informazione la si trova nel listino che fa riferimento ad un anno di pubblicazione preciso. Nell'esempio analizzato, con i dati simulati, si è deciso di inserire un solo anno (2018) e di conseguenza il filtro applicato, relativo alla raccomandazione per novità (tempo più recente), non va a modificare l'ordinamento della lista che si è definita tramite gli step precedenti, che riguardavano la disponibilità del prodotto a listino in primis, e il prezzo inferiore (come da richieste della società affidataria del progetto).
Per far sì che questi criteri non creino conflitti di interessi tra di loro, si esegue la raccomandazione seguendo l'ordine di filtro espresso in precedenza (Cluster, Disponibilità, Prezzo inferiore, Novità).
Una volta eseguiti tutti questi passi si raggiunge un output di un singolo prodotto, che nella funzione prende il nome di $racc\_fin$.
Viene ora riportata la seconda funzione di raccomandazione.

```
def raccommandation(data, df, df1, listino, cat=[], id_utente=[]):


    if cat==[] and id_utente!=[]:
        profilo=[]
        profilo=profilo + np.unique(data.loc[data.iloc[:, 0]==
   id_utente].iloc[:, 2]).tolist()
        profilo=np.unique(profilo).tolist()
        prod_id=data[data["User_id"]==id_utente].product_id.tolist()
        idx_ut=data[data["User_id"]==id_utente].index
        cat_ut=data.iloc[idx_ut]["categoria_utente"].tolist()
        cat_ut_un=np.unique(cat_ut)
        df_racc1=pd.DataFrame()
        #def del df da cui prendere la racc
        for i in range(len(cat_ut_un)):
            df_racc1=df_racc1.append(df[df["categoria_utente"]==
   cat_ut_un[i]])
        #elimino righe con stesso id utente
```

```
17          for p in range(len(prod_id)):
18              df_racc1=df_racc1.drop(df_racc1[df_racc1["product_id"]==
    prod_id[p]].index)
19          df_racc = pd.merge(df_racc1, listino[['product_id', "disp",
    "Anno"]], left_on='product_id', right_on='product_id')
20          # subset per disponibilita
21          df_racc=df_racc[df_racc["disp"]==1]
22
23          # raccomandazoine per prezzo
24          racc=[]
25          for i in range(len(np.unique(df_racc.iloc[:, 0]))):
26              for x in range(len(np.unique(df_racc[df_racc["
    categoria_utente"]==np.unique(df_racc["categoria_utente"])[i]].k)
    )):
27                  tmp_racc=df_racc[(df_racc["categoria_utente"]==np.
    unique(df_racc["categoria_utente"])[i]) &
28                                   (df_racc["k"]==np.unique(df_racc[
    df_racc["categoria_utente"]==np.unique(df_racc["categoria_utente"
    ])[i]].k)[x])]
29
30
31                  #....
32                  idx_min=tmp_racc.prezzo.idxmin()
33                  racc_list_prod=tmp_racc.loc[idx_min, "product_id"].
    tolist()
34                  racc.append(racc_list_prod)
35                  #altra racc per prezzo
36          racc_cc=racc1(listino, df1, profilo)
37
38          index_racc=[]
39          for i in range(len(racc)):
40              index_racc.append(listino.loc[listino.iloc[:, 0]==racc[i
    ]].index.tolist()[0])
41
42          anno_item=listino.loc[index_racc, "Anno"].tolist()
43
44          racc_fin=[x for _, x in sorted(zip(anno_item, racc), key=
    lambda pair: pair[0])][0]
45
46          return racc_fin, racc_cc
47
48      elif cat!=[] and id_utente!=[]: #in cat ci deve essere anche
    quella del prodotto acquistato dall'id utente
49          profilo=[]
50          profilo=profilo + np.unique(data.loc[data.iloc[:, 0]==
    id_utente].iloc[:, 2]).tolist()
```

```
51          profilo=np.unique(profilo).tolist()
