Relazione progetto

Mamone Maximiliano - 308214
e-mail: maximilianonicolas.mamone@studenti.unipr.it

Marzo 2022

Indice

1 Introduzione

Nel corso della relazione tenterò di spiegare come mi sono approcciato allo svolgimento del progetto. Introdurrò brevemente il dataset e il procedimento per poi andare nel dettaglio delle varie fasi.

1.1 Dataset

Il dataset utilizzato per il progetto comprende un insieme di più di 30'000 osservazioni e 15 feature riguardanti il reddito annuale dei cittadini americani nel 1994. Introduco brevemente le feature:

- **age**, età della persona
- **workclass**, la tipologia di impiego (impiegato governativo, dipendente, ecc)
- **fnlwgt** o **final weight**, cioè una stima fatta dal governo sul numero di persone che tale osservazione rappresenta
- **education**, più alto livello di educazione conseguito
- **education.num**, un valore numerico gerarchico che indica il grado di educazione
- **marital.status**, stato civile
- **occupation**, lavoro effettivo della persona
- **relationship**, relazione sentimentale
- **race**, **etnia** della persona
- **sex**, genere della persona
- **capital.gain**, guadagno capitale
- **capital.loss**, perdita capitale
- **hours.per.week**, ore lavorative settimanali
- **native.country**,nazionalità
- **income**, reddito

1.2 Procedimento

Il task è di classificazione binaria e l'obbiettivo è tentare di predire il label "Income", il quale può assumere due valori "$\leq$50K" o "$>$50K". Per fare ciò seguirò alcuni step:

- **Esplorazione dei dati**, cercare di conoscere in generale il dataset tramite informazioni sulla struttura dello stesso e grafici.
- **Preprocessing**, tramite vari sottopassaggi riarrangiare e modificare il dataset per renderlo più idoneo all'elaborazione da parte di un algoritmo di machine learning o deep learning.
- **Scelta del modello**, utilizzando *RandomSearch*, *GridSearch* e tanti modelli, cercare di ottenere contemporaneamente il miglior modello possibile con i migliori parametri possibili.
- **Validazione finale del modello**, verificare che il modello sia correttamente in grado di generalizzare sul test set e traendo qualche conclusione dai risultati.

2 Esplorazione dei dati

Prima di tutto vogliamo avere un idea generale di come sia strutturato il dataset e se ci sono problemi come valori nulli, incomprensibili o duplicati.

2.1 Dataset

Come si vede in Figura 1 il nostro dataset non presenta valori nulli e come detto prima è composto da 15 feature, 9 categoriche (tra cui anche **income**) e 6 numeriche. A tutte queste dedicheremo una sezione a breve per visualizzarle e cercare di capire se si può svolgere qualche tipo di operazione su di esse per cercare di ottenere nuove informazioni.

Dalla Figura 2 notiamo invece che le feature **workclass, occupation e native.country** contengono valori "?", quindi anche se non ci sono valori nulli riconosciuti da *Pandas* per noi queste istanze sono un problema che sarà necessario risolvere prima di procedere con la scelta e la creazione di un modello. Inoltre il label *income* assume due valori diversi da 0 e 1 è sarà quindi necessario codificarlo per poter effettuare agevolmente le predizioni.

```
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             32561 non-null  int64
 1   workclass       32561 non-null  object
 2   fnlwgt          32561 non-null  int64
 3   education       32561 non-null  object
 4   education.num   32561 non-null  int64
 5   marital.status  32561 non-null  object
 6   occupation      32561 non-null  object
 7   relationship    32561 non-null  object
 8   race            32561 non-null  object
 9   sex             32561 non-null  object
 10  capital.gain    32561 non-null  int64
 11  capital.loss    32561 non-null  int64
 12  hours.per.week  32561 non-null  int64
 13  native.country  32561 non-null  object
 14  income          32561 non-null  object
dtypes: int64(6), object(9)
```

