



UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Corso di Laurea Magistrale
in Ingegneria Informatica

Relazione dell'elaborato per il corso di
Machine Learning e Data Mining

Titanic - Machine Learning from Disaster

Marco Sartorelli

Matricola n. 719788

Marcello Manenti

Matricola n. 719487

Anno Accademico 2020/2021

INTRODUZIONE

La seguente relazione descrive il processo di trattamento e utilizzo dei dati per la partecipazione alla competizione Kaggle "*Titanic - Machine Learning from Disaster*".

Nella competizione, sono stati forniti dei dataset contenenti informazioni sui passeggeri presenti sul Titanic, come nome, età, genere, condizione socio-economica e altro.

L'obiettivo della competizione è quello di costruire un modello predittivo che preveda nel modo più accurato possibile la sopravvivenza o la morte di un passeggero date le sue informazioni generali.

La prima fase è stata dedicata all'analisi dei dati disponibili, all'individuazione di correlazioni e all'estrazione di dati utili dagli attributi presenti.

Nella seconda fase invece sono stati applicati diversi algoritmi di previsione, e generati i rispettivi files per la sottomissione alla competizione Kaggle.

STRUMENTI UTILIZZATI

Il lavoro è stato svolto sulla piattaforma Google Colab, il linguaggio utilizzato è il Python, con le seguenti librerie:

- matplotlib
- numpy
- pandas
- seaborn
- sklearn
- keras

I DATI

I dati forniti nella competizione sono divisi in due set, training set e test set.

Il training set è un dataset di 891 righe e 12 colonne, che sono:

- PassengerId → codice univoco assegnato ad ogni passeggero
- Survived → il target, cioè la sopravvivenza di ogni passeggero (0 = No, 1 = Sì)
- Pclass → la classe del biglietto (1 = prima, 2 = seconda, 3 = terza)
- Name → nome del passeggero
- Sex → genere del passeggero
- Age → età del passeggero
- SibSp → numero di fratelli/coniugi a bordo del Titanic
- Parch → numero di genitori/figli a bordo del Titanic
- Ticket → numero del biglietto
- Fare → prezzo del biglietto
- Cabin → numero della cabina
- Embarked → porto di imbarco ('C'=Cherbourg, 'Q'=Queenstown, 'S'=Southampton)

Il test set è un dataset di 417 righe e 11 colonne (senza la colonna target)

EXPLORATORY DATA ANALYSIS

Come prima cosa, analizziamo le differenze nei dati in base al sesso.

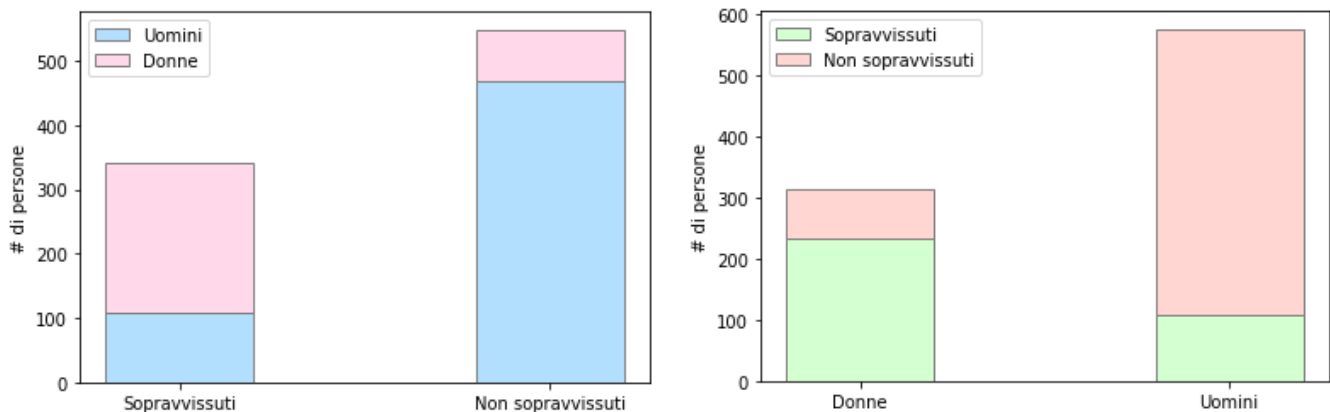
Con una semplice operazione, calcoliamo le percentuali di sopravvivenza per ogni genere rispetto al totale, e riusciamo chiaramente a vedere come i passeggeri di sesso femminile abbiano una percentuale di sopravvivenza sostanzialmente più alta (74,2%) rispetto ai passeggeri di sesso maschile (18,9%).

Ciò può essere dovuto principalmente a due fattori:

1. alla politica di emergenza che viene adottata spesso in caso di disastri, del “prima donne e bambini”
2. al fatto che una più grande percentuale di passeggeri di sesso femminile era in possesso di biglietti di prima e seconda classe (che potrebbero risultare in una possibilità di sopravvivenza maggiore)

```
Ratio di sopravvivenza femminile: 0.7420382165605095
Ratio di sopravvivenza maschile: 0.18890814558058924
```

Vediamo anche graficamente, le quote di entrambi i sessi sui passeggeri sopravvissuti e non, e visualizziamo il ratio calcolato sopra

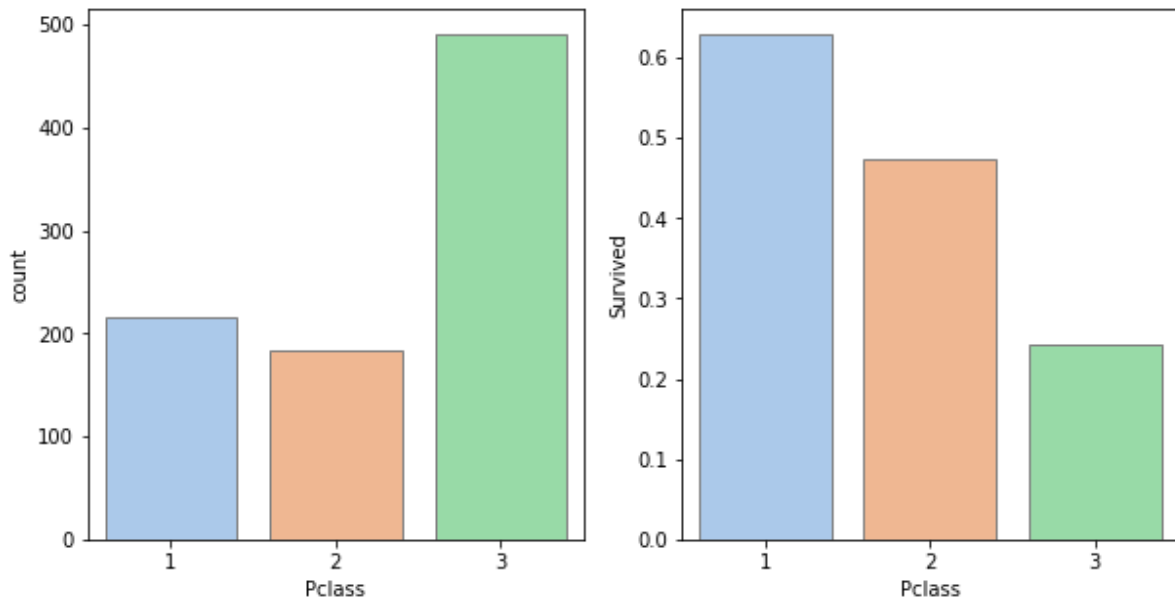


Andiamo quindi ad esaminare i dati relativi alla classe del biglietto, e a verificare se sia fondata l'assunzione fatta sopra nel punto 2.