52          prod_id=data[data["User_id"]==id_utente].product_id.tolist()
53          idx_ut=data[data["User_id"]==id_utente].index
54          df_racc1=pd.DataFrame()
55          #def del df da cui prendere la racc
56          for i in range(len(cat)):
57              df_racc1=df_racc1.append(df[df["categoria_utente"]==cat[
    i]])
58
59          #elimino righe con stesso id utente
60          for p in range(len(prod_id)):
61              df_racc1=df_racc1.drop(df_racc1[df_racc1["product_id"]==
    prod_id[p]].index)
62          df_racc = pd.merge(df_racc1, listino[['product_id', "disp",
    "Anno"]], left_on='product_id', right_on='product_id')
63          # subset per disponibilita
64          df_racc=df_racc[df_racc["disp"]==1]
65          #raccomandazoine per prezzo
66          racc=[]
67          for i in range(len(np.unique(df_racc.categoria_utente))):
68              for x in range(len(np.unique(df_racc[df_racc["
    categoria_utente"]==np.unique(df_racc.categoria_utente)[i]].k))):
69                  tmp_racc=df_racc[(df_racc["categoria_utente"]==np.
    unique(df_racc.categoria_utente)[i]) &
70                                   (df_racc["k"]==np.unique(df_racc[
    df_racc["categoria_utente"]==np.unique(df_racc.categoria_utente)[
    i]].k)[x])]
71                  idx_min=tmp_racc.prezzo.idxmin()
72                  racc_list=tmp_racc.loc[idx_min, "product_id"].tolist
    ()
73                  racc.append(racc_list)
74          racc_cc=racc1(listino, df1, profilo)
75          index_racc=[]
76          for i in range(len(racc)):
77              index_racc.append(listino.loc[listino.iloc[:, 0]==racc[i
    ]].index.tolist()[0])
78
79          anno_item=listino.loc[index_racc, "Anno"].tolist()
80
81          racc_fin=[x for _, x in sorted(zip(anno_item, racc), key=
    lambda pair: pair[0])][0]
82
83          return [racc_fin, racc_cc]
84
85      elif cat!=[] and id_utente==[]:
86          df_racc1=pd.DataFrame()
```

```
87          #def del df da cui prendere la racc
88          for i in range(len(cat)):
89              df_racc1=df_racc1.append(df[df["categoria_utente"]==cat[
    i]])
90
91          df_racc = pd.merge(df_racc1, listino[['product_id', "disp",
    "Anno"]], left_on='product_id', right_on='product_id')
92
93          # subset per disponibilita
94          df_racc=df_racc[df_racc["disp"]==1]
95          #raccomandazoine per prezzo
96          racc=[]
97          for i in range(len(np.unique(df_racc.categoria_utente))):
98              for x in range(len(np.unique(df_racc[df_racc["
    categoria_utente"]==np.unique(df_racc.categoria_utente)[i]].k))):
99                  tmp_racc=df_racc[(df_racc["categoria_utente"]==np.
    unique(df_racc.categoria_utente)[i]) &
100                                  (df_racc["k"]==np.unique(df_racc[
    df_racc["categoria_utente"]==np.unique(df_racc.categoria_utente)[
    i]].k)[x])]
101                 idx_min=tmp_racc.prezzo.idxmin()
102                 racc_list=tmp_racc.loc[idx_min, "product_id"].tolist
    ()
103                 racc.append(racc_list)
104         index_racc=[]
105         for i in range(len(racc)):
106             index_racc.append(listino.loc[listino.iloc[:, 0]==racc[i
    ]].index.tolist()[0])
107
108         anno_item=listino.loc[index_racc, "Anno"].tolist()
109
110         racc_fin=[x for _, x in sorted(zip(anno_item, racc), key=
    lambda pair: pair[0])][0]
111         return [racc_fin]
112
113     elif cat==[] and id_utente==[]:
114         print("Please insert at least one product category")
```

