Progetto Big Data e Business Intelligence

Classificazione sulla sopravvivenza dei passeggeri del Titanic

Andrea Adorni mat.267672

Il dataset è recuperabile da https://www.kaggle.com/c/titanic

Il disastro del Titanic è un ottimo esempio di come si possono ricavare informazioni dai dati in relazione al target di riferimento.

1)Look at the big picture

Il nostro task è un task di classificazione supervisionato. Il nostro obiettivo è predire la sopravvivenza di un passeggero attraverso un algoritmo di Machine Learning.

Le misure di performance possibili sono quelle relative alla classificazione: Accuracy (che accantoneremo perché le classi target sono troppo sbilanciate e non darebbe una misura più contestualizzata delle performance), Precision, Recall, F1 e matrice di confusione.

2) Get the data

(0-traintestsplit.py)
Abbiamo un totale di dieci feature iniziali:

- **PassengerId**: semplicemente un identificativo univoco, non interessante
- **Pclass**: La classe del passeggero possono essere 1 per prima,2 per seconda,3 per terza. Potrebbe essere una informazione importante riguardo il ceto sociale del passeggero
- **Name**: identità del passeggero. Apparentemente indifferente al fine del task, ma possiamo ricavarne il titolo sociale per avere una info ulteriore (**Title**)
- **Sesso**: male o female
- **Età**: Età del passeggero. Per neonati sotto l'anno d'età viene messa una frazione tra 0 e 1, ci sono dei numeri che sono segnati nella forma xx.5. Ciò sta a indicare che l'età è stimata
- **Sibsp**: numero di famigliari di una certa categoria che viaggiano col passeggero sib sta per "sibling" e sp sta per "spouse". Quindi fratelli, fratellastri, mariti e mogli (fidanzati non compresi)
- **Parch**: numero di famigliari di una certa categoria insieme al passeggero. Par sta per parent, ch sta per child. Quindi genitori e figli
- **Ticket**: codice identificativo del biglietto. Non viene fornita la logica dietro esso e non sembra fornire informazioni utili
- **Fare**: La tariffa del passeggero. Altra informazione sul ceto sociale del passeggero

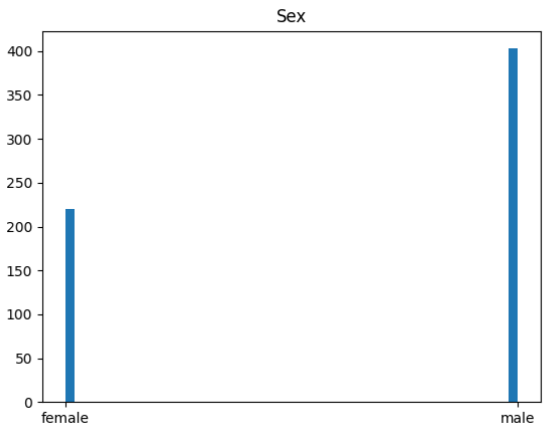
- **Cabin**: Numero della cabina in cui alloggia il passeggero. Forse informazione spaziale
- **Embarked**: Porto di imbarcazione. C sta per Cherbourg, Q sta per Queenstown e S sta per Southampton
- **Survived**: Target di riferimento. 1 sopravvissuto, 0 deceduto.

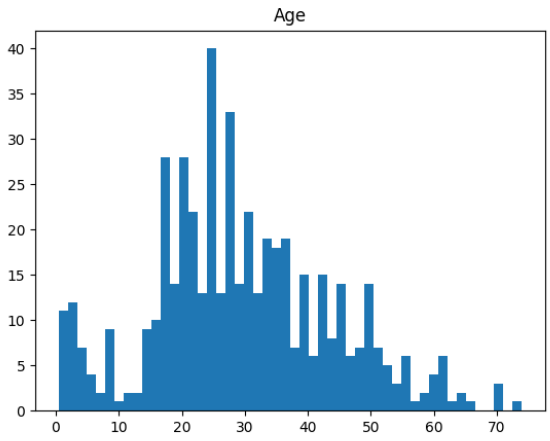
Il dataset è composto da **891 istanze**, con dati in stringa, float e int. Ci sono alcuni dati mancanti che andremo a elaborare in funzione del training
Il dataset è stato diviso in **Training** e **Test set**. Il test occupa il **30% del totale, campionato casualmente. NON BILANCIATO** (trainingset.csv, testset.csv, dataset.csv