Dopo aver contato i sopravvissuti per ogni classe, calcoliamo le percentuali, e vediamo che per la prima (63%) e per la seconda (47,3%) classe, la porzione di sopravvissuti è maggiore rispetto alla terza (24,2%)

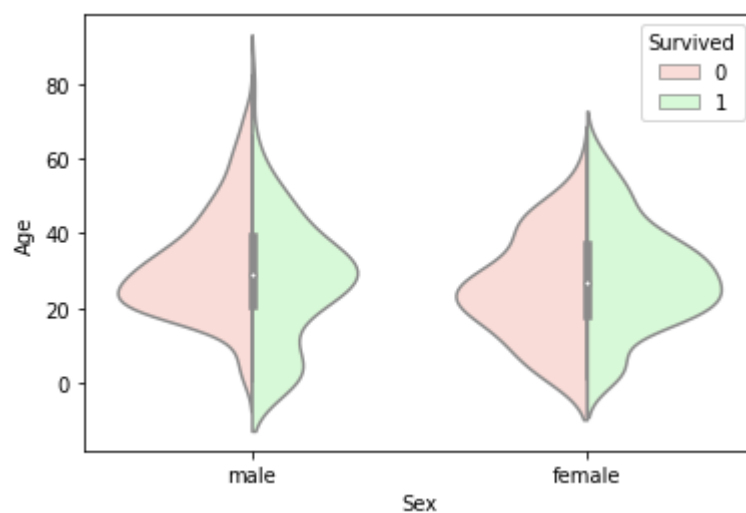
```
Sopravvissuti in prima classe : 136 con un ratio : 0.6296296296296297
Sopravvissuti in seconda classe : 87 con un ratio : 0.47282608695652173
Sopravvissuti in terza classe : 119 con un ratio : 0.24236252545824846
```

Nei due grafici sottostanti vediamo rappresentati il conteggio dei passeggeri per ogni classe e la percentuale di sopravvivenza.

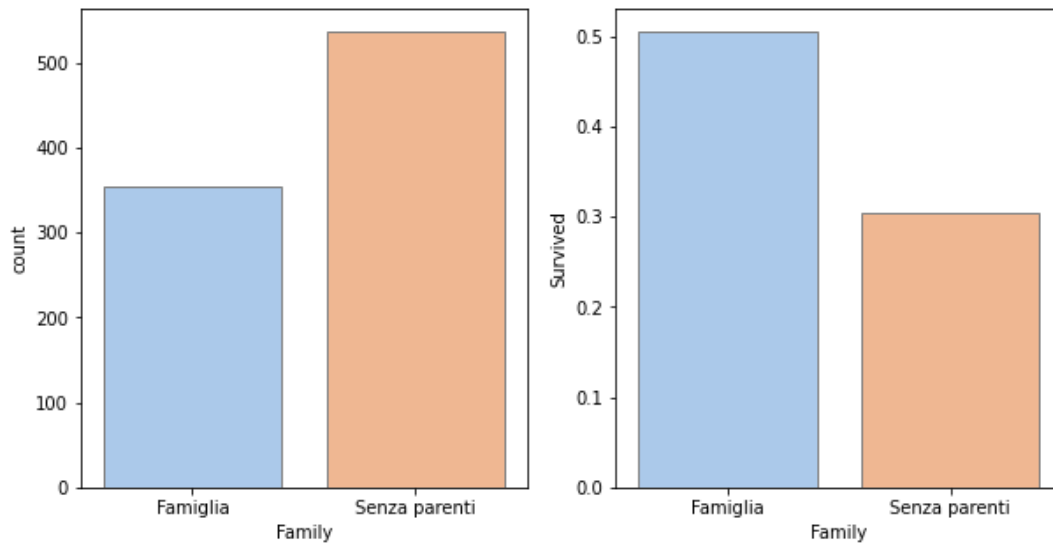


Un altro fattore che può aver influenzato la sopravvivenza di un passeggero è l'età.

Dai violin plot riportati sotto, vediamo come ci sia un picco di passeggeri uomini morti tra i 20 e i 40 anni di età, il che può essere ricondotto ancora alla procedura del “prima donne e bambini” nel caso di salvataggi.



Vediamo inoltre come la presenza di familiari a bordo sembra possa aver influenzato la sopravvivenza dei passeggeri, favorendo chi aveva parenti a bordo.



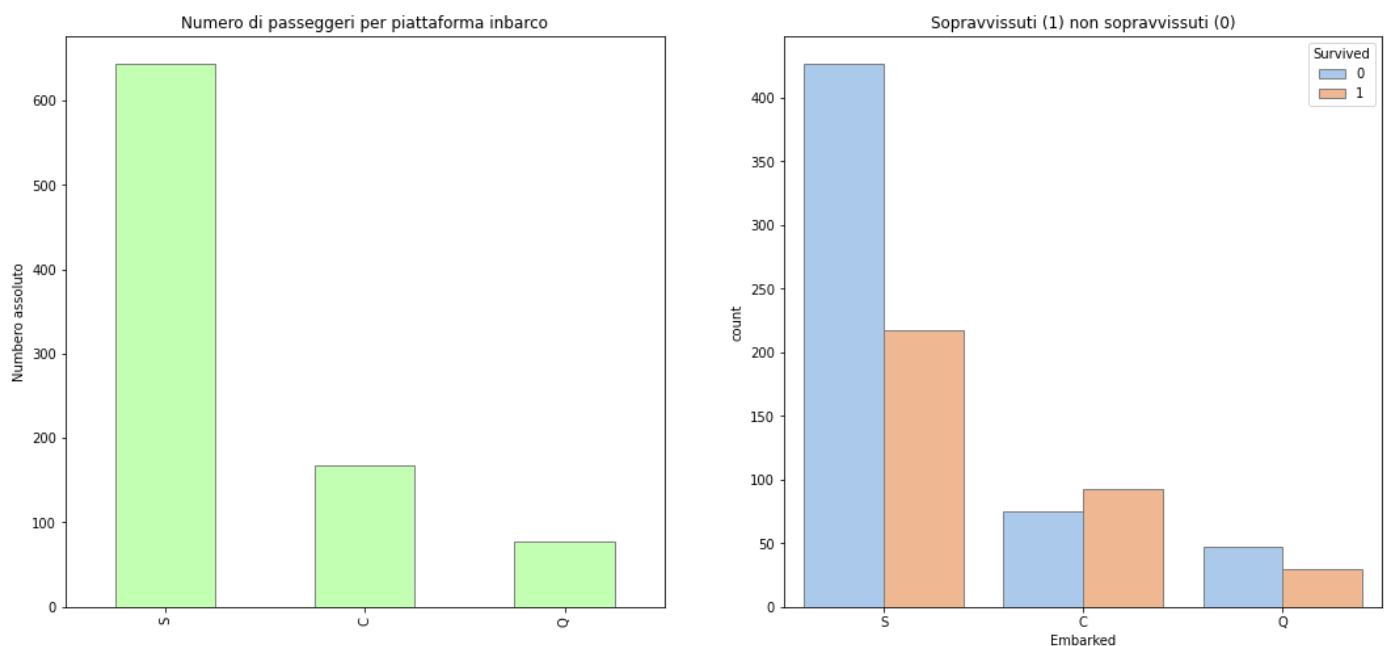
Ciò potrebbe derivare ancora dalla procedura di “prima donne e bambini”, quindi andiamo a calcolare anche la percentuale di uomini adulti con e senza famiglia.

