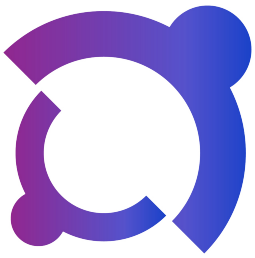


ZaMatcher
Modulo di Machine Learning

Repository github

`https://github.com/sl1mSha4dey/ZaMatcher`

OpenMeet Team

Angelo Nazzaro	0512110391
Roberto Della Rocca	0512110802
Yuri Brandi	0512109740
Francesco Granozio	0512111903

13 febbraio 2023

Indice

Glossario . . . 3

1 Introduzione **4**
1.1 Cos'è ZaMatcher? . . . 4
1.2 Definizione del problema . . . 4
1.2.1 Obiettivi . . . 4
1.2.2 Specifica P.E.A.S. . . . 4
1.2.3 Caratteristiche dell'ambiente . . . 5
1.3 Analisi del problema . . . 5

2 Il Machine Learning **7**
2.1 Cos'è il Machine Learning? . . . 7
2.2 CRISP-DM . . . 8
2.3 Business Understanding . . . 9
2.4 Data Understanding . . . 10
2.4.1 Quali dati raccogliere? . . . 10
2.4.1.1 Digressione sull'orientamento e identità di genere 11
2.4.2 Come raccogliere i dati? . . . 12
2.4.3 Struttura del dataset . . . 12
2.5 Data Preparation . . . 14
2.5.1 Data Cleaning . . . 14
2.5.2 Feature Scaling . . . 17
2.5.3 Feature Extraction . . . 18
2.5.4 Data Balancing . . . 18
2.5.5 Principal Component Analysis . . . 19
2.6 Modeling . . . 21
2.6.1 Clustering . . . 21
2.6.2 Gli algoritmi di clustering . . . 21
2.6.3 K-means . . . 22
2.6.3.1 Somma degli errori quadrati . . . 22
2.6.3.2 Passaggi e metriche . . . 23
2.6.3.3 Implementazione e addestramento . . . 23
2.6.4 DBSCAN . . . 25
2.6.4.1 Come stimare i valori dei parametri? . . . 25
2.6.4.2 Implementazione e addestramento . . . 26

2.6.5 Clustering Gerarchico Agglomerativo 29
2.6.5.1 Passaggi e metriche 29
2.6.5.2 Implementazione e addestramento 30
2.6.6 Classificazione 36
2.6.7 Gli algoritmi di classificazione 36
2.6.8 Training set e Test set 37
2.6.9 Random Forest Classifier 37
2.6.9.1 Come funziona un decision tree? 37
2.6.9.2 Implementazione e addestramento 38
2.6.10 KNN Classifier 39
2.6.10.1 Implementazione e addestramento 39
2.6.11 SVC Classifier 40
2.6.11.1 Implementazione e addestramento 40
2.6.12 MLP Classifier 40
2.6.12.1 Implementazione e addestramento 41
2.6.13 Gradient Boosting Classifier 41
2.6.13.1 Implementazione e addestramento 41
2.6.14 Grid Search 41
2.6.14.1 Random Forest con Grid Search 42
2.6.14.2 KNN con Grid Search 42
2.6.14.3 SVC con Grid Search 43
2.6.14.4 Digressione sui parametri usati nella Grid Search 43
2.7 Evaluation 44
2.7.1 Cross-validation 44
2.7.2 Come valutare un classificatore? 45
2.7.2.1 Cos'è una matrice di confusione? 45
2.7.2.2 Metriche 46
2.7.3 Valutazione 47
2.7.4 Esportazione del modello 48
2.8 Deployment 48

3 Conclusioni **52**
3.1 Cos'abbiamo imparato? 52
3.2 Obiettivi futuri 52
Ringraziamenti 54

Glossario

Termine	**Definizione**
Meeter	È l'utente finale dell'app mobile OpenMeet.
Match	Corrispondenza di valutazioni positive tra due utenti.
GPL	GNU General Public License.

Capitolo 1

Introduzione

1.1 Cos'è ZaMatcher?

OpenMeet è un applicativo **mobile** per incontri, **open-source** e pubblicato sotto **licenza GPL v3.0**.
In OpenMeet, gli utenti, detti **Meeters**, valutano gli altri profili utente esprimendo una valutazione positiva (**like**) o negativa (**skip**). OpenMeet utilizza un sistema **double opt-in** dove entrambi gli utenti devono scambiarsi una valutazione positiva (**Match**) prima di poter iniziare a messaggiare.

ZaMatcher è l'intelligenza artificale di OpenMeet atta a migliorare e personalizzare gli utenti proposti ad un Meeter apprendendo continuamente i suoi gusti nel tempo attraverso le sue valutazioni.

1.2 Definizione del problema

1.2.1 Obiettivi

Lo **scopo** del progetto è quello di realizzare un **agente intelligente capace di apprendere**, che sia in grado di consigliare ad un Meeter un elenco di Meeter compatibili sulla base delle proprie preferenze ed interessi;

1.2.2 Specifica P.E.A.S.

Di seguito è riportata la descrizione P.E.A.S. dell'ambiente operativo.

- **Performance**
  La misura di prestazione adottata dall'agente è la sua capacità di avvicinarsi quanto più possibile ad una situazione ideale nella quale i nuovi utenti proposti al Meeter saranno valutati sempre positivamente.

- **Environment**
  L'ambiente in cui opera l'agente è l'insieme dei Meeter costituito dai dati dei singoli utenti, dalle loro preferenze e dai loro interessi.
- **Actuators**
  L'agente agisce sull'ambiente fornendo un elenco di suggerimenti.
- **Sensors**
  L'agente percepisce l'ambiente tramite i dati immessi dall'utente in fase di registrazione.

1.2.3 Caratteristiche dell'ambiente

L'ambiente operativo individuato è:

- **Completamente Osservabile**
  L'ambiente è completamente osservabile in quanto l'agente è a conoscenza dei dati di tutti i Meeters.
- **Deterministico**
  L'ambiente è deterministico in quanto il suo stato successivo è determinato dalle azioni eseguite dall'agente.
- **Episodico**
  L'ambiente è episodico poiché i suggerimenti forniti dall'agente non influenzeranno le sue azioni future.
- **Dinamico**
  L'ambiente è dinamico in quanto nel corso dell'elaborazione dell'agente, un Meeter potrebbe cambiare le proprie preferenze.
- **Discreto**
  L'ambiente è discreto in quanto le caratteristiche dell'insieme dei dati e le azioni attuabili all'interno di esso sono finite.
- **Singolo Agente**
  L'ambiente è singolo agente perché l'unico agente che opera nell'ambiente è l'agente stesso.

1.3 Analisi del problema

Il problema sarebbe potuto essere approcciato sviluppando un algoritmo di ricerca per tags basato sugli interessi dei singoli Meeter. Tale soluzione, per quanto semplice, presentava varie criticità:

- Non sarebbe stato possibile fornire un suggerimento 'veriterio' in quanto l'algoritmo non avrebbe tenuto conto di altri fattori quali la distanza, l'età e l'orientamento sessuale del Meeter;

- Non sarebbe stato possibile tenere conto del differente peso delle correlazioni dei precedenti fattori;
- Col crescere del quantitativo di dati, la computazione richiesta sarebbe stata troppo onerosa sia dal punto di vista temporale che spaziale;

Per tali motivazioni, si è deciso di adottare un **modello di machine learning** per affrontare il problema. Nello specifico, si è deciso di trattare il problema combinando tecniche di **apprendimento non supervisionato** ed **apprendimento supervisionato**. Volendo fornire un'anticipazione sull'analisi della soluzione proposta, sono stati utilizzati algoritmi basati sui concetti di **clustering** e **classificazione**.

L'idea alla base è stata quella di suddividere gli utenti in gruppi con interessi e posizione simili, denotando le caratteristiche più comuni di ogni gruppo tramite il clustering e consigliare un elenco di utenti ad ogni Meeter in base al gruppo di appartenenza tramite la classificazione.
Le scelte implementative e le motivazioni dietro di esse saranno approfondite nelle sezioni successive.

Capitolo 2

Il Machine Learning

2.1 Cos'è il Machine Learning?

Il **Machine Learning** (**ML**) è una branca dell'**Intelligenza Artificiale** (**IA**) che si occupa dello studio e della costruzione di algoritmi che siano in grado di **imparare dai dati**.

I principali tipi di algoritmi di machine learning attualmente utilizzati sono:

- **Algoritimi di Apprendimento Supervisionato**
  Gli algoritmi di machine learning supervisionato sono i più utilizzati. Con questo modello, un data scientist agisce da guida e insegna all'algoritmo i risultati da generare. Esattamente come un bambino impara a identificare i frutti memorizzandoli in un libro illustrato, nel machine learning supervisionato **l'algoritmo apprende da un set di dati già etichettato e con un output predefinito**.

- **Algoritmi di Apprendimento Non Supervisionato**
  Il machine learning non supervisionato utilizza un approccio più indipendente, in cui un computer impara a identificare processi e schemi complessi senza la guida attenta e costante di una persona. Il machine learning non supervisionato implica una **formazione basata su dati privi di etichette e per i quali non è stato definito un output specifico**.

- **Algoritmi di Apprendimento Semi-Supervisionato**
  Gli algoritmi di machine learning semi-supervisionato implicano una formazione basata su **dati misti, alcuni etichettati ed altri non etichettati**.

- **Algoritmi di Apprendimento Per Rinforzo**
  I modelli di machine learning di apprendimento per rinforzo compiono azioni in maniera sequenziale e, al termine di ogni sequenza, gli viene assegnata una **ricompensa** che ha lo scopo di incoraggiare comportamenti corretti.

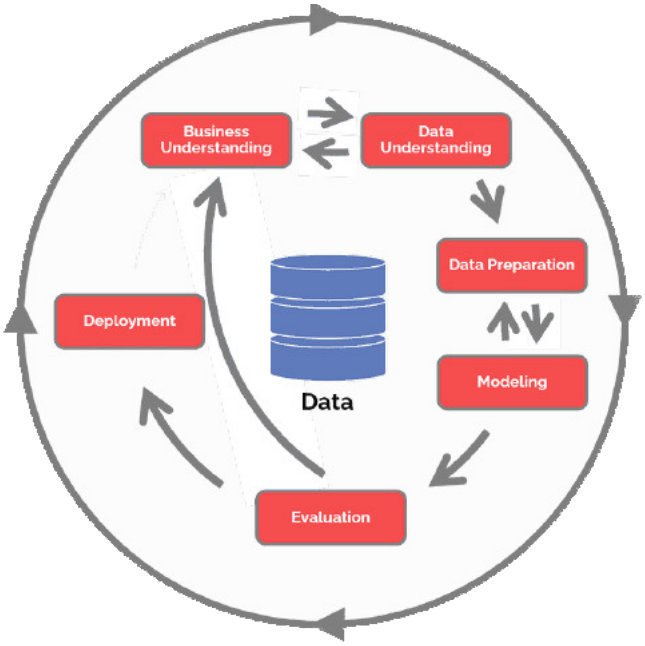
2.2 CRISP-DM

Nello sviluppo di qualsiasi sistema software è **estremamente importante** utilizzare un approccio ingegneristico poiché i sistemi software non adeguatamente ingegnerizzati falliscono nel 94% dei casi, rischiando di danneggiare cose, persone e ambiente circostante.

Quando si progetta una soluzione basata su machine learning è essenziale pensare all'approccio ingegneristico da usare sia dal punto di vista del software sia dal punto di vista dei dati. L'approccio più comunemente usato è quello fornito dal modello **CRISP-DM**.

Il modello **CRISP-DM** (**Cross-Industry Standard Process For Data Mining**) fornisce una rappresentazione del ciclo di vita di progetti basati su intelligenza artificiale e data science. È suddiviso in **sei fasi non sequenziali che possono essere eseguite un numero illimitato di volte**. Questa sua caratteristica lo rende un modello al contempo tradizionale e flessibile.

Nelle sezioni successive andremo ad analizzare le varie fasi del CRISP-DM e cosa è stato fatto in ognuna di esse.



2.3 Business Understanding

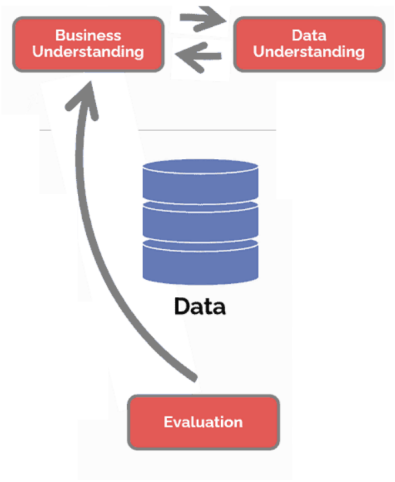
La prima fase prevista dal CRISP-DM è la fase di **Business Understanding** durante la quale si raccolgono i requisiti, si definiscono gli obiettivi di business e tecnici, la disponibilità delle risorse, si stimano i rischi, i costi e si selezionano le tecnologie e i tool necessari.

- **Obiettivi di business**
  Come introdotto nella sezione 1.2.1, l'obiettivo è quello di realizzare un modello di machine learning in grado di raccomandare un elenco di Meeters compatibili ad un utente prendendo in considerazione gli interessi e le preferenze.

