



**EXPLORE
WORLDS**

ALFIERI RICCARDO
MASCOLO FRANCESCO

INTRODUZIONE

Quante persone vorrebbero viaggiare? Quante di queste persone effettivamente viaggiano? Quante di queste viaggiano l'intero pianeta? All'incirca solo 400 persone nella storia hanno visitato tutti gli Stati del mondo.

Da un punto di vista tecnico, la superficie complessiva della Terra è di 509 600 000 km².

Volendo effettuare una separazione, la superficie "calpestabile" è di 148 326 000 km², mentre la superficie ricoperta da acqua è di 361 740 000 km².

Supponendo che una persona viva in media 80 anni, vorrebbe dire che per esplorare ogni angolo della terra emersa, si dovrebbe viaggiare ad una velocità di circa **211.65 km/h per tutta la vita** (volendo attraversare anche solo uno dei chilometri quadrati in linea retta).

Ovviamente ciò è impossibile.

Come risolvere il problema? Seguire la nostra pagina Instagram @ExploreWorld.



@EXPLOREWORLD

BROOKLYN
BRIDGE
NEW YORK

CHE COS'È EXPLOREWORLD

ExploreWorld è una piccola applicazione che permette di effettuare automaticamente i post su Instagram con poche istruzioni.

Una volta deciso il luogo, automaticamente vengono fornite delle foto, e una volta “setacciate” tramite l’intelligenza artificiale è possibile postarle su Instagram.

Inoltre, per assicurarci che il nostro post raggiunga più persone possibili, è possibile anche avere una stima sul numero di like che il post potrebbe ottenere.

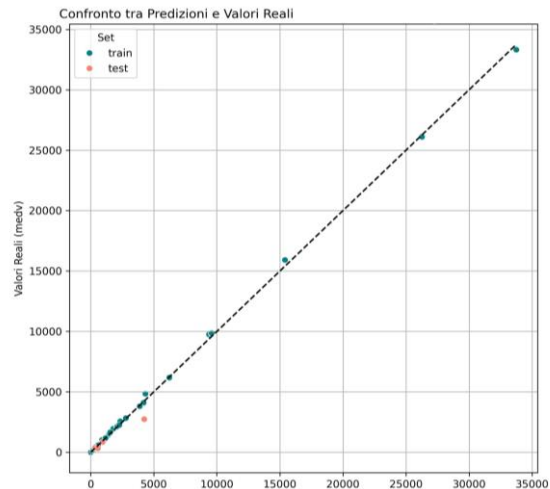
ExploreWorld nasce per diversi motivi:

- Condividere con altre persone luoghi che abbiamo visitato
- Far conoscere luoghi particolari e poco conosciuti ad altri utenti
- Mettere sotto sfida la propria conoscenza a livello geografico

L’obiettivo finale? Esplorare tutto il mondo.

OBIETTIVI DEL PROGETTO

Quali tecniche possiamo utilizzare per risolvere i problemi presi in considerazione?