Dalle percentuali di sopravvivenza di uomini con e senza famiglia, non sembra ci sia una

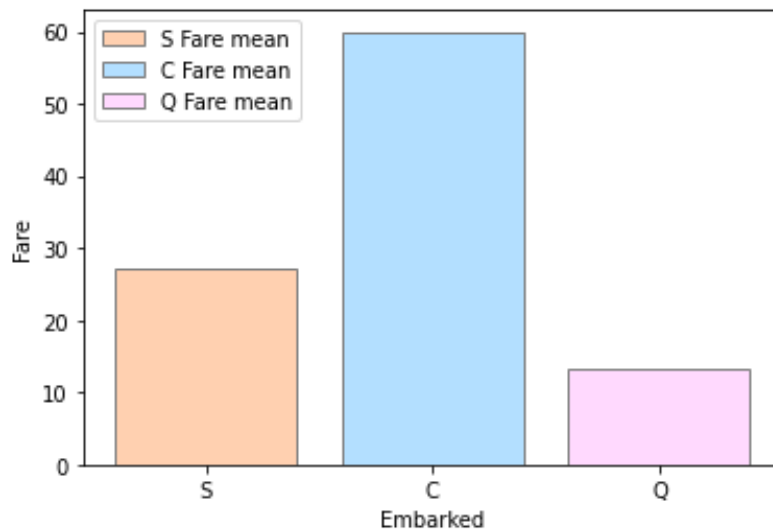
```
Nuomero uomini soli : 292 , numero uomini con famiglia : 103
-----
Sopravvivenza uomini soli : 0.17123287671232876
Sopravvivenza uomini con famiglia : 0.1941747572815534
```

correlazione evidente tra presenza di parenti e sopravvivenza

Ora andiamo invece a controllare se il sito di imbarcazione ha qualche influenza sul target

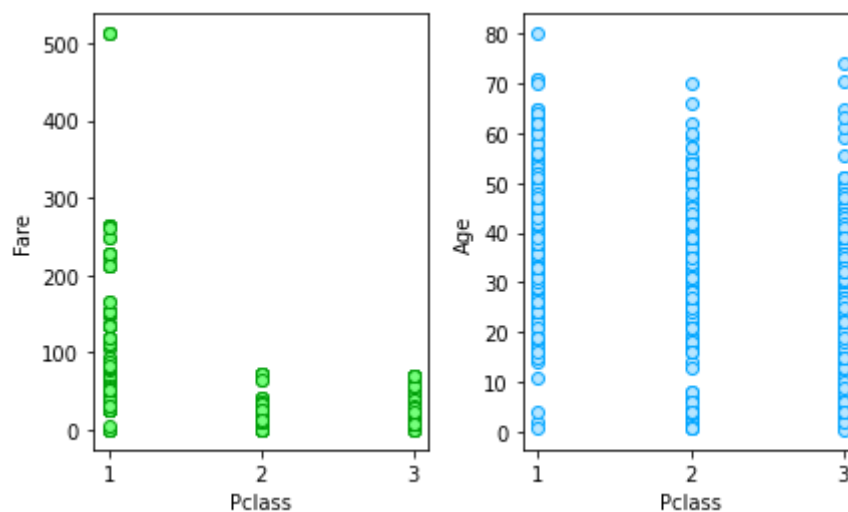


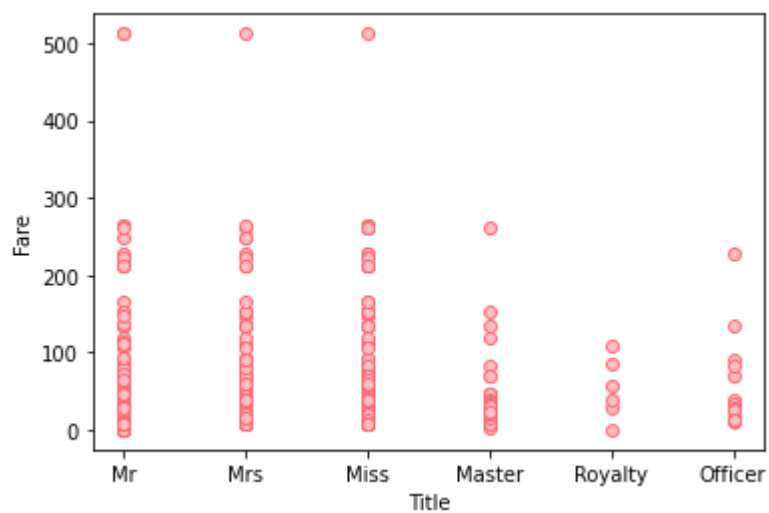
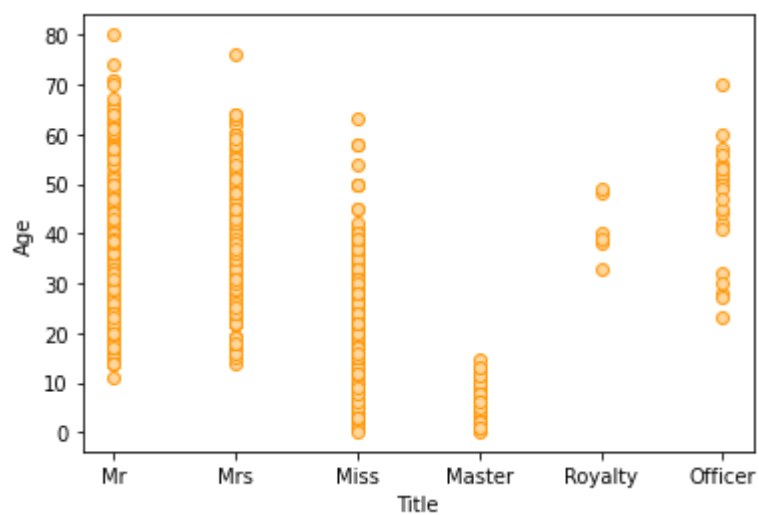
Sembrerebbe esserci un rapporto tra imbarcati e sopravvissuti maggiore sull'imbarco di Cherbourg (C). Questo potrebbe essere relativo al maggior costo del biglietto delle persone imbarcate a C invece dell'imbarco effettivo, fatto che possiamo vedere nel grafico sottostante, nel quale sono plottate le medie dei prezzi per i vari imbarchi.



Nei grafici scatter sottostanti, abbiamo provato ad individuare degli outliers.

Nonostante dal grafico che mette in relazione Pclass e Fare (verde) possa sembrare che ci sia un outlier in prima classe vicino alla Fare 500, dal grafico che mette in relazione i titoli e la Fare notiamo che il prezzo vicino al 500 è presente in tre titoli diversi.





FEATURE ENGINEERING

Partiamo dividendo il dataset in train e test, l'obiettivo è non far conoscere informazioni del train al test e viceversa durante le operazioni di imputazione.