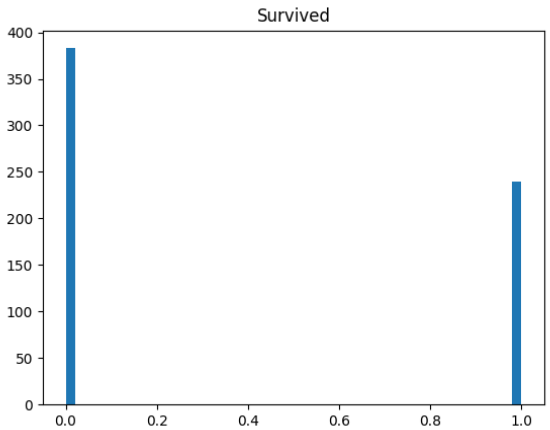
3) Explore the data

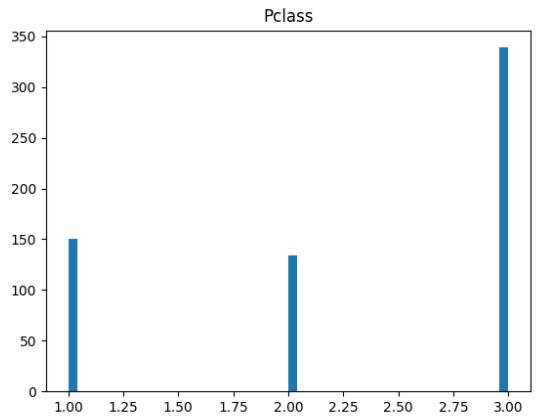
(1-dataexploration.py)

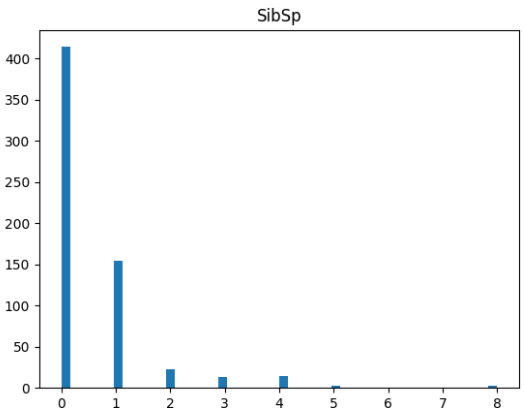
Di seguito abbiamo dei grafici rappresentativi delle frequenze per ogni feature più rappresentativa:

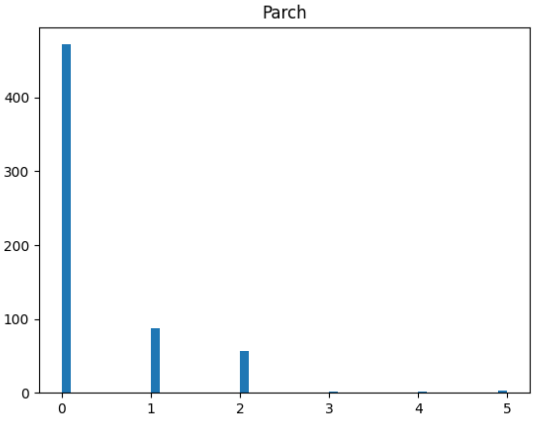


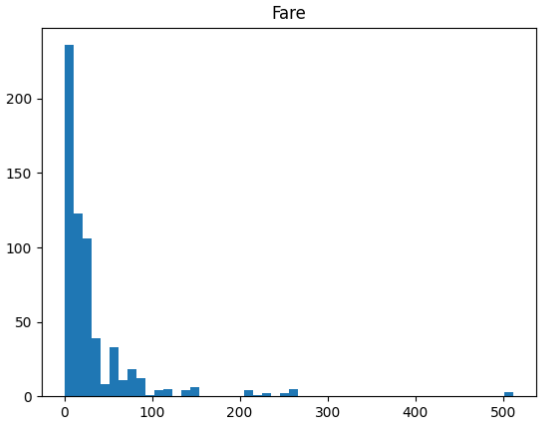


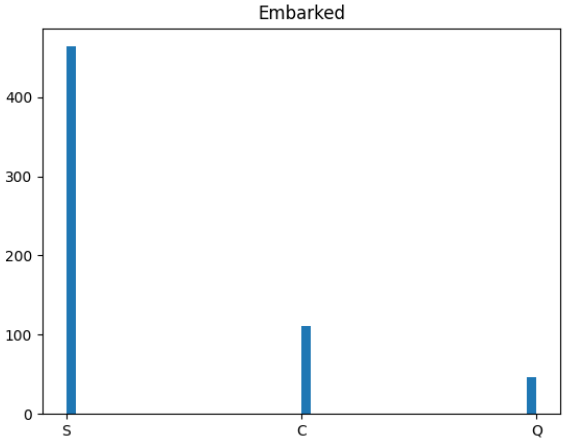


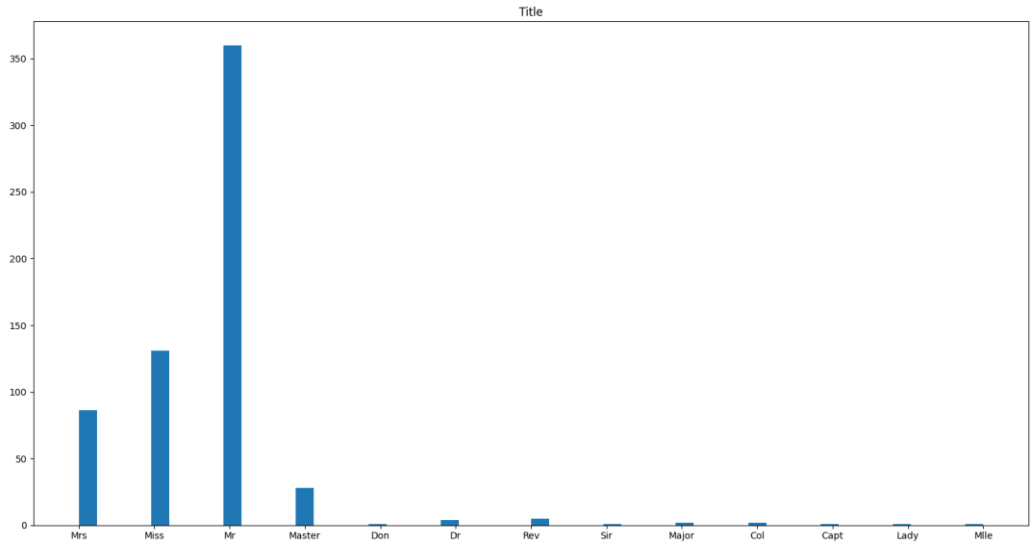












Dai grafici iniziali del training possiamo dedurre alcune informazioni:

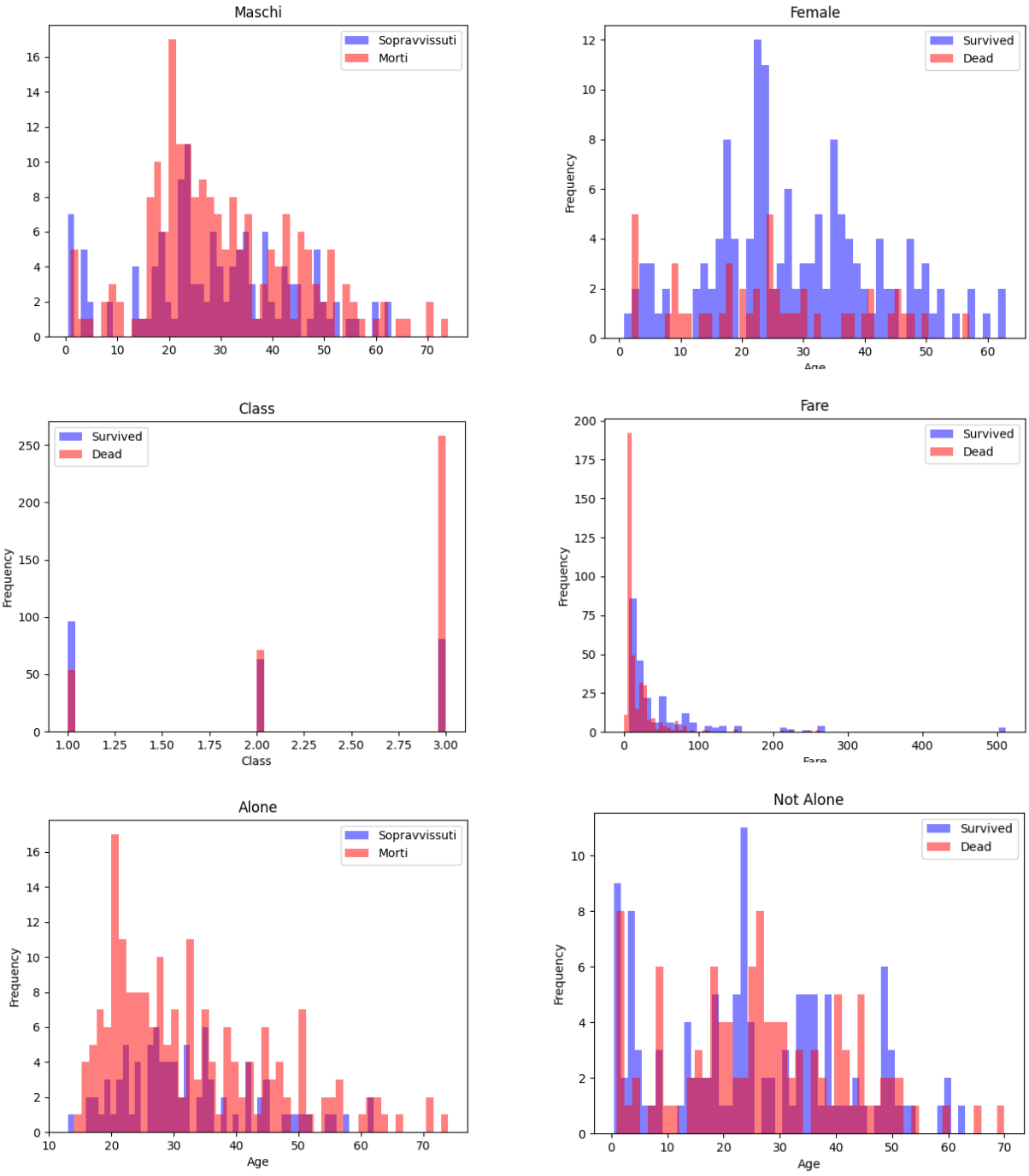
- Ci sono **più passeggeri di sesso maschile** che femminile
- I **deceduti sono quasi il doppio** dei sopravvissuti in proporzione
- La **maggior parte** dei passeggeri **viaggia da sola**
- La distribuzione dei passeggeri si concentra principalmente nei **Fare più bassi e nella terza classe**
- Seconda e prima classe hanno frequenze simili
- Maggiore concentrazione nel titolo Mrs, Mr e Miss
- Age ha distribuzione che tende alla gaussiana, quindi **maggiore concentrazione tra i 20 e 40 anni**, e Fare tende ad una esponenziale inversa

Nel training abbiamo un totale di 623 istanze, con **124 valori mancanti in Age, 483 in Cabin e 2 in Embarked**.

Ho provato a estrarre due nuove feature da Sibsp e Parch, perché ho ipotizzato potesse essere una informazione rilevante capire se un passeggero viaggiasse da solo oppure in famiglia (più probabilità di sopravvivere?)