Listing 7: Raccomandazione da *cluster*

La funzione descritta nel Listing 7, che si basa sulla similarità tra utenti, ultima la raccomandazione definendo 3 tipologie di ricerche che l'utente potrebbe effettuare.

- Caso in cui l'utente inserisce solo il proprio id utente: in questa situazione l'algoritmo è in grado di risalire alla categoria utente di appartenenza, e di conseguenza, andrà a raccomandare un prodotto appartenente alla stessa cate-

goria utente, seguendo gli stessi principi di filtro definiti nella funzione *racc*1. In questo caso la componente di clustering viene a meno, infatti dovendo raccomandare un solo prodotto, la raccomandazione non si diversifica per tipologia di prodotto, ma solo per tipologia di utente. Nel caso in cui si volesse raccomandare più di due prodotti, la raccomandazione si potrebbe basare sui cluster di appartenenza dei vari prodotti, già filtrati per categoria utente, così da andare a sottolineare la differenza di PC che c'è all'interno di una singola categoria utente;

- Caso in cui l'utente inserisce sia il proprio id utente, che una categoria utente (che sarà diversa dalla sua di appartenenza): in questo caso la raccomandazione si basa esclusivamente sulla categoria utente che viene inserita dall'utente. Infatti si immagina che un utente che inserisce una categoria utente precisa, abbia necessità di acquistare dei prodotti che fanno parte di quella categoria. L'id utente in questione viene richiesto per poter accedere alla lista di prodotti precedentemente acquistati dall'utente ed eliminarli dalla lista di possibili raccomandazione finali. Come è stato specificato in precedenza, questa tipologia di ricerca dà la precedenza alla categoria utente, mentre l'id utente viene utilizzato per non raccomandare prodotti già acquistati. Per quanto riguarda il processo di raccomandazione, si fa sempre riferimento a ciò che è stato specificato sotto lo script "Raccomandazione da $clustofclust$";

- Caso in cui viene inserita solo la categoria utente: in questo caso si tratta di un utente nuovo, senza uno storico di acquisti e di conseguenza senza un id utente. Per questo motivo la ricerca può essere effettuata esclusivamente attraverso la categoria utente (rese note dall'azienda), in modo tale da indicare una linea guida all'utente che non può essere descritto dal suo id utente. Nel caso appena descritto, la raccomandazione può fare affidamento esclusivamente alla categoria di utente inserito dall'utente e di conseguenza la raccomandazione non farà riferimento al listino dei prodotti ($clsutofclust$) ma solo al sistema di fatturazione ($cluster$) all'interno del quale è specificata la categoria utente;

- Caso in cui non viene inserito né l'id utente né la categoria utente: in questa situazione il sistema di raccomandazione sviluppato fino ad ora non sarebbe funzionale, in particolare il sistema ha sempre bisogna di una delle due variabili in input (Categoria Utente o Id Utente), senza di essere non sarebbe possibile andare ad identificare l'utente che effettua la ricerca e di conseguenza la raccomandazione non soddisferebbe le necessità dell'utente.

Nella funzione sopra riportata viene inoltre definito un parametro *profilo* che fa riferimento allo storico degli acquisti dell'id utente che, se presente, richiede la raccomandazione.
Questa tipologia di raccomandazione è stata sviluppata attraverso delle precise richieste della società che ha commissionato il progetto. I fattori modificabili nel processo di raccomandazione possono però essere molteplici, dal numero di raccomandazione da presentare in output, piuttosto che le modalità di ordine con cui i prodotti vengono inseriti in lista (prezzo max, prezzo medio, prodotto più venduto, ecc.), o ancora aggiungere una raccomandazione che non ha bisogno degli input nominati in precedenza, bensì estrarrebbe i prodotti più venduti e consiglierebbe esclusivamente quelli, senza conoscere l' esigenze del cliente.
Queste sono ipotesi di come il ranking dei risultati potrebbe cambiare, ma naturalmente ogni ipotesi andrebbe studiate e accettata dalla società che ha commissionato il lavoro, potrebbe risultare che non tutte le ipotesi formulate siano consone al sistema di raccomandazione che si vuole produrre.

5.3.4 Risultati

Come ultimo paragrafo della descrizione di questo algoritmo, si riportano alcuni risultati ottenuti attraverso dei tentativi svolti dagli sviluppatori.
L'interfaccia che si è creata vuole dare la possibilità di avvicinarsi il più possibile a una possibile interfaccia di un sito e-commerce attraverso il quale l'utente può decidere che tipologia di ricerca avviare.

Esempio 1

ACQUISTI PRECEDENTI							
	Product Id	**Processore**	**Ram**	**Archiviazione**	**Prezzo**	**Disp**	**Anno**
Acq. 1	5447	i7	32	3072	€1.249,57	1	2019
Acq. 2	9861	R5	8	1024	€938,66	0	2019
RACCOMANDAZIONI							
	Product Id	**Processore**	**Ram**	**Archiviazione**	**Prezzo**	**Disp**	**Anno**
Racc. 1	9794	i5	4	1024	€704,27	1	2018
Racc. 2	98	i5	4	256	€948,23	1	2019

Tabella 3: Raccomandazione tramite id utente

In questo esempio l'utente decide di fare una ricerca attraverso il proprio id utente (31) siccome in passato aveva già acquistato presso lo stesso rivenditore. L'algoritmo restituisce in output un riepilogo degli acquisti passati ("Acquisto Precedenti") e successivamente elenca le due raccomandazioni ("Raccomandazione/i"):

- La prima raccomandazione fa riferimento al prodotto ricercato all'interno della stessa categoria utente;
- La seconda raccomandazione fa riferimento al prodotto simile a quelli comprati in precedenza dall'utente.