Figura 1: Informazioni sui valori nulli e i tipi delle feature

index	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	race	sex	capital.gain	capital.loss	hours.per.week	native.country	income
0	90	?	77053	HS-grad	9	Widowed	?	Not-in-family	White	Female	0	4356	40	United-States	<=50K
1	82	Private	132870	HS-grad	9	Widowed	Exec-managerial	Not-in-family	White	Female	0	4356	18	United-States	<=50K
2	66	?	186061	Some-college	10	Widowed	?	Unmarried	Black	Female	0	4356	40	United-States	<=50K
3	54	Private	140359	7th-8th	4	Divorced	Machine-op-inspct	Unmarried	White	Female	0	3900	40	United-States	<=50K
4	41	Private	264663	Some-college	10	Separated	Prof-specialty	Own-child	White	Female	0	3900	40	United-States	<=50K
5	34	Private	216864	HS-grad	9	Divorced	Other-service	Unmarried	White	Female	0	3770	45	United-States	<=50K
6	38	Private	150601	10th	6	Separated	Adm-clerical	Unmarried	White	Male	0	3770	40	United-States	<=50K
7	74	State-gov	88638	Doctorate	16	Never-married	Prof-specialty	Other-relative	White	Female	0	3683	20	United-States	>50K
8	68	Federal-gov	422013	HS-grad	9	Divorced	Prof-specialty	Not-in-family	White	Female	0	3683	40	United-States	<=50K
9	41	Private	70037	Some-college	10	Never-married	Craft-repair	Unmarried	White	Male	0	3004	60	?	>50K
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
32555	53	Private	321865	Masters	14	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	40	United-States	>50K
32556	22	Private	310152	Some-college	10	Never-married	Protective-serv	Not-In-Family	White	Male	0	0	40	United-States	<=50K
32557	27	Private	257302	Assoc-acdm	12	Married-civ-spouse	Tech-support	Wife	White	Female	0	0	38	United-States	<=50K
32558	40	Private	154374	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husband	White	Male	0	0	40	United-States	>50K
32559	58	Private	151910	HS-grad	9	Widowed	Adm-clerical	Unmarried	White	Female	0	0	40	United-States	<=50K
32560	22	Private	201490	HS-grad	9	Never-married	Adm-clerical	Own-Child	White	Male	0	0	20	United-States	<=50K

Figura 2: Head e Tail del dataset originale - In rosso le feature con valori sconosciuti al loro interno

2.2 Data Visualization

Facciamo ora una veloce visualizzazione delle varie feature, tramite grafici a barre per le variabili categoriche e istogrammi per le variabili numeriche, interessandoci della frequenza delle occorrenze rispettivamente al corrispondente valore di **income**

