Titanic Survivability

Di Tanna Alessandro - Favale Enrico - Di Canio Emanuele - Palella Piergaetano

Dataset e preprocessing

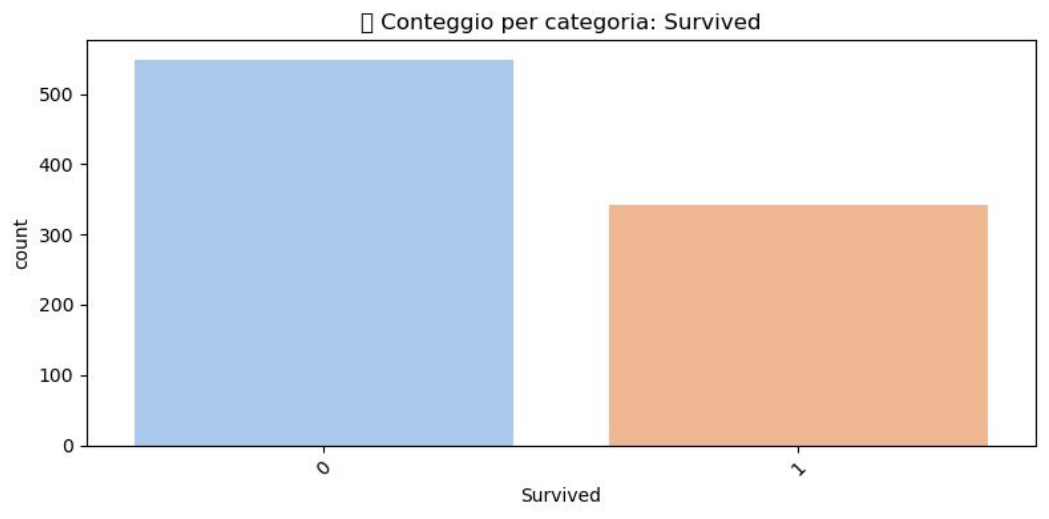
Dataset

Istanze: 891

Feature: 12

- → PassengerId : int
- → Survived : [Yes, No]
- → Pclass : [1, 2, 3]
- → Name : str
- → Sex : [male, female]
- → Age : float
- → SibSp : int
- → Parch : int
- → Ticket : str
- → Fare : float
- → Cabin : str
- → Embarked : [C, Q, S]

Suddivisione delle classi



Classi

1	Sopravvissuto - 342
0	Non Sopravvissuto - 549

Preprocessing

1. Rimozione di colonne inutili
2. Gestione dei valori mancanti
3. Aggiunta di feature riassuntive
4. Normalizzazione dei dati

Rimozione delle colonne inutili

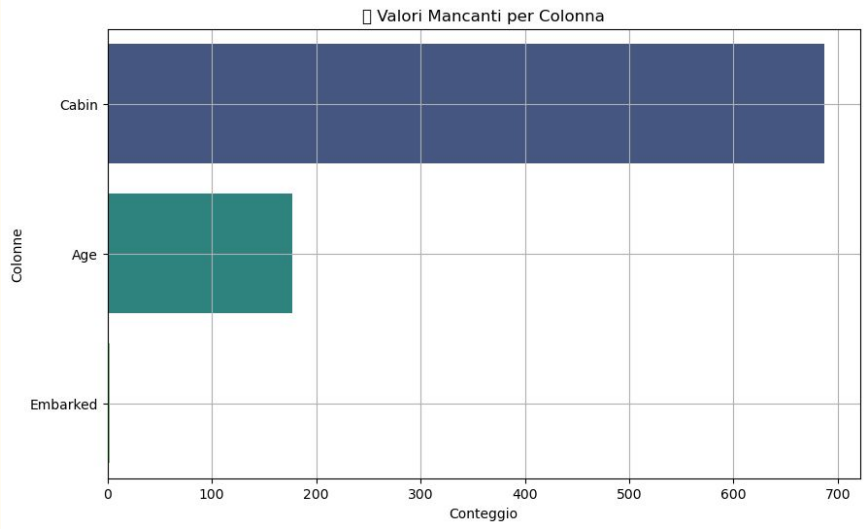
Colonne eliminate:

1. PassengerId
2. Name
3. Ticket
4. Cabin

Queste colonne contengono **dati univoci** che non apportano valore predittivo al modello. La loro rimozione ha aiutato a **ridurre l'overfitting**, evitando che il modello impari a memoria **informazioni irrilevanti.**

Gestione dei valori mancanti

- → Cabin - creazione di una feature binaria '`HasCabin`' per determinare se la cabina è presente
- → Age - gestione dei valori mancanti con la mediana
- → Embarked - gestione dei valori nulli con la **moda** (valore più frequente).



La distribuzione dei **valori mancanti** per colonna.

Aggiunta di feature riassuntive

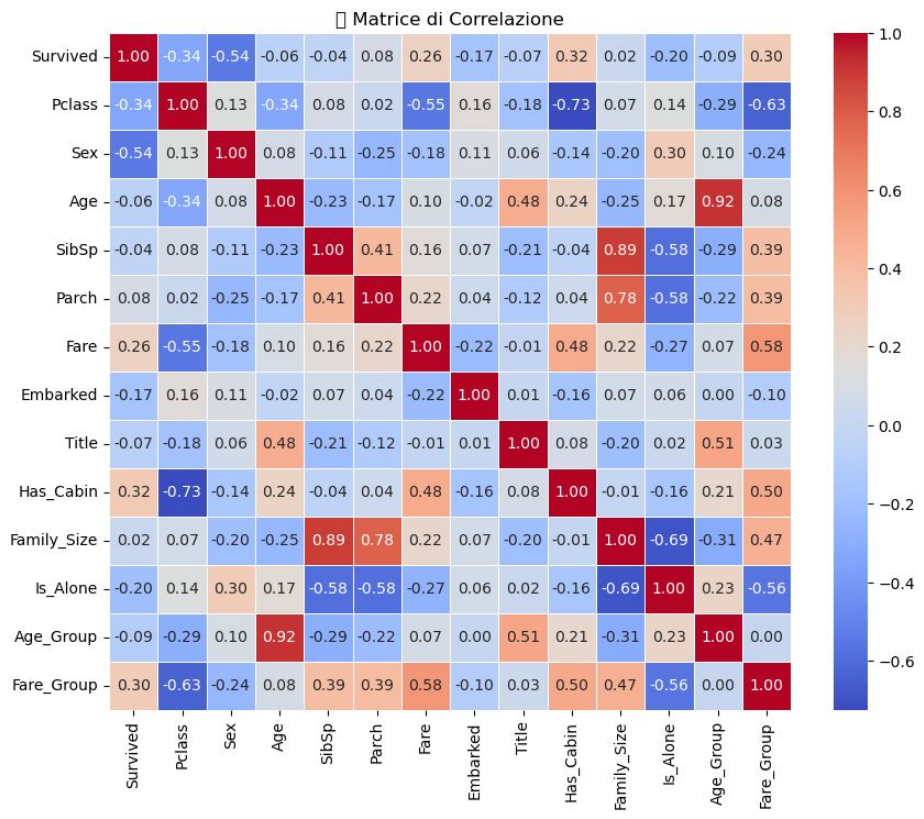
Title - estrazione dei titoli [‘ Rare ’, ‘ Miss ’, ‘ Mr ’] dal nome dei passeggeri. Utile nella definizione della classe sociale del passeggero insieme ad altre feature.

Family_Size, Is_Alone - feature complementari derivanti da **SibSp** e **Parch.**

Age_Group - creazione di 5 fasce di età con soglie: [0, 12, 18, 35, 60, 100].

Fare_Group - creazione di 4 fasce di tariffa.