Quindi ho creato la feature **Relatives** che è la somma di sibsp e parch e la feature booleana **Alone**, che è 1 se relatives = 0

Analizziamo l'incidenza dei sopravvissuti con le feature più promettenti:



Si deduce questo:

- Nei passeggeri di sesso maschile c'è una **maggiore prevalenza di morti**, tranne in età infantile
- Nei passeggeri di sesso femminile c'è una **maggiore prevalenza di sopravvissuti**
- I passeggeri di **ceto più basso** hanno **più probabilità** di non sopravvivere
- **Più probabilità** di morire nei passeggeri che viaggiano **da soli**

Questi dati vengono anche in parte confermati a livello numerico. Le **femmine** hanno un ratio di **0.72** contro i **maschi** con **0.20**. La terza classe è quella con il ratio più basso con 0.24. Chi è **da solo** ha un ratio di **0.30** e chi ha **famigliari** a bordo **0.50**.

4) Prepare your data for the ML algorithms

Cerchiamo ora di analizzare i campi mancanti:

- Età: Abbiamo 124 valori mancanti, però è un campo troppo significativo per ignorarlo. Ho riempito, quindi, i dati mancanti con un valore casuale in un intorno della media che ha la deviazione standard come estremi
- Cabin: Abbiamo 483 valori mancanti, che è un numero veramente importante. Inizialmente pensavo di togliere direttamente la feature, però ho voluto vedere come si distribuivano i valori mancanti in base alla classe e ho visto che più del 90% dei passeggeri in terza classe è senza cabina e lo stesso vale per la seconda classe. Quindi ho pensato avesse un senso e ho deciso quindi di riempire i campi vuoti con None
- Embarked: 2 valori mancanti. Abbiamo due possibilità: eliminare quelle righe oppure affidare un valore. Ho deciso di affidare il valore S che è quello più frequente.

Ho rimosso la feature Ticket perché ha veramente tanti valori unici (circa 500) e non vedo una utilità dietro a questa feature (a meno che non ci sia una relazione tra biglietto e qualche feature ma non sembra).

Per quanto riguarda la codifica:

- **Cabin**: **Ordinal Encoding**, questo perché le cabine sono anche ordinate in base a lettera e numero. Plausibile relazione spaziale tra cabine con stessa lettera o numeri simili.
- **Sex**: **One hot encoding** semplicissimo. 1 femmina, 0 maschio.
- **Alone**: medesimo di Sex. 1 Yes, 0 No
- **Embarked**: **Ordinal Encoding**. 0 è S, C è 1 e Q è 2.
- **Title**: **Ordinal Encoding**. Ho deciso di non optare per la one hot perché si sarebbero create troppe features

I dataset puliti sono stati salvati in trainingclean.csv e testclean.csv

(2-preprocessing.py)
Sono state effettuate delle misurazioni di feature selection con **chi2 e mutual information**. Dalle misurazioni la feature con scoring più basso è stata **Sibsp**, ho deciso quindi di rimuoverla.

Ho deciso di standardizzare il dataset nelle feature non categoriche, insieme a Cabin. Questo perché Cabin arriva a cifre di scala più alta rispetto alle altre categoriche. Ho usato la **standardizzazione Z-Score**. (trainingstd.csv, teststd.csv)

5) Model Selection

(3-model.py)
Come è chiaro dal task, i modelli che sono andato a confrontare sono tutti modelli di classificazione, buona parte riguardano l'Ensemble Learning.