REGRESSIONE



CLASSIFICAZIONE



**DENSITY
BASED
CLUSTERING**

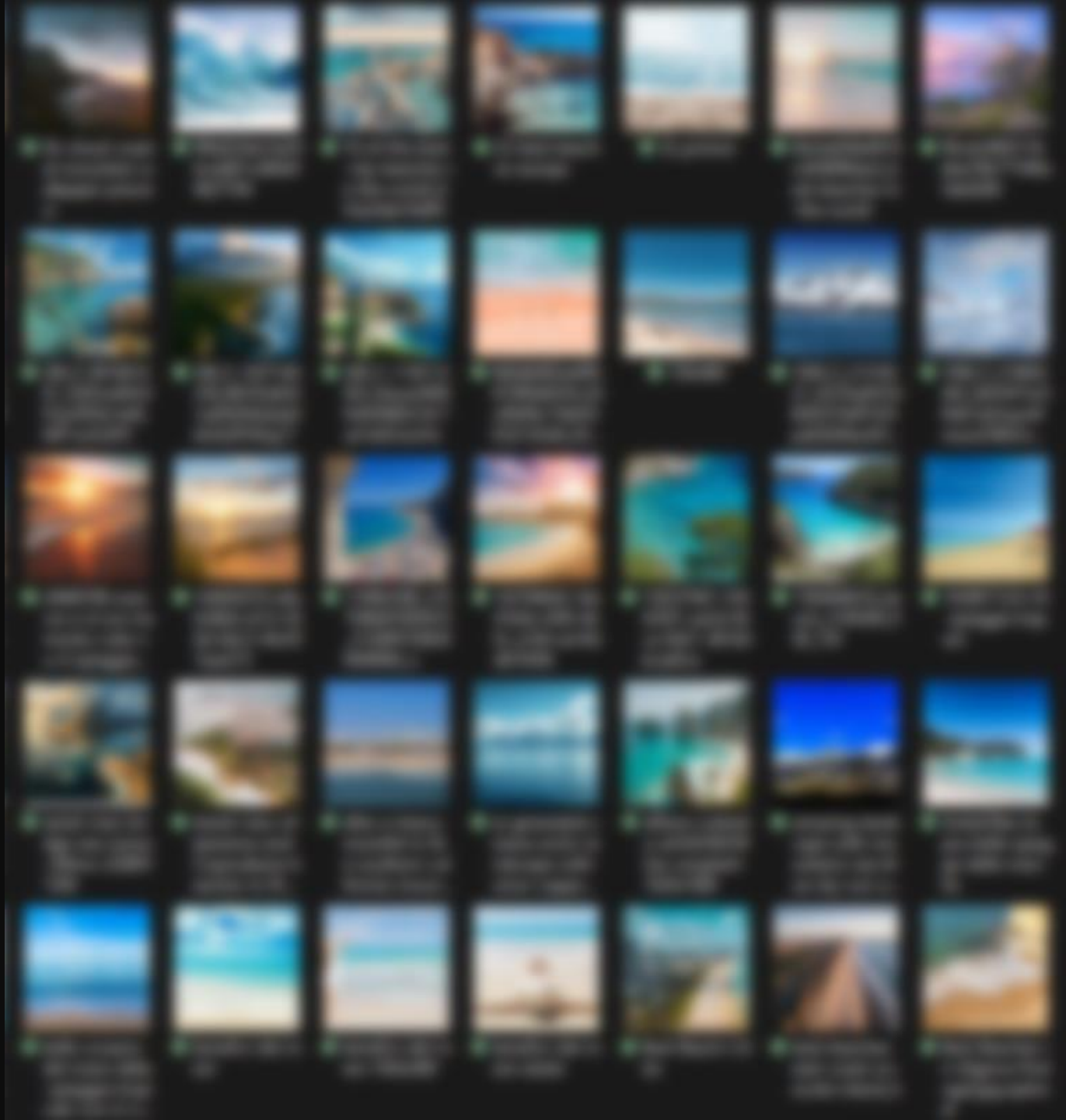
DATASET


I dati sono di fondamentale importanza per addestarre i modelli d'intelligenza artificiale.

Per il progetto ExploreWorld, tutti i dataset sono stati creati dai membri del progetto tramite lo scraping di pagine internet.

Lo scraping di siti web è il processo di estrazione automatica di dati da pagine web utilizzando strumenti o script per raccogliere informazioni strutturate da utilizzare per analisi o applicazioni.


Per i dataset delle immagini è stato effettuato lo scraping di GoogleImmagini. Per il dataset contenente informazioni dei post, è stato effettuato lo scraping di Instagram.






- ☐ Mattina
- ☐ Alba/Tramonto
- ☐ Notte
- ☐ Persone
- ☐ No Persone
- ☐ Mare
- ☐ Montagna
- ☐ Città

Ricerca




- ☐ Mattina
- ☐ Alba/Tramonto
- ☐ Notte
- ☐ Persone
- ☐ No Persone
- ☐ Mare
- ☐ Montagna
- ☐ Città

Ricerca




- ☐ Mattina
- ☐ Alba/Tramonto
- ☐ Notte
- ☐ Persone
- ☐ No Persone
- ☐ Mare
- ☐ Montagna
- ☐ Città

Ricerca



- ☐ Mattina
- ☐ Alba/Tramonto
- ☐ Notte
- ☐ Persone
- ☐ No Persone
- ☐ Mare
- ☐ Montagna
- ☐ Città

Ricerca



- ☐ Mattina
- ☐ Alba/Tramonto
- ☐ Notte
- ☐ Persone
- ☐ No Persone
- ☐ Mare
- ☐ Montagna
- ☐ Città


Ricerca

Scrivi qui la descrizione...

