RELAZIONE MACHINE LEARNING E DATA MINING

Ingegneria informatica

Nicola Pasolini | Andrea Bonomi

INDICE

1 INTRODUZIONE

- Obiettivi del progetto
- Descrizione del dataset utilizzato
- Librerie utilizzate

2 ANALISI E PREPARAZIONE DATI

- Distribuzione e correlazione dati
- Gestione feature
- Preprocessoring

3 MODELLI E ALGORITMI

- Metriche di valutazione
- Random forest
- Xgboost
- SVM
- Stacking

4 OPTUNA

- Risultati
- Resampling

5 conclusioni

1. Introduzione

1.1 OBIETTIVI DEL PROGETTO

Il progetto di machine learning qui presentato ha come obiettivo principale la costruzione di un modello predittivo utilizzando il dataset "bank-additional-full.csv". Questo dataset contiene informazioni raccolte da una campagna di marketing diretto di un istituto bancario portoghese, basata su chiamate telefoniche a persone già clienti, offrendo la possibilità di sottoscrivere un deposito a termine bancario. L'obiettivo è prevedere l'esito positivo o negativo della sottoscrizione, in base ai dati a disposizione sul cliente.

1.2 DESCRIZIONE DEL DATASET UTILIZZATO

Il dataset "bank-additional-full.csv" è composto da 41,188 istanze e 21 feature, le quali includono attributi demografici, socioeconomici e relativi alla campagna stessa. Di seguito sono riportate alcune delle principali caratteristiche del dataset:

- **Variabili Categoriche**:
    - job: tipo di lavoro del cliente (es. 'admin.', 'blue-collar', etc.)
    - marital: stato civile del cliente (es. 'married', 'single', etc.)
    - education: livello di istruzione del cliente (es. 'basic.4y', 'high.school', etc.)
    - default: indica se il cliente ha un credito in default (es. 'yes', 'no')
    - housing: indica se il cliente ha un mutuo sulla casa (es. 'yes', 'no')
    - loan: indica se il cliente ha un prestito personale (es. 'yes', 'no')
    - contact: tipo di contatto comunicativo (es. 'cellular', 'telephone')
    - month e day_of_week: mese e giorno della settimana in cui è stato effettuato il contatto
    - poutcome: esito della campagna di marketing precedente (es. 'failure', 'success', etc.)

- **Variabili Numeriche**:
    - age: età del cliente
    - duration: durata dell'ultimo contatto in secondi
    - campaign: numero di contatti effettuati durante questa campagna
    - pdays: numero di giorni trascorsi dall'ultimo contatto con il cliente
    - previous: numero di contatti effettuati prima di questa campagna

    - emp.var.rate: tasso di variazione dell'occupazione
    - cons.price.idx: indice dei prezzi al consumo
    - cons.conf.idx: indice di fiducia dei consumatori
    - euribor3m: tasso Euribor a 3 mesi
    - nr.employed: numero di dipendenti

- **Target**:
    - y: variabile target che indica se il cliente ha sottoscritto un deposito a termine (es. 'yes', 'no')

Il dataset è stato scelto per la sua ricchezza di informazioni e per la complessità del problema predittivo che rappresenta. La sfida consiste nel trattare sia variabili numeriche che categoriali, gestire valori mancanti, e costruire un modello robusto che possa generalizzare bene su dati non visti.

In questo progetto verranno applicate diverse tecniche di preprocessing dei dati, selezione e valutazione di modelli e ottimizzazione degli iperparametri, con l'obiettivo finale di ottenere un modello accurato ed efficiente.

1.3 LIBRERIE UTILIZZATE

Le seguenti librerie sono state importate e utilizzate nel progetto:

- **Os e Sys**: Utilizzate per l'interazione con il sistema operativo e la gestione dei percorsi dei file
- **Pandas** e **NumPy**: Utilizzate per la manipolazione di dataset e l'analisi dei dati, operazioni matematiche ed efficienti sugli array.
- **Matplotlib e Seaborn**: Utilizzate per la visualizzazione dei dati, permettono di creare agilmente grafici e diagrammi per esplorare e capire meglio i dati.
- **Scikit-learn**: Utilizzata per la costruzione e la validazione di modelli di machine learning.
- **Optuna**: Utilizzata per automatizzare e ottimizzare la ricerca degli iperparametri migliori dei vari modelli utilizzati.
- **Joblib**: Utilizzata per il salvataggio e il caricamento dei modelli. Permette di serializzare oggetti Python che possono essere riutilizzati successivamente.
- **Ydata Profiling**: Utilizzata per creare report di profilazione dei dati, permettendo una panoramica iniziale molto ampia

- **Imbalanced-learn**: Utilizzata per affrontare il problema dello squilibrio di classe nei dati. Fornisce tecniche di campionamento come oversampling e undersampling.
- **XGBoost**: Una libreria ottimizzata e distribuita che fornisce tecniche di gradient boosting.