Analisi della correlazione tra feature



Alcune variabili mostrano forti correlazioni con la probabilità di sopravvivenza:

→ Sex

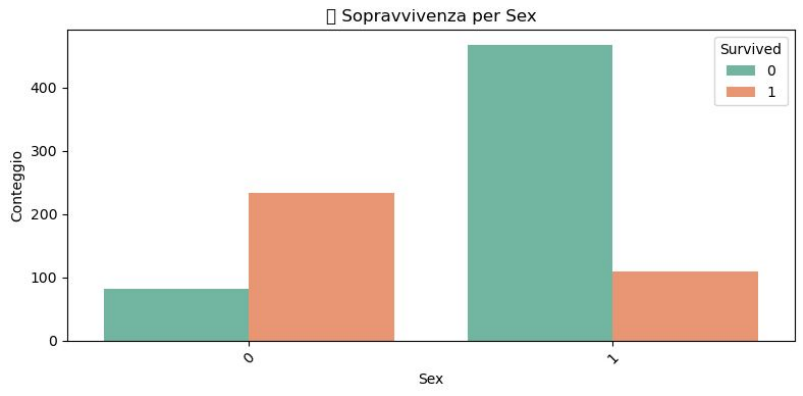
→ Fare / Fare_Group

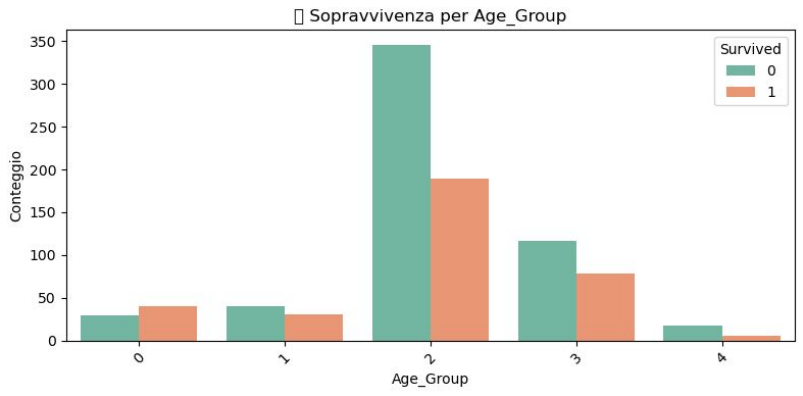
→ Has_Cabin

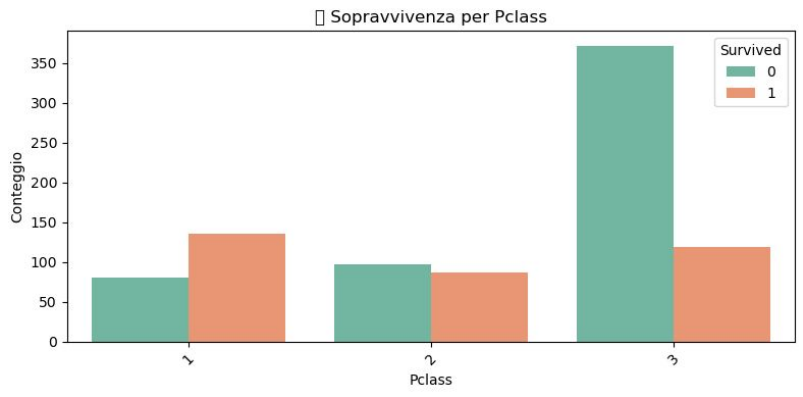
→ Pclass

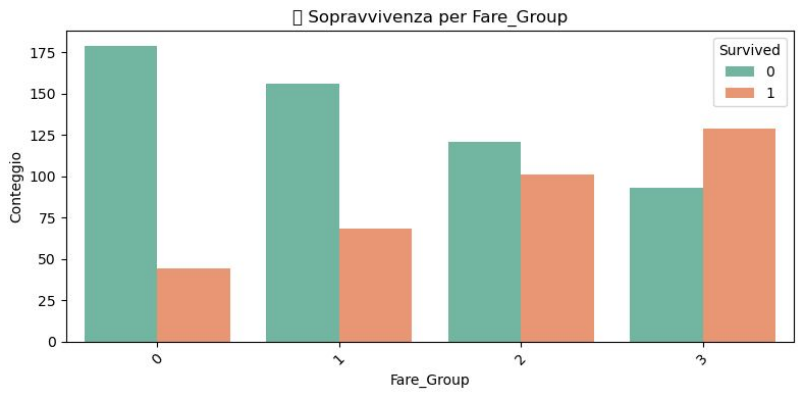
Queste feature sono di particolare interesse nella fase di selezione del modello, poiché potrebbero contribuire significativamente alla predizione dell'output (`Survived`).