- **Disponibilità delle risorse**
  Per costruire ed addestrare il modello di machine learning abbiamo necessità di un dataset dal quale il modello può apprendere. Dopo una serie di ricerche, purtroppo, infruttifere, si è deciso di optare per la creazione di un dataset personalizzato.

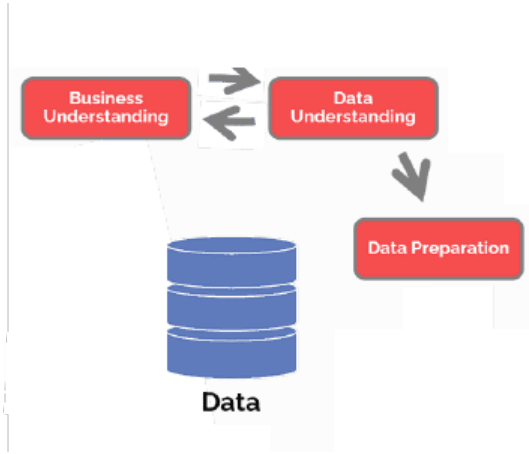
- **Stima dei rischi**
  La creazione di un dataset personalizzato rappresenta un rischio di particolare rilevanza siccome comporta la raccolta e la costruzione di dati non verificati.

- **Tecnologie e tool**
  Per l'acquisizione dei dati si è deciso di utilizzare OpenAI [1] tramite il **conversational agent ChatGPT** [2] e la libreria python **Faker** [3]. Per l'analisi, la modellazione dei dati e per l'implementazione e l'addestramento del modello di machine learning abbiamo usato **Python** in combinazione con le librerie **pandas** [4] , **seaborn** [5], **numpy** [6], **sklearn** [7], **yellowbrick** [8], **kneed** [9], **joblib** [10] e **Flask** [11].



2.4 Data Understanding

Sulla base degli obiettivi definiti nella fase precedente, il secondo passo consiste nell'identificazione, collezione e analisi dei dataset che possono portare al raggiungimento degli obiettivi.



L'intelligenza artificiale, in particolare il machine learning, è una scienza che richiede un ampio utilizzo di dati e *senza dati non si cantano messe*!

Non nascondiamo che la creazione di un dataset, senza alcuna base di partenza, sia stata un'impresa alquanto criptica. I metodi per la creazione del dataset che avevamo a disposizione erano due:

- Integrare ZaMatcher all'interno di OpenMeet procedendo alla raccolta dei dati degli utenti registrati sulla piattaforma;
- Generare un dataset pseudo-casuale;

A causa della **mancanza di risorse**, **tempo** e **dell'incertezza sulla quantità di dati effettivamente collezionabile** si è deciso di optare per la generazione di un dataset pseudo-casuale.

2.4.1 Quali dati raccogliere?

Per garantire il corretto funzionamento del modello di machine learning, sono state individuate le seguenti features:

- **Name**
  È il nome del Meeter.
  Non è rilevante ai fini dell'addestramento del modello.
- **Surname**
  È il cognome del Meeter.
  Non è rilevante ai fini dell'addestramento del modello.

- **Gender**
  È il genere del Meeter. Può assumere i seguenti valori:
  - **M**: Male
  - **F**: Female
  - **N**: Non Binary
- **Searching Gender**
  Rappresenta il genere verso il quale l'utente è interessato. Può assumere i seguenti valori:
  - **M**: Male
  - **F**: Female
  - **B**: Both (Male & Female)
  - **N**: Non Binary
  - **A**: All
- **Age**
  È l'età del Meeter; varia tra 18 e 40.
- **Interests**
  È l'insieme degli interessi e/o degli hobbies posseduti dal Meeters. Il numero di interessi posseduti varia tra 3 e 6.
- **Latitude**
  Rappresenta la latitudine del luogo in cui si trova il Meeter.
  Viene usata assieme alla longitudine per raggruppare i Meeter in base alla posizione.
- **Longitude**
  Rappresenta la longitude del luogo in cui si trova il Meeter.
  Viene usata assieme alla latitudine per raggruppare i Meeter in base alla posizione.

2.4.1.1 Digressione sull'orientamento e identità di genere

Con una sempre crescente attenzione nei confronti della comunità LGBTQIA+, nella scelta della raccolta dei dati è stato necessario prendere delle scelte etiche che mirassero all'**inclusività**, senza però "etichettare" troppo gli utenti, correndo il rischio di discriminarli.

Ad esempio: considerando che ci siano persone di genere maschile, non interessate a persone transessuali indipendentemente dal loro genere, al fine di migliorare l'esperienza della prima categoria di utenti potrebbe sembrare una buona idea etichettare un utente come "transessuale". Tuttavia richiedere ad un utente di immettere un'informazione di questo tipo, cambiando anche la sua esperienza sulla piattaforma (incontrerà persone differenti), creerebbe una forte **sensazione di discriminazione**.

L'aspetto etico nella raccolta dei dati di un utente, rappresenta quindi un importante **trade-off** rispetto all'efficacia nel clustering degli utenti.

2.4.2 Come raccogliere i dati?

La raccolta, o meglio, la generazione dei dati è stata effettuata combinando diversi strumenti e tecnologie, come già anticipato a 2.3. Nello specifico, sono stati utilizzati **OpenAI**, **Python** e la libreria **Faker**.

OpenAI è stata utilizzata per generare l'insieme degli **interessi** tramite il **conversational agent ChatGPT**. Sono stati generati 156 interessi distinti.

La libreria Faker è stata usata per la generazione di nomi e cognomi fittizi che fossero adeguati al genere dell'utente.

Per la raccolta delle coordinate geografiche è stato usato un dataset contenente più di 30.000 città americane [10].

2.4.3 Struttura del dataset

Il dataset è stato generato seguendo un formato tabellare per un totale di 5.000 righe:

name	surname	gender	searching_gender	age	interests	latitude	longitude
Alice	Smith	N	A	32	Art, Painting	36.5367	-95.9264
...	...	...	...	...	...	...	...

Tabella 2.1: Rappresentazione tabulare del dataset

Di seguito è riportato lo script in python per la generazione del dataset:

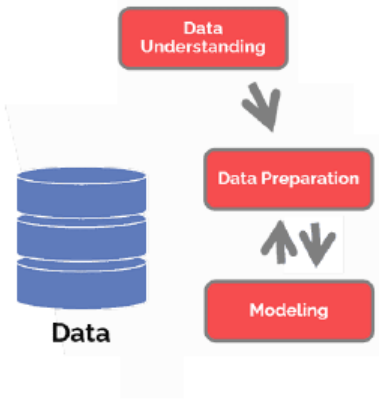
```python
import pandas as pd
import random
from faker import Faker

# CONSTANTS SECTION
# M: Male, F: Female, B: Bisexual, N : Non Binary, A: All
GENDER_MALE, GENDER_FEMALE, GENDER_NON_BINARY, = "M", "F", "N"
SEARCHING_GENDER = ["M", "F", "B", "N", "A"]
MIN_AGE, MAX_AGE = 18, 40
INTEREST_MIN_SIZE, INTEREST_MAX_SIZE = 3, 6
DS_SIZE = 5_000

faker = Faker()

# load datasets
# taken from https://simplemaps.com/data/us-cities
cities_df = pd.read_csv("./us_cities.csv")
interests = pd.read_csv("./interests.csv")['interest'].tolist()

ds_list = []

```

```python
for i in range(DS_SIZE):
    row = {}

    match random.randrange(1, 4):
        case 1:
            row["name"] = faker.first_name_male()
            row["surname"] = faker.last_name_male()
            row["gender"] = GENDER_MALE
        case 2:
            row["name"] = faker.first_name_female()
            row["surname"] = faker.last_name_female()
            row["gender"] = GENDER_FEMALE
        case 3:
            row["name"] = faker.first_name_nonbinary()
            row["surname"] = faker.last_name_nonbinary()
            row["gender"] = GENDER_NON_BINARY

    row["searching_gender"] = random.choice(SEARCHING_GENDER)
    row["age"] = random.randrange(MIN_AGE, MAX_AGE + 1)
    row_interests_size = random.randrange(INTEREST_MIN_SIZE,
    INTEREST_MAX_SIZE + 1)

    row['interests'] = ', '.join(random.sample(interests,
    row_interests_size))
    cities_random_idx = random.randrange(cities_df.index.start,
    cities_df.index.stop)
    row['latitude'] = cities_df['lat'][cities_random_idx]
    row['longitude'] = cities_df['lng'][cities_random_idx]

    ds_list.append(row)

random.shuffle(ds_list)
df = pd.DataFrame(ds_list)

print(f"Gender distribution: \n{df['gender'].value_counts(normalize=
    True) * 100}")
print(f"Searching Gender distribution: \n{df['searching_gender'].
    value_counts(normalize=True) * 100}")

df.to_csv('dataset.csv', index=False, header=True)
```

Listing 2.1: create_dataset

2.5 Data Preparation

L'obiettivo di questa fase è quello di preparare i dati in maniera tale che possano essere utilizzati nei successivi passi del processo. Questo include un processo fondamentale che è noto come **feature engineering**, ovvero la selezione delle caratteristiche del problema che hanno maggiore potenza predittiva. Inoltre, questa fase include l'implementazione dei processi di pulizia dei dati sulla base dei problemi di qualità riscontrati nella fase precedente.



2.5.1 Data Cleaning

La fase di *data cleaning* è il processo di **correzione** o **rimozione** di **dati errati**, **corrotti**, **formattati in modo errato**, **duplicati** o **incompleti** all'interno di un dataset. Ha come obiettivo quello di fornire un dataset dotato di una qualità ritenuta adeguata per poter procedere con le fasi successive.

Nel nostro caso è stato necessario effettuare un processo di rimozione degli **outliers** per ottenere una corretta distribuzione degli utenti in base alla locazione solo per scopi relativi all'analisi dei dati.
Un outlier è un punto dei dati notevolmente diverso dal resto. Rappresenta errori nella misurazione, cattiva raccolta dei dati o semplicemente mostra variabili non considerate durante la raccolta dei dati. L'individuazione e la rimozione degli outliers è stata effettuata sulle colonne `latitude` e `longitude` tramite l'uso di **boxplots**, **violinplots** e calcolando l'**IQR** (**InterQuartile Range**). Di seguito sono riportati i violinplots e i boxplots della latitudine e della longitudine:

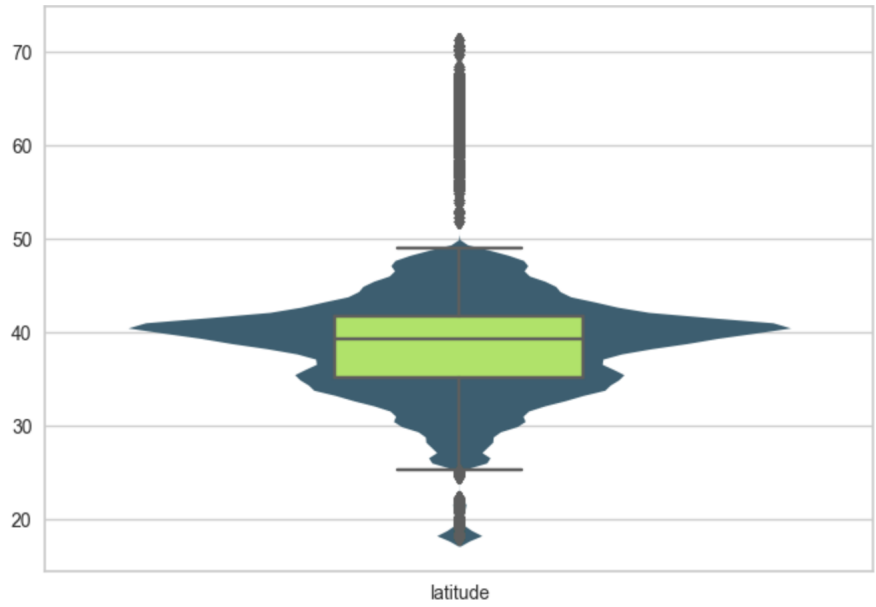


Figura 2.1: Latitude Outliers

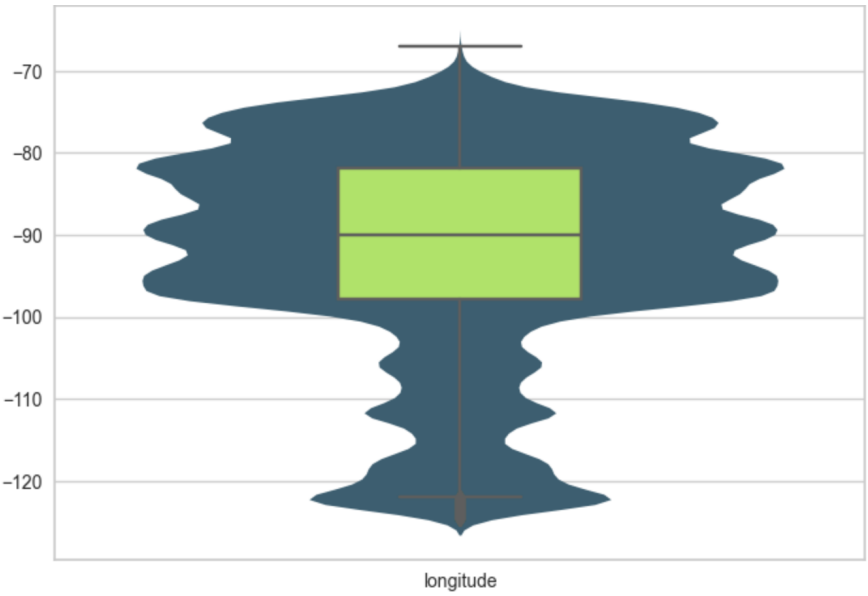


Figura 2.2: Longitude Outliers

I violinplots e i boxplots forniscono una rappresentazione della distribuzione dei dati in tre quartili che purtroppo non è affidabile al 100%; per tale motivo andremo a calcolare l'IQR per assicurarci di eliminare tutti gli outliers reali. Gli step necessari al calcolo dell'IQR sono i seguenti:

1. Calcola il primo e il terzo quartile (Q1 e Q3);
2. Calcolare l'IQR: $IQR = Q3 - Q1$;
3. Calcolare il limite inferiore: $Q1 * 1.5$;
4. Calcolare il limite superiore: $Q3 * 1.5$;
5. Sostituire i punti che riesdono al di fuori del limite inferiore e superiore con un valore nullo;
6. Rimuovere tutti i valori nulli;

```
q75, q25 = np.percentile(df.loc[:, column], [75, 25])  # step 1
intr_qr = q75 - q25  # step 2

max = q75 + (1.5 * intr_qr)  # step 3
min = q25 - (1.5 * intr_qr)  # step 4

df.loc[df[column] < min, column] = np.nan  # step 5
df.loc[df[column] > max, column] = np.nan  # step 5

df.dropna(axis=0) # step 6
```

Listing 2.2: remove_outliers

Ricalcoliamo i violinplots e i boxplots dopo la rimozione degli outliers:

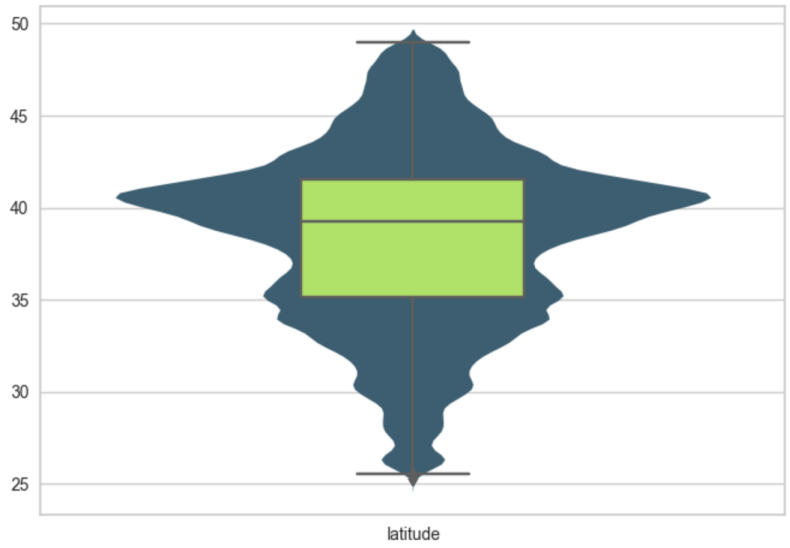


Figura 2.3: Latitude No Outliers

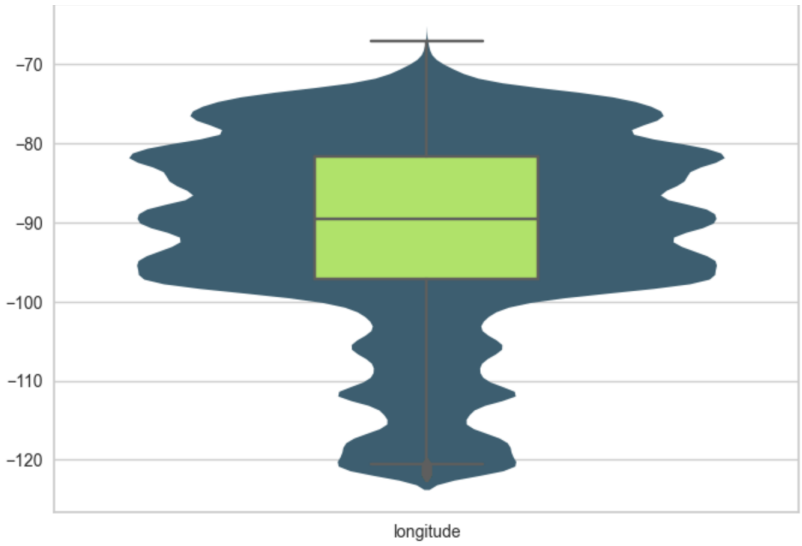


Figura 2.4: Longitude No Outliers

2.5.2 Feature Scaling

Se l'insieme dei valori per una determinata caratteristica è molto diverso rispetto ad un altro, c'è il rischio di "far confondere" l'algoritmo di apprendimento; l'algoritmo potrebbe sottostimare/sovrastimare l'importanza di una caratteristica poiché questa ha una scala di valori molto inferiore/superiore rispetto alle altre.
Il *feature scaling* è l'insieme di tecniche che consentono di **normalizzare o scalare l'insieme di valori di una caratteristica**.
La normalizzazione dei dati è stata effettuata sulle features `latitude` e `longitude` usando la **min-max normalization**. La **min-max normalization** è il metodo più comune per normalizzare:

$$x^1 = a + \frac{(x - \min(x))(b - a)}{\max(x) - \min(x)}$$

Dove a e b rappresentano i valori minimo e massimo che vogliamo ottenere dalla normalizzazione. Di default, il `MinMaxScaler` fornito da `sklearn` normalizza nell'intervallo $[0, 1]$.

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
X[X.columns] = scaler.fit_transform(X[X.columns])
```

Listing 2.3: normalize_latitude_longitude

2.5.3 Feature Extraction

La *feature extraction* rientra nell'ambito del **feature engineering**, cioè il processo nel quale un progettista usa la propria conoscenza del dominio per determinare le caratteristiche (feature) dai dati grezzi estraibili tramite tecniche di data mining. La feature extraction è un processo tramite il quale vengono generate automaticamente delle features dai dati.

Nello specifico è stata utilizzata la classe `CountVectorizer` sulla feature `interests`. La classe `CountVectorizer`, messa a disposizione da `sklearn`, converte un insieme di documenti testuali in una matrice di tokens.

```
import re
from sklearn.feature_extraction.text import CountVectorizer

vect = CountVectorizer(tokenizer=lambda interests: interests.split(", "
    ), lowercase=True)
vect_labels = vect.fit_transform(X["interests"])

X = pd.concat([X, pd.DataFrame(data=vect_labels.toarray(), columns=vect
    .get_feature_names_out())], axis=1)
X = X.drop(columns=["interests"])
```

Listing 2.4: feature_extraction

Per ogni interesse presente nella colonna `interests` è stata generata un colonna dove ogni cella può assumere i valori: 0 e 1. Se la cella assume valore 1 implica che quel determinato Meeter possiede quell'interesse, 0 altrimenti.

acting	advertising	air hockey	american football	art	art history	astronomy	astrophysics	...
0	0	0	0	0	0	0	0	...
0	0	0	0	0	0	0	0	...
0	0	0	0	0	0	0	0	...
0	0	0	0	0	0	0	0	...
0	1	0	0	0	1	0	0	...

Figura 2.5: Feature Extraction: Count Vectorizer

2.5.4 Data Balancing

La maggior parte dei modelli di machine learning funzionano bene solo quando il numero di esempi di una certa classe (ad esempio, il fatto che una mail sia classifica come 'spam') è simile al numero di esempi di un'altra classe (la classe 'no-spam'). Dunque, è **estremamente importante bilanciare i dati di un problema**.

Il *data balancing* è l'insieme di tecniche per convertire un dataset sbilanciato in un dataset bilanciato.

Siccome la generazione del dataset è stata gestita internamente non è stato necessario applicare alcuna tecnica di data balancing. Tuttavia, il dataset presenta una dimensionalità pari a 158 features. Nella sezione successiva andremo ad effettuare la **Principal Component Analysis** sul dataset al fine di ridurne la dimensionalità.

2.5.5 Principal Component Analysis

La **Principal Component Analysis**, o **PCA**, è un metodo per la riduzione della dimensionalità che viene spesso usato per ridurre la dimensionalità di grandi datasets, transformando un grande insieme di variabili in uno più piccolo che contiene la maggior parte delle informazioni dell'insieme di partenza.
La riduzione del numero di variabili di un dataset va naturalmente a scapito dell'accuratezza, ma il trucco nella riduzione della dimensionalità è scambiare un po' di accuratezza con la semplicità. Siccome i datasets più piccoli sono più facili da esplorare e visualizzare, rendono l'analisi dei dati molto più semplice e veloce per gli algoritmi di apprendimento senza variabili estranee da elaborare.

Per riassumere, l'idea dietro alla PCA è semplice - **ridurre il numero di variabili del dataset conservando quante più informazioni possibili**.
La PCA restituisce una matrice di covarianza tramite la quale è possibile osservare se esistono relazioni tra le variabili considerate. La matrice di covarianza è una matrice simmetrica $p \times p$, dove p è il numero delle dimensioni, che ha come celle le covarianze associate con tutte le possibili combinazioni delle variabili iniziali. Ad esempio, per una dataset 3-dimensionale con tre variabili x, y e z, la matrice di covarianza è una matrice 3×3 avente questa forma:

$$\begin{bmatrix} Cov(x,x) & Cov(x,y) & Cov(x,z) \\ Cov(y,x) & Cov(y,y) & Cov(y,z) \\ Cov(z,x) & Cov(z,y) & Cov(z,z) \end{bmatrix}$$

Il segno della covarianza ci indica come due variabili sono correlate. Se è positivo, le due variabili aumentano e diminuiscono assieme (sono correlate). Se è negativo, una cresce e l'altra descresce (sono inversamente correlate).
Nel seguente script andremo ad effettuare la PCA sul nostro dataset cercando di mantenere il 99% della varianza.

```
from sklearn.decomposition import PCA

pca = PCA()

pca_df = pca.fit_transform(X)

#  Plotting to determine how many features should the dataset be
    reduced to
```

```
plt.style.use("bmh")
plt.figure(figsize=(14,4))
plt.plot(range(1,training_df.shape[1]+1), pca.explained_variance_ratio_
    .cumsum())
plt.show()

# Finding the exact number of features that explain at least 99% of the
     variance in the dataset
total_explained_variance = pca.explained_variance_ratio_.cumsum()
n_over_99 = len(total_explained_variance[total_explained_variance
    >=.99])
n_to_reach_99 = training_df.shape[1] - n_over_99

print(f"Number features: {n_to_reach_99}\nTotal Variance Explained: {
    total_explained_variance[n_to_reach_99]}")
```

Listing 2.5: pca

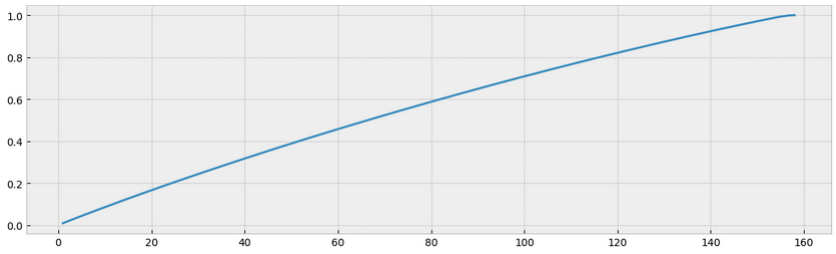


Figura 2.6: Principal Component Analysis

```
Number features: 154
Total Variance Explained: 0.9933861399213042
```

Listing 2.6: pca_estimation

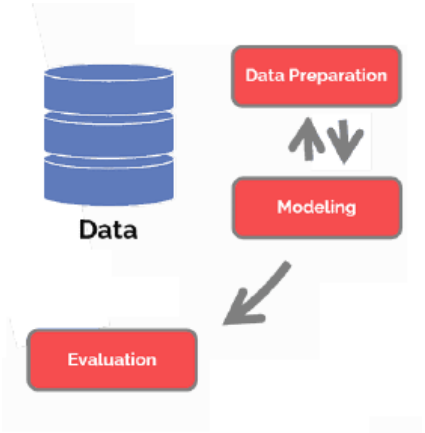
Come si evince, la PCA afferma che il 99% della varianza dell'informazione del dataset è contenuto in 154 features. Andiamo quindi a ridurre la dimensionalità del dataset fino al numero di features determinate dalla PCA e andremo ad estimare la varianza effettiva.