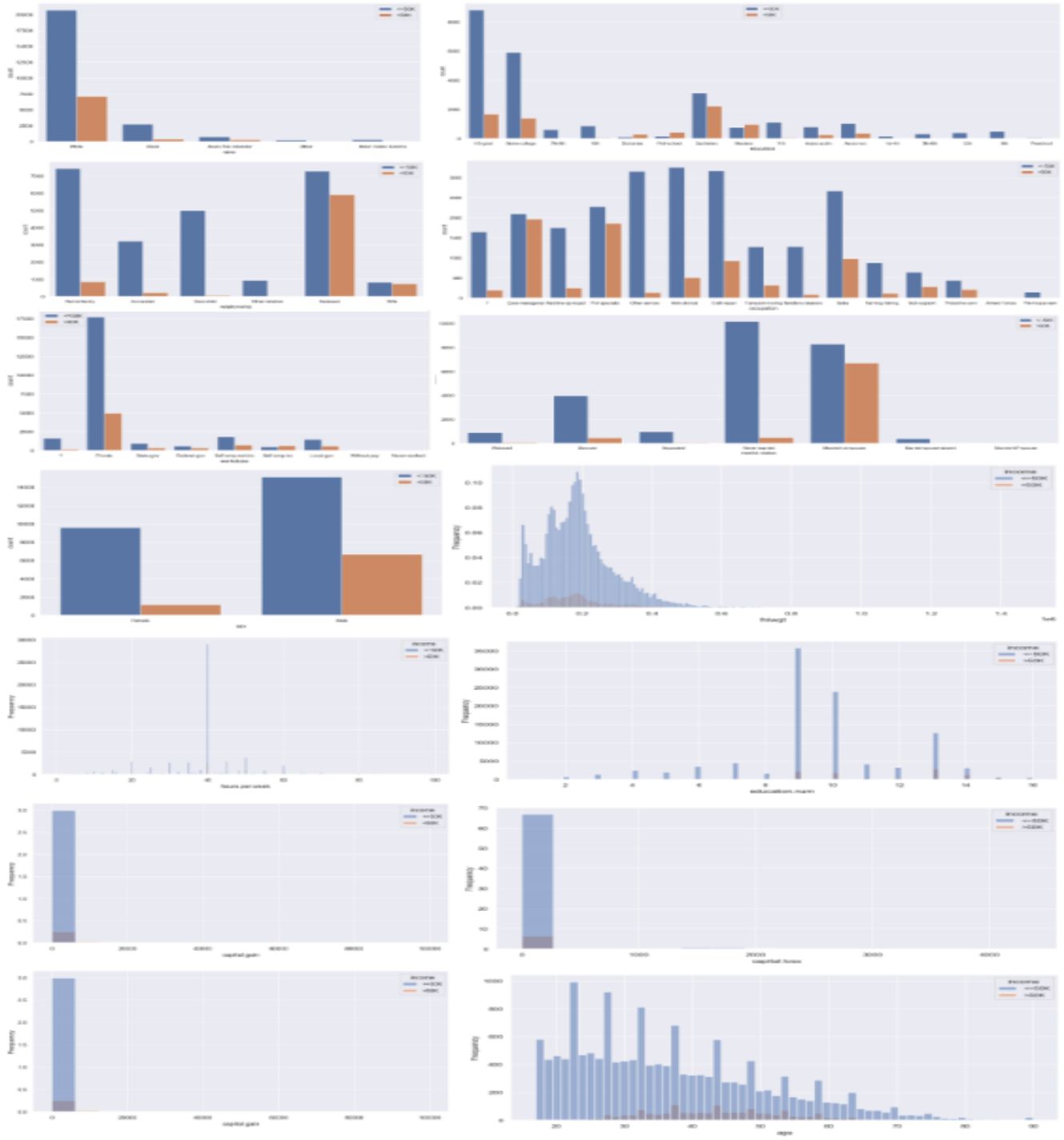

Figura 3: Data visualization tramite bar chart e histogram di tutte le feature

Da questi grafici risulta evidente che le feature numeriche avranno bisogno di essere sottoposte ad una qualche tipologia di feature scaling.