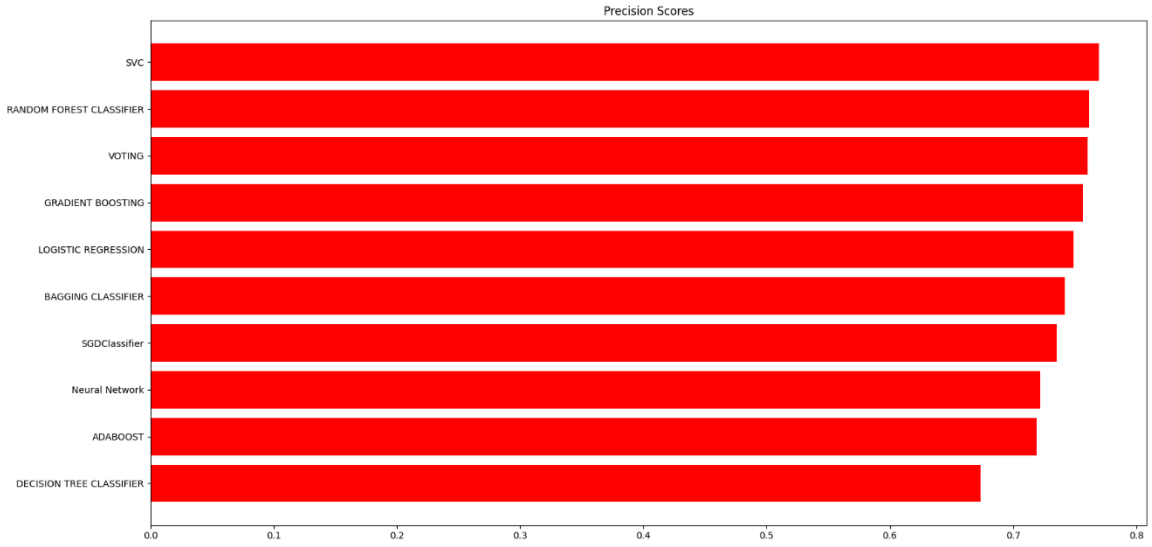
I modelli che ho utilizzato sono questi:

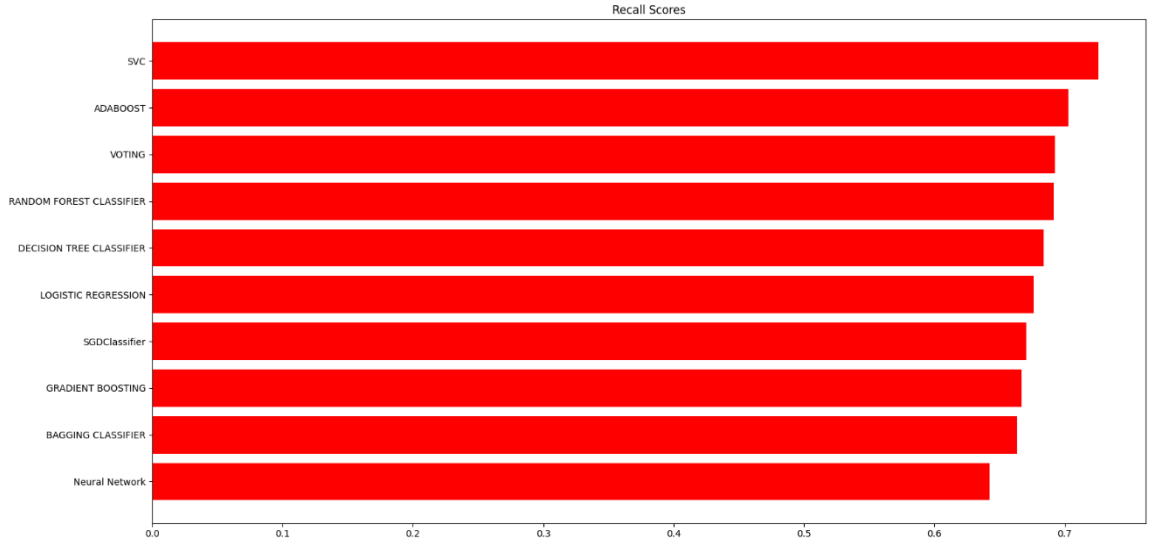
- **Bagging Classifier**
- **Random Forest Classifier**
- **Decision Tree Classifier**
- **Adaboost**

