



ModuloFIA: Prodotti Consigliati

De Simone Tearosa	0512115755
Gaudiosi Gabriele	0512113948

anno accademico 2023/2024

Indice

1	Definizione del problema.	3
1.1	Introduzione	3
1.2	Specifica dell'Ambiente	3
1.2.1	Specifica PEAS	3
1.3	Caratteristiche dell'Ambiente	3
1.4	Analisi del Problema	4
2	Data Understanding	5
2.1	Scelta del Dataset	5
2.2	Esplorazione dei Dati	5
2.3	Descrizione dei Dati	5
2.3.1	Variabili Categoriche	5
2.3.2	Variabili Numeriche	5
2.4	Analisi della Qualità dei Dati	6
3	Data Preparation	7
3.1	Data Cleaning	7
3.2	Feature scaling	8
3.3	Feature selection	8
3.3.1	Matrice di Correlazione	8
3.4	Data Balancing	9
4	Data Modeling	11
4.1	LabelEncoder	11
4.2	Configurazione per Addestramento e Validazione	11
4.3	Decision Tree	12
4.4	Random Forest	13
4.4.1	LightGBM	14
4.5	Scelta del classificatore	15
5	Integrazione	16
5.1	Deployment	16
5.2	Implementazione	16
6	Conclusioni	18
6.1	Diversità dei Modelli	18
6.2	Prospettive Future	18

1 Definizione del problema

1.1 Introduzione

Si vuole realizzare un sistema che mira a promuovere uno stile di vita più sostenibile, attraverso la condivisione di oggetti e il riutilizzo di risorse. L'obiettivo principale è ridurre l'impatto ambientale delle attività quotidiane sul pianeta, tramite la creazione di una piattaforma digitale che faciliti la condivisione ed il noleggio di oggetti inutilizzati.

Il sistema consente di prendere in prestito oggetti resi disponibili mediante annunci, promuovendo così la riduzione della necessità di creare nuovi oggetti e di conseguenza gli sprechi di produzione. Il codice sarà reperibile sulla repository di GitHub.

1.2 Specifica dell'Ambiente

1.2.1 Specifica PEAS

- **Performance:** La misura di prestazione utilizzata per valutare l'operato dell'agente sarà basata sulla correttezza dell'identificazione della categoria per il dato utente durante la fase di testing del modello che addestreremo.
- **Environment:** L'ambiente in cui opera l'agente sarà una piattaforma web che opera in ambito commerciale.
- **Actuators:** Gli attuatori che consentiranno all'agente di eseguire azioni saranno costituiti da semplici metodi in grado di effettuare inferenze sui dati e fornire un riscontro successivamente.
- **Sensors:** I sensori a disposizione dell'agente, che riceveranno gli input, saranno rappresentati da un form composto da domande a risposta multipla sul lato utente, i cui dati verranno inseriti in appositi contenitori sul lato dell'agente.

1.3 Caratteristiche dell'Ambiente

- **Completamente osservabile:** all'agente sono disponibili tutte le informazioni utili per effettuare la valutazione.
- **Deterministico:** lo stato successivo dell'ambiente dipenderà solamente dalle azioni compiute dall'agente, poiché sarà l'unico a poter agire liberamente durante la predizione.
- **Sequenziale:** le azioni successive dell'agente dipenderanno da quelle compiute in precedenza, poiché sono basate sulle inferenze dai dati con cui esiste una forte dipendenza.
- **Statico:** mentre l'agente prende decisioni, l'ambiente in cui opera rimane immutato fino a quando non termina.
- **Discreto:** il numero di azioni disponibili per l'agente sarà limitato, principalmente perché potrà solamente rispondere con un set preimpostato di dati in output in base alle informazioni ottenute dai dati.
- **Singolo-Agente:** l'ambiente in cui opera l'agente non prevede la presenza di altri agenti che possano ostacolarlo o aiutarlo durante la predizione della categoria.

1.4 Analisi del Problema

Lo scopo di questo progetto è sviluppare un modulo di Intelligenza Artificiale basato su un modello di Machine Learning che, analizzando i dati ricevuti, sarà in grado di predire quale categoria di oggetti una persona potrebbe essere più propensa a noleggiare.

Questo renderà molto più semplice proporre ad ogni utente prodotti che potrebbero essere di suo interesse evitando di farlo navigare tra migliaia di annunci e facilitando la possibilità dell'effettiva visualizzazione e magari successivo noleggio.

Gli strumenti utilizzati per la realizzazione sono elencati di seguito:

- **Google Colab** come strumento per la scrittura del modello.
- **Python** come linguaggio di programmazione.
- **PyCharm** come IDE.
- **LaTeX** per la documentazione.
- **Overleaf** per scrittura in LaTeX.
- **GitHub** per il versioning.
- **Discord** per la comunicazione tra noi sviluppatori.