Feature di maggiore rilevanza









Normalizzazione dei dati

Obiettivo: portare le feature su una scala comune per migliorare le prestazioni degli algoritmi.

→ viene istanziato uno **scaler** tramite `StandardScaler()`.

→ in seguito viene applicato il metodo `.fit_transform()` sui dati di training.

Risultato:

Media = **0**
Deviazione standard = **1**

Le metriche

1. **Accuracy:** quanto spesso il modello predice correttamente (su tutti i campioni).

2. **Precision**: quanti tra i positivi predetti sono davvero positivi.

3. **F1-score**: media armonica tra precision e recall; utile in caso di classi sbilanciate.

4. **Loss**: misura dell'errore complessivo del modello durante addestramento e validazione.

Struttura della rete

Rete Fully Connected

→ Layer di input con **160 neuroni**, funzione di attivazione `ReLU` e `Dropout` al **10%**.

→ **3** *hidden layer* da **104**, **48** e **112** neuroni, con `ReLU` e `Dropout` rispettivamente al **40%**, **40%** e **20%**.

→ Output layer con **1 unità** e funzione di attivazione `Sigmoid`.

Ogni layer è seguito da una BatchNormalization per migliorare la stabilità dell'allenamento.

Training e Tuning

Training e Tuning

1. Numero di epoche: **37**
2. Batch Size: **64**
3. Training set: **70%**
4. Validation set: **20%**
5. Test set: **10%**
6. Learning rate: **5e-4**
7. Optimizer: ***Adam***
8. Loss: ***BinaryCrossentropy***

Inizializzazione del modello

```python
model = TitanicSurvivalModel(dataset_path = 'data/dataset.csv',
                             epochs = 37,
                             batch_size = 64,
                             test_split = 0.1,
                             validation_split = 0.2,
                             learning_rate = 5e-4,
                            )
```

La configurazione ottimale del modello è stata individuata tramite **auto-tuning** con Keras Tuner, per determinare:

- il numero di layer,
- i neuroni per layer,
- il tasso di dropout,
- il learning rate.

```python
self.model = tf.keras.models.Sequential([
    # Input layer
    tf.keras.layers.Dense(160, activation='relu', input_shape=(self.X_train.shape[1],)),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.1),

    # Hidden layers
    tf.keras.layers.Dense(104, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.4),

    tf.keras.layers.Dense(48, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.4),

    tf.keras.layers.Dense(112, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.2),

    # Output layer
    tf.keras.layers.Dense(1, activation='sigmoid')
])

optimizer = tf.keras.optimizers.Adam(learning_rate=self.learning_rate)
loss = tf.keras.losses.BinaryCrossentropy()

self.model.compile(
    optimizer=optimizer,
    loss=loss,
    metrics=[
        tf.keras.metrics.Accuracy(name="accuracy"),
        tf.keras.metrics.Precision(name="precision"),
        tf.keras.metrics.F1Score(name="f1_score")
    ]
)
```

Autotuning

`RandomSearch` è un algoritmo che esplora **combinazioni casuali** di iperparametri per trovare la configurazione ottimale del modello.

La metrica scelta da ottimizzare è la **validation accuracy**, con l'obiettivo di ***massimizzarla***.