Inserisci il Numero di Follower Della Tua Pagina Instagram:

In base alle informazioni del tuo post la predizione del numero dei like è:

Predict Like

 Pubblica su Instagram

INTERFACCIA DELL'APPLICAZIONE

Ricerca delle immagini
Utilizzo di modelli di classificazione per selezione delle immagini
Utilizzo di modelli di regressione per stima di like
Pulsante per postare

6

RICERCA DELLE IMMAGINI

Una volta deciso il luogo che si desidera, si inserisce all'interno dell'area di testo dedicate e si preme la lente d'ingrandimento per effettuare la ricerca.

Inserisci un luogo



COME FUNZIONA? Scraping

Nel nostro caso, lo scraping si utilizza per ottenere gli URL delle immagini provenienti da GoogleMaps e GoogleImmagini

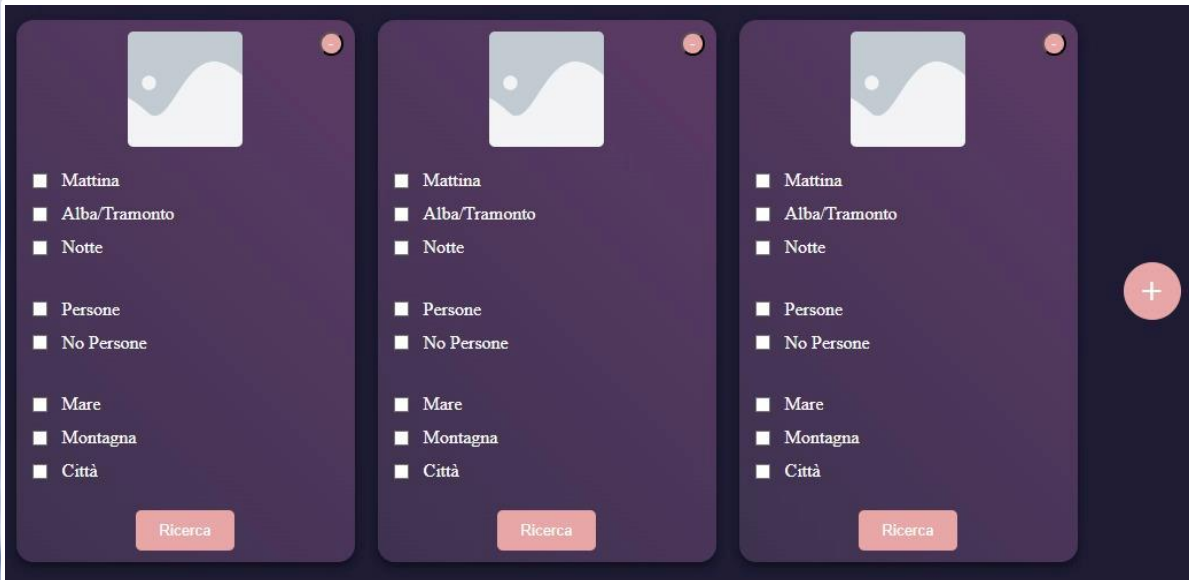
```
def getURLs(Luogo, LuogoType):  
  
    driverMaps=createDriver()  
    driverImages=createDriver()  
  
    result_list = []  
  
    # Creiamo un lock per garantire lo scraping dei due siti in contemporanea  
    lock = threading.Lock()  
  
    Luogo=getLuogo(driverMaps, Luogo, LuogoType)  
    coordinate=getCoordinates(driverMaps.current_url)  
  
    # Crea i thread per eseguire le funzioni contemporaneamente  
    thread1 = threading.Thread(target=getURLGoogleMaps, args=(driverMaps,result_list,lock,))  
    thread2 = threading.Thread(target=getURLGoogleImage, args=(driverImages,Luogo,result_list,lock,))  
  
    # Facciamo partire i thread  
    thread1.start()  
    thread2.start()  
  
    # Attendiamo la terminazione di entrambe le funzioni  
    thread1.join()  
    thread2.join()  
  
    return result_list,coordinate,Luogo
```