```
pca = PCA(n_components=n_to_reach_99)

# Fitting and transforming the dataset to the stated number of features
     and creating a new DF
pca_df = pca.fit_transform(training_df)

# Seeing the variance ratio that still remains after the dataset has
    been reduced
print(pca.explained_variance_ratio_.cumsum()[-1])

0.9890603474159855
```

Listing 2.7: fitting_pca

2.6 Modeling

Una volta sistemati i dati, è ora di iniziare la fase di modellazione. In primo luogo, va selezionata la tecnica o l'algoritmo da utilizzare; dopodiché, si passa alla fase di addestramento. In questo caso, si configurano i parametri del modello selezionato, si addestra il modello e si descrivono i risultati ottenuti in fase di addestramento.



Come già anticipato a 1.3 il problema è stato affrontanto sviluppando una soluzione ibrida, combinando **clustering** e **classificazione**.
La soluzione proposta consiste nell'individuare pattern simili nelle preferenze degli utenti tramite il clustering e successivamente, tramite un classificatore addestrato sui dati clusterizzati in precedenza, fare in modo che il modello impari a predire e consigliare Meeters che abbiano preferenze simili all'utente preso in considerazione.

2.6.1 Clustering

Il clustering è una tecnica di apprendimento non supervisionato che classifica i dati senza assegnarvi etichette, creando gruppi di oggetti dalle simili caratteristiche. Gli algoritmi di clustering si basano sul concetto di **similarità**, che deve essere alta tra gli elementi del gruppo, e bassa tra diversi gruppi.

2.6.2 Gli algoritmi di clustering

I principali algoritmi di clustering utilizzati sono:

- **K-means**
  L'algoritmo k-means è l'algoritmo a partizionamento iterativo ad errore quadratico più famoso che sfrutta i **centroidi**. L'algoritmo sceglie i centroidi in maniera casuale o pseudo-casuale, genera un partizionamento assegnando ogni campione al centroide più vicino, calcola i nuovi centroidi e ripete fino al cambiamento dei centroidi. Ha una buona efficienza,

e con cluster ben fatti restituisce buoni risultati: bisogna però impostare un valore di k cluster a priori, che può essere stabilito su dati empirici ma anche sfruttando alcune metriche. Aumentando k si potrebbero migliorare le prestazioni dell'algoritmo.

- **Clustering gerarchico**
  Il clustering gerarchico cerca di considerare raggruppamenti multi-livello, ovvero a diversi livelli di similarità.
  Forma un **dendrogramma**, che come un albero decisionale, può essere navigato per decidere quanti cluster sono necessari per i dati di input. In base alla similarità di partenza si parla di clustering gerarchico divisivo e agglomerativo.

- **Density-based clustering - DBSCAN**
  Il DBSCAN raggruppa pattern considerando la loro densità nella distribuzione. Si basa su due parametri principali: $minPts$, che stabilisce il numero minimo di punti per considerare un intorno di un punto denso, ed ϵ che definisce un intorno circolare di ciascun punto dai suoi vicini. Aumentando o riducendo il valore di questi parametri, si potrebbe migliorare la qualità del clustering.

2.6.3 K-means

Come anticipato precedentemente, l'algoritmo k-means è un algoritmo di clustering a partizionamento iterativo ad **errore quadratico**.

2.6.3.1 Somma degli errori quadrati

Il criterio della **somma degli errori quadrati** (**squared sum estimate**) è uno dei criteri più semplici ed utilizzati per l'ottimizzazione dei cluster. Supponiamo di avere un insieme $D = x_1, x_2, ..., x_n$ composto da n campioni che vogliamo partizionare in esattamente k insiemi disgiunti $D_1, D_2, ...D_k$. Ogni sottoinsieme rappresenta un cluster, con i campioni nello stesso cluster che sono per qualche motivo più simili l'un l'altro rispetto ai campioni negli altri cluster. Supponiamo che l'insieme dato di n campioni sia stato partizionato in qualche modo in k cluster $D_1, D_2, ..., D_k$ e che n_i sia il numero in D_i e che m_i sia la media artimetica dei campioni, ovvero:

$$m_i = \frac{1}{n_i} \sum_{x \in D_i} x$$

Allora la somma degli errori quadrati sarà uguale a:

$$j_e = \sum_{i=1}^{k} \sum_{x \in D_i} |x - m_i|$$

Per un dato cluster D_i, il vettore delle medie m_i (detti **centroidi**) è **la miglior rappresentazione dei campioni nel dataset**.

2.6.3.2 Passaggi e metriche

L'algoritmo k-means compie i seguenti passaggi:

1. Seleziona k centroidi in maniera casuale generando k pattern;
2. Genera un partizionamento assegnando ogni campione al centroide più vicino;
3. Calcola i nuovi centroidi del cluster, considerando la media dei valori dei cluster generati al punto #2;
4. Ripete il punto #2 finché non cambiano i centroidi;

L'algoritmo ha una buona efficienza; tuttavia bisogna impostare un valore k di cluster a priori. A tal proposito andremo ad utilizzare una metrica di stima, detta **punto di gomito** o **Elbow point**. Il punto di gomito è utilizzato per valutare il miglior numero di cluster da generare negli algoritmi di clustering partizionale.

2.6.3.3 Implementazione e addestramento

Prima di tutto andiamo calcolare il punto di gomito per identificare il numero migliore di cluster da generare. Eseguiremo più volte l'algoritmo k-means in quanto tende a convergere prematuramente.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from yellowbrick.cluster import KElbowVisualizer, SilhouetteVisualizer

X = df.copy()

k_clusters = 10
model = KMeans(n_clusters=k_clusters, random_state=0, n_init=10)

visualizer = KElbowVisualizer(model, timings=True)
visualizer.fit(X)
visualizer.show()
```

Listing 2.8: distance_elbow_point

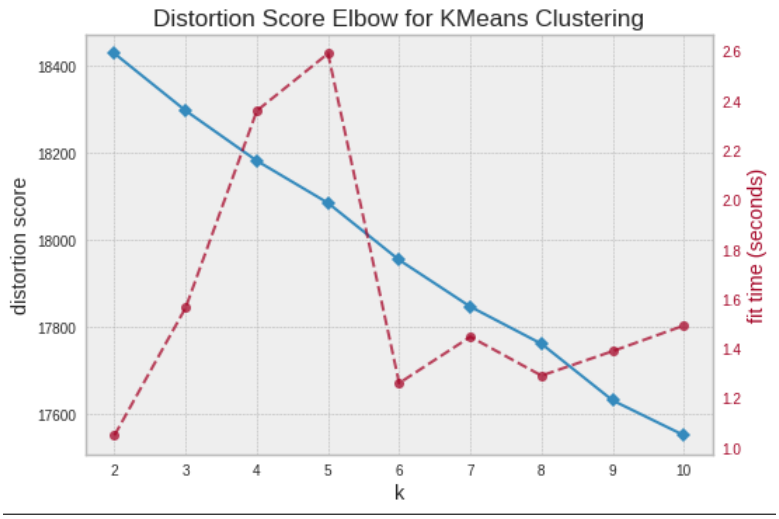


Figura 2.7: Elbow point

Dal grafico, si evince che non è stato possibile individuare un valore per il punto di gomito il che implica che l'algoritmo k-means non si adatta molto bene al problema in esame.
Sebbene non sia stato possibile usare l'algoritmo k-means per affrontare il problema, è stato comunque utilizzato per ottenere una visione della distribuzione degli utenti in base alla distanza che potrà ritornare utile in una successiva fase di analisi dei dati.

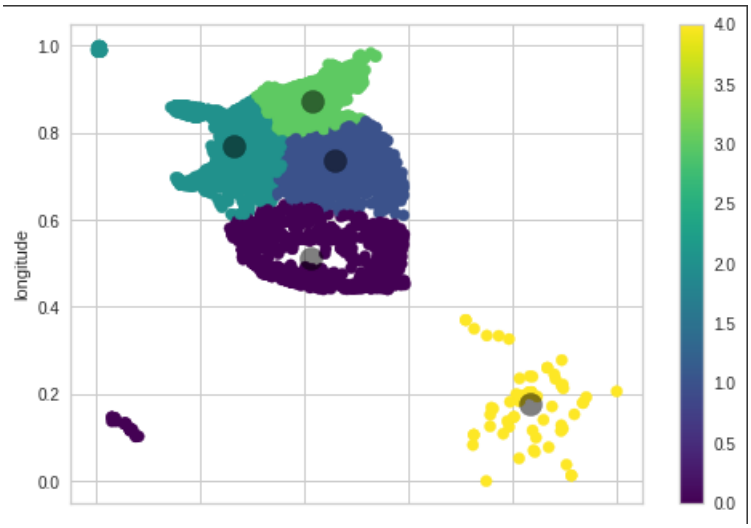


Figura 2.8: Distribuzione degli utenti in base alla locazione

2.6.4 DBSCAN

Come specificato prima, il DBSCAN raggruppa pattern considerando la loro densità di distribuzione e si basa su due parametri principali: $minPts$, che stabilisce il numero minimo di punti per considerare un intorno di un punto denso, ed ϵ che definisce un intorno circolare di ciascun punto dai suoi vicini. DBSCAN, tuttavia, non è in grado di produrre valori ottimali di ϵ e $minPts$.

2.6.4.1 Come stimare i valori dei parametri?

Per stimare il valore ottimale di ϵ andremo ad utilizzare l'algoritmo descritto nel seguente articolo [13].

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
from yellowbrick.cluster import KElbowVisualizer, SilhouetteVisualizer

neigh = NearestNeighbors(n_neighbors=len(X.columns) * 2)
nbrs = neigh.fit(X)
distances, indices = nbrs.kneighbors(X)

distances = np.sort(distances, axis=0)
distances = distances[:,1]
plt.plot(distances)
```

Listing 2.9: neighbours

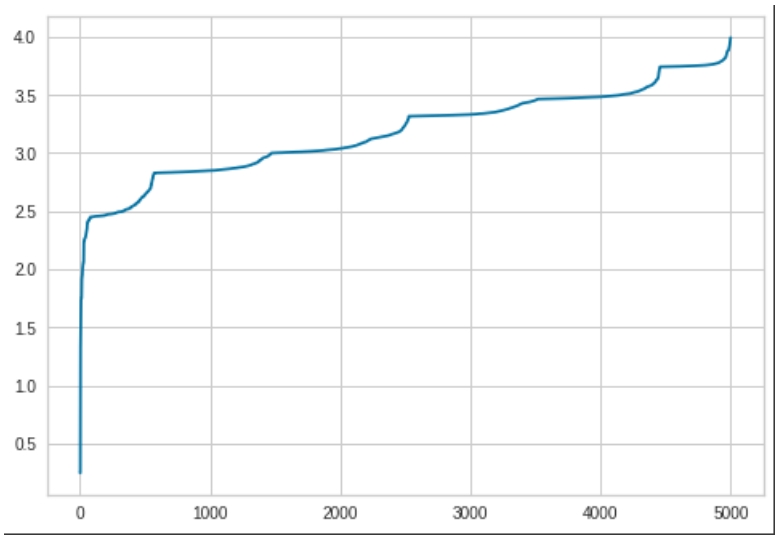


Figura 2.9: Single level density negihbours

In seguito andiamo a plottare il valore ottimale di ϵ tramite la libreria `kneed`.

```
from kneed import KneeLocator

kneedle = KneeLocator(range(1,len(distances)+1),  #x values
                      distances, # y values
                      S=1.0, #parameter suggested from paper
                      curve="convex", #parameter from figure
                      direction="decreasing") #parameter from figure
kneedle.plot_knee_normalized()
```

Listing 2.10: epsilon

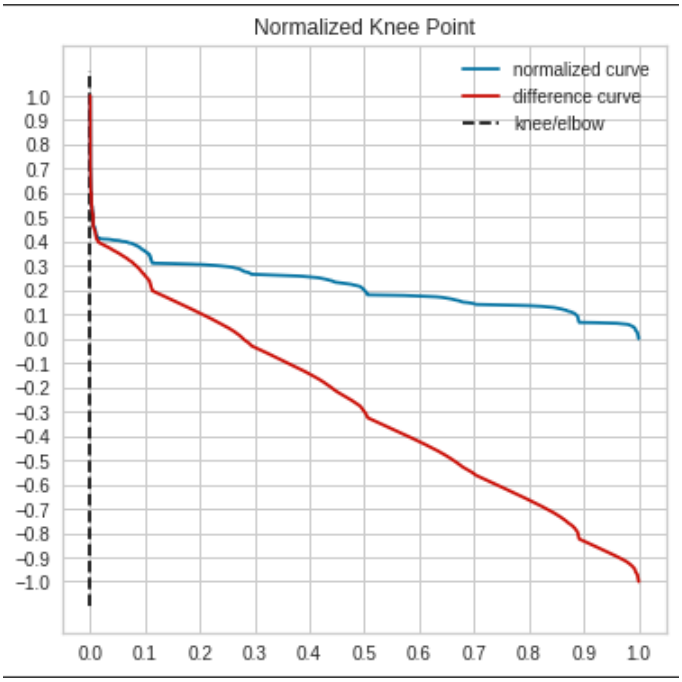


Figura 2.10: DBSCAN: valore di ϵ

Il valore ottimale per ϵ è pari a 0.2500478278861643 ed è contenuto nell'attributo knee_y. Per la stima del parametro $minPts$ si ricorre ad un metodo euristico il quale afferma che sia D la dimensionalità del dataset, allora $minPts >= D+1$. Per dataset di larghe dimensioni, una stima valida di $minPts$ è pari a $D*2$.

2.6.4.2 Implementazione e addestramento

Andiamo a stimare il numero di cluster e di **noise points**. Un **noise point** è un punto che non è associato con nessun cluster e rappresenta un outlier. Procediamo alla stima dei noise points in modo da poterli rimuovere e riaddestrare il modello; i noise points sono individuati dalla label **-1**.