2 Data Understanding

2.1 Scelta del Dataset

L'acquisizione dei dati è il processo di raccolta ed organizzazione dei dati necessari per andare a creare un modello di ML. Con gli obiettivi chiari e definiti, siamo andati alla ricerca di un dataset fino a trovarne uno molto interessante su Kaggle. Il dataset preso in considerazione è presente al link: <https://www.kaggle.com/datasets/vetrirah/customer>.

2.2 Esplorazione dei Dati

Il dataset in esame contiene tre diverse tabelle di cui solo la prima era di nostro interesse. In questa c'erano 2627 osservazioni e 10 caratteristiche.

Sebbene come è possibile leggere nella descrizione del dataset gli utenti siano stati divisi in quattro sottoinsiemi, noi abbiamo notato che i dati erano anche ulteriormente suddivisi su una variabile generica che ne avrebbe rappresentato una categoria. Abbiamo usato quella variabile per creare le corrispondenze con le categorie del nostro sito ed utilizzare quindi questo dataset. 'Var_1', da noi poi rinominata 'Categories', è quindi diventata la colonna che useremo per predire a quale categoria di prodotti il nuovo utente potrebbe essere maggiormente interessato. Ora andremo ad analizzare, ed esplorare nel dettaglio, i dati per comprenderli meglio.

2.3 Descrizione dei Dati

Come già detto il dataset contiene 10 caratteristiche che sono poi rappresentate attraverso variabili categoriche (sottoinsieme finito di valori alfanumerici) e variabili numeriche.

2.3.1 Variabili Categoriche

- **Gender:** rappresenta il sesso degli utenti.
- **Ever_Married:** da noi sostituito con **Married**, contiene l'informazione di se un utente è mai stato sposato.
- **Graduated:** se l'utente si è mai diplomato.
- **Profession:** che lavoro fa l'utente.
- **Spending_Score:** da noi è stata rinominata "Budget", esprime quanto l'utente può o vuole spendere.
- **Var_1:** da noi rinominata "Categories", come detto prima, rappresenta una generica categoria a cui l'utente è associato e allo stesso tempo l'output del modello di Machine Learning.

2.3.2 Variabili Numeriche

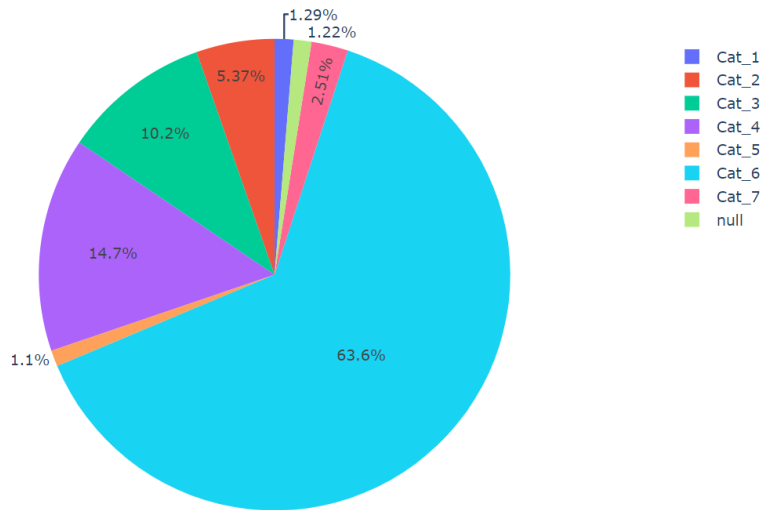
- **ID:** valore unico per ogni osservazione presente nel dataset.
- **Age:** l'età dell'utente.

- **Work_Experience:** l'esperienza lavorativa dell'utente espressa in anni di lavoro.
- **Family_Size:** la dimensione del nucleo familiare.