CLASSIFICAZIONE DELLE FOTO

Per effettuare la classificazione, si scelgono gli elementi che devono essere presenti nell'immagine.

Si sceglie l'orario della giornata, se ci sono persone oppure no, e quali sono le caratteristiche geomorfologiche.

Per effettuare la classificazione vengono utilizzate delle reti neurali convoluzionali (CNN) addestrate su diversi dataset per riconoscere gli elementi.



CLASSIFICAZIONE DELLE FOTO

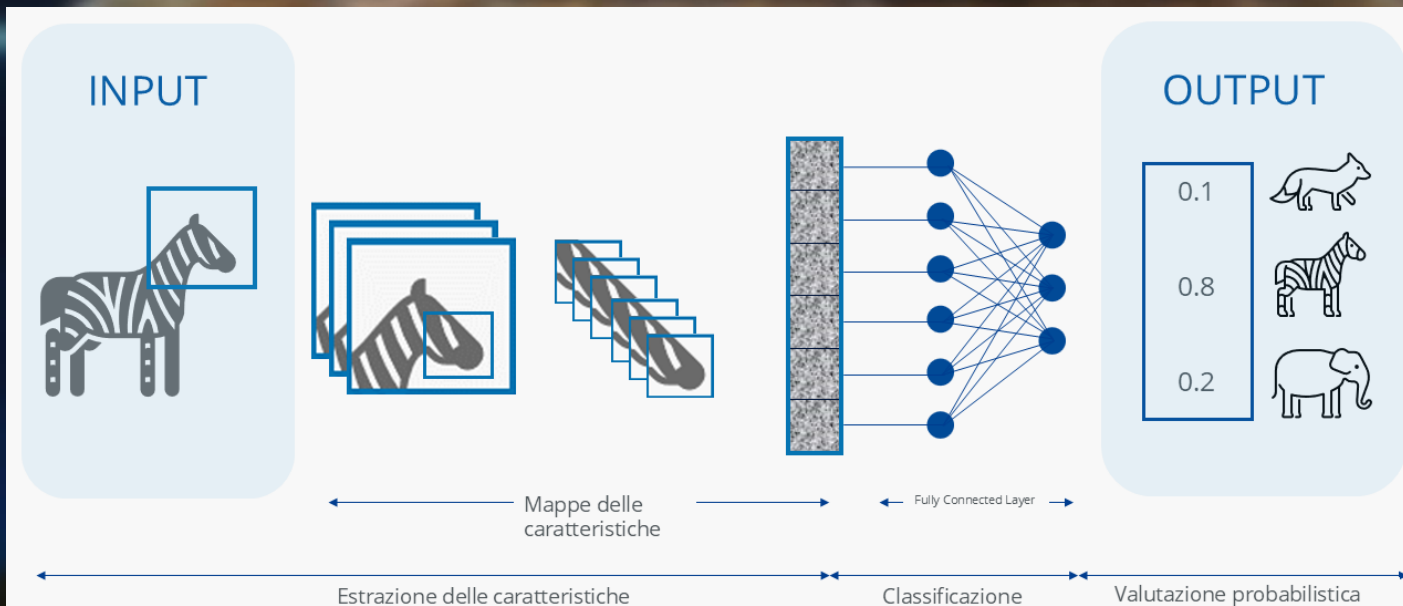
```
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(244, 244, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(1, activation='sigmoid') # Output binario (0 o 1)
])
```