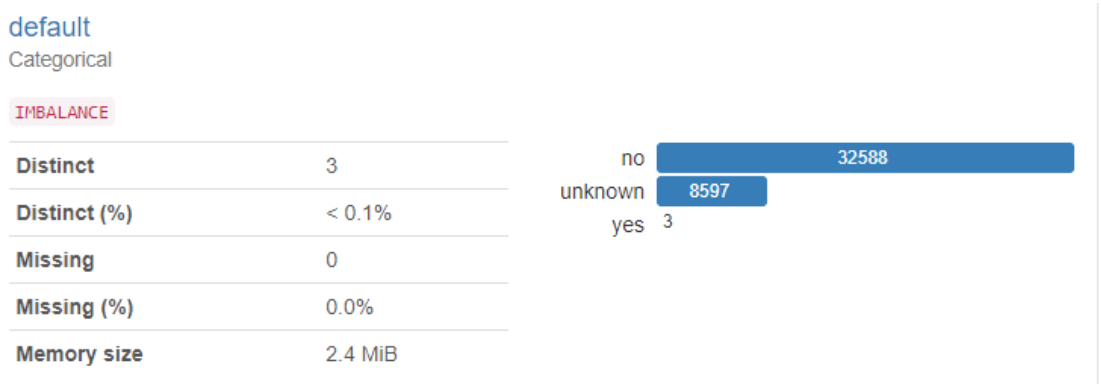
2. Analisi Esplorativa dei Dati

2.1 DISTRIBUZIONE E CORRELAZIONE DATI

Per una prima analisi del dataset, oltre ai metodi messi a disposizione da pandas è stata utilizzata la classe ProfileReport di ydata-profiling, che fornisce una panoramica anche grafica sulle caratteristiche di ogni feature presente nel dataset.

Analizzando i dati iniziali ci siamo resi conto di alcune anomalie:

- La colonna "default", oltre ad essere semanticamente binaria e non categorica, è fortemente sbilanciata dal momento che i "sì" rappresentano una percentuale minore allo 0.1% dei casi totali. Dal momento che il numero di valori positivi è non sufficiente per avere una classe rappresentativa delle istanze abbiamo deciso di escluderla dal dataset.



- I mesi nei quali sono state svolte le chiamate sono pressoché limitati al periodo

Value	Count	Frequency (%)
may	13769	33.4%
jul	7174	17.4%
aug	6178	15.0%
jun	5318	12.9%
nov	4101	10.0%
apr	2632	6.4%
oct	718	1.7%
sep	570	1.4%
mar	546	1.3%
dec	182	0.4%

estivo:

- "Pdays" assume il valore 999 quando il cliente non era mai stato precedentemente contattato e nel 96.3% dei casi così è stato.

  Successivamente andremo a codificare questa informazione con "0" e "1".

Value	Count	Frequency (%)
999	39673	96.3%
3	439	1.1%
6	412	1.0%
4	118	0.3%
9	64	0.2%
2	61	0.1%
7	60	0.1%
12	58	0.1%
10	52	0.1%
5	46	0.1%
Other values (17)	205	0.5%