Vengono innanzitutto droppate delle colonne :

PassengerId → dal train perchè è stato reputato non aggiungesse significato ai dati, essendo solo una chiave primaria di quello che ipotizziamo essere un database

Survived → dal train per rendere i dataframes train e test di uguale numero di features e potergli operare con più semplicità.

Possiamo, tramite il comando `head()`, andare a vedere la forma dei nostri dati.

| Pclass | Name | Sex | Age | SibSp | Parch | Ticket | Fare | Cabin | Embarked | Family |
|--------|---|--------|------|-------|-------|------------------|---------|-------|----------|--------|
| 3 | Braund, Mr. Owen Harris | male | 22.0 | 1 | 0 | A/5 21171 | 7.2500 | NaN | S | 1 |
| 1 | Cumings, Mrs. John Bradley (Florence Briggs Th... | female | 38.0 | 1 | 0 | PC 17599 | 71.2833 | C85 | C | 1 |
| 3 | Heikkinen, Miss. Laina | female | 26.0 | 0 | 0 | STON/O2. 3101282 | 7.9250 | NaN | S | 0 |
| 1 | Futrelle, Mrs. Jacques Heath (Lily May Peel) | female | 35.0 | 1 | 0 | 113803 | 53.1000 | C123 | S | 1 |
| 3 | Allen, Mr. William Henry | male | 35.0 | 0 | 0 | 373450 | 8.0500 | NaN | S | 0 |

Vediamo subito che abbiamo diverse features qualitative, le quali dovranno essere convertite. Inoltre, sono presenti alcuni valori NaN già tra i pochi plottati.

Andiamo a vedere quanti sono i missing values, così da decidere che tecniche utilizzare per il riempimento (o l'eventuale drop):

Missing values del train

| | | | |
|-------------|-----|----------------------|------------------|
| PassengerId | --- | missing values: 0, | percentage 0.00 |
| Pclass | --- | missing values: 0, | percentage 0.00 |
| Name | --- | missing values: 0, | percentage 0.00 |
| Sex | --- | missing values: 0, | percentage 0.00 |
| Age | --- | missing values: 177, | percentage 19.87 |
| SibSp | --- | missing values: 0, | percentage 0.00 |
| Parch | --- | missing values: 0, | percentage 0.00 |
| Ticket | --- | missing values: 0, | percentage 0.00 |
| Fare | --- | missing values: 0, | percentage 0.00 |
| Cabin | --- | missing values: 687, | percentage 77.10 |

| | | | |
|----------|-----|--------------------|-----------------|
| Embarked | --- | missing values: 2, | percentage 0.22 |
| Family | --- | missing values: 0, | percentage 0.00 |

Missing values del test

| | | | |
|-------------|-----|----------------------|------------------|
| PassengerId | --- | missing values: 0, | percentage 0.00 |
| Pclass | --- | missing values: 0, | percentage 0.00 |
| Name | --- | missing values: 0, | percentage 0.00 |
| Sex | --- | missing values: 0, | percentage 0.00 |
| Age | --- | missing values: 86, | percentage 20.57 |
| SibSp | --- | missing values: 0, | percentage 0.00 |
| Parch | --- | missing values: 0, | percentage 0.00 |
| Ticket | --- | missing values: 0, | percentage 0.00 |
| Fare | --- | missing values: 1, | percentage 0.24 |
| Cabin | --- | missing values: 327, | percentage 78.23 |
| Embarked | --- | missing values: 0, | percentage 0.00 |
| Family | --- | missing values: 0, | percentage 0.00 |

Distinguiamo 3 categorie :

1. **Embarked**, con un numero di missing values bassissimo, in questo caso riteniamo che riempire questa feature con il valore più utilizzato nel dataframe (che ha 891-2 valori) sia una tecnica accettabile.
2. **Age**, in questo caso il numero di valori nulli inizia ad essere più consistente, abbiamo ipotizzato due strade :
 - un riempimento con pescaggio casuale e reinserimento in base alla distribuzione di probabilità dei valori a nostra conoscenza, oppure
 - un riempimento tramite media gruppando i valori a nostra conoscenza per 'Titolo' della persona e 'Sesso. Vedremo successivamente in dettaglio entrambi i metodi.
3. **Cabin**, questa feature è quasi del tutto assente, decidiamo di utilizzare un riempimento tramite pescaggio casuale con reinserimento pesato sulla distribuzione di probabilità della feature per i pochi valori a nostra conoscenza.

Cabin

Questa feature è composta da una lettera seguita da delle cifre, è quindi di tipo qualitativo e dobbiamo trasformarla.

Decidiamo di mantenere solo la prima lettera per rendere la manipolazione più facile, ci basiamo sull'assunzione che sia questa la parte più importante della feature e non il numero.

Inizio ora il riempimento, innanzitutto calcolo la distribuzione di probabilità dei valori presenti, per farlo associo ad ogni valore unico della feature un numero che sarà la somma delle volte che questo appare all'interno della colonna diviso per il numero totali di samples della colonna. Cioè:

$$\frac{n \cdot \text{Elemento}X}{\text{TotElementi}}$$

Ora facciamo passare iterativamente tutta la feature, quando incontriamo un valore nullo peschiamo con reinserimento in maniera pesata un valore con cui sostituiamo il NaN presente. Per fare ciò ci avvaliamo della funzione `Random.choice()` che riceve in input un array di valori unici e un array della stessa dimensione che abbina ad ogni valore il suo peso (da 0 ad 1).

```
feature[i] = random.choice(feature_unique, size=None, replace=True, p=a)
```

Fatto ciò posso trasformare la feature in una dummy variable (per trasformare qualitativo in quantitativo)

Fare

Questa come già anticipato richiede poco lavoro, i missing values sono solo due e i valori son già quantitativi.

```
data.Fare.fillna(data.Fare.median(), inplace=True)
```

Il parametro `inplace` settato a "True" indica che vogliamo modificare il dataframe direttamente

Name

Questa feature è molto particolare: di per sé, il nome non sembra possa essere utile per capire l'eventuale sopravvivenza o morte di una persona. Al massimo potrebbe essere usata per ricavare il sesso (che abbiamo visto essere molto influente), ma noi questa feature l'abbiamo già.

Consultando altri kernels, abbiamo visto che potrebbe aiutare estrarre il titolo della persona per categorizzare meglio i gruppi di persone e le loro appartenenze.

Molti dei titoli presenti erano usati da pochissime persone, abbiamo quindi cercato di creare dei gruppi che mantenessero la categorizzazione senza avere un grado di dettaglio troppo elevato.

```
"Capt": "Officer",
"Col": "Officer",
"Major": "Officer",
"Jonkheer": "Royalty",
"Don": "Royalty",
"Dona": "Royalty",
"Sir" : "Royalty",
"Dr": "Officer",
"Rev": "Officer",
"the Countess": "Royalty",
"Mme": "Mrs",
"Ms": "Miss",
"Ms": "Mrs",
"Mr" : "Mr",
"Mrs" : "Mrs",
"Miss" : "Miss",
"Master" : "Master",
"Lady" : "Royalty",
```

Riporto l'attuale situazione del dataframe per chiarezza, qui è stato applicato un `group_by` e `count` sulla feature `title`

| Title | Sex | |
|---------|--------|-----|
| Master | male | 61 |
| Miss | female | 262 |
| Mr | male | 757 |
| Mrs | female | 200 |
| Officer | female | 1 |
| | male | 22 |
| Royalty | female | 3 |
| | male | 3 |

Infine creo le dummy variables per title e passo alla feature successiva

Age

La feature età aveva un numero di missing value medio, abbiamo provato ad applicare due tecniche : scelta random con reinserimento pesata e mediana in base a sesso e titolo.