```
model = DBSCAN(eps=kneedle.knee_y + 0.01, min_samples=4)
model.fit(Y)
labels = model.labels_

n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
n_noise_ = list(labels).count(-1)
print("Estimated number of clusters: %d" % n_clusters_)
print("Estimated number of noise points: %d" % n_noise_)


Estimated number of clusters: 0
Estimated number of noise points: 5000
```

Listing 2.11: clusters_noise_points_estimation

Purtroppo il modello identifica l'intero dataset come noise points. Andremo comunque a plottare il risultato del modello per osservare la distribuzione dei dati. **I noise points saranno individuati dai punti di colore nero**.

```
unique_labels = set(labels)
core_samples_mask = np.zeros_like(labels, dtype=bool)
core_samples_mask[model.core_sample_indices_] = True

colors = [plt.cm.Spectral(each) for each in np.linspace(0, 1, len(
    unique_labels))]
for k, col in zip(unique_labels, colors):
    if k == -1:
        # Black used for noise.
        col = [0, 0, 0, 1]

    class_member_mask = labels == k

    xy = Y[class_member_mask & core_samples_mask]
    plt.plot(
        xy[:, 0],
        xy[:, 1],
        "o",
        markerfacecolor=tuple(col),
        markeredgecolor="k",
        markersize=14,
    )

    xy = Y[class_member_mask & ~core_samples_mask]
    plt.plot(
        xy[:, 0],
        xy[:, 1],
        "o",
        markerfacecolor=tuple(col),
        markeredgecolor="k",
        markersize=6,
    )

plt.show()
```

Listing 2.12: dbscan

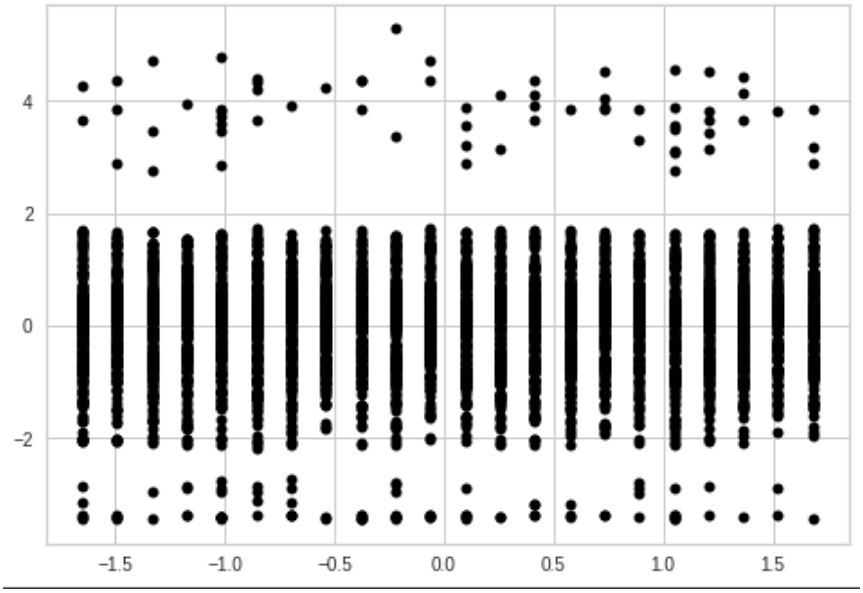


Figura 2.11: DBSCAN

2.6.5 Clustering Gerarchico Agglomerativo

L'algoritmo che, per sperimentazione empirica, si adatta meglio al problema in esame è l'algoritmo di **clustering gerarchico** nella sua versione **agglomerativa**.

2.6.5.1 Passaggi e metriche

Mentre gli algoritmi partizionali, come l'algoritmo k-means, restituiscono delle partizioni disgiunte, il clustering gerarchico cerca di considerare raggruppamenti multi-livello, ovvero a diversi livelli similarità. Il primo livello conterrà n cluster, ogni cluster conterrà un solo elemento. Il secondo conterrà $n-1$ cluster, ogni cluster sarà formato sulla base delle similarità delle caratteristiche e così via. Il processo andrà avanti fino a quando tutti gli elementi non formeranno un unico cluster.

Al fine valutare il numero ottimale di cluster da generare andremo a considerare varie metriche: **l'indice di Davies-Bouldin**, **Silhouette score** e **l'indice di Calinski-Harabasz**.

L'indice di Davies-Bouldin è una metrica di convalida che viene spesso utilizzata per valutare il numero ottimale di cluster da utilizzare. È definito come un il rapporto tra la dispersione del cluster e la separazione del cluster. Valori più bassi indicano una migliore clusterizzazione.
Dati n punti dimensionali, sia C_i il cluster dei punti e sia X_i un vettore di features n-dimensionale assegnato al cluster C_i.

$$S_i = (\frac{1}{T_i}\sum_{j=1}^{T_i} \|X_j - A_i\|_p^q)^{\frac{1}{q}}$$

A_i è il centroide di C_ie T_i è la dimensione di C_i. S_i è la q-esima radice del q-esimo momento dei punti in C_i rispetto alla media. Se $q = 1$, allora S_i è la distanza media tra i vettori delle caratteristiche nel cluser C_i e il centroide.

L'indice di Calinski-Harabasz, o **criterio del rapporto di varianza**, è un misura di quanto un oggetto è simile al proprio cluster (coesione) rispetto ad altri cluster (separazione). Qui la coesione è stimata in base alle distanze dai punti dati in un cluster al suo centroide del cluster e la separazione si basa sulla distanza dei centroidi del cluster dal centroide globale. Valori maggiori dell'indice CH significa che i cluster sono densi e ben separati. L'indice di Calinkski-Harabasz una forma del tipo $(a.Separazione)/(b.Coesione)$ dove a e b sono i pesi.

$$CH = [\frac{\sum_{k=1}^{K} n_k \|c_k - c\|^2}{K-1}]/[\frac{\sum_{k=1}^{K} n_k \sum i = 1^{n_k} \|d_i - c_k\|^2}{N-K}]$$

dove, n_k e c_k sono rispettivamente il numero di punti e il centroide del k-esimo cluster, c è il centroide globale, N è il numero totale dei dati.

Il silhouette score o **silhouette coefficient** quantifica quanto i dati siano ben disposti nei cluster generati. Il coefficiente varia tra -1 e +1. Valori alti indicano condizioni maggiori di coesione e di separazione dei dati. Il coefficiente, per un punto i-esimo appartenente al cluster c, è calcolato tramite la seguente formula:

$$s(i) = \frac{b(i) - a(i)}{max(b(i), a(i))}$$

con:

- $a(i)$ che rappresenta la distanza media dell'i-esimo punto rispetto a tutti gli altri punti appartenente allo stesso cluster;
- $b(i)$ che rappresenta la distanza media dell'i-esimo punto rispetto a tutti gli altri punti appartenente al cluster più vicino del cluster a cui è stato assegnato;
- $s(i)$ che rappresenta il coefficiente di silhouette dell'i-esimo punto;

Il valore finale di silhouette è dato dalla **media dei coefficienti di silhouette calcolati per ogni elemento del problema**.

2.6.5.2 Implementazione e addestramento

Prima di tutto, andiamo a generare un **dendrogramma** per osservare il comportamento del modello di machine learning ed individuare il numero ottimale di cluster da generare.

```
import scipy.cluster.hierarchy as sch

dendrogram = sch.dendrogram(sch.linkage(X, method="ward"))

plt.title("Interests, latitude, longitude Dendrogram")
plt.xlabel("Interests, latitude, longitude")
plt.ylabel("Distance")
plt.show()
```

Listing 2.13: dendogram

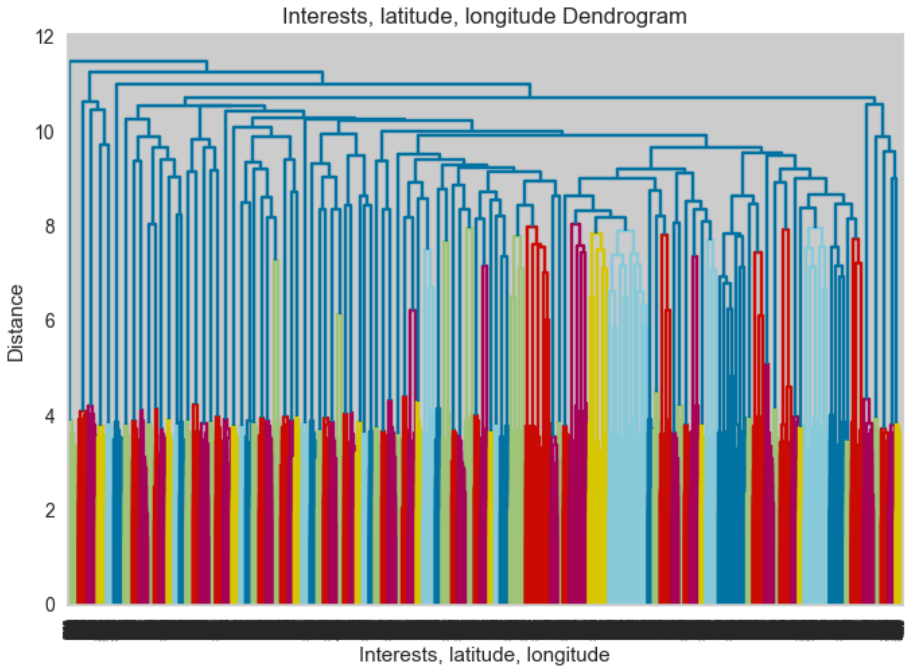


Figura 2.12: Dendrogramma

Siccome il dendrogramma non è facilmente interpretabile, andiamo a calcolare l'indice di Davies Bouldin, Calinski-Harabasz e il Silhouette score per individuare il numero ottimale di cluster da generare k.

```
from sklearn.cluster import AgglomerativeClustering

from sklearn.metrics import davies_bouldin_score, silhouette_score,
    calinski_harabasz_score

cluster_count = range(2, 200)

davies_scores = []
silhouette_scores = []
calinski_harabasz_scores = []

for i in cluster_count:
    hac_model = AgglomerativeClustering(n_clusters = i)
    hac_model.fit(training_df)
    hac_assignments = hac_model.labels_
    davies_scores.append(davies_bouldin_score(training_df,
    hac_assignments))
    silhouette_scores.append(silhouette_score(training_df,
    hac_assignments))
    calinski_harabasz_scores.append(calinski_harabasz_score(training_df
```