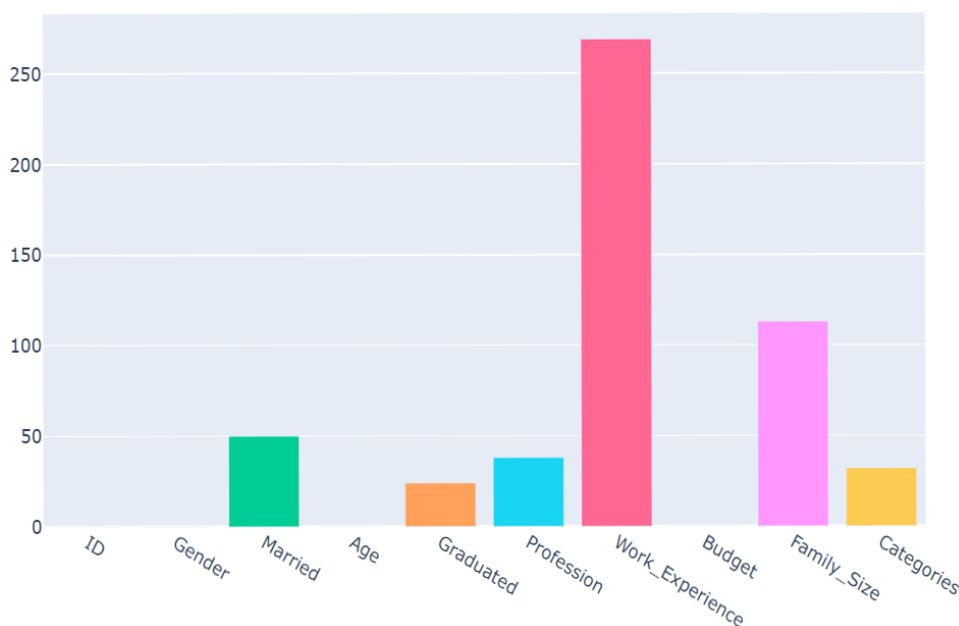
2.4 Analisi della Qualità dei Dati

In questa fase andiamo ad analizzare i problemi di qualità dei dati rilevati durante la fase di esplorazione. La prima cosa che abbiamo scoperto è che il dataset è fortemente sbilanciato verso una delle categorie, infatti analizzando i numeri per ogni categoria vediamo che nel dataset sono presenti:

- Cat_1 34
- Cat_2 141
- Cat_3 267
- Cat_4 386
- Cat_5 29
- Cat_6 1672
- Cat_7 66
- null 32



Questo indica un forte sbilanciamento dei dati, ed è quindi un problema che dovrà essere risolto. Si nota inoltre che sono presenti dei dati null nella nostra caratteristica da prevedere che vanno rimossi. Notando questo siamo andati ad analizzare il dataset per cercare di trovare tutti i dati null presenti al suo interno. Da questa indagine abbiamo notato che per le varie caratteristiche le osservazioni in cui queste erano nulle erano:



- ID 0
- Gender 0
- Married 50
- Age 0
- Graduated 24
- Profession 38
- Work_Experience 269
- Budget 0
- Family_Size 113
- Categories 32

Questa problematica è stata risolta nella fase successiva, cioè la Data Preparation.

3 Data Preparation

3.1 Data Cleaning

In questa fase vanno corretti i problemi individuati in fase di Data Understanding. Per risolvere il problema dei valori null ci siamo preoccupati di come gestirli in maniera singola per ogni caratteristica. La colonna `Work_Experience` per il nostro problema era una feature non rilevante, di conseguenza abbiamo completamente eliminato dal dataset la caratteristica. Ci siamo poi occupati della colonna `Family_Size` decidendo di riempire le righe con dei valori calcolati partendo dalla sua media in relazione alla feature `Age`.

```
# Creo di una variabile per la grandezza media di un nucleo familiare in base  
# all'età di quell'utente  
media_FamilySize = dataset.groupby('Age')['Family_Size'].mean()  
media_FamilySize = (media_FamilySize.round()).astype(int)  
  
# Assegno la grandezza media del nucleo familiare dell'utente ogni volta che incontro  
# un null  
def fill_family_size(row):  
    if pd.isnull(row['Family_Size']):  
        age = row['Age']  
        return media_FamilySize[age]  
    else:  
        return int(row['Family_Size'])  
  
# Applico la modifica alla caratteristica  
dataset['Family_Size'] = dataset.apply(fill_family_size, axis=1)
```

Utilizziamo un processo simile per trovarci i dati per la categoria `Married`, in particolare per `Married` impostiamo a `Yes` se il nucleo familiare è composto da almeno 2 persone.

```
# Assegno valore 'Yes' ogni volta che incontro un null e il nucleo familiare ha più  
# di 2 persone  
def fill_Married(row):  
    if pd.isnull(row['Married']):  
        FSize = row['Family_Size']  
        if FSize >= 2:  
            return 'Yes'  
        else:  
            return 'No'  
    else:  
        return row['Married']  
  
# Applico la modifica alla caratteristica  
dataset['Married'] = dataset.apply(fill_Married, axis=1)
```

Abbiamo infine cancellato tutte le osservazioni in cui i valori di Graduated, Profession o Categories fossero null.