MODELLO

```
# Data augmentation for training
train_datagen = ImageDataGenerator(
    rescale=1./255, # Normalize pixel values
    rotation_range=40, # Random rotations
    width_shift_range=0.2, # Random horizontal shifts
    height_shift_range=0.2, # Random vertical shifts
    shear_range=0.2, # Random shearing
    zoom_range=0.2, # Random zoom
    horizontal_flip=True, # Random horizontal flip
    fill_mode='nearest' # Fill pixels after transformation
)
```

DATA AUGMENTATION

RETE NEURALE CONVOLUZIONALE



Sono un tipo di rete neurale progettata specificamente per elaborare dati con una struttura a griglia, come le immagini. Sono particolarmente efficaci nel riconoscimento di caratteristiche visive.

Componenti fondamentali:

- **Strati convoluzionali:** Applicano filtri per estrarre caratteristiche locali, come bordi, texture o pattern.
- **Strati di pooling:** Ridimensionano i dati riducendo la dimensionalità e mantenendo le informazioni più rilevanti.
- **Strati fully connected:** Collegano tutte le unità per effettuare la classificazione finale.

STIMA DEI LIKE DEL POST

Per far sì che il post otenga più like, c'è bisogno di una giusta descrizione.

Nell'interfaccia è presente appositamente un riquadro per essa.

Una volta inserita, verrà stimato il numero di like valutando la descrizione, numero di follower e numero di immagini scelte.

Per condividere il post su Instagram, basterà premere il pulsante «Pubblica su Instagram»

```
def predictLikes(post):

    num_follower=post['Followers']

    # Dataset
    current_dir = os.path.dirname(os.path.abspath(__file__)) # Directory corrente del file regressione.py
    project_root = os.path.dirname(current_dir) # Salire di un livello (ExploreWorld)
    file_path = os.path.join(project_root, "Datasets", "postDataset", "instagram_posts.csv") # Percorso assoluto

    data = load_dataset(file_path)
    X, y = preprocess_data(data)
    X_train, X_test, y_train, y_test = split_data(X, y)

    model_high_followers = train_model(X_train, y_train) # Modello per follower > 46700
    model_low_followers = model_operations() # Modello per follower <= 46700

    if num_follower > 47600:
        new_data_df = pd.DataFrame([post])
        predicted_likes = predict_likes(model_high_followers, new_data_df)
        predicted_likes = predicted_likes[0] # Estrai il primo elemento
        return predicted_likes

    else:
        predicted_likes = predictFewFollowerLike(model_low_followers, post)
        return predicted_likes
```

Scrivi qui la descrizione...

Inserisci Il Numero di Follower Della Tua Pagina Instagram:

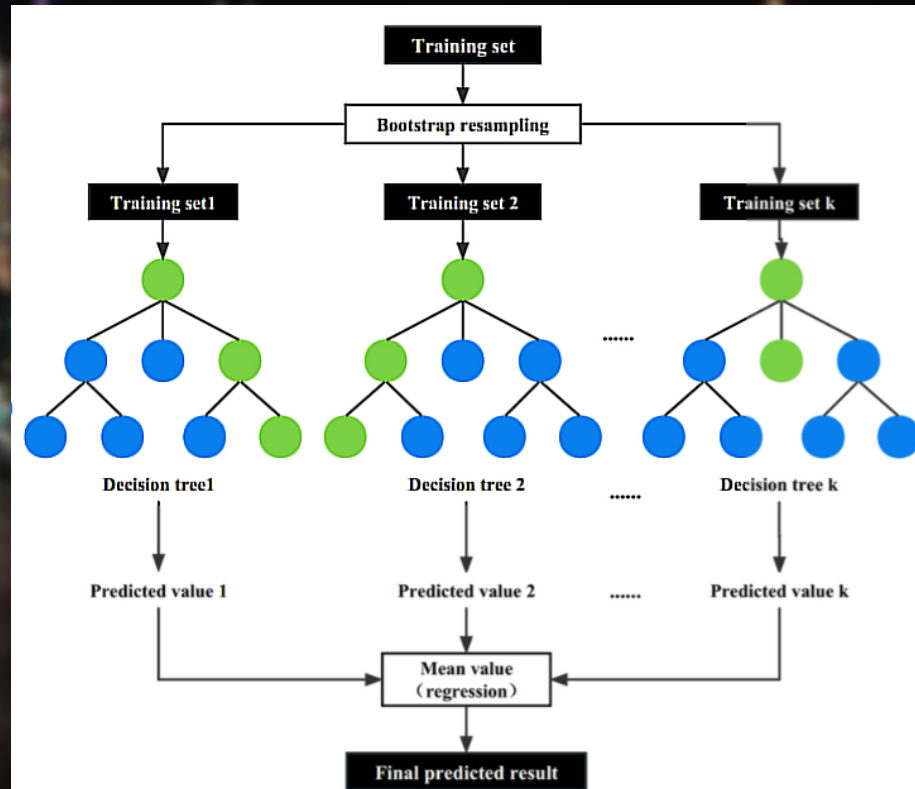
In base alle informazioni del tuo post la predizione del numero dei like é:

Predict Like

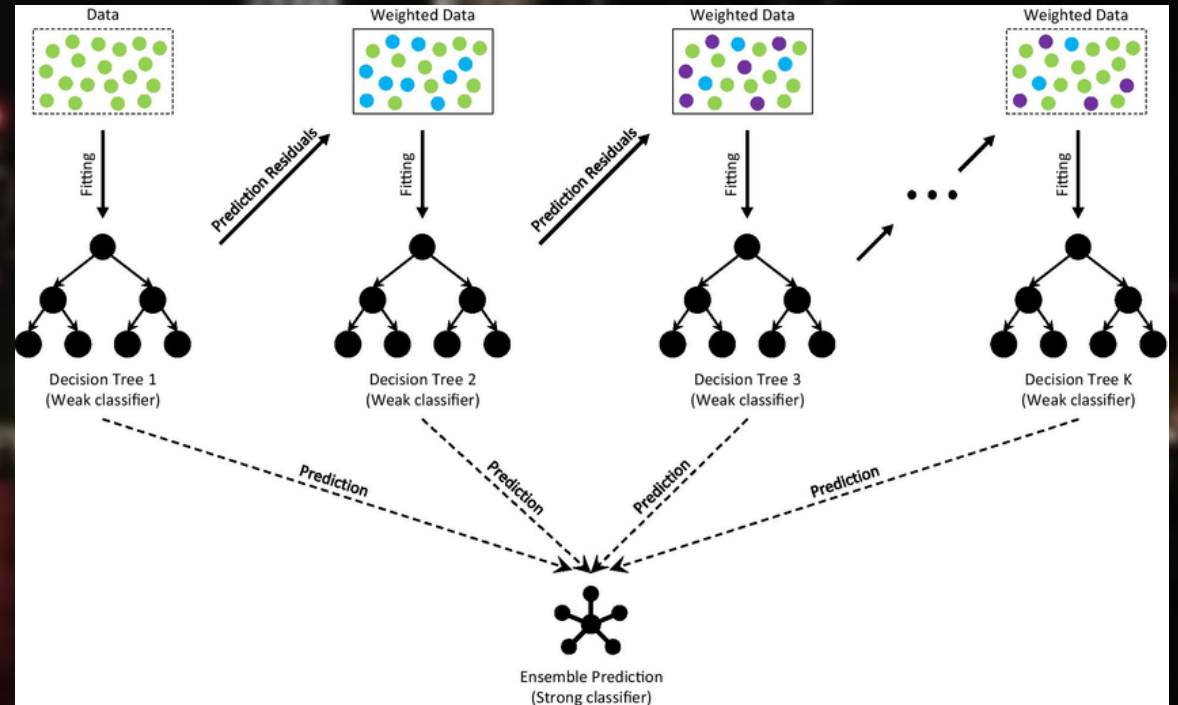
 Pubblica su Instagram

MODELLI USATI PER LA REGRESSIONE

RandomForestRegressor:



Gradient Boosting Regression (GBR):