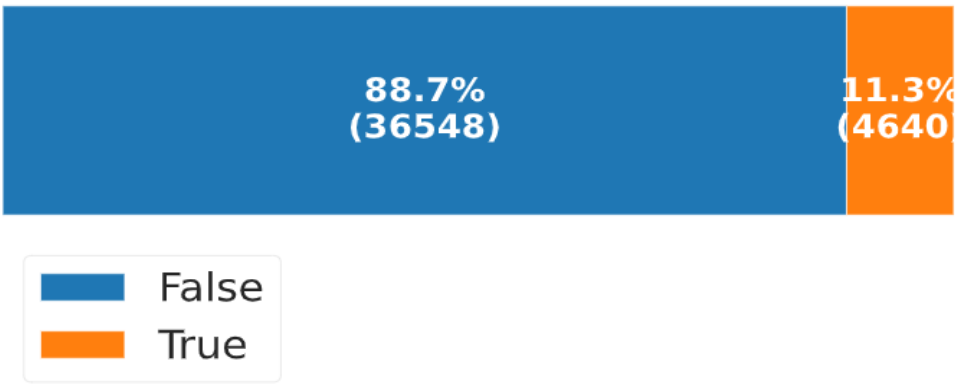
- Dall'analisi di "previous" si capisce che in genere i clienti vengono ricontattati al più una volta, ma ci sono anche rari casi che estendono il follow-up:

Value	Count	Frequency (%)
0	35563	86.3%
1	4561	11.1%
2	754	1.8%
3	216	0.5%
4	70	0.2%
5	18	< 0.1%
6	5	< 0.1%
7	1	< 0.1%

- Dei tentativi precedenti solo il 3.3% ha avuto successo:

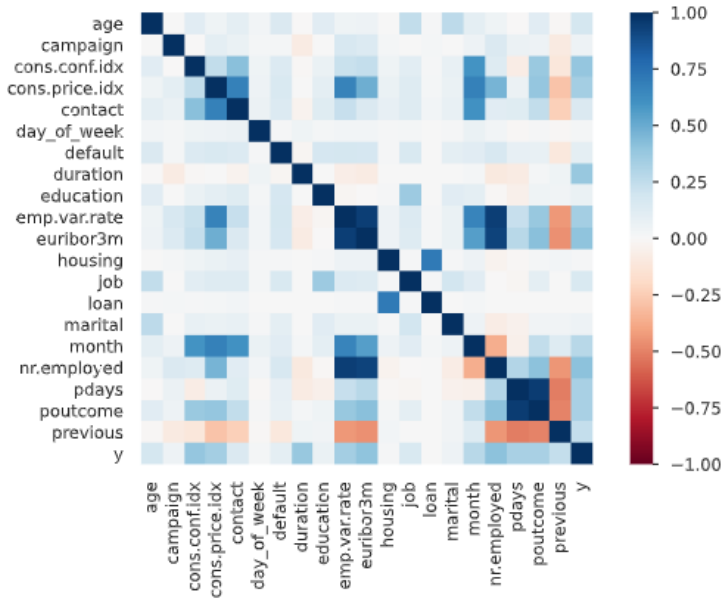
failure	4252	10.3%
success	1373	3.3%

- La variabile target "y" è sbilanciata con l'88.7% dei record registrati come negativi.



Questo ha rappresentato un ostacolo nella creazione di un modello efficace ma era quello che ci aspettavamo da un dataset contenente dati reali.

Heatmap della correllazione tra le variabili:



Dall'analisi delle correlazioni non sono emersi tendenze interessanti utili per la stima della variabile target y.

2.3 GESTIONE FEATURE

Come prima operazione abbiamo eliminato i pochi duplicati presenti nel dataset

```
dataset.drop_duplicates()
```

Ai fini dell'analisi e della corretta implementazione degli algoritmi abbiamo trasformato i valori "unknown" in valori np.nan, ovvero la classe di valori "Not a

number" di numpy, che fornisce diversi meccanismi di gestione degli stessi. In fase di preprocessing utilizzeremo un imputer per sostituirli e poter far lavorare gli algoritmi di apprendimento.

```
dataset = dataset.replace("unknown", np.nan)
```

Successivamente, abbiamo codificato le feature categoriche binarie con i valori interi "0" e "1" attraverso delle labda functions.

```
dataset["pdays"] = dataset["pdays"].apply(lambda x: 0 if x == 999 else 1)
dataset["housing"] = dataset["housing"].apply(lambda x: 0 if x == 'no' else 1)
dataset["loan"] = dataset["loan"].apply(lambda x: 0 if x == 'no' else 1)
dataset["contact"] = dataset["contact"].apply(lambda x: 0 if x == 'telephone' else 1)
dataset["y"] = dataset["y"].apply(lambda x: 0 if x == 'no' else 1)
```