I due prodotti raccomandati in questo caso sono molto simili a quelli acquistati in precedenza dall'utente, ciò significa che l'algoritmo di clustering effettuato ha funzionato e di conseguenza si ritrova similarità tra prodotti acquistati e raccomandati.

Esempio 2

ACQUISTI PRECEDENTI							
	Product Id	**Processore**	**Ram**	**Archiviazione**	**Prezzo**	**Disp**	**Anno**
Acq. 1	3033	i7	16	3072	€1.424,44	1	2019
Acq. 2	4821	R9	16	2048	€2.372,5	1	2019
RACCOMANDAZIONI							
	Product Id	**Processore**	**Ram**	**Archiviazione**	**Prezzo**	**Disp**	**Anno**
Racc. 1	3528	R3	8	256	€456,24	1	2018
Racc. 2	94	i7	16	2048	€1.407,21	1	2019

Tabella 4: Raccomandazione tramite id utente e categoria utente

In questo secondo esempio viene inserito dall'utente sia il proprio id (241), appartenente alla categoria utente 6, che una categoria utente (3) all'interno della quale effettuare la ricerca. Come si può notare lo schema di presentazione dell'interfaccia è uguale al primo caso, la differenza sta nella raccomandazione che non fa più riferimento agli acquisti passati, bensì fa riferimento alla categoria utente inserita.

Esempio 3

ACQUISTI PRECEDENTI							
	Product Id	**Processore**	**Ram**	**Archiviazione**	**Prezzo**	**Disp**	**Anno**
-	-	-	-	-	€-	-	-
RACCOMANDAZIONI							
	Product Id	**Processore**	**Ram**	**Archiviazione**	**Prezzo**	**Disp**	**Anno**
Racc. 1	5222	i9	64	2048	€1.837,73	1	2019

Tabella 5: Raccomandazione tramite categoria utente

Nell'ultimo esempio, caso in cui viene inserita solo la categoria utente (9), la raccomandazione è singola. Questo risultato lo si ottiene dal momento in cui, senza l'id utente, l'unica raccomandazione proposto fa riferimento alla similarità tra utenti, ovvero i prodotti venduti all'interno di una stessa categoria utente.
Di conseguenza la raccomandazione non terrà conto di prodotti non venduti in passato, per il semplice motivo che un prodotto è assegnato a una categoria utente solamente una volta che viene acquistato.
Queste tre brevi tabelle contenenti i risultati non possono ovviamente essere definite delle verifiche sistematiche dell'algoritmo, bensì hanno lo scopo di riassumere in breve le funzionalità e il prodotto dell'algoritmo che, come si può notare dai risultati appena mostrati (tabelle 3, 4 e 5), funziona e soddisfa le esigenza iniziale del cliente richiedente il sistema di raccomandazione.