La prima tecnica è già stata esplorata per la feature cabin, il funzionamento sarebbe uguale quindi non lo riporto.

La seconda tecnica si avvale di questo pezzo di codice

```
grouped = data.groupby(['Sex', 'Title'])  
data.Age = grouped.Age.apply(lambda x: x.fillna(x.median()))
```

La lambda function ha lo scopo di passare tutti i valori e applicare il calcolo descritto su quelli settati a NaN

Abbiamo usato la mediana e non la media per evitare che pochi valori troppo elevati (piccoli o grandi) potessero influenzare tutta la feature

Essendo già una feature quantitativa il lavoro è finito.

Embarked

Anche in questo caso come 'Fare' i missing values sono molto pochi, decidiamo quindi di prendere quello più utilizzato e riempire così.

La feature è qualitativa, la trasformiamo in dummy variable prima di procedere.

Pclass

Non abbiamo missing values da trattare, è una feature qualitativa che trasformo come sopra.

Sex

Trasformo sex da maschio/femmina a 0/1, questo è preferibile all'utilizzo di una dummy variable quando possibile.

Ticket

Questa feature è abbastanza complessa, non sembra trasmettere informazioni utili per il nostro fine.

Anche controllando quali fossero gli usi standard di features di questo tipo in internet, il più delle volte veniva droppata o calcolata la lunghezza dei caratteri alfanumerici del biglietto/codice identificativo.

Il secondo metodo non ci è sembrato di molto utilizzo nel nostro specifico caso, abbiamo quindi optato per il drop di questa feature.

I nostri dataframes sono ora pronti per la fase successiva, non abbiamo più missing values/features qualitative.

| # | Column | Non-Null Count | Dtype |
|-----|---------|----------------|---------|
| --- | ----- | ----- | ----- |
| 0 | Sex | 891 non-null | int64 |
| 1 | Age | 891 non-null | float64 |
| 2 | SibSp | 891 non-null | int64 |
| 3 | Parch | 891 non-null | int64 |
| 4 | Fare | 891 non-null | float64 |
| 5 | Cabin_A | 891 non-null | uint8 |
| 6 | Cabin_B | 891 non-null | uint8 |

| | | | |
|----|---------------|--------------|-------|
| 7 | Cabin_C | 891 non-null | uint8 |
| 8 | Cabin_D | 891 non-null | uint8 |
| 9 | Cabin_E | 891 non-null | uint8 |
| 10 | Cabin_F | 891 non-null | uint8 |
| 11 | Cabin_G | 891 non-null | uint8 |
| 12 | Cabin_T | 891 non-null | uint8 |
| 13 | Title_Master | 891 non-null | uint8 |
| 14 | Title_Miss | 891 non-null | uint8 |
| 15 | Title_Mr | 891 non-null | uint8 |
| 16 | Title_Mrs | 891 non-null | uint8 |
| 17 | Title_Officer | 891 non-null | uint8 |
| 18 | Title_Royalty | 891 non-null | uint8 |
| 19 | FamilySize | 891 non-null | int64 |
| 20 | Embarked_C | 891 non-null | uint8 |
| 21 | Embarked_Q | 891 non-null | uint8 |
| 22 | Embarked_S | 891 non-null | uint8 |
| 23 | Pclass_1 | 891 non-null | uint8 |
| 24 | Pclass_2 | 891 non-null | uint8 |
| 25 | Pclass_3 | 891 non-null | uint8 |

MODELLIZZAZIONE

All'inizio della sezione della modellizzazione, abbiamo diviso il training set fornito dalla competizione in un training set più piccolo e un test set, in modo da avere un set di dati con il target da poter usare per la validazione dei modelli.

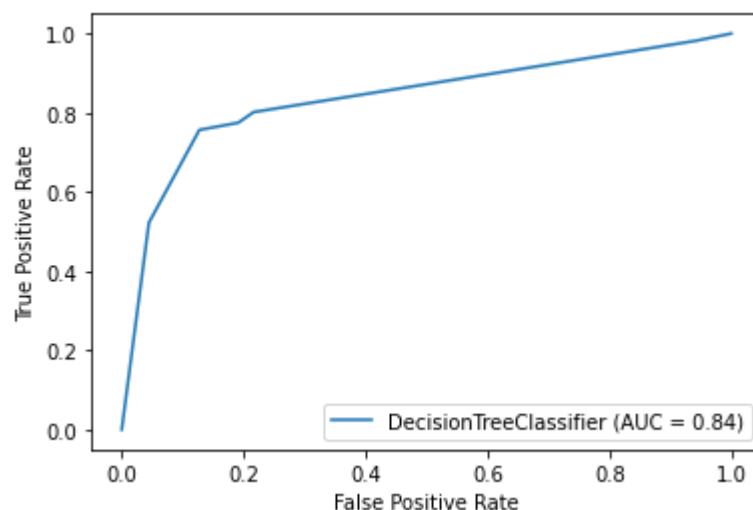
Come proporzione di split, abbiamo usato il 70% del training set iniziale come training, e il restante 30% come test.

Decision tree classifier

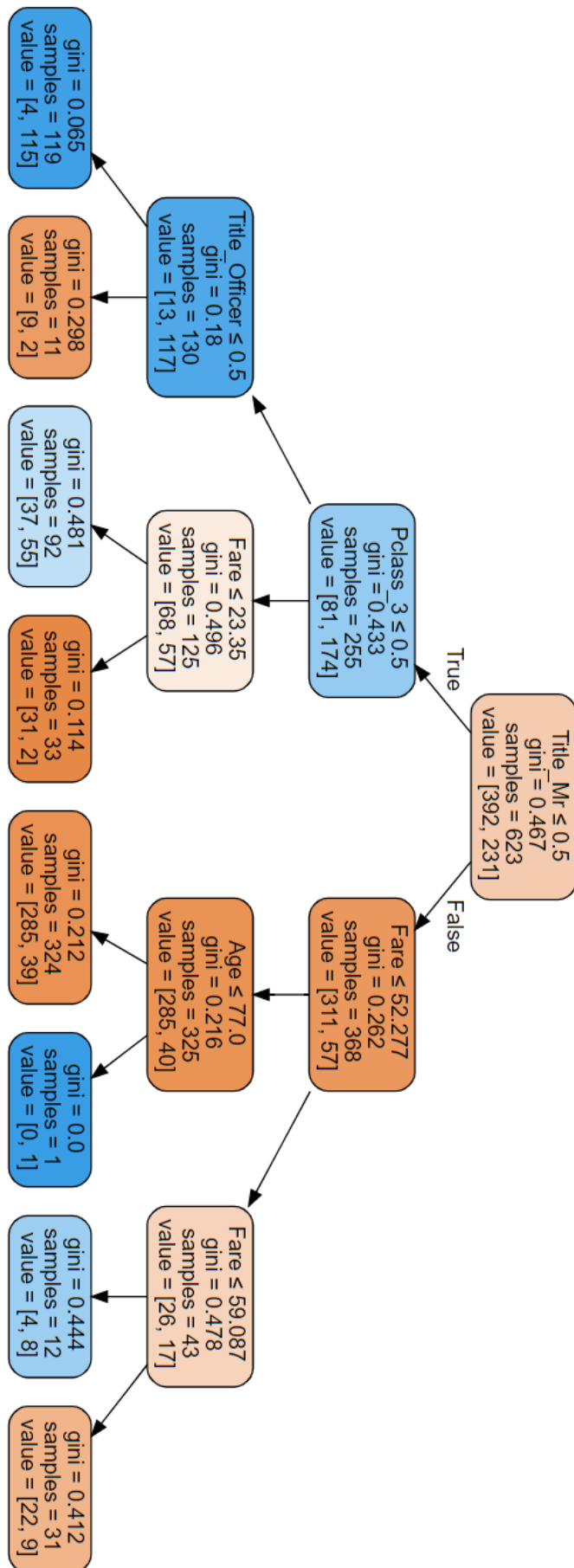
Per l'applicazione degli alberi di decisione, abbiamo iniziato dall'implementazione più semplice, applicando il metodo `DecisionTreeClassifier` con `max_depth=3`.