Abbiamo poi riclassificato le variabili in modo automatico in base al numero di valori distinti che presentano, dividendole in binarie, categoriche e numeriche:

```
BINARY_COLS = [col for col, val in dataset.nunique().items() if
val == 2]
BINARY_COLS
```

['housing', 'loan', 'contact', 'pdays', 'y']

```
BINARY_THRESHOLD = 2
CATEGORICAL_COLS = [
    col
    for col in dataset.select_dtypes(include='object')
    if dataset[col].nunique() > BINARY_THRESHOLD

]
CATEGORICAL_COLS
```

['job', 'marital', 'education', 'month', 'day_of_week', 'poutcome']

```
NUMERICAL_COLS = [
    col
    for col in dataset.select_dtypes(include=['int64', 'float64']
)
    if dataset[col].nunique() > BINARY_THRESHOLD
]
NUMERICAL_COLS
```

['age', 'duration', 'campaign', 'previous', 'emp.var.rate', 'cons.price.idx', 'cons.conf.idx', 'euribor3m']

2.4 PREPROCESSOR

Abbiamo diviso i dati di training e di test con un rapporto 80/20 attraverso il metodo `train_test_split()`

Successivamente abbiamo create una pipeline per ogni categoria di feature, in modo da creare una successione di operazioni di preprocessing che preparassero i dati per la fase successiva, ovvero il training degli algoritmi di apprendimento.

Abbiamo gestito le colonne binarie con un Simple Imputer:

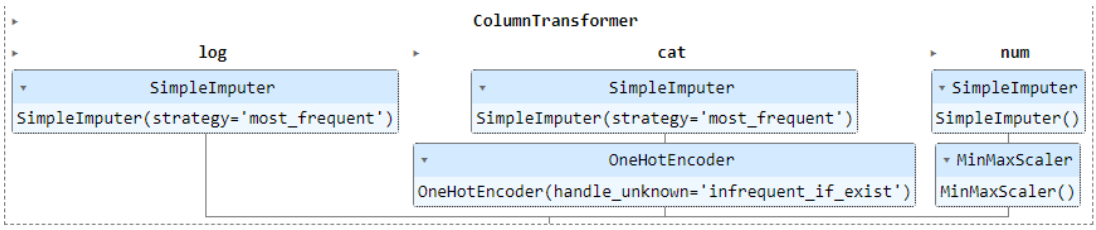
```
logical_pipeline = Pipeline(
    [("imputer", SimpleImputer(strategy="most_frequent"))]
    )
```

Per le colonne categoriche abbiamo aggiunto la codifica "one hot encoder":

```
categorical_pipeline = Pipeline(
    [
        ("imputer", SimpleImputer(strategy="most_frequent")),
        ( "one_hot_encoder",
OneHotEncoder(handle_unknown="infrequent_if_exist"))
    ]
)
```

Le colonne numeriche invece utilizzano un imputer e uno scaler, così da semplificare le computazioni successive:

```
numerical_pipeline = Pipeline(
    [
        ("imputer", SimpleImputer(strategy="mean")),
        #("abs_transformer", FunctionTransformer(np.abs,
validate=True)),
        ("scaler", MinMaxScaler(), )
    ]
)
```



3. Modelli e Algoritmi

Come passo successivo, sono state create ulteriori pipeline, composte dal preprocessor sopra descritto e dal modello scelto per il training e la successiva classificazione. Come esempio, riportiamo il codice con la pipeline della random forest

```
rndf_clf = RandomForestClassifier(random_state=RANDOM_STATE)

pipeline_rndf_clf = Pipeline(
    [
        ("preprocessor", preprocessor),
        ("classifier", rndf_clf)
        ]
)
```

Le altre pipeline seguono la stessa logica, sostituendo di volta in volta il classifier.

La trattazione prosegue mostrando i risultati ottenuti da quattro dei principali modelli di apprendimento automatico: Random forest, Xgboost, Support vector machine e Stacking. In questa fase viene passato loro un solo parametro, il random_state, in modo da ottenere risultati riproducibili, lasciando gli altri iperparametri a valore di default. La scelta degli iperparametri viene delegata ad una sezione successiva, attraverso l'uso di Optuna.