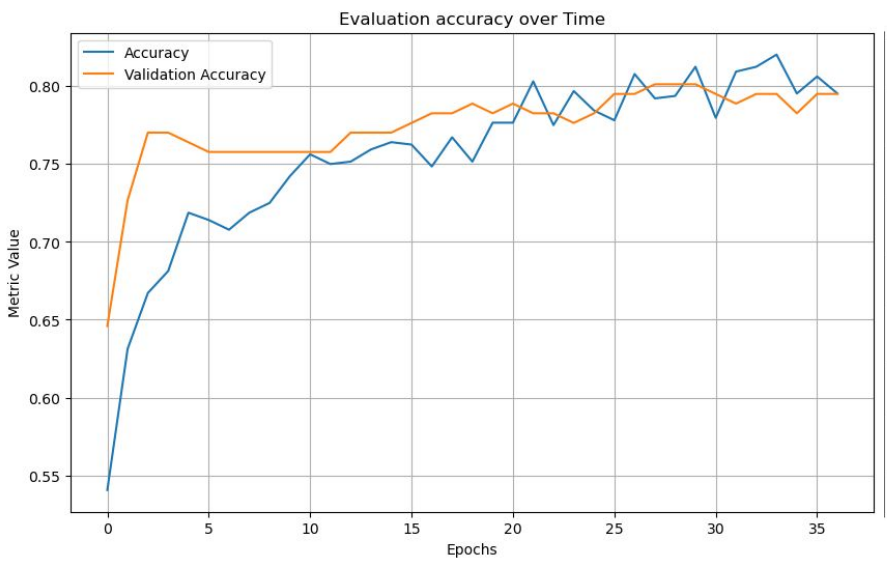
Al termine della ricerca,

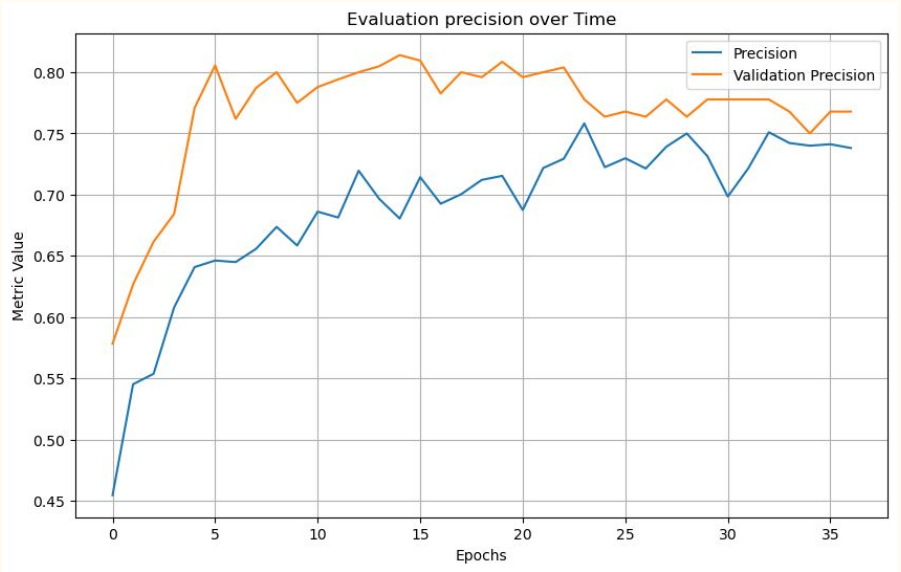
```python
tuner = kt.RandomSearch(
    self.model_builder,
    objective=kt.Objective('val_accuracy', direction='max'),
    max_trials=self.max_trials, # 10
    executions_per_trial=2,
    directory='keras_tuner',
    project_name='titanic_tuning'
)

tuner.search(
    self.X_train,
    self.y_train,
    epochs=self.epochs, # 100
    alidation_split=self.validation_split, # 0.2
    callbacks=[tf.keras.callbacks.EarlyStopping(patience=10)]
)
```

il tuner restituisce i **migliori iperparametri**, che verranno utilizzati per addestrare il modello finale.