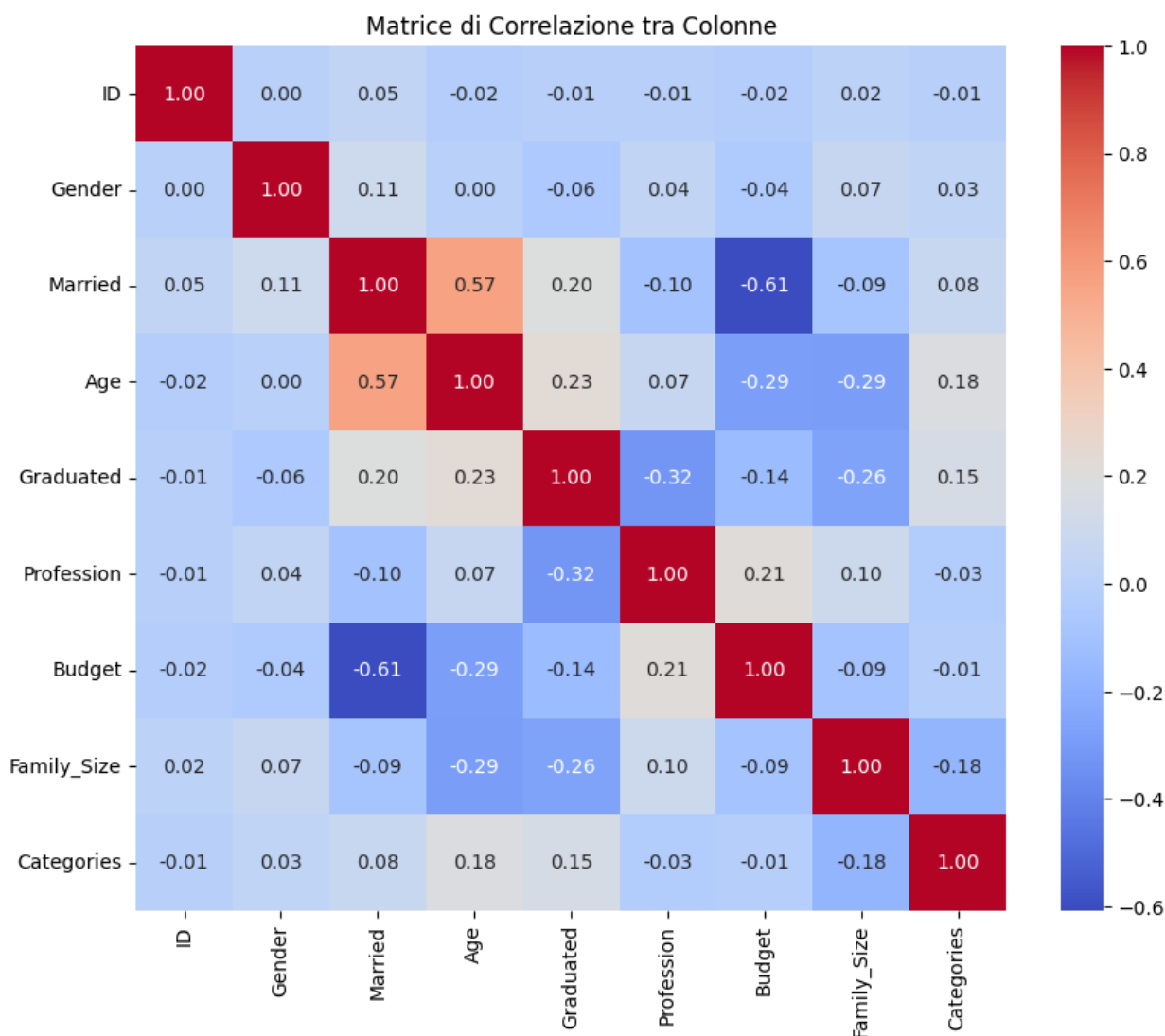
3.2 Feature scaling

Non è stato necessario effettuare alcun tipo di Feature Scaling poiché non ci sono dati numerici che avrebbero potuto influenzare negativamente il modello di Machine Learning.

3.3 Feature selection

3.3.1 Matrice di Correlazione

Utilizziamo una matrice per misurare il grado di correlazione tra ogni coppia di variabili. Bisogna tenere a mente che i **coefficienti di Pearson**, mostrati in ogni cella della matrice, variano tra 1 e -1. Se il valore è ad uno dei due estremi indica una correlazione perfetta, mentre 0 indica assenza di correlazione.



Come possiamo notare osservando la matrice di correlazione le uniche caratteristiche che hanno un grado di correlazione più elevato sono Age e Married, e ciò deriva dall'operazione effettuata durante la fase di Data cleaning. Se ci fossero state due caratteristiche altamente correlate, sarebbe stato opportuno ridurre la dimensionalità unendole in una sola; tuttavia, con un grado di correlazione massimo di 0.57 tale necessità non sussiste.

Considerando anche la matrice di correlazione abbiamo concordato sul fatto che non fosse necessario selezionare delle categorie, ma che avremmo potuto tenerle tutte per sfruttarle poi nel modello.

3.4 Data Balancing

Come analizzato nella fase di Data Understanding, il dataset è decisamente sproporzionato su una delle categorie. Addestrare il modello sul dataset porterebbe ad un elevato rischio di overfitting. Per ridurlo, è necessario andare a bilanciare il dataset.

Per farlo abbiamo scelto di usare un metodo di oversampling, questo perché se avessimo deciso di utilizzare un metodo di undersampling ci saremmo ritrovati ad eliminare quasi totalmente il dataset rimanendo con 203 osservazioni (29 per ogni categoria), decisamente non sufficienti ad addestrare il modello.