3.1 METRICHE DI VALUTAZIONE

La valutazione delle performance di un modello di machine learning è cruciale per determinare la sua efficacia nel compito di classificazione.

```
def result_analysis(pipeline):
  score = (pipeline.score(X_val, y_val))
  print("score = " + str(score))

  y_pred = pipeline.predict(X_val)

  c = confusion_matrix(y_val, y_pred)
  disp = ConfusionMatrixDisplay(confusion_matrix=c)
  disp.plot()

  plt.show()

  f1 = f1_score(y_val, y_pred)
  print('f1 = '+str(f1))

  print("\n"+"Classification Report:")
  print(classification_report(y_val, y_pred ))
```

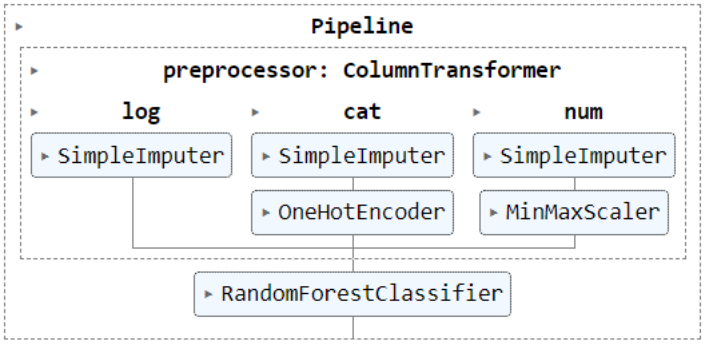
La funzione result_analysis inizia calcolando lo score del modello, che rappresenta l'accuratezza, ovvero la proporzione di previsioni corrette rispetto al totale delle osservazioni.

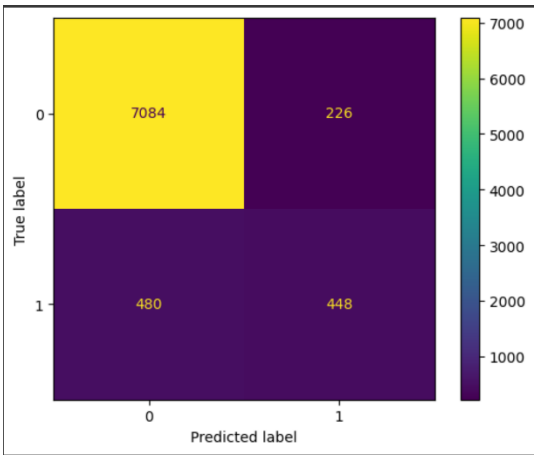
Successivamente, la funzione genera e visualizza una matrice di confusione mostrando il numero di veri positivi, veri negativi, falsi positivi e falsi negativi.

L'F1-score è un'altra metrica utilizzata, che è la media armonica della precision e recall. Questa metrica è particolarmente utile quando si lavora con dati sbilanciati, poiché considera sia le false positive che le false negative.

Infine, la funzione stampa un report di classificazione dettagliato che comprende le metriche appena descritte.