```
    , hac_assignments))
```

Listing 2.14: scores_evaluation

Andiamo a plottare i risultati ottenuti ed estimare il numero ottimale di cluster da generare.

Davies-Bouldin Index

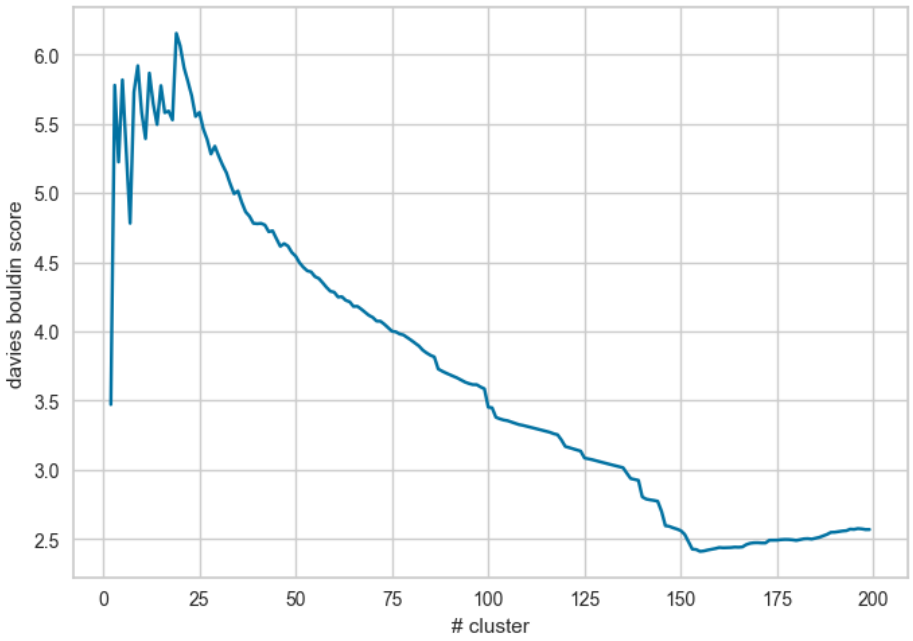


Figura 2.13: Davies-Bouldin Index

Da un'analisi più approfondita si evince che il valore ottimale di k proposto dall'indice di davies bouldin è pari a 154.

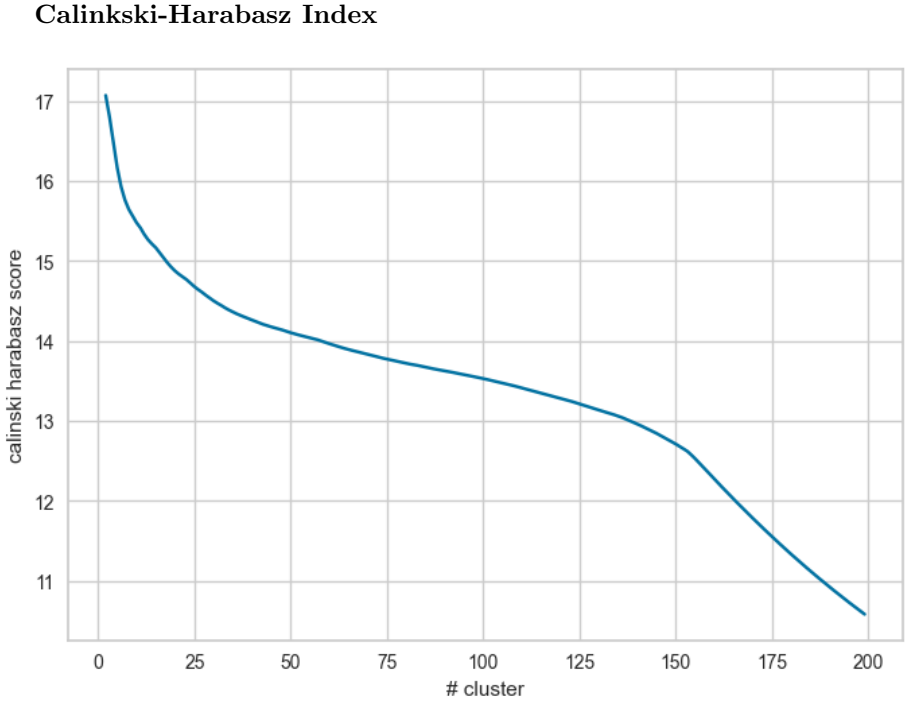


Figura 2.14: Calinkski-Harabasz Index

Da un'analisi più approfondita si evince che il valore ottimale di k proposto dall'indice di calinkski harabasz è pari a 25.

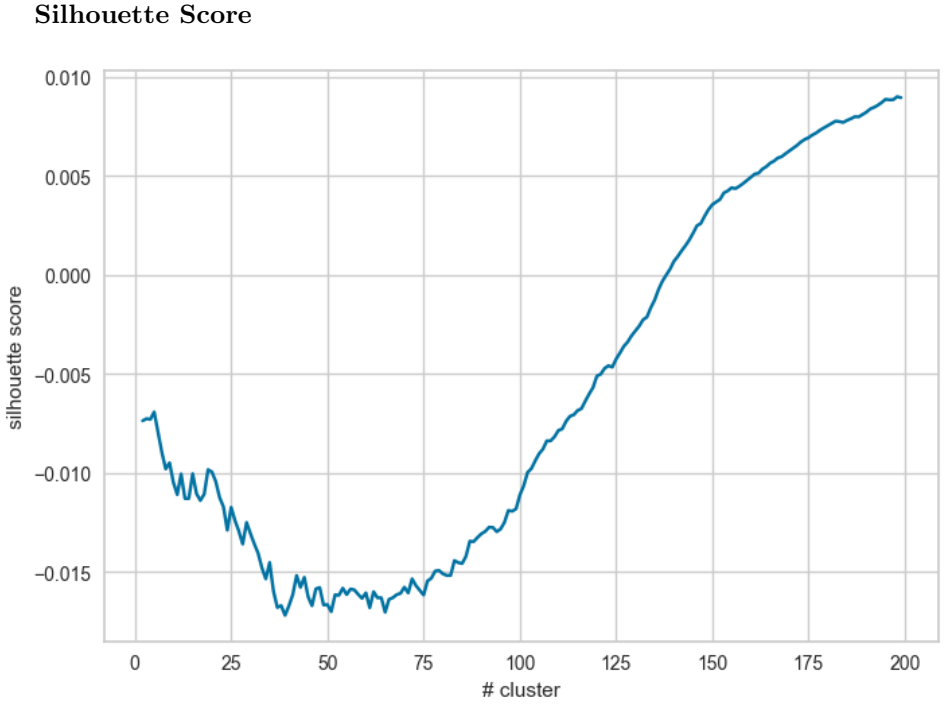


Figura 2.15: Silhouette Score

Da un'analisi più approfondita si evince che il valore ottimale di k proposto dal silhouette coefficient è pari a 200.

Siccome tutte e tre le metriche riportano risultati diversi, dopo una serie di sperimentazioni empiriche, si è stabilito che il miglior compromesso è dato da k = 154.

```
from scipy.cluster.hierarchy import dendrogram

def plot_dendrogram(model, **kwargs):
    # Create linkage matrix and then plot the dendrogram

    # create the counts of samples under each node
    counts = np.zeros(model.children_.shape[0])
    n_samples = len(model.labels_)
    for i, merge in enumerate(model.children_):
        current_count = 0
        for child_idx in merge:
            if child_idx < n_samples:
                current_count += 1  # leaf node
            else:
                current_count += counts[child_idx - n_samples]
        counts[i] = current_count
```

```
17
18      linkage_matrix = np.column_stack(
19          [model.children_, model.distances_, counts]
20      ).astype(float)
21
22      # Plot the corresponding dendrogram
23      dendrogram(linkage_matrix, **kwargs)
24
25  hac_model = AgglomerativeClustering(compute_distances=True, n_clusters
        = 154)
26  hac_model.fit(training_df)
27
28  plot_dendrogram(hac_model, truncate_mode="level", p=3)
29  plt.xlabel("Number of points in node (or index of point if no
        parenthesis).")
30  plt.show()
31
32  print(f"DB Score: {davies_bouldin_score(training_df, hac_model.labels_)
        }")
33  print(f"Silhouette Score: {silhouette_score(training_df, hac_model.
        labels_)}")
34  print(f"Calinski Harabasz Score: {calinski_harabasz_score(training_df,
        hac_model.labels_)}")
```

Listing 2.15: agglomerative_clustering

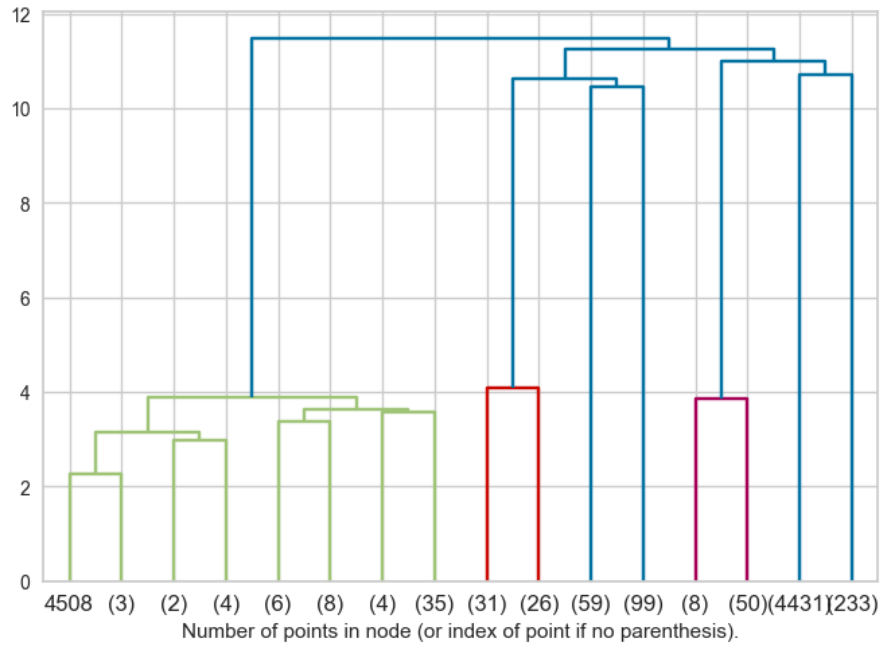


Figura 2.16: Dendrogramma con 154 cluster

```
DB Score: 2.424067990561999
Silhouette Score: 0.004250730189086839
Calinski Harabasz Score: 12.575345880105381
```

Listing 2.16: agglomerative_clustering_scores

L'ultimo step che ci rimane da fare è l'esportazione del dataset contenente le cluster labels.

```
X = df[['name', 'surname', 'gender', 'searching_gender', 'interests', '
    age']].join(pd.DataFrame(scaler.inverse_transform(X), columns=
    training_df.columns, index=training_df.index))
X['cluster #'] = hac_model.labels_
X.to_csv('clustered_dataset.csv', index=False, header=True)
```

Listing 2.17: export_clustered_dataset

2.6.6 Classificazione

La classificazione è una tecnica di apprendimento supervisionato in cui il modello va a predire il valore di una variabile categorica, detta **variabile dipendente**, **target** o **classe**, tramite l'uso di un **traning set**, ovvero un insieme di osservazioni per cui la variabile target è nota. Il modello di machine learning costruito per problemi di classificazione prende il nome di **classificatore**.

2.6.7 Gli algoritmi di classificazione

I due principali classificatori che sono stati presentati durante il corso sono stati:

- **Naive Bayes**
  L'algoritmo Naive Bayes considera le caratteristiche della nuova istanza da classificare e calcola la probabilità che queste facciano parte di una classe tramite l'applicazione del teorema di Bayes:

  $$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

  L'algoritmo è chiamato **naive** (**ingenuo**) perchè non tiene conto della correlazione delle varie caratteristiche e dunque non valuta la potenziale utilità data dalle combinazione di più caratteristiche.

- **Decision Tree**
  L'algoritmo Decision Tree mira a creare un albero i cui nodi rappresentano un sotto-insieme di caratteristiche del problema e i cui archi rappresentano delle decisioni. L'obiettivo di un decision tree è predire il valore della variabile categorica apprendendo semplici regole di decisione inferite dai dati di training. I decision tree sono particolarmente utili per la facilità di lettura che possiedono: navigando l'albero è possibile comprendere il motivo per cui è stata fatta una determinata predizione. (**explainability**)

Per affrontare il problema proposto si è deciso di valutare vari classificatori, tra cui il **Random Forest** e il **Gradient Boosting classificatori ensemble** che combinano più classificatori insieme. Nelle sezioni successive andremo descrivere i vari classificatori valutati nel corso dello sviluppo del progetto.

2.6.8 Training set e Test set

Addestrare e validare un modello di machine learning sullo stesso dataset è un grosso errore metodologico che porta ad avere risultati totalmente inaffidabili. Per tal motivo andremo **inizialmente** a dividere il dataset di partenza in maniera tale da formare due insiemi distinti:

- **Training set**
  Composto dalle istanze che l'algoritmo userà per l'addestramento.
- **Test set**
  Composto dalle istanze per cui l'algoritmo addestrato dovrà predire la classe di appartenenza.

Il modello è stato addestrato considerando il 67% del dataset e validato col rimanente 33%.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
    =0.33, random_state=42)
```

Listing 2.18: split_dataset

2.6.9 Random Forest Classifier

Il Random Forest Classifier è un classificatore basato sull'**ensemble learning** che opera costruendo una moltitudine di decision trees.

2.6.9.1 Come funziona un decision tree?

L'algoritmo per la costruzione di un decision tree opera come segue:

1. Posiziona la miglior caratteristica del traning set come radice dell'albero;
2. Divide il traning set in sotto-insiemi. Ogni sotto-insieme dovrebbe essere composto di valori simili per una certa caratteristica;
3. Ripete gli steps #1 e #2 su ogni sotto-insieme fin quando non viene raggiungo un nodo foglia in ogni sotto-albero.

L'attributo sulla base del quale dividere il dataset viene selezionato in base all'**information gain** che questo possiede. L'information gain misura il **grado di purezza** di un attributo, ovvero quanto questo sarà in grado di dividere adeguatamente il dataset. Alla base del calcolo dell'information gain vi è l'**entropia**

che indica in che misura un messaggio è ambiguo e difficile da capire. L'entropia viene calcoalta come segue:

$$H(D) = -\sum_{c} p(c) \times \log_2 p(c)$$

dove $p(c)$ è la proporzione della classe c nel dataset D. Calcolata l'entropia, per ogni attributo del dataset viene calcolato l'information gain come segue:

$$Gain(D, A) = H(D) - \sum_{v \in values(A)} \frac{|D_v|}{|D|} \times H(D_v)$$

dove:

- D è l'entropia del dataset;
- D_v è il sotto-insieme di D per cui l'attributo A ha valore v;
- $|D_v|$ è il numero di elemento di D_v;
- $|D|$ è il numero di elementi del dataset;

2.6.9.2 Implementazione e addestramento

Di seguito è riportato lo script riguardante l'implementazione e l'addestramento del classificatore basato su Random Forest.

```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(random_state=42)
rf.fit(X_train, y_train)

#Make prediction
y_pred = rf.predict(X_test)

report_classifier(rf, y_test, y_pred)

Accuracy score: 0.4684848484848485
```

Listing 2.19: random_forest

Nella sezione Evalutation verranno descritti i metodi e i criteri che hanno portato alla scelta del classificatore più adatto.

2.6.10 KNN Classifier

La **Neighbors-based classification** è un tipo di apprendimento basato su istanze o apprendimento non generalizzante: non tenta di costruire un modello interno generale, ma memorizza semplicemente istanze dei dati di addestramento. La classificazione è calcolata da un semplice voto di maggioranza dei vicini più prossimi di ciascun punto: a un punto di interrogazione viene assegnata la classe di dati che ha il maggior numero di rappresentanti all'interno dei vicini più prossimi del punto.
La tecnica più comunemente utilzzata è la k**-neighbors classification** o k**-nearest-neighbors**. Una scelta ottimal di k è strettamente dipendente dai dati: in generale valori maggiori di k sopprimono gli effetti del rumore, ma rende i confini della classificazione meno distinti.