Visualizziamo ora la frequenza dei valori "<=50k" e ">50k" del label **income**

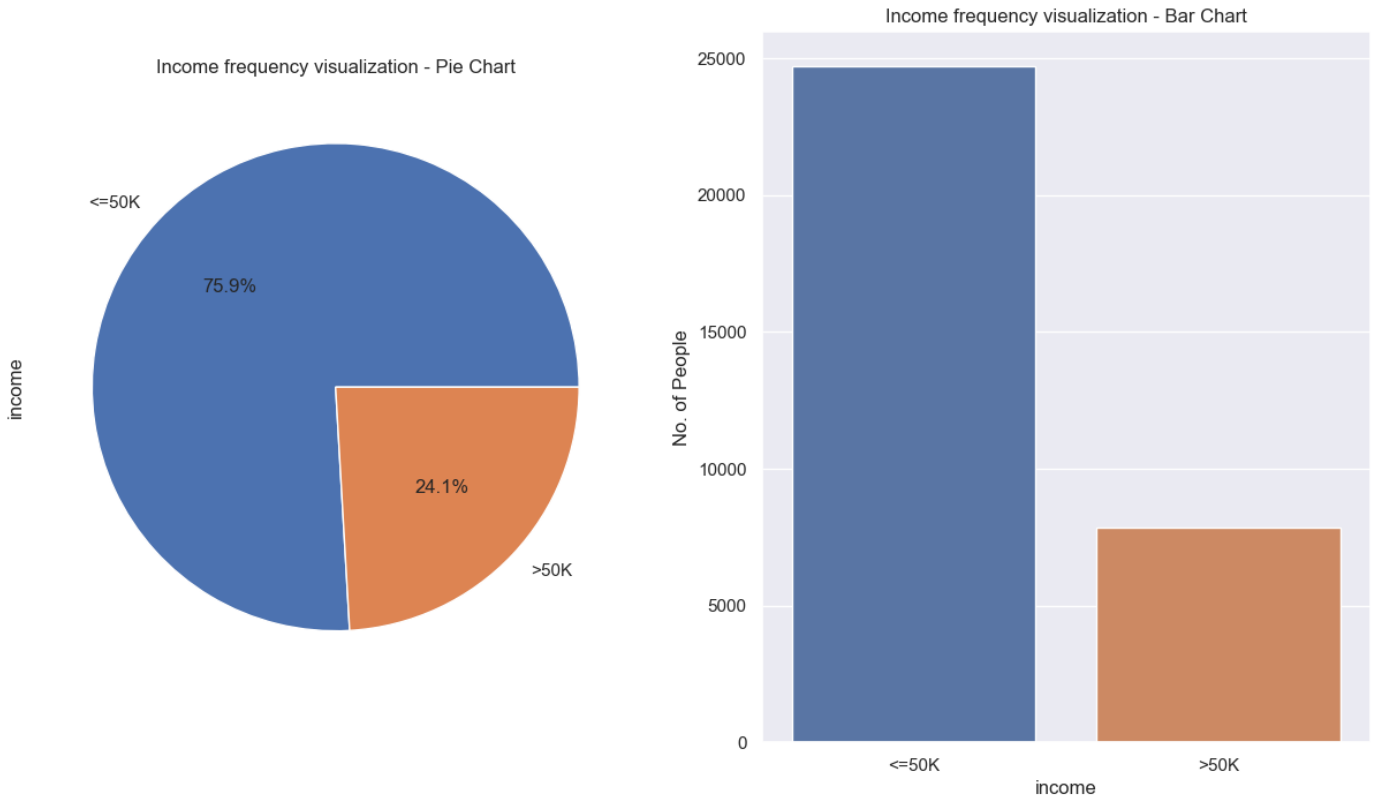

Figura 4: Data visualization tramite bar chart e pie chart della frequenza dei valori per il label **income**

Il dataset è quindi pesantemente sbilanciato. Questo potrebbe limitare le nostre possibilità di ottenere un modello preciso ma, utilizzando i giusti strumenti non dovrebbe essere un problema.

3 Preprocessing

In questa fase del progetto agisco sul dataset per prepararlo all'elaborazione cercando di ottimizzare i dati per essere più facilmente gestibili dai vari modelli.

3.1 Data Cleaning

Pulisco il dataset innanzitutto eliminando possibili osservazioni duplicate e risolvendo il problema dei valori "?" sostituendoli in ogni colonna con il corrispettivo valore della *moda* per tale feature.

3.2 Feature Engineering

Osservando la *Figura 2* decido di agire sulla feature **marital.status** per cercare di ottimizzarne la forma. I valori possibili sono:

- Never-married
- Divorced
- Separated
- Widowed
- Married-civ-spouse
- Married-spouse-absent
- Married-AF-spouse

Noto che è possibile raggrupparli in due categorie che rappresentano la situazione "attuale" della persona, **Married** e **Single**. Di conseguenza sostituisco i primi 4 valori con **Single** e i restanti 3 con **Married**.
Successivamente mi occupo dell'encoding delle due categorie con 0 e 1, e anche di tutte le altre feature categoriche compreso il label. Per farlo sfrutto un `OrdinalEncoder` e un `LabelEncoder` entrambi della libreria `sklearn`.