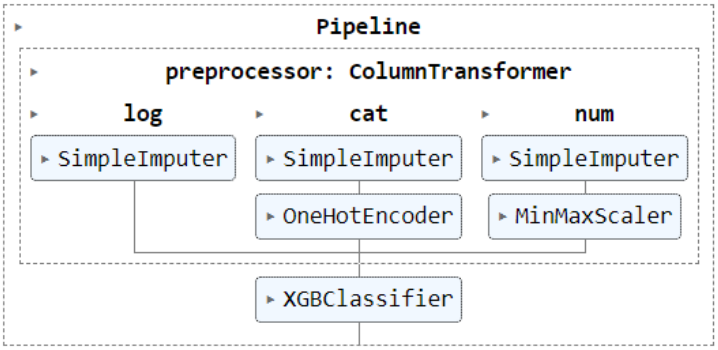
3.2 RANDOM FOREST

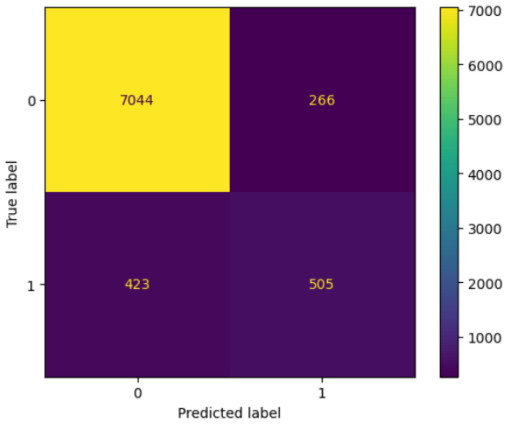




Score: 0.91 F1: 0.55

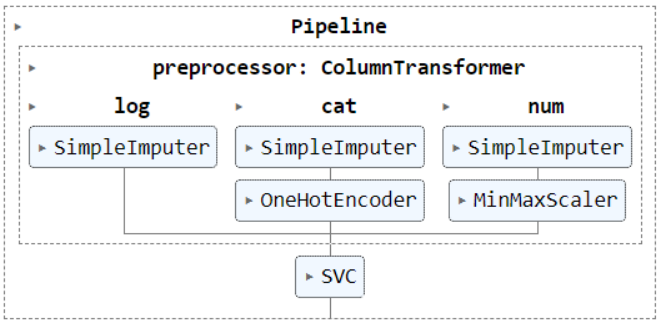
3.3 XGBOOST

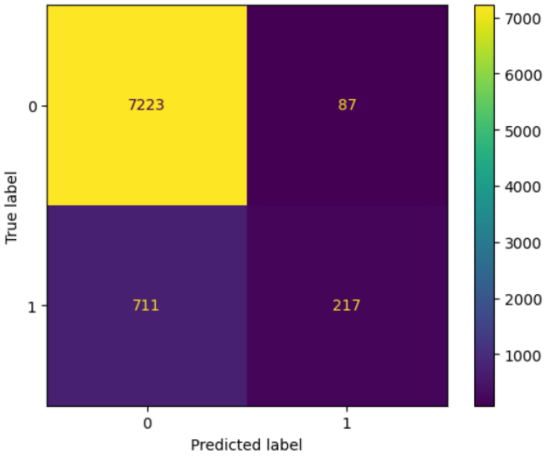




Score: 0.91 F1: 0.59

3.4 SVM





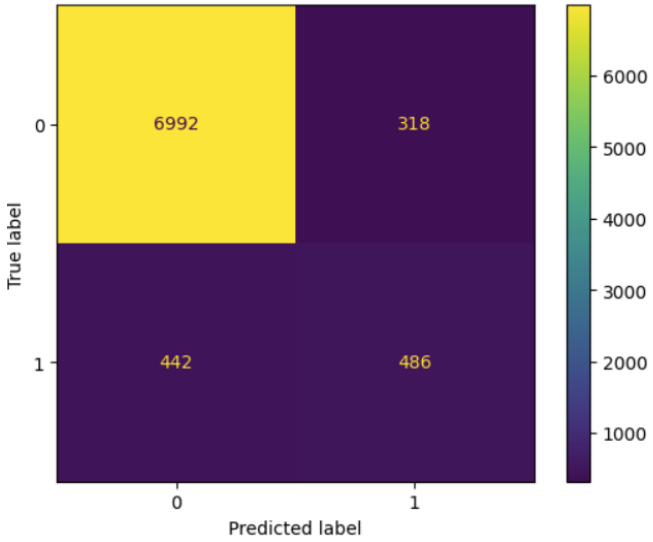
Score: 0.90 F1: 0.35

3.5 STACKING

Lo stacking combina diversi algoritmi di machine learning per costruire un modello più potente e robusto. La figura allegata illustra la pipeline di stacking che ho utilizzato nel progetto.

Abbiamo utilizzato due classificatori di base: `AdaBoostClassifier` e `XGBClassifier`. Questi modelli sono stati addestrati indipendentemente sui dati di addestramento e le loro predizioni sono state utilizzate come input per un meta-modello, un `RandomForestClassifier`, che ha il compito di combinare in modo ottimale le predizioni dei modelli di primo livello.

Purtroppo, l'applicazione della tecnica di stacking non ha migliorato i risultati ottenuti con la Random Forest, ipotizziamo che il meta-modello potrebbe dare un peso significativo alle proprie previsioni rispetto a quelle degli altri modelli, risultando in prestazioni simili a quelle della Random Forest da sola.



Score = 0.9 `f1=0.56`

4. OPTUNA

Abbiamo utilizzato Optuna per l'ottimizzazione iperparametrica dei modelli.