2.6.10.1 Implementazione e addestramento

Di seguito è riportato lo script riguardante l'implementazione e l'addestramento del classificatore basato su KNN.

```
from sklearn.neighbors import KNeighborsClassifier

knnc = KNeighborsClassifier()
knnc.fit(X_train, y_train)
y_pred = knnc.predict(X_test)

report_classifier(knnc, y_test, y_pred)

Accuracy score: 0.503030303030303
```

Listing 2.20: knn

Nella sezione Evalutation verranno descritti i metodi e i criteri che hanno portato alla scelta del classificatore più adatto.

2.6.11 SVC Classifier

Il **Support Vector Machines**, o **SVC**, classifier tenta di segregare i dati nel miglior modo possibile. La distanza tra l'uno o l'altro punto più vicino è noto come margine. L'obiettivo è selezionare un iperpiano con il massimo margine possibile tra i vettori di supporto nel dataset dato. SVM cerca l'iperpiano col massimo margine:

1. Generando iperpiani che separano le classe nel miglior modo possibile;
2. Selezionando il giusto iperpiano con la massima segregazione dai punti più vicini;

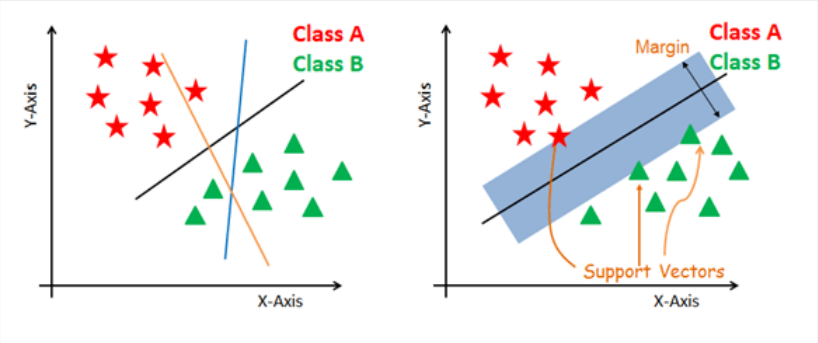


Figura 2.17: Esempio di SVM

2.6.11.1 Implementazione e addestramento

Di seguito è riportato lo script riguardante l'implementazione e l'addestramento del classificatore basato su SVC.

```
from sklearn.svm import SVC

svc = SVC()
svc.fit(X_train, y_train)
y_pred = svc.predict(X_test)

report_classifier(knnc, y_test, y_pred)

Accuracy score: 0.6715151515151515
```

Listing 2.21: svc

Nella sezione Evalutation verranno descritti i metodi e i criteri che hanno portato alla scelta del classificatore più adatto.

2.6.12 MLP Classifier

Il **Multi-Layer Perceptron**, o **MLP**, classifier si basa su una **rete neurale** al fine di classificare i dati.

2.6.12.1 Implementazione e addestramento

Di seguito è riportato lo script riguardante l'implementazione e l'addestramento del classificatore basato su MLP.

```
from sklearn.neural_network import MLPClassifier

mlpc = MLPClassifier()
mlpc.fit(X_train, y_train)
y_pred = mlpc.predict(X_test)

report_classifier(mlpc, y_test, y_pred)

Accuracy score:  0.6254545454545455
```

Listing 2.22: mlpc

Nella sezione Evalutation verranno descritti i metodi e i criteri che hanno portato alla scelta del classificatore più adatto.

2.6.13 Gradient Boosting Classifier

Il Gradient Boosting è un classificatore basato sull'**ensemble learning** che opera costruendo una moltitudine di decision trees **deboli**. Quando un decision tree è il *weak-learner*, l'algoritmo risultante è chiamato **gradient-boosted trees**. Un modello gradient-boosted trees è costruito in modo graduale come in altri metodi di potenziamento. In generale, è più performante del Random Forest.

2.6.13.1 Implementazione e addestramento

Di seguito è riportato lo script riguardante l'implementazione e l'addestramento del classificatore basato su Gradient Boosting.

```
from sklearn.ensemble import GradientBoostingClassifier

gbc = GradientBoostingClassifier()
gbc.fit(X_train, y_train)
y_pred = gbc.predict(X_test)

report_classifier(gbc, y_test, y_pred)

Accuracy score:  0.5763636363636364
```

Listing 2.23: gradient_boosting

Nella sezione Evalutation verranno descritti i metodi e i criteri che hanno portato alla scelta del classificatore più adatto.

2.6.14 Grid Search

La maggior parte dei modelli di machine learning contiene parametri che possono essere regolati per variare il modo in cui il modello apprende. Ad esempio, il modello di regressione logistica, da `sklearn`, ha un parametro `C` che controlla la regolarizzazione, che influisce sulla complessità del modello.

Come scegliamo il valore migliore per `C`? Il valore migliore dipende dai dati usati per addestrare il modello.
La **Grid Search** è una ricerca esaustiva sui valori dei parametri specificati al fine di migliorare le prestazioni del modello. Di seguito sono riportati gli script e i risultati ottenuti dalla grid search su Random Forest, KNN ed SVC.

2.6.14.1 Random Forest con Grid Search

Di seguito è riportato lo script riguardante l'implementazione e l'addestramento del classificatore basato su Random Forest con Grid Search.

```
from sklearn.model_selection import GridSearchCV

rf_parameters = {
    'n_estimators': [250, 500, 750, 1000, 1250],
    'max_features': ['sqrt', 'log2'],
    'criterion': ['gini', 'entropy'],
    'max_depth': [3, 5, 10, None]
}

rfc_gs = GridSearchCV(estimator = rf, param_grid = rf_parameters, cv =
    10, verbose=2)
rfc_gs.fit(X_train, y_train)
rfc_gs.best_params_

rf = RandomForestClassifier(criterion='gini', max_features='log2',
    n_estimators=500, max_depth=None)
rf.fit(X_train, y_train)

#Make prediction
y_pred = rf.predict(X_test)
report_classifier(rf, y_test, y_pred)

Accuracy went from 0.50 to 0.52 (+0.02)
```

Listing 2.24: rf_grid_search

2.6.14.2 KNN con Grid Search

Di seguito è riportato lo script riguardante l'implementazione e l'addestramento del classificatore basato su KNN con Grid Search.

```
from sklearn.model_selection import GridSearchCV

neighbors = range(1, 31)

knn_params = {
    'n_neighbors': list(neighbors)
}

knn_gs = GridSearchCV(estimator = knnc, param_grid = knn_params, cv =
    10, verbose=2)
knn_gs.fit(X_train, y_train)
knn_gs.best_params_

```

```
knnc = KNeighborsClassifier(n_neighbors=1)
knnc.fit(X_train, y_train)
y_pred = knnc.predict(X_test)

report_classifier(knnc, y_test, y_pred)

Accuracy went from 0.50 to 0.53 (+0.03)
```

Listing 2.25: knn_grid_search

2.6.14.3 SVC con Grid Search

Di seguito è riportato lo script riguardante l'implementazione e l'addestramento del classificatore basato su SVC con Grid Search.

```
from sklearn.model_selection import GridSearchCV

svc_params = {
    'C':[1, 10, 100, 1000],
    'gamma':[1, 0.1, 0.5, 0.08, 0.001, 0.01, 0.0001],
    'kernel':['linear','rbf', 'poly', 'sigmoid'],
    'degree': [2, 3, 4],
}

scv_fs = GridSearchCV(estimator = svc, param_grid = svc_params, cv =
    10, verbose=2)
scv_fs.fit(X_train, y_train)
scv_fs.best_params_

svc = SVC(C=1, gamma=0.1, kernel='rbf', degree=2)
svc.fit(X_train, y_train)

#Make predition
y_pred = svc.predict(X_test)
report_classifier(svc, y_test, y_pred)

Accuracy went from 0.6715 to 0.6745 (+0.003)
```

Listing 2.26: svc_grid_search

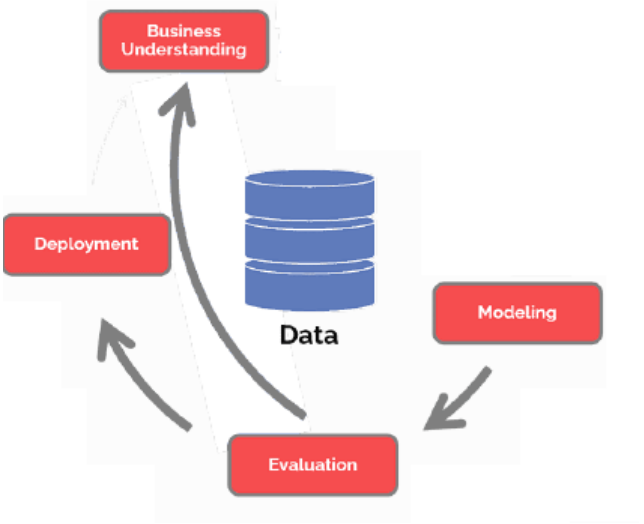
La grid search non è stata effettuata sui classificatori basati su MLP e Gradient Boosting in quanto troppo esigente in termini di risorse hardware.

2.6.14.4 Digressione sui parametri usati nella Grid Search

Per l'individuazione dei parametri da utilizzare nella grid search per ogni classificatore non esiste una vera e propria euristica; di conseguenza sono stati utilizzati i parametri che sembravano avere la maggior influenza sulle prestazioni dei vari classificatori in base a quanto specificato nella documentazione di `sklearn`.

2.7 Evaluation

La fase di evalutation ha l'obiettivo di valutare se i risultati sono chiari, se sono in linea con gli obiettivi di business e se rivelano delle prospettive aggiuntive alle quali il progettista non aveva pensato.



Nella sezione seguente andremo a descrivere com'è stato validado e valutato il modello finale basato su classificazione.

2.7.1 Cross-validation

In seguito all'uso della **Grid Search** 2.6.14 sui vari classificatori, al fine della validazione dei risultati si è deciso di utilizzare il metodo della **convalida incrociata** o **cross-validation** fornito dall'implementazione di `sklearn`.

La **cross-validation** è un metodo statistico che consiste nella **ripetuta partizione** e **valutazione** dell'insieme dei dati di partenza. La cross-validation prevede i seguenti passi:

1. Mischiare in maniera casuale i dati di partenza;
2. Dividere i dati in k gruppi;
3. Per ogni gruppo:
   (a) Considerare il gruppo corrente come test set;
   (b) Considerare i rimanenti $k - 1$ gruppi come training set;
   (c) Addestrare il modello con i dati del training set;
   (d) Valutare le prestazioni del modello e cancellarlo;

È comune utilizzare un $k = 10$ ed è proprio il valore che abbiamo utilizzato. In questo caso si parla di **10-fold cross validation**.
Siccome la cross-validation è casuale potrebbe inavvertitamente portare un **vantaggio** al classificatore, ad esempio, una divisione dei gruppi irrealistica. Per tal motivo si è deciso di far in modo di avere un **campionamento stratificato** che porta i sottoinsiemi creati ad avere un numero simile di istanze delle diverse classi.
Il metodo di validazione finale viene detto **stratified 10-fold validation**.

2.7.2 Come valutare un classificatore?

A prescindere dalla procedura di validazione usata, è necessario avere strumenti adatti per valutare la bontà delle predizioni. Per tal fine è sono state utilizzate le **matrice di confusione** restituite da ogni classificatore e le metriche da esse derivanti.

2.7.2.1 Cos'è una matrice di confusione?

Una **matrice di confusione**, detta anche **tabella di errata classificazione**, restituisce una rappresentazione dell'accuratezza di un classificatore.

	Istanze realmente positive	Istanze realmente negative
Istanze predette come positive	Veri positivi	Falsi positivi
Istanze predette come negative	Falsi negativi	Veri negativi

Figura 2.18: Matrice di confusione

La matrice di confusione è una matrice con cui poter indicare se e in quanti casi il classificatore ha predetto correttamente o meno il valore di un'etichetta del test set.

La matrice sovrastante a 2.18 è una rappresentazione di una matrice di confusione per un problema di classificazione binaria. Nel nostro caso, le matrici di confusioni usate per la valutazione del modello risultano essere più complesse in quanto derivanti da un problema di classificazione multi-etichetta.

2.7.2.2 Metriche

Sulla base dei valori della matrice di confusione, possono essere calcolate diverse metriche.
Definendo TP come il numero di veri positivi, FP come il numero di falsi positivi, TN come il numero di veri negativi e FN come il numero di falsi negativi possiamo calcolare, tra le metriche principali:

- Precision;
- Recall;
- Specificity;
- Accuracy;
- MCC;

Ogni metrica fornisce un'indicazione complementare per la valutazione del modello di classificazione. Ad esempio, in un problema binario (classificazione true/false):

La precision indica il numero di predizioni corrette per la classe `true` rispetto a tutte le predizioni fatte dal classificatore. Indica quanti errori ci saranno nelle lista delle predizioni fatte dal classificatore.

$$Precision = \frac{TP}{(TP + FP)}$$

La recall indica il numero di predizioni corrette per la classe `true` rispetto a tutte le istanze positive di quella classe. Indica quante istanze positive nell'intero dataset il classificatore può determinare.

$$Recall = \frac{TP}{(TP + FN)}$$

La specificity è il rapporto tra i veri negativi e tutti gli esiti negativi. Indica quante istanze negativi nell'intero dataset il classificatore può determinare.

$$Specificity = \frac{TN}{(TN + FP)}$$

Il **coefficiente di correlazione di Matthews** (**MCC**) è un indice statistico più affidabile che produce un punteggio elevato solo se la previsione ha ottenuto buoni risultati in tutte e quattro le categorie della matrice di confusione (veri positivi, falsi negativi, veri negativi e falsi positivi) , proporzionalmente sia alla dimensione degli elementi positivi sia alla dimensione degli elementi negativi nel set di dati.