Avendo scelto l'overstampng abbiamo dovuto affrontare un altro problema. Come aggiungere almeno 1300 osservazioni a categoria senza assicurarci di finire in overfitting? Per attenuare questo problema abbiamo deciso di utilizzare una tecnica chiamata: **Synthetic Minority Oversampling Tecnique** o più semplicemente **SMOTE**. Questa è una tecnica di oversampling che non crea copie delle osservazioni esistenti, ma invece genera nuove istanze, anche dette sintetiche attraverso l'interpolazione permettendoci di mitigare il rischio di overfitting.

```
from collections import Counter
from imblearn.over_sampling import SMOTE
import numpy as np

X = dataset[['Gender', 'Married', 'Graduated', 'Profession', 'Budget', 'Age',
'Family_Size']]
y = dataset['Categories']

categories_to_oversample = [0, 4, 6]
indices_to_oversample =
dataset.index[dataset['Categories'].isin(categories_to_oversample)]

# Concatena le features e i target selezionati
X_oversample = X.loc[indices_to_oversample]
y_oversample = y.loc[indices_to_oversample]

oversampling_count = 1600
```

```
sampling_strategy =
{category: oversampling_count for category in categories_to_oversample}

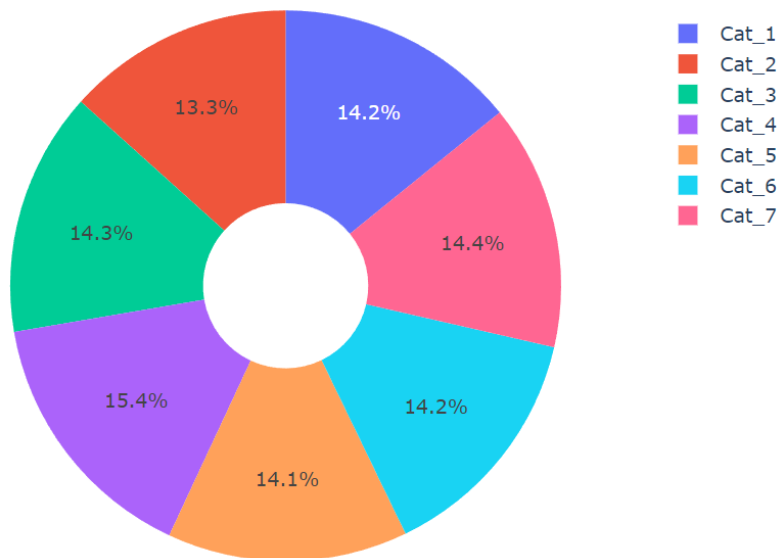
smote = SMOTE(sampling_strategy=sampling_strategy)
X_resampled, y_resampled = smote.fit_resample(X_oversample, y_oversample)

X_oversampled = np.concatenate([X, X_resampled])
y_oversampled = np.concatenate([y, y_resampled])

new_dataset1 = pd.DataFrame(X_oversampled, columns=X.columns)
new_dataset1['Categories'] = y_oversampled
# Il codice di ripete identico per le altre tre classi di minoranza
```

Alla fine abbiamo aggiunto 1600 osservazioni per le classi di "Cat_1", "Cat_5", "Cat_7" e 1400 alle altre tre classi di minoranza raggiungendo questo stato:

- Cat_1 1634
- Cat_2 1539
- Cat_3 1654
- Cat_4 1773
- Cat_5 1628
- Cat_6 1672
- Cat_7 1663



4 Data Modeling

4.1 LabelEncoder

Visto che il nostro dataset aveva una maggioranza di dati categorici, per assicurarci la sicurezza che gli algoritmi potessero lavorare correttamente con le variabili categoriche e trarre informazioni utili dai dati, ci siamo preoccupati di trasformare almeno i dati binari, per esprimerle con variabili numeriche; per farlo abbiamo usato **LabelEncoder**.

Questa viene utilizzato quando si ha una variabile categorica con almeno due categorie e si desidera convertirla in una forma numerica, e si associa quindi lo 0 ad un risultato e l'uno all'altro.

4.2 Configurazione per Addestramento e Validazione

Per la validazione del modello abbiamo scelto di utilizzare la Stratified k-fold validation, o convalida incrociata stratificata. Questa consiste in:

1. Mischiare in maniera casuale i dati di partenza;
2. Dividere i dati in k gruppi, nel nostro caso 10;
3. Per ogni gruppo:
 - (a) Considerare il gruppo 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 ed eliminarlo.

```
from sklearn.model_selection import StratifiedKFold
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score,
confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
import lightgbm as lgb

# Estrai le feature e la variabile target
X = new_dataset[['Gender', 'Married', 'Graduated', 'Profession', 'Budget', 'Age',
'Family_Size']]
y = new_dataset['Categories']

# Inizializza lo StratifiedKFold con k=10
skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
```

Abbiamo poi addestrato tre diversi modelli e li abbiamo confrontati per avere maggiore sicurezza di quale fosse il migliore da usare.