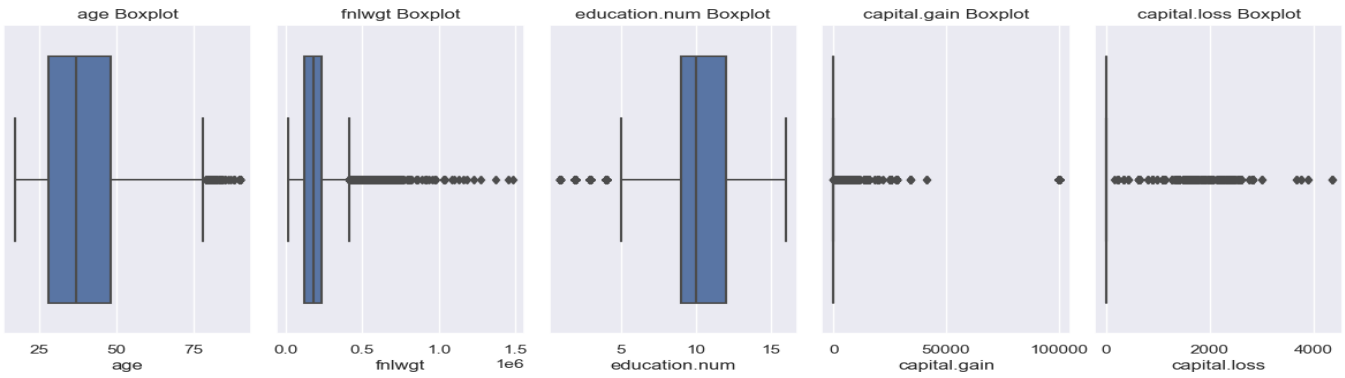

Figura 5: Boxplots relativi ad alcune feature numeriche

Cerco ora di indivduare gli outliers in alcune delle feature numeriche, **age, fnlwgt, education.num, capital.gain, capital.loss**. Ho selezionato queste feature basandomi sui grafici in *Figura 3*, ad esempio i valori di **capital.gain e capital.loss** risultano evidentemente "ammassati" verso sinistra, così come quelli di **fnlwgt**, ciò mi fa pensare che la bassissima quantità di osservazioni esterne al range, dove le osservazioni sono più concentrate, (gli outlier appunto) risulterebbe dannosa al modello.

Devo ora decidere come ripulire il dataset, scelgo di sfruttare lo *z-score* dei singoli dati come fattore determinante e cerco quindi un threshold adeguato che non mi faccia perdere una quantità troppo elevata di osservazioni. Data la natura sbilanciata del dataset perdere più dell'8% ho pensato sarebbe stato un problema e di conseguenza elimino tutte le righe dove *z-score*>4.2.

```
INFORMATION ABOUT Z-SCORE AND CORRELATED DATA LOSS
With threshold 3.0 data loss is 14.71%
With threshold 3.2 data loss is 11.12%
With threshold 3.4 data loss is 10.33%
With threshold 3.6 data loss is 9.68%
With threshold 3.8 data loss is 8.88%
With threshold 4.0 data loss is 8.37%
With threshold 4.2 data loss is 7.59%
With threshold 4.4 data loss is 6.02%
With threshold 4.6 data loss is 4.64%
With threshold 4.8 data loss is 3.29%
```

Figura 6: Data loss rispettivamente al threshold di z-score selezionato

A questo punto voglio migliorare ancora il mio dataset, agisco quindi sulla *skewness* dei miei dati. Con questo termine si indica una misura della deviazione assimetrica da una ideale distribuzione simmetrica. Per farlo sfrutto la funzione `PowerTransformer` sempre della libreria `sklearn`.

Questo é un buon momento per dividere il dataset in Training set e Test set in un rapporto 70% Training e 30% Test. Tutte le prossime fasi, se non diversamente specificato, saranno svolte a partire dal Training set.