Dopo averlo addestrato sulla porzione di training set, la score relativa alla porzione di training iniziale dedicata al test è risultata 0,8246.

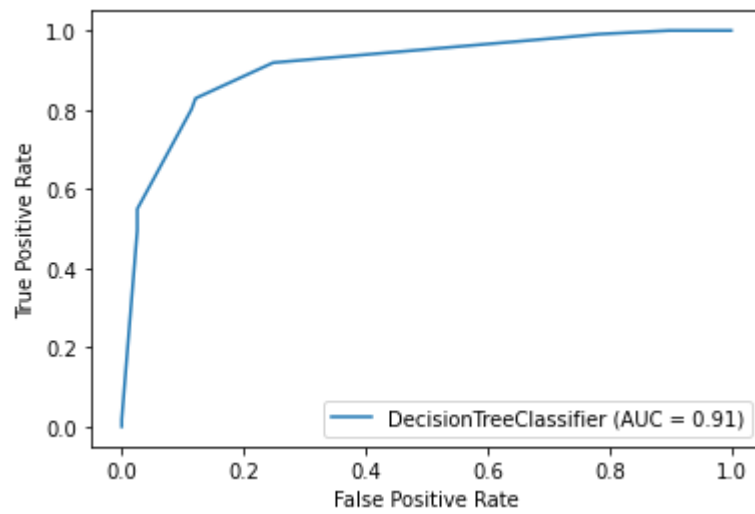
La roc curve relativa a questo modello:



Segue la rappresentazione dell'albero di decisione generato dall'algoritmo:



Ecco la ROC curve risultante dal modello generato dalla grid search



La submission fatta utilizzando il modello ha ottenuto un punteggio di 0,76315

| Your most recent submission | | | | |
|--|-----------|-----------|----------------|---------|
| Name | Submitted | Wait time | Execution time | Score |
| dtc_grid.csv | just now | 1 seconds | 0 seconds | 0.76315 |
| Complete | | | | |
| Jump to your position on the leaderboard ▼ | | | | |

Ensemble Methods

Voting Classifier

Per l'applicazione dei metodi d'insieme, abbiamo iniziato con un semplice voting classifier con hard e soft voting.

Gli algoritmi presi in considerazione sono stati 4:

1. Decision tree
2. Logistic regression
3. Random forest
4. SVC

Con hard voting i risultati sono stati:

```
DecisionTreeClassifier 0.8171641791044776
LogisticRegression 0.8134328358208955
RandomForestClassifier 0.7947761194029851
SVC 0.6604477611940298
VotingClassifier 0.7910447761194029
```

Mentre con soft voting:

```
DecisionTreeClassifier 0.8171641791044776
LogisticRegression 0.8134328358208955
RandomForestClassifier 0.832089552238806
SVC 0.6604477611940298
VotingClassifier 0.7947761194029851
```

Abbiamo provato una submission alla competizione per entrambe le versioni.

Per il modello con hard voting, il risultato è stato 0,77511

| Your most recent submission | | | | |
|--|-----------|-----------|----------------|---------|
| Name | Submitted | Wait time | Execution time | Score |
| hardVoting.csv | just now | 1 seconds | 0 seconds | 0.77511 |
| Complete | | | | |
| Jump to your position on the leaderboard ▼ | | | | |

Mentre il modello con soft voting ha ottenuto un punteggio di 0,76555

| Your most recent submission | | | | |
|--|-----------|-----------|----------------|---------|
| Name | Submitted | Wait time | Execution time | Score |
| softVoting.csv | just now | 1 seconds | 0 seconds | 0.76555 |
| Complete | | | | |
| Jump to your position on the leaderboard ▼ | | | | |

Bagging Classifier

Il bagging classifier, usando i decision tree classifiers come weak learners, ha ottenuto un accuracy score sui dati di test di 0,82836

Abbiamo anche provato ad applicare l'out of bag evaluation, ottenendo uno score di 0,82089

La submission effettuata usando questo metodo invece, ha ottenuto un punteggio di 0,77272

| Your most recent submission | | | | |
|--|-----------|-----------|----------------|---------|
| Name | Submitted | Wait time | Execution time | Score |
| bagging.csv | just now | 1 seconds | 0 seconds | 0.77272 |
| Complete | | | | |
| Jump to your position on the leaderboard ▼ | | | | |

Random Forest Classifier

Il random forest classifier, sui dati di test, ha ottenuto un'accuracy di 0,83209

Il punteggio della submission relativa a questo modello è stato di 0,78468

| Your most recent submission | | | | |
|--|-----------|-----------|----------------|---------|
| Name | Submitted | Wait time | Execution time | Score |
| randomforest.csv | just now | 1 seconds | 0 seconds | 0.78468 |
| Complete | | | | |
| Jump to your position on the leaderboard ▼ | | | | |

AdaBoost

L'applicazione del boosting con l'algoritmo AdaBoost, ha prodotto un accuracy score sui dati di test di 0,80970

La submission relativa a questo algoritmo, invece, ha ottenuto un punteggio di 0,77751

| Your most recent submission | | | | |
|--|-------------------|-----------|----------------|---------|
| Name | Submitted | Wait time | Execution time | Score |
| adaBoost (1).csv | a few seconds ago | 8 seconds | 0 seconds | 0.77751 |
| Complete | | | | |
| Jump to your position on the leaderboard ▼ | | | | |

Gradient Boosting

L'algoritmo GradientBoostingRegressor, per l'applicazione del gradient boosting, ha ottenuto uno score sui dati di test di 0,83209

La submission alla competizione del target prodotto con questo algoritmo, ha ottenuto un punteggio di 0,77511

| Your most recent submission | | | | |
|--|-----------|-----------|----------------|---------|
| Name | Submitted | Wait time | Execution time | Score |
| gBoosting (1).csv | just now | 1 seconds | 0 seconds | 0.77511 |
| Complete | | | | |
| Jump to your position on the leaderboard ▼ | | | | |

XGBoost

Infine, l'algoritmo XGBoost, ha prodotto un'accuracy di 0,81343 con un punteggio sulla competizione kaggle di 0,76076

| Your most recent submission | | | | |
|--|-----------|-----------|----------------|---------|
| Name | Submitted | Wait time | Execution time | Score |
| XGB.csv | just now | 1 seconds | 0 seconds | 0.76076 |
| Complete | | | | |
| Jump to your position on the leaderboard ▼ | | | | |

Neural Network

Questa parte è stata svolta in un notebook a parte.