4.3 Decision Tree

Il Decision Tree è un modello di Machine Learning che utilizza una struttura ad albero per prendere decisioni basate su condizioni delle feature, che può essere usato sia per problemi di regressione che di classificazione. Durante l'addestramento, il modello suddivide il dataset in modo iterativo fino a ottenere foglie omogenee rispetto alla variabile target: ogni nodo rappresenta una condizione su una feature, ogni ramo i risultati delle decisioni, ed ogni foglia contiene una predizione finale.

```
# Inizializza il modello Decision Tree
decision_tree_model = DecisionTreeClassifier(random_state=42)

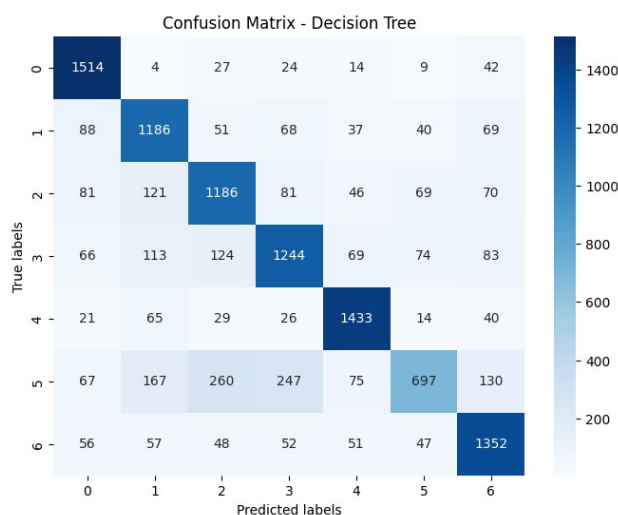
# Liste per memorizzare le performance del modello Decision Tree
decision_tree_accuracies = []
decision_tree_precisions = []
decision_tree_recalls = []

# Liste per memorizzare le previsioni e le etichette reali
decision_tree_predictions_all = []
y_true_all = []

# Itera attraverso i fold
for train_index, test_index in skf.split(X, y):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    # Addestra il modello Decision Tree
    decision_tree_model.fit(X_train, y_train)

    # Fai le previsioni
    decision_tree_predictions = decision_tree_model.predict(X_test)
    decision_tree_predictions_all.extend(decision_tree_predictions)
    y_true_all.extend(y_test)
```



- Accuracy: 0.7944992791394033
- Precision: 0.7939003061724089
- Recall: 0.7944992791394033

4.4 Random Forest

Il Random Forest Classifier è un modello di Machine Learning che si basa su un insieme di alberi decisionali. Durante l'addestramento, vengono creati diversi alberi, ognuno su un sottoinsieme casuale dei dati. Durante la previsione, ogni albero fornisce una previsione e la classe più frequente diventa la previsione finale. Questo approccio riduce il rischio di overfitting rispetto a un singolo albero.

```
# Inizializza il modello Random Forest
random_forest_model = RandomForestClassifier(n_estimators=200, random_state=42)

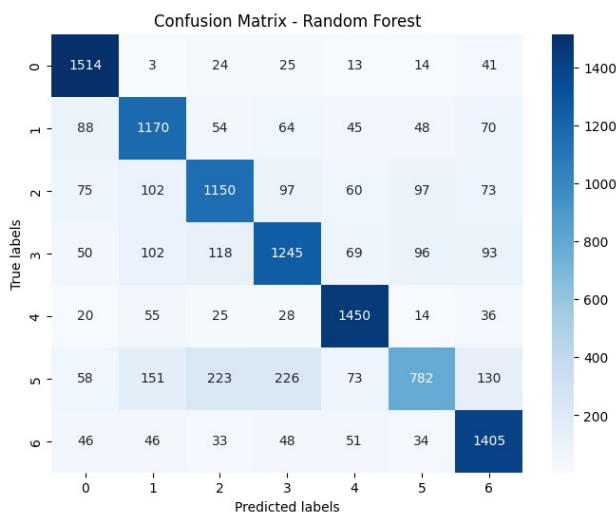
# Liste per memorizzare le performance del modello Random Forest
random_forest_accuracies = []
random_forest_precisions = []
random_forest_recalls = []

# Liste per memorizzare le previsioni e le etichette reali
random_forest_predictions_all = []
y_true_all = []

# Itera attraverso i fold
for train_index, test_index in skf.split(X, y):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    # Addestra il modello Random Forest
    random_forest_model.fit(X_train, y_train)

    # Fai le previsioni
    random_forest_predictions = random_forest_model.predict(X_test)
    random_forest_predictions_all.extend(random_forest_predictions)
```



- Accuracy: 0.8066618609293556
- Precision: 0.805708488767411
- Recall: 0.8066618609293556