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP) \times (TP + FN) \times (TN + FP) \times (TN + FN)}}$$

L'accuracy indica il numero totale di predizioni corrette identificate dal classificatore, sia della classe positiva che della classe negativa.

$$Accuracy = \frac{(TP + TN)}{(TP + TN + FP + FN)}$$

È importante ricordare che le metriche forniscono solo indicazioni e vanno interpretate correttamente sulla base del problema o dei problemi in esame.

2.7.3 Valutazione

Per l'effettiva valutazione dei classificatori sono state prese in considerazione la matrice di confusione e le seguenti metriche: *precision*, *recall*, *accuracy* e *support*. Il support è il numero di occorrenze di ogni classe in `y_true` ed è una metrica fornita da `sklearn`.

Dopo una serie di sperimentazioni empiriche e vari confronti tra i classificatori, si è deciso di adottare il classificatore basato su SVC come modello finale di machine learning. Di seguito sono riportate la matrice di confusione e le metriche derivanti dal classificatore SVC.

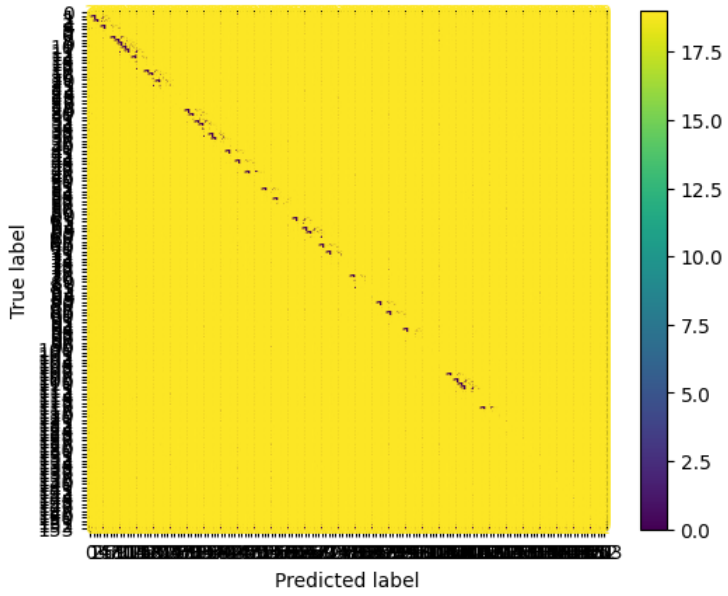


Figura 2.19: Matrice di confusione SVC

precision	recall	f1-score	support
0.71	0.65	0.64	1650

Tabella 2.2: Valore medio delle metriche dell'SVC

2.7.4 Esportazione del modello

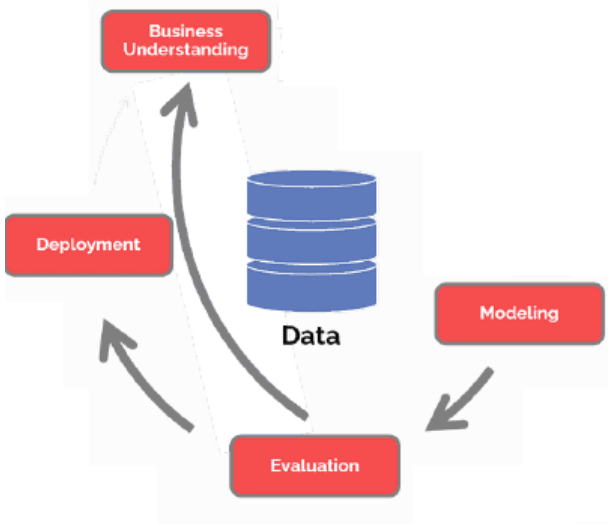
Di seguito è riportato lo script per l'esportazione del modello che verrà usato nella fase di deployment.

```
1 import joblib
2
3 joblib.dump(svc, "model.sav")
```

Listing 2.27: exporting_model

2.8 Deployment

La fase di deployment ha l'obiettivo di mettere in funzione l'approccio definito, e quindi, renderlo usabile. Questa fase prevede il passaggio dall'ingegnere del machine learning all'**ingegneria del software e all'ingegneria dell'usabilità**.



Per rendere disponibile il modello è stato realizzato un **web service** usando `Flask` col quale **OpenMeet** si interfaccia.
Al web service viene inviata una richiesta contenente il Meeter preso in considerazione. Il Meeter viene convertito in un oggetto di tipo `DataFrame` e viene sottoposto allo stesso processo di data preparation descritto precedentemente.

Successivamente, viene caricato il classificatore e viene usato per effettuare il predict della cluster label del Meeter.

```
@app.route('/', methods=['POST'])
def process_request():
    df_user = pd.DataFrame.from_dict({**request.args}, orient='index').
    T

    clustered = pd.read_csv("../clustered_dataset.csv")

    vect = CountVectorizer(tokenizer=lambda interests: interests.split(
    ", "), lowercase=True)
    vect_labels = vect.fit_transform(df_user["interests"])

    df_user = pd.concat([df_user, pd.DataFrame(data=vect_labels.toarray
    (), columns=vect.get_feature_names_out())],
                        axis=1)

    merged = pd.concat([clustered, df_user], ignore_index=True)
    merged = merged.fillna(0)

    del merged['name']
    del merged['surname']
    del merged['gender']
    del merged['searching_gender']
    del merged['age']
    del merged['Cluster #']
    del merged['interests']

    scaler = MinMaxScaler()
    merged[merged.columns] = scaler.fit_transform(merged[merged.columns
    ])

    row = merged.iloc[-1:]

    clf = joblib.load('../model.sav')
    value = clf.predict(row)
    n_cluster = value[0]

    cluster_group = clustered.loc[clustered['Cluster #'] == n_cluster]

    result = apply_filters(clustered, df_user, n_cluster)

    new_df = result[["name", "surname", "age", "gender", "
    searching_gender", "latitude", "longitude", "interests"]]

    return new_df.to_json(orient='records')
```

Listing 2.28: web_service

Una volta effettuato il predict, i risultati vengono filtrati in base al `searching_gender` e al `gender` del Meeter seguendo la seguente *tabella dei generi*:

	Male	Female	Both (Male & Female)	Non-Binary	All
Male	(Male, Male), (Male, Both), (Male, All)	(Female, Male), (Female, Both), (Female, All)	(Male, Male), (Male, Both), (Male, All), (Female, Male), (Female, Both), (Female, All)	(Non-Binary, Male), (Non-Binary, Both), (Non-Binary, All)	(Male, Male), (Male, Both), (Male, All), (Female, Male), (Female, Both), (Female, All), (Non-Binary, Male),(Non-Binary, Both), (Non-Binary, All)
Female	(Male, Female), (Male, Both), (Male, All)	(Female, Female), (Female, Both), (Female, All)	(Male, Female), (Male, Both), (Male, All), (Female, Female), (Female, Both), (Female, All)	(Non-Binary, Female),(Non-Binary, Both), (Non-Binary, All)	(Male, Female), (Male, Both), (Male, All), (Female, Female), (Female, Both), (Female, All), (Non-Binary, Female),(Non-Binary, Both), (Non-Binary, All)
Non-Binary	(Male, Non-Binary)	(Female, Non-Binary)	(Male, Non-Binary), (Female, Non-Binary)	(Non-Binary, Non-Binary)	(Male, Non-Binary), (Female, Non-Binary), (Non-Binary, Non-Binary)

Figura 2.20: Tabella dei generi

```
def apply_filters(clustered, df_user, result_from_classifier):
    user_searching_gender = clustered.loc[clustered["Clustered #"] ==
    result_from_classifier]

    match df_user["searching_gender"][0]:
        case "M"
            user_searching_gender = user_searching_gender.loc[
    user_searching_gender["gender"] == "M"]
        case "F"
            user_searching_gender = user_searching_gender.loc[
    user_searching_gender["gender"] == "F"]
        case "N"
            user_searching_gender = user_searching_gender.loc[
    user_searching_gender["gender"] == "N"]
        case "B"
            user_searching_gender = user_searching_gender.loc[
    user_searching_gender["gender"] == "B"]

    users_searching_user_gender = pd.DataFrame(columns=
    user_searching_gender.columns)

    for index, row in user_searching_gender.iterrows():
         case "M"
            if row["searching_gender"] == "M" or row["searching_gender"
    ] == "B" or row["searching_gender"] == "A":
                users_searching_gender = pd.concat([
    users_searching_user_gender, row.to_frame().T])
        case "F"
            if row["searching_gender"] == "F" or row["searching_gender"
    ] == "B" or row["searching_gender"] == "A":
                users_searching_gender = pd.concat([
    users_searching_user_gender, row.to_frame().T])
        case "N"
            if row["searching_gender"] == "N" or row["searching_gender"
    ] == "A":
```

```
25                  users_searching_gender = pd.concat([
    users_searching_user_gender, row.to_frame().T])
26
27      return users_searching_user_gender
```

Listing 2.29: apply_filters

Ad esempio, per il Meeter:

```
1 meeter = {"name": "Pippo", "surname": "Pluto", "gender": "M", "
    searching_gender": "A", "interests": 'Gaming, Movies, Judo, Acting'
    , "age": "45", "latitude": "48.0572", "longitude": "-117.7466"}
```

Il classificatore predice la cluster label **109** e vengono restituiti i seguenti Meeters:

	name	surname	gender	searching_gender	age	interests	latitude	longitude	acting	advertising	...	web design	weightlifting	windsurfing	wine tasting	woodworking	wrestling	writing	yachting	yoga	Cluster #
0	Beth	Shaw	F	M	23	Roller hockey, Movies, Gaming, Paragliding, Or...	42.4355	-83.489	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	109
57	Rebecca	Green	F	M	26	Advertising, Movies, Equestrian, Filmmaking, M...	28.8563	-81.6771	0.0	1.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	109
189	Dana	Reid	F	M	31	Music production, Movies, Financial analysis	39.452	-97.53	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	109
354	Cody	Owens	M	M	20	Movies, Carpentry, Financial analysis	39.3433	-89.2186	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	109
539	John	Mueller	M	A	19	Water polo, Theatre, Movies	42.9258	-83.6181	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	109
592	Latasha	Ellis	N	M	27	Painting, Movies, Watercolor, Yachting, Jujitsu	39.8056	-75.4862	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	109
806	Christine	Lyons	F	M	24	Yoga, Running, Movies, Rock Climbing	39.5861	-82.5262	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	109

Figura 2.21: Utenti suggeriti

Capitolo 3

Conclusioni

Siamo giunti alla conclusione di un lungo viaggio alla fine del quale ci riteniamo soddisfatti di quanto ottenuto poiché sono stati raggiunti gli obiettivi che ci eravamo prefissati. Un aspetto di cui andiamo particolarmente fieri è **l'inclusività** del modello.

3.1 Cos'abbiamo imparato?

Durante lo sviluppo del progetto abbiamo sperimentato di prima persona i processi, o almeno la superifice, che vengono effettutati dai data scientists per progettare modelli di machine learning. Inoltre, siamo venuti a conoscenza di ulteriori metriche, euristiche, modelli ed algoritmi che non erano stati studiati durante il corso.

Forse, la nozione più importante che si è appresa è quella di non addestrare un modello di machine learning sulla propria macchina - chiedere al collega Della Rocca.

3.2 Obiettivi futuri

Per sviluppi e miglioramenti futuri al progetto ci proponiamo di:

- Passare ad un modello basato su **Active learning**, per far sì che il modello possa imparare anche in base alle interazioni dell'utente personalizzando ancor di più i suggerimenti. L'active learner comporterebbe l'installazione di un modello su ogni client mobile;
- Prevedere una fase di **Retrain** del modello nel caso in cui si dovessero aggiungere nuove località e interessi o dovesse cambiare il "pattern" delle interazioni dell'utente;
- Addestrare il modello su dataset verificati che rispecchino una situazione reale;

Bibliografia

[1] OpenAI, https://openai.com/

[2] ChatGPT, https://openai.com/blog/chatgpt/

[3] Faker, https://fakerjs.dev/

[4] pandas, https://pandas.pydata.org/docs/

[5] seaborn, https://seaborn.pydata.org/

[6] numpy https://numpy.org/

[7] sklearn, https://scikit-learn.org

[8] yellowbricl, https://www.scikit-yb.org/en/latest/

[9] kneed, https://pypi.org/project/kneed/

[10] joblib, https://joblib.readthedocs.io/en/latest/

[11] Flask https://flask.palletsprojects.com/en/2.2.x/

[12] United States Cities Database, https://simplemaps.com/data/us-cities.

[13] Determination of Optimal Epsilon (Eps) Value on DBSCAN Algorithm to Clustering Data on Peatland Hotspots in Sumatra, https://iopscience.iop.org/article/10.1088/1755-1315/31/1/012012/pdf

Ringraziamenti

Un ringraziamento speciale all'associazione di **Arcigay**, in particolar modo all'associazione **Arcigay Salerno "Marcella Di Folco"** per la disponibilità alla collaborazione e al confronto sugli aspetti etici dell'**inclusività** del progetto.