Useremo le librerie sklearn (per il pre-processing) e tensorflow (keras sopra esso).

Innanzitutto, definiamo una funzione per scalare i dati, questa scalerà i dati nel range 0-1 in base al minimo e massimo nel dataframe.

Definisco una funzione per la creazione della rete neurale, i parametri che ci interessano sono numero di features (che sarà definito dal dataframe), il numero di strati nascosti (in un caso di classificazione si stà di solito tra l'1 e 6, scegliamo 3 come media), il numero di neuroni degli strati nascosti (anche qui non abbiamo un vincolo, vedremo successivamente che abbiamo provato diversi tentativi), il dropout (vediamo una spiegazione successivamente) e la learning_rate (questo parametro è molto importante, in base all'ottimizzatore un valore troppo alto qua potrebbe rovinare tutto l'allenamento. Vedremo poi che con Adam questo non si verifica in maniera tangibile e mantenendo 0.003 riusciamo a convergere al minimo).

Inizio l'effettiva creazione, la rete sarà completamente connessa come vista a lezione.

Usiamo come funzione di attivazione in tutti i neuroni la 'ReLU' tranne nel neurone di output dove usiamo la sigmoide perché vogliamo una percentuale (vivo/morto).

Infine come loss function usiamo binary crossentropy perché il problema è di classificazione e ha solo due possibili stati.

Prendiamo i dati di train e li dividiamo tra train e validation con una percentuale del 25%.

Ora definisco una funzione per calcolare l'accuracy del modello, da ricordare che la sigmoide mi da un valore tra 0 e 1 float, starà a me convertirlo con una mia regola prima di valutare l'output. Scegliamo che se il valore è ≥ 0.5 allora sopravvive sennò muore.

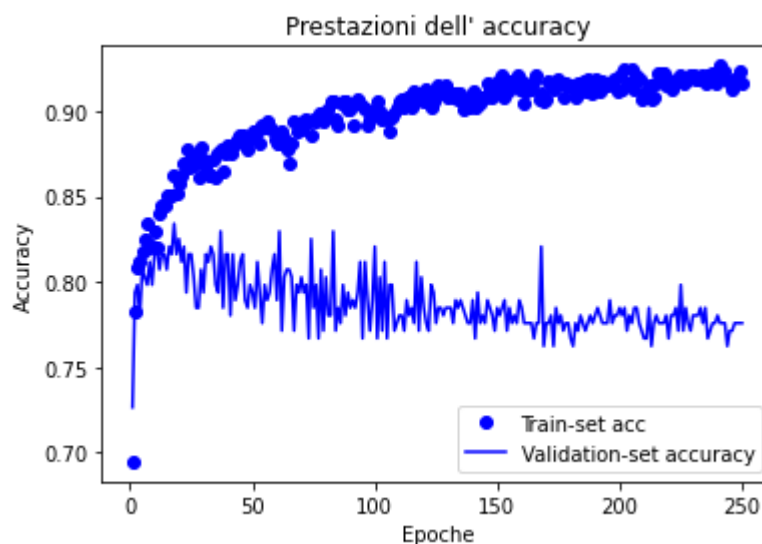
Ora conoscendo il target calcolo l'accuratezza sia su train che validation per poter controllare anche se il modello va in over-fit.

Keras con il metodo `model.fit()` ci restituisce un dictionary :

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

Tramite i valori che questo ci fornisce possiamo creare un grafico per vedere ogni epoca come impara il nostro modello

Un caso molto frequente di overfit si ha quando il modello è grande rispetto ai dati con cui possiamo allenarlo, è secondo noi quello che è successo con il primo modello che abbiamo allenato.

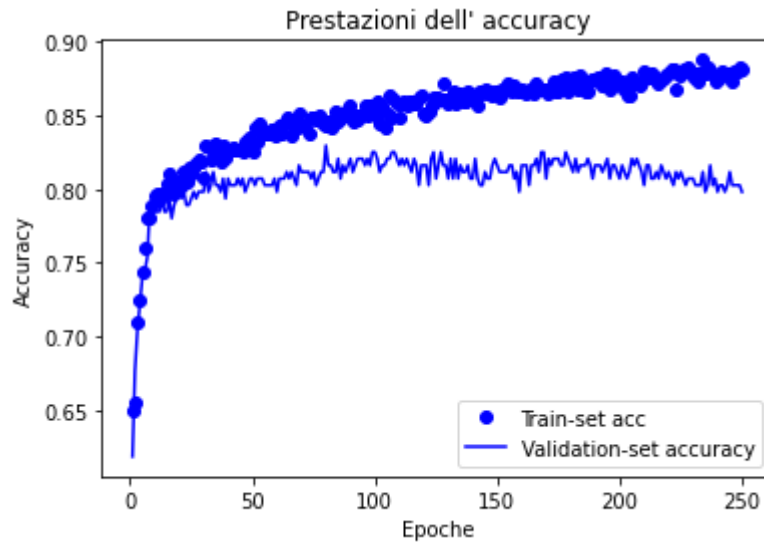


Training accuracy 0.919

Test accuracy 0.776

Si nota subito che l'aumento delle epoche non sta giovando per il trattamento di casi generali del problema, ma stiamo sempre più specificandolo per trattare i casi del training (impara anche a trattare i rumori).

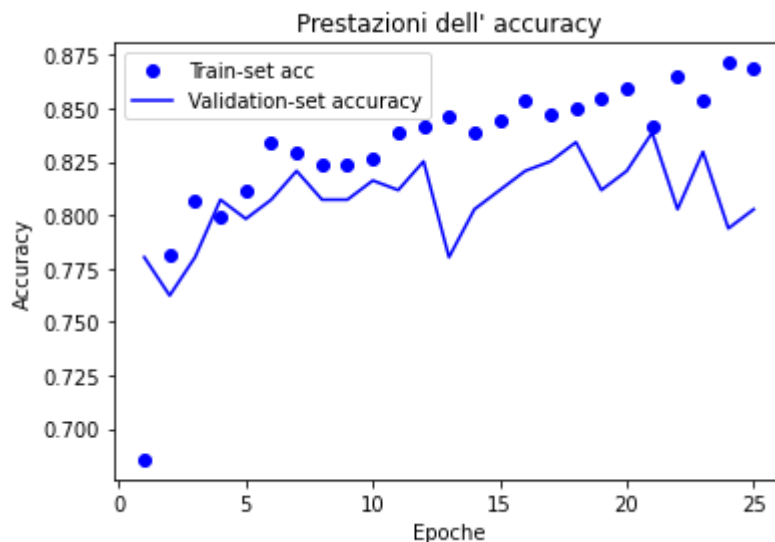
Ci sono diversi modi per trattare l'overfit, per prima cosa abbiamo provato a snellire la rete neurale, abbiamo usato un solo strato nascosto, numero di neuroni sceso a 32. Abbiamo però mantenuto le epoche a 250 per vedere l'andamento



Training accuracy 0.886

Test accuracy 0.798

La situazione migliora, possiamo provare ora a diminuire le epoche, infatti dall'immagine sopra vediamo che dopo un certo punto il validation non migliora più.