4.4.1 LightGBM

LightGBM è un framework di Machine Learning basato su algoritmi di gradient boosting (tecnica di ensemble learning). Nel gradient boosting, viene addestrata una sequenza di modelli deboli, dove ogni nuovo modello cerca di correggere gli errori commessi dai modelli precedenti. Questo viene fatto facendo in modo che ciascun nuovo modello si concentri sugli esempi che sono stati classificati erroneamente o per i quali la previsione ha un grande errore residuo. In particolare abbiamo scelto il LightGBM poiché supporta la parallelizzazione del processo di addestramento, rendendolo estremamente veloce e adatto per applicazioni su larga scala.

```
# Inizializza il modello LightGBM
lgb_model = lgb.LGBMClassifier(random_state=42)

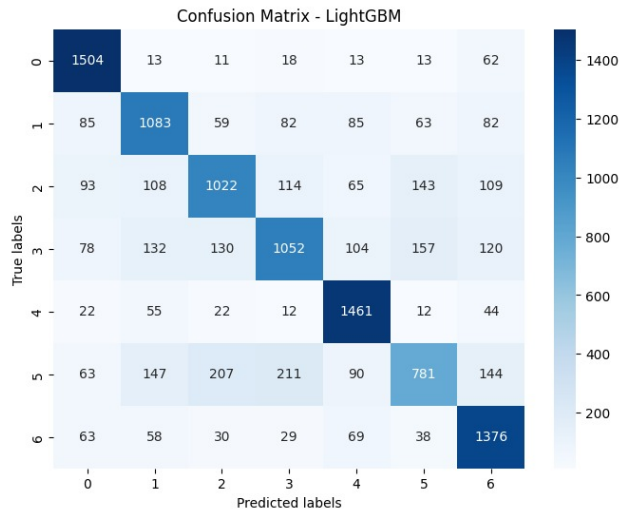
# Liste per memorizzare le performance del modello LightGBM
lgb_accuracies = []
lgb_precisions = []
lgb_recalls = []

# Liste per memorizzare le previsioni e le etichette reali
lgb_predictions_all = []
y_true_all = []

# Itera attraverso i fold
for train_index, test_index in skf.split(X, y):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    # Addestra il modello LightGBM
    lgb_model.fit(X_train, y_train)

    # Fai le previsioni
    lgb_predictions = lgb_model.predict(X_test)
    lgb_predictions_all.extend(lgb_predictions)
```



- Accuracy: 0.7394920705334368
- Precision: 0.7352969870698135
- Recall: 0.7394920705334368

4.5 Scelta del classificatore

Il Random Forest è una tecnica di ensemble learning che combina diversi alberi decisionali per ridurre l'overfitting e migliorare la generalizzazione. Rispetto al Decision Tree singolo, il Random Forest può gestire meglio la complessità del modello.

Quando confrontato con LightGBM, la scelta è dipesa dalle dimensioni del dataset e dalla complessità del problema. LightGBM è noto per l'efficienza computazionale su grandi dataset, ma è spesso più sensibile al rumore. Se il dataset è sufficientemente grande e si desidera una maggiore velocità di addestramento, LightGBM potrebbe essere preferibile, ma noi abbiamo preferito optare per un modello più robusto; quindi, come si vede anche dai dati fuoriusciti dal testing dei modelli, il **Random Forest Model** era quello più adatto alle nostre esigenze.