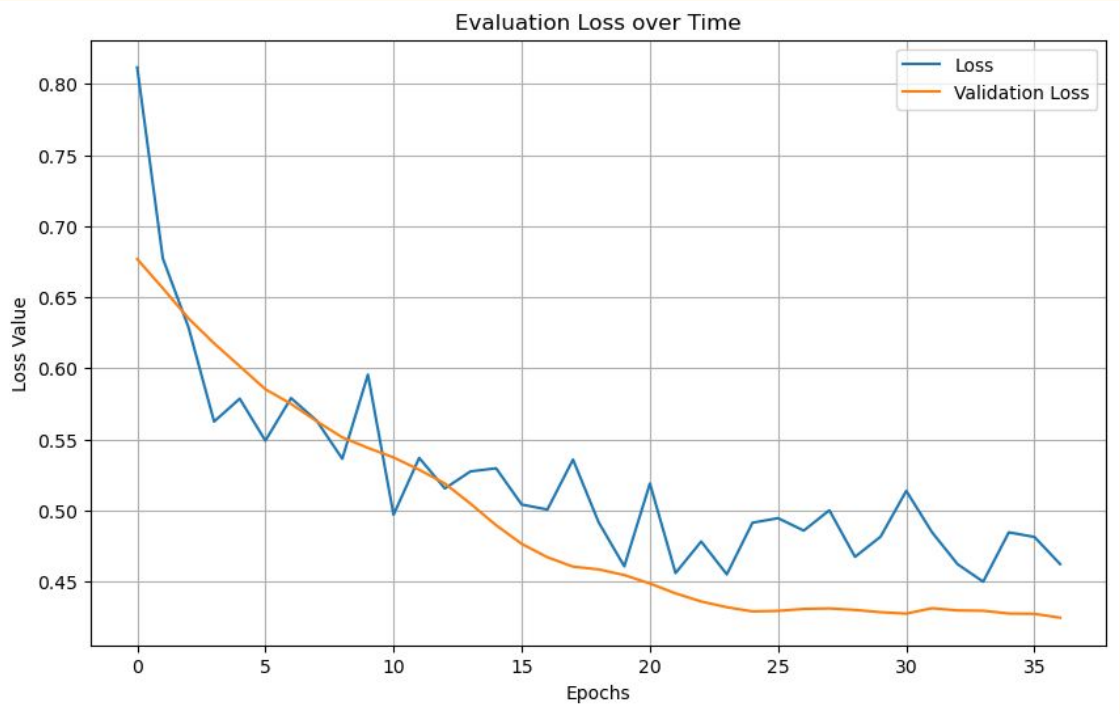
Accuracy e Precision





Sia Accuracy che Precision migliorano progressivamente durante le epoche, indicando che il modello **apprende** i dati in modo efficace.

Loss - BinaryCrossentropy



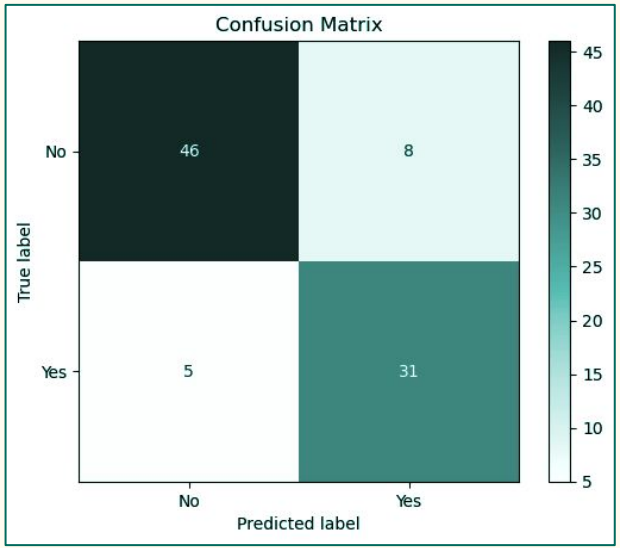
La **BinaryCrossentropy** quantifica l'errore tra probabilità predette (un valore che varia tra 0 e 1) e valori reali (0 o 1), penalizzando le predizioni **sicure ma sbagliate**.

Risultati

Metriche di test

Accuracy	Precision	F1-Score	Loss
84%	87%	57%	0.38

Matrice di confusione



- → **True Negative**: Predetti non sopravvisuti, realmente non sopravvissuti.
- → **False Negative**: Predetti non sopravvissuti, in realtà sopravvissuti.
- → **True Positive**: Predetti sopravvissuti, realmente sopravvissuti.
- → **False Positive**: Predetti sopravvissuti, in realtà non sopravvissuti.