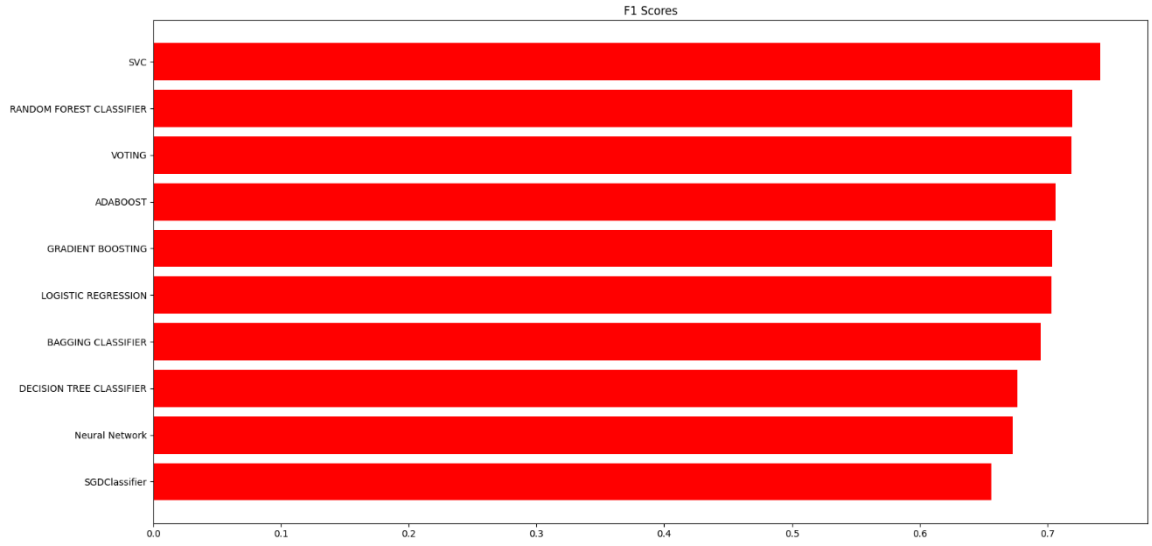
- **Gradient Boosting**
- **Logistic Regression**
- **Neural Network**
- **Hard Voting Classifier**
- **SVC**
- **SGDClassifier**

Tutti i modelli sono stati addestrati e comparati con una **K-Fold di 10** prendendo la media dei rispettivi score per ogni fold. I modelli hanno utilizzato tutti i parametri di default. La rete neurale (neuralnetwork.py) è stata impostata con uno strato di input di 30 neuroni, con una funzione di uscita Relu e un neurone nello strato di output che ha una funzione sigmoide, visto che è un problema di classificazione binaria.

F1	Model
0.656050	SGDClassifier
0.673041	Neural Network
0.676600	DECISION TREE CLASSIFIER
0.694729	BAGGING CLASSIFIER
0.703369	LOGISTIC REGRESSION
0.703916	GRADIENT BOOSTING
0.706229	ADABOOST
0.718902	VOTING
0.719481	RANDOM FOREST CLASSIFIER
0.741520	SVC







SVC è il modello migliore in tutti e 3 gli score stando tra 0.7 e 0.8. Visto che inizialmente con la k-fold sbagliata avevo addestrato la **Random Forest**, ho tenuto i risultati anche del tuning (rifatto) di quel modello

Tuning dei parametri

(4-gridSVC.py)

Per SVC ho deciso di concentrarmi sui parametri che mi sembravano più promettenti che potessero influenzare di più il risultato.

SVC è un modello che fa parte dei Support vector machines (SVMs)

La Grid Search ha utilizzato questa griglia:

- C: parametro di regularization. 1,5,10,15,20,25,30,35,40

- Kernel: funzione di decisione dell'algoritmo. Linear o rbf
- Gamma: coefficiente della rbf. 0.1, 1, 10, 100, scale, auto
- class_weight: balanced o None

I risultati sono stati salvati all'interno di scorestuningSVC.csv. I migliori parametri in ordine di rank score sono:

{'C': 5, 'class_weight': None, 'gamma': 'scale', 'kernel': 'rbf'}
{'C': 1, 'class_weight': None, 'gamma': 'scale', 'kernel': 'rbf'}
{'C': 1, 'class_weight': None, 'gamma': 'auto', 'kernel': 'rbf'}
{'C': 1, 'class_weight': None, 'gamma': 0.1, 'kernel': 'rbf'}
{'C': 1, 'class_weight': 'balanced', 'gamma': 0.1, 'kernel': 'rbf'}

Tutti e 5 i risultati si aggirano a una F1 medio di 0.75.