Il **RandomForestRegressor** è un modello di regressione basato sugli alberi decisionali. La sua forza sta nel combinare le previsioni di molti alberi decisionali per ottenere risultati più robusti e precisi.

```
# 1. Caricamento del dataset
def load_dataset(file_path):
    data = pd.read_csv(file_path)
    return data

# 2. Pre-processing dei dati
def preprocess_data(data):
    # Converti le colonne numeriche in tipo numerico
    numeric_columns = ['Like', 'Followers', 'NumImmagini', 'Numero di Parole', 'Numero di Emoticon',
                        'Numero di Menzioni', 'Numero di Hashtag', 'Associazione a luogo', 'Occorrenze del Luogo']
    for col in numeric_columns:
        data[col] = pd.to_numeric(data[col], errors='coerce')

    # Rimuovi righe con valori nulli
    data = data.dropna()

    # Separazione delle feature e del target
    X = data[['Followers', 'NumImmagini', 'Numero di Parole', 'Numero di Emoticon', 'Numero di Menzioni',
              'Numero di Hashtag', 'Associazione a luogo', 'Occorrenze del Luogo']]
    y = data['Like']

    return X, y

# 3. Divisione dei dati
def split_data(X, y):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
    return X_train, X_test, y_train, y_test

# 4. Addestramento del modello
def train_model(X_train, y_train):
    model = RandomForestRegressor(random_state=42, n_estimators=100)
    model.fit(X_train, y_train)
    return model
```

Il **Gradient Boosting Regression (GBR)** è un modello di regressione basato sempre sugli alberi decisionali, ma agisce in modo diverso rispetto al Random Forest Regressor poiché si basa sulla combinazione di modelli deboli (generalmente alberi di decisione poco profondi) per creare un modello predittivo robusto e accurato.

Tale modello viene utilizzato per account un numero di follower basso andando a considerare solo le caratteristiche della descrizione e numero di immagini

```
def model_operations():
    # Caricamento dei dati

    # Dataset
    current_dir = os.path.dirname(os.path.abspath(__file__)) # Directory corrente del file regressione.py
    project_root = os.path.dirname(current_dir) # Salire di un livello (ExploreWorld)
    file_path = os.path.join(project_root, "Datasets", "postDataset", "instagram_posts.csv") # Percorso assoluto

    data = pd.read_csv(file_path)

    # Selezione delle colonne rilevanti
    features = ['NumImmagini', 'Numero di Parole', 'Numero di Emoticon', 'Numero di Menzioni', 'Numero di Hashtag',
                'Occorrenze del Luogo']
    target = 'Like'

    # Trasformazione Logaritmica dei Like
    data['Like'] = np.log1p(data['Like'])

    X = data[features]
    y = data['Like']

    # Preprocessing: scaling delle caratteristiche numeriche
    preprocessor = ColumnTransformer(
        transformers=[('num', StandardScaler(), features)]
    )

    # Modello
    model = Pipeline(steps=[
        ('preprocessor', preprocessor),
        ('regressor', GradientBoostingRegressor(n_estimators=200, learning_rate=0.1, random_state=42))
    ])

    # Suddivisione del dataset in training e test set
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Addestramento del modello
    model.fit(X_train, y_train)

    return model
```

MAPPA DEI LUOGHI

Per creare la mappa dei luoghi che sono stati postati su Instagram, utilizziamo un density-based clustering, con un eps-distance a scelta (in foto 350 km) e un numero minimo di samples pari a 1.