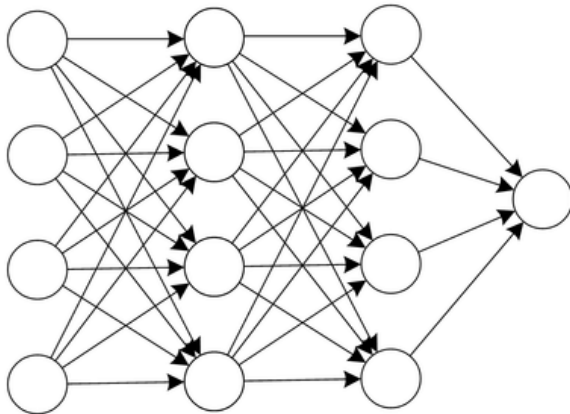
Training accuracy 0.879

Test accuracy 0.803

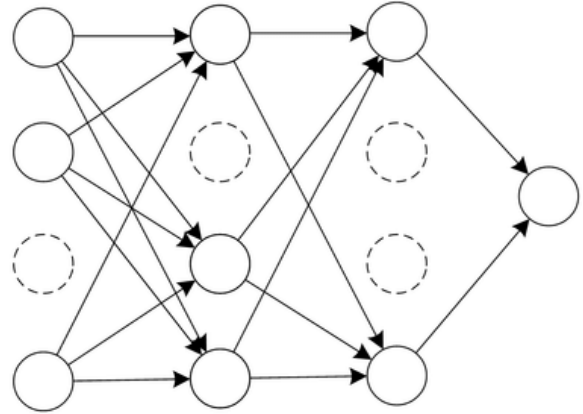
Proviamo un'ultima tecnica prima di unirle tutte insieme.

Aggiungiamo i dropout, questa tecnica consiste nel porre ad ogni epoca un tot% (scelto da noi) di pesi a 0 (droppiamo un neurone praticamente), avendo noi aggiunto un layer di dropout ad ogni layer normale, toglieremo il tot% di pesi ad ogni livello influenzando sia l'input che l'output.

Questo dovrebbe fare in modo di rompere quelle situazioni dove gli strati di una rete si co-adattano per correggere gli errori degli strati precedenti (sto imparando il rumore statistico, giungendo all'overfit), rendendo quindi il modello più generale.

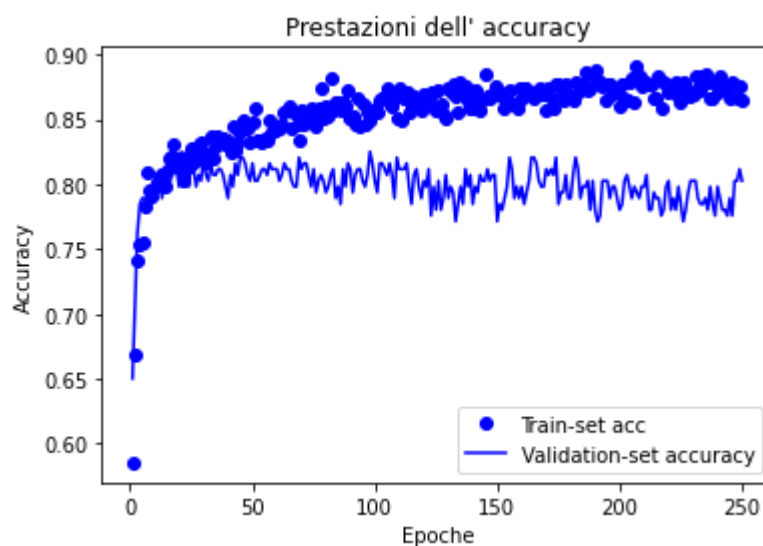


(a) Standard Neural Network



(b) Network after Dropout

Situazione con un dropout del 50%



Training accuracy 0.889

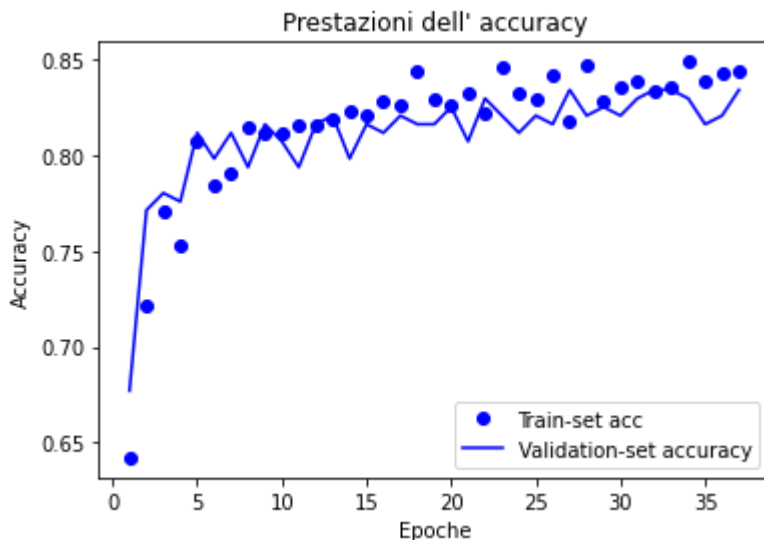
Test accuracy 0.803

Infine provo a trovare il meglio delle tecniche viste.

Alleggerisco leggermente il modello ponendo strati nascosti a 2, neuroni a 64. Non posso porre lo stesso alleggerimento di prima perchè qui aggiungendo il dropout rischio che quando i nodi vengono spenti si semplifichi troppo e non apprenda.

Dropout 30%.

Inoltre pongo le epoche a 250 però mi faccio aiutare da keras, se trova un massimo del validation test (accuracy) allora prova ancora 25 epoche (momentum, non voglio incagliarmi in un minimo locale) e se non aumenta si ferma e salva il modello al suo massimo.



Training accuracy 0.832

Test accuracy 0.816

Aggiungo che il dropout spegne i nodi solo nell'allenamento. Durante la predizione si usa la rete per intero.

La submission alla competizione effettuata con questo metodo, ha ottenuto uno score di 0,78708 (la migliore).

Your most recent submission

| | | | | |
|---------------|-----------|-----------|----------------|---------|
| Name | Submitted | Wait time | Execution time | Score |
| NeuralNet.csv | just now | 1 seconds | 0 seconds | 0.78708 |



Complete

[Jump to your position on the leaderboard](#) ▼

CONCLUSIONE

La possibilità di cimentarci in questa competizione con le conoscenze derivate dal corso, è stata molto utile per assimilare solidamente i concetti pratici e metterci alla prova con migliaia di altri partecipanti.

Nonostante la nostra inesperienza nel campo, grazie alle esercitazioni svolte in questo semestre siamo stati in grado di ottenere un punteggio di 0,78708.

| | | | | | |
|------|------------|--|---------|----|----|
| 6055 | UniBS_MLDM |   | 0.78708 | 39 | 3d |
|------|------------|--|---------|----|----|

La posizione ottenuta nella competizione è stata la 6055^a su 47731 partecipanti, cioè nel top 13% dei risultati.

C'è tuttavia da considerare che le prime posizioni sono occupate da utenti che hanno appositamente scaricato il dataset completo da altre fonti, poiché i dati reali sono pubblici, per poi pubblicare un risultato perfetto in una sola entry, falsando la competizione.

Considerando questo, e considerando le posizioni a pari merito con il nostro punteggio, il nostro risultato rientra nella top 9% dei risultati.