(4-randomforesttuning.py)

Passiamo ora al tuning della Random Forest, che ha necessitato dipiù tempo e di una ricerca più approfondita.
Ho iniziato con una **Random Search da 1000 prove**, usando sempre come score di valutazione la F1
I parametri che ho impostato sono i seguenti:

- n_estimators: da 100 a 500 divisi in 15 elementi equidistanti
- max_depth: da 10 a 110 divisi in 11 elementi equidistanti + None
- min_samples_split: 2,5,10
- min_samples_leaf: 1, 2, 4
- max_features: auto, sqrt
- bootstrap: True, False

Salvati i risultati nel file randomtuningforest.csv, i primi 5 migliori sono i seguenti:

{'n_estimators': 100, 'min_samples_split': 10, 'min_samples_leaf': 1, 'max_features': 'sqrt', 'max_depth': 90, 'bootstrap': True}
{'n_estimators': 442, 'min_samples_split': 2, 'min_samples_leaf': 2, 'max_features': 'sqrt', 'max_depth': 110, 'bootstrap': True}
{'n_estimators': 271, 'min_samples_split': 5, 'min_samples_leaf': 4, 'max_features': 'auto', 'max_depth': 90, 'bootstrap': True}
{'n_estimators': 157, 'min_samples_split': 5, 'min_samples_leaf': 2, 'max_features': 'auto', 'max_depth': 70, 'bootstrap': True}
{'n_estimators': 357, 'min_samples_split': 10, 'min_samples_leaf': 2, 'max_features': 'sqrt', 'max_depth': 20, 'bootstrap': True}

Dopo la random search ho provato a fare una **Grid Search** (5-gridforestuning.py) su una cerchia più ristretta di parametri, con una cross validation di 5:

- n_estimators: da 100 a 500 diviso in 50 elementi equidistanti
- max_features: auto o sqrt
- max_depth: da 60 a 110 diviso in 10 elementi
- min_samples_split: 2, 5, 10

- min_samples_leaf: 1,2,4
- bootstrap: True

Salvati i risultati nel file gridtuningforest.csv, i primi 5 migliori sono i seguenti:

{'bootstrap': True, 'max_depth': 65, 'max_features': 'auto', 'min_samples_leaf': 2, 'min_samples_split': 5, 'n_estimators': 100}

{'bootstrap': True, 'max_depth': 82, 'max_features': 'sqrt', 'min_samples_leaf': 4, 'min_samples_split': 2, 'n_estimators': 108}

{'bootstrap': True, 'max_depth': 87, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 5, 'n_estimators': 108}

{'bootstrap': True, 'max_depth': 87, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 100}

{'bootstrap': True, 'max_depth': 60, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 5, 'n_estimators': 157}

6) Final evaluation

(6-final.py)

Sia per SVC che per la Random Forest ho utilizzato la griglia di parametri con il rank più alto dai file csv.

Addestrando i modelli e facendo predizioni per 10 volte ho calcolato la media degli score.

Per la Random Forest:

- **Precision**: 0.8897293947555707
- **Recall**: 0.7578431372549019
- **F1**: 0.8183966686264554

Per SVC:

- **Precision**: 0.8297872340425532
- **Recall**: 0.7647058823529411
- **F1** 0.7959183673469388

Le misurazioni mostrano come la Random Forest, nonostante nel model comparison fosse nettamente più bassa rispetto a SVC, il tuning dei parametri è stato molto più efficace, portandolo ad avere una precisione da 0.89 e una F1 di 0.81

SVC si comporta leggermente meglio della Forest nella recall. Entrambi i modelli sono più abili nel limitare i falsi positivi e meno i falsi negativi.

Il tuning dei parametri ha portato Random Forest ad avere risultati migliori sia dei valori di default, sia della SVC, che a sua volta, invece, ha ottenuto dei risultati molto simili a quelli di default nonostante la grid search.

Comunque sia, ora si spiega come mai i valori nel model comparison, con la vecchia k-fold, sembravano mostrare un overfitting così alto.