Per fare ciò, abbiamo definito uno spazio di ricerca per ciascun iperparametro e abbiamo lasciato che Optuna esplorasse automaticamente queste combinazioni per trovare la configurazione ottimale. Questo processo ha permesso di migliorare le prestazioni del modello riducendo al contempo il tempo e lo sforzo necessari rispetto all'ottimizzazione manuale.

Abbiamo definito una parte di configurazione comune a tutti i modelli ed una dedicata ai parametri specifici per il modello utilizzato; in particolare, il lavoro si è concentrato sulla random forest e l'Xgboost.

Nel caso della random forest, l'ottimizzazione dei parametri ci ha permesso di ottimizzare i risultati della metrica F1.

Random Forest

```
optuna_param_distributions_rndf = {
    "classifier__n_estimators": IntDistribution(100, 400),
    "classifier__max_depth": IntDistribution(10, 25),
    "classifier__min_samples_split": IntDistribution(5, 20),
    "classifier__min_samples_leaf": IntDistribution(5, 20),
    "classifier__max_features": FloatDistribution (0.1, 1.0),
    "classifier__bootstrap": CategoricalDistribution([True, False]),
    "classifier__criterion": CategoricalDistribution(["gini",
"entropy"])
```

Score: 0.915 F1: 0.590

XgBoost

```
optuna_param_distributions_xgb = {
    "classifier__n_estimators": IntDistribution(50, 300),
    "classifier__max_depth": IntDistribution(3, 15),
    "classifier__learning_rate": FloatDistribution(0.01, 0.3),
    "classifier__subsample": FloatDistribution(0.5, 1.0),
    "classifier__colsample_bytree": FloatDistribution(0.5, 1.0),
    "classifier__gamma": FloatDistribution(0, 5),
    "classifier__min_child_weight": IntDistribution(1, 10)
}
```

Score: 0.917 F1: 0.590

4.1 RESAMPLING

Per migliorare ulteriormente i risultati compensando lo squilibrio del dataset, abbiamo implementato sia tecniche di over-sampling che di under-sampling unita all'ottimizzazione parametrica di optuna. Utilizzando la libreria imbalanced-learn, basata su sklearn, l'obiettivo è migliorare la capacità del modello di predire correttamente la classe minoritaria, maggiormente trascurata dai modelli a causa della rappresentazione.

Optuna, unito al resampling, ha ottimizzato i parametri per massimizzare l'F1-Score, migliorando la capacità del modello di identificare correttamente gli esempi della classe minoritaria a discapito dell'accuracy che evidentemente era troppo influenzata dalla presenza della classe maggioritaria.

Under Sampling

```
from imblearn.under_sampling import RandomUnderSampler

rus = RandomUnderSampler(random_state=RANDOM_STATE)
X_under_resampled, y_under_resampled = rus.fit_resample(X_train,
y_train)
```

Otteniamo un set di forma (7424, 19). Rieseguendo il fit con optuna, otteniamo:

	Score	F1
Random Forest	0. 0.85	0. 0.58
XgBoost	0.86	0.60

Over Sampling

```
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=0)
X_over_resampled, y_over_resampled = ros.fit_resample(X_train,
y_train)

X_over_resampled.shape
```

Otteniamo un set di (58476, 19). Rieseguendo il fit su optuna, otteniamo

	Score	F1
Random Forest	0.88	0.63
XgBoost	0.90	0.62

5. CONCLUSIONI

Il progetto ha evidenziato la difficoltà di gestire dati sbilanciati, abbiamo testato le principali tecniche di pre-processing e machine learning raggiungendo risultati soddisfacenti.

L'algoritmo XgBoost si è dimostrato molto efficace e performante anche senza ulteriori ottimizzazioni.

I modelli di Svm e Stacking, a fronte di tempi computazionali maggiori, hanno fornito risultati simili o leggermente peggiori rispetto agli altri due algoritmi di apprendimento. Proseguendo nella ricerca di ottimizzazione degli iperparametri con optuna, è stato possibile migliorare la classificazione, mantenendo comunque contenuti i tempi computazionali.

Il resampling, come prevedibile, ha portato due risultati diversi: l'under-sampling, pur diminuendo il tempo di calcolo, non è riuscito a migliorare le metriche, mentre attraverso l'over-sampling, con una computazione più complessa e costosa, abbiamo ottenuto i migliori risultati, grazie ad un dataset di training più bilanciato.