6 Conclusione e sviluppi futuri

Il lavoro descritto in questo elaborato di tesi fornisce un esempio sull'utilizzo degli algoritmi di clustering all'interno dei sistemi di raccomandazione, in particolare attraverso un Recommender System ibrido che combina sia l'approccio Content Based sia l'approccio Collaboritve Filtering. In secondo luogo si è cercato di creare, partendo da un set di dati simulati, un sistema di raccomandazione basato sul raggruppamento dei prodotti e degli utenti a disposizione.
Nel caso in questione si è andati a lavorare nell'ambito dei siti e-commerce, in particolare nella categoria elettronica e informatica. L'algoritmo implementato consente, all'interno di cataloghi estremamente ricchi di prodotti, di trovare il prodotto più adatto alle esigenze del cliente.
L'analisi è partita dallo studio dei dati reali dai quali in seguito si sono generati i dataset simulati di Listino e Vendite. La scelta di utilizzare dati simulati nasce dal fatto che il sistema di fatturazione, al momento della commissione del lavoro, non era ancora collegabile, con i listini delle società fornitrici di computer; senza questo importante passaggio non era possibile procedere con lo sviluppo del sistema con i dati reali, per questo motivo si sono simulati dei dati che replicassero il sistema di fatturazione come sarebbe dovuto diventare in futuro per essere applicabile al progetto. Per quanto riguarda invece il listino dei prodotti, si è deciso di simularlo a causa del problema precedentemente citato, l'impossibilità di collegarlo attraverso identificatori al sistema di fatturazione e di conseguenza l'impossibilità di utilizzarlo come dataset per il sistema di raccomandazione.
I dati simulati hanno avuto l'unico scopo di testare e allenare l'algoritmo sviluppato, per poi in futuro applicarlo ai dati reali.
Dopo questa prima parte di analisi dei dati, ci si è concentrati sull'utilizzo dei modelli di clustering più importanti e citati in letteratura, K-Means, Hierarchical e Mini-Batch. Attraverso delle metriche di valutazione della separazione dei cluster, Silhouette e Indice di Calinski-Harabasz, si è giunti alla conclusione che i gruppi trovati all'interno dei dataset siano adeguatamente separati per procedere col sistema di raccomandazione. Si ipotizza, e i primi risultati non contraddicono tale ipotesi, che i prodotti che vengono raccomandati appartengono ai dei cluster sensati, sulla base di un giudizio di similarità tra prodotti comprati e raccomandati .
Infine, nell'elaborato viene descritto il sistema di raccomandazione sviluppato in Python. Per quanto il sistema creato non è ancora stato testato nella realtà dalla società proprietaria, gli output che si ottengono dal sistema evidenziano una certa similarità tra prodotti comprati in precedenza e prodotti raccomandati, andando a sottolineare quello che era l'obiettivo del progetto vale a dire creare un sistema di

raccomandazione in grado di soddisfare al meglio le richieste del cliente sulla base dei sui acquisti passati e sulla base della sua identificazione a livello lavorativo (tipologia di lavoro svolto), nonché unico dato relativo alla persona e non ai suoi acquisti.
Alcuni miglioramenti e sviluppi futuri potrebbero riguardare prendere in considerazione ulteriori caratteristiche degli utenti per mezzo di ulteriori metadati. Un utente descritto non solo dalla categoria lavorativa, può essere classificato in maniera differente e migliore all'interno del sistema di raccomandazione e, di conseguenza, le raccomandazioni potranno essere più precise e corrette.
Naturalmente l'inserimento del Recommender System descritto in questo elaborato all'interno del sistema e-commerce della società richiedente il progetto, potrebbe avviare un processo virtuoso di perfezionamento della qualità dei dati raccolti e degli algoritmi impiegati, che nel tempo innalzerebbe la qualità delle raccomandazioni effettuate.

Riferimenti bibliografici

[1] Probabilistic Robotics
Sebastian Thrun, Wolfram Burgard Dieter Fox
2005

[2] J. B. MacQueen (1967): "Some Methods for classification and Analysis of Multivariate Observations", Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability, Berkeley, University of California Press

[3] Characterization and evaluation of similarity measures for pairs of clusterings
Darius Pfitzner, Richard Leibbrandt, David Martin Ward Powers
June 2009 Knowledge and Information Systems

[4] Finding groups in data : an introduction to cluster analysis
Kaufman, Leonard ; Rousseeuw, Peter J
New York etc : Wiley, 1990

[5] Recommender Systems and Collaborative Filtering
Fernando Ortega e Ángel González-Prieto
2020

[6] Recommender Systems Handbook
Francesco RicciLior RokachBracha ShapiraPaul B. Kantor
2011

[7] Thomas M. Mitchell. 1997. Machine Learning (1st. ed.). McGraw-Hill, Inc., USA

[8] Johannes Ledolter, Data Mining and Business Analytics with R, John Wiley and Sons, 2013

[9] Andrew R. Webb, Keith D. Copsey, Statistical Pattern Recognition, Third Edition, John Wiley and Sons, 2011

[10] Pang-Ning Tan, Michael Steinbach, Vipin Kumar, Introduction to Data Mining, Pearson Education, 2006

[11] E. B. Fowlkes, C. L. Mallows A Method for Comparing Two Hierarchical Clusterings, Journal of the American Statistical Association 78.383 pp.553-69, 1983