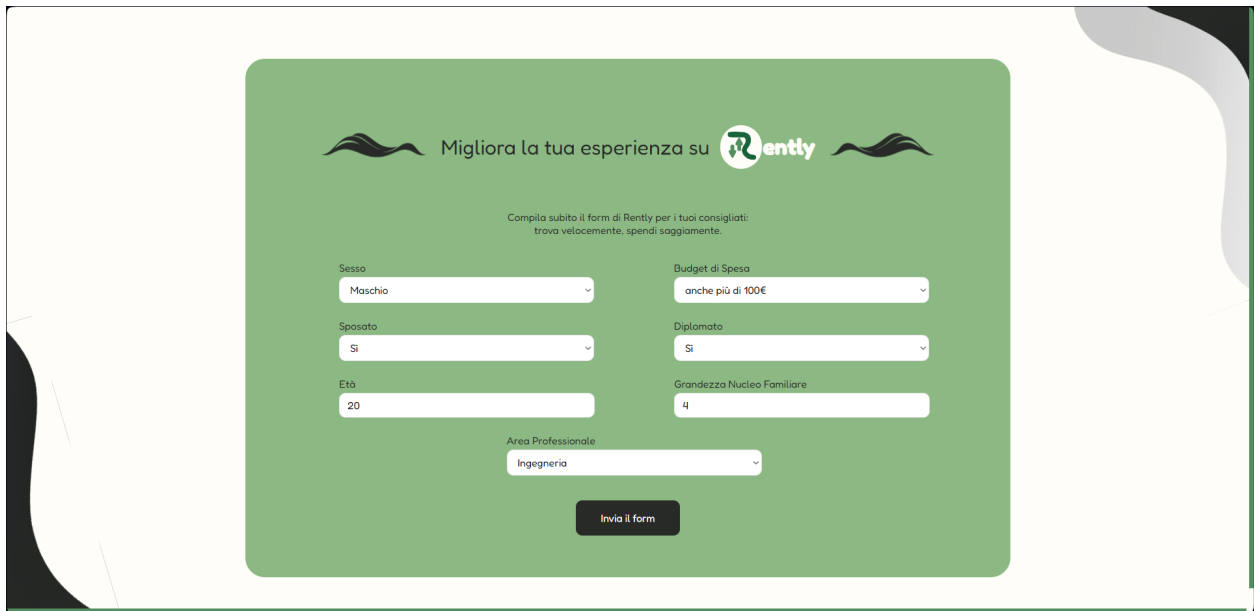
5 Integrazione


5.1 Deployment

Il nostro modello sarà integrato nella piattaforma web Rently, un'applicazione che permette agli utenti di noleggiare una vasta gamma di oggetti. All'interno di Rently, il modello avrà una sezione dedicata dove gli utenti potranno compilare un modulo specifico. Una volta che il modulo viene inviato, un controller apposito instraderà la richiesta al nostro modello. Quest'ultimo elaborerà i dati forniti e restituirà come risultato la categoria di annunci più appropriata per quell'utente. Infine, questo risultato sarà visualizzato all'utente per ulteriori azioni.

5.2 Implementazione

Per soddisfare le esigenze del nostro progetto, abbiamo integrato il modello di Machine Learning all'interno di un'applicazione Flask basata su Python. Questa ospita il modello addestrato per effettuare previsioni sulla categoria di utenti in base a variabili specifiche. Queste variabili vengono fornite al back-end tramite la compilazione di un form lato client.



Migliora la tua esperienza su 

Compila subito il form di Rently per i tuoi consigliati:
trova velocemente, spendi saggiamente.

Sesso	Budget di Spesa
Maschio	anche più di 100€
Spasato	Diplomato
Sì	Sì
Età	Grandezza Nucleo Familiare
20	4
Area Professionale	
Ingegneria	

Invia il form

Utilizzando la libreria scikit-learn, il modello viene caricato in Flask e reso disponibile tramite un endpoint RESTful.

```
@app.route('/predictCategoria', methods=['POST'])
def predict():

    colonne_vuote = ['Gender', 'Married', 'Graduated', 'Profession', 'Budget',
                    'Age', 'Family_Size']
    dataset_colonne = pd.DataFrame(columns=colonne_vuote)

    dati_utente = request.json

    user_input_df = pd.DataFrame([dati_utente])

    for col in dataset_colonne.columns:
        if col in user_input_df.columns:
            dataset_colonne[col] = user_input_df[col]
        else:
            dataset_colonne[col] = 0

    input_df_template = dataset_colonne.reindex(columns=colonne_vuote).fillna(0)

    categoria_predetta_codificata = model.predict(input_df_template)

    categoria_predetta =
    label_encoder.inverse_transform(categoria_predetta_codificata)[0]

    return jsonify({'categoria_predetta': categoria_predetta})
```

Dall'altra parte, l'applicazione Spring Boot in Java è stata progettata per inviare richieste HTTP POST all'applicazione Flask, fornendo i dati utente necessari per le previsioni. L'utilizzo di RestTemplate o WebClient di Spring ha semplificato il processo di invio di dati JSON all'applicazione Flask e la gestione delle risposte JSON ricevute.

Una volta ricevuta la risposta dall'applicazione Flask, contenente la categoria prevista per l'utente, l'applicazione Spring Boot può elaborare ulteriormente i dati e intraprendere le azioni necessarie. Ad esempio, i dati previsti possono essere salvati su un database o utilizzati per personalizzare l'esperienza dell'utente nell'applicazione.

L'integrazione tra queste due tecnologie ha consentito di sfruttare al meglio le risorse e le competenze disponibili, creando un sistema robusto e flessibile per la previsione di categorie utente in tempo reale. La separazione delle responsabilità tra le due applicazioni ha favorito la scalabilità e la manutenibilità del sistema nel lungo termine.

6 Conclusioni

6.1 Diversità dei Modelli

L'opportunità di esplorare e confrontare tre modelli diversi ci ha fornito una comprensione più approfondita delle loro peculiarità. Ogni modello ha presentato vantaggi e limitazioni, e la diversità tra di essi ha arricchito la nostra esperienza di apprendimento. Inoltre, l'analisi dettagliata delle matrici di confusione ci ha permesso di identificare aree specifiche in cui i modelli avevano difficoltà. Questa approfondita esplorazione delle previsioni errate è stata preziosa per capire meglio il comportamento dei modelli in diverse situazioni.

6.2 Prospettive Future

Guardando al futuro, ci rendiamo conto che ci sono ancora opportunità per migliorare e affinare i modelli. Possibili iterazioni del progetto potrebbero includere l'ingegneria delle feature più avanzata, l'utilizzo di ulteriori tecniche di ensemble per combinare i modelli, e la raccolta di ulteriori dati per migliorare la generalizzazione.

In sintesi, questo progetto ci ha fornito non solo competenze pratiche nell'implementazione di modelli di machine learning, ma anche una maggiore consapevolezza delle sfide e delle decisioni coinvolte in un progetto di questo genere. Siamo entusiasti di continuare a esplorare il vasto campo dell'apprendimento automatico e di applicare le nostre conoscenze a sfide sempre più complesse.