La mappa è caricata con la libreria Folium, e su di essa, tramite le coordinate, sono posizionati i punti rappresentanti i luoghi.

```
# Funzione per calcolare la matrice delle distanze haversine
def haversine_distance_matrix(coordinates):
    return pairwise_distances(coordinates, metric=lambda x, y: haversine(x, y))

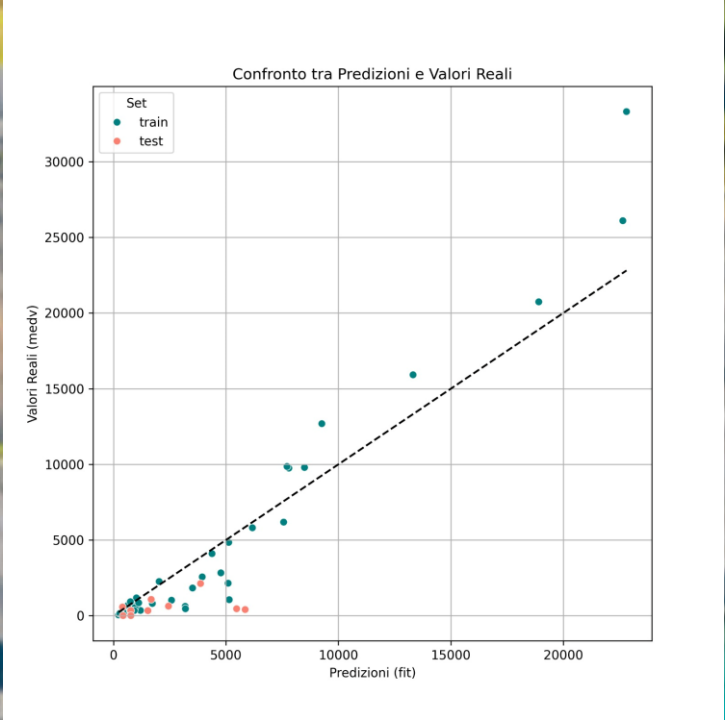
# Calcola la matrice delle distanze in chilometri
distance_matrix = haversine_distance_matrix(coords)

# Applica il clustering DBSCAN
eps_distance = 350 # Cambia la soglia di distanza (in km)
dbscan = DBSCAN(eps=eps_distance, min_samples=1, metric="precomputed")
cluster_labels = dbscan.fit_predict(distance_matrix)

# Crea la mappa centrata sul primo punto
m = folium.Map(location=coords[0].tolist(), zoom_start=6)
```

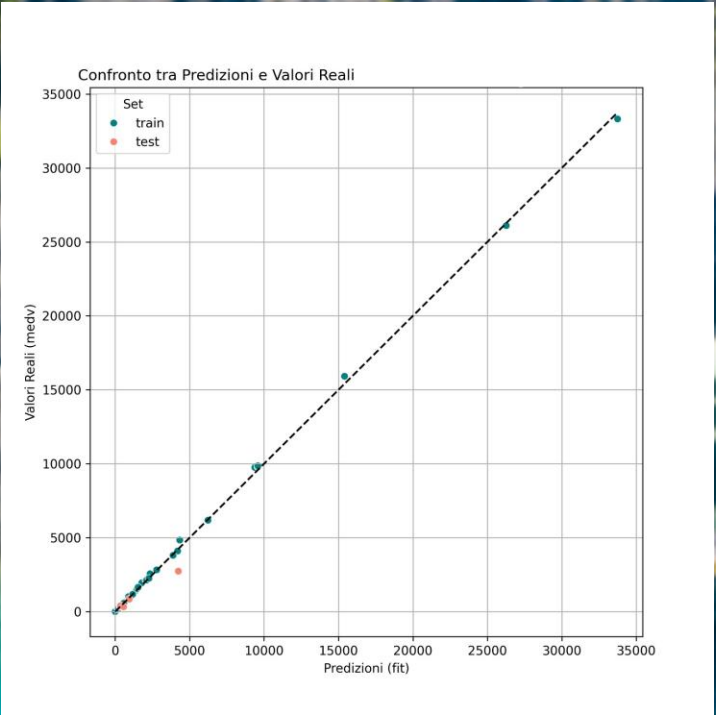


VALUTAZIONE DEI MODELLI



RandomForestRegressor

Mean Absolute Error: 1640.318181
Root Mean Squared Error: 2415.69009



Gradient Boosting Regression

Mean Absolute Error: 467.64384005208854
Root Mean Squared Error : 762.8216279610828

Modello	Accuracy
CittaModel	0.80
MareModel	0.83
MontagneModel	0.77
PeopleModel	0.79
PhotoOrNotModel	0.87
TimeModel	0.86



@EXPLOREWORLD



@EXPLOREWORLD



@EXPLOREWORLD



GRAZIE MILLE PER L'ATTENZIONE