Infine voglio risolvere il problema del bilanciamento del dataset, decido di utilizzare un algoritmo di oversampling per generare nuove osservazioni. Tra le varie possibilità offerte dalla libreria `imblearn` (Imbalanced Learning) scarto il *random over-sampling* perchè semplicemente duplica alcune istanze in maniera randomica, e scelgo invece di utilizzare `SMOTE` (Synthetic Minority Oversampling Technique) che effettua la generazione di nuove istanze tramite un processo di interpolazione dei dati già presenti. Questa operazione viene effettuata solo sul training set per non "inquinare" i dati del Test.

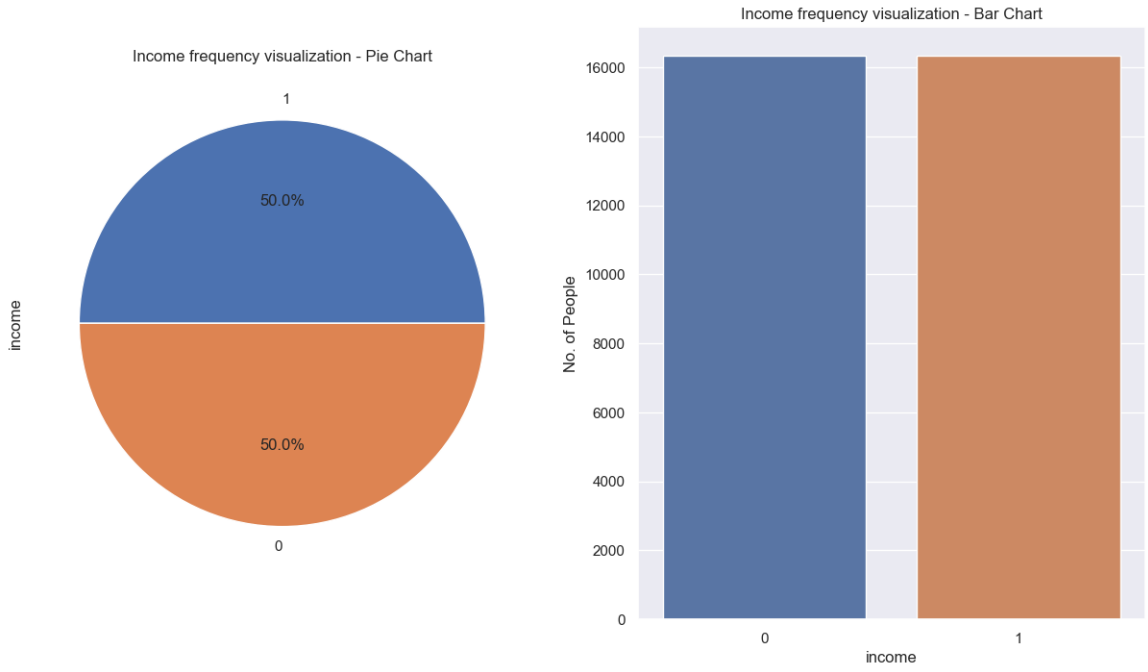

Figura 7: Nuovi grafici delle frequenza dei valori del label nel *Training set*

3.3 Feature Selection

Prima di dedicarmi a *Feature scaling*, volevo svolgere un operazione di selezione delle feature che contengono più informazione. A questo punto tramite la funzione `SelectKBest` (`sklearn`) valuto la mutua informazione tra tutte le feature all'interno del Training e il label **income**. Scelgo come threshold 0.04 perchè non ritengo necessario ridurre più di tanto le feature e di conseguenza seleziono (sia nel Training che nel Test):

age, education, education.num, marital.status, occupation, relationship, race, sex, capital.gain, hours.per.week.

	Specs	Score
0	age	0.372614
1	workclass	0.076415
2	fnlwgt	0.041794
3	education	0.104319
4	education.num	0.078860
5	marital.status	0.129128
6	occupation	0.145739
7	relationship	0.237875
8	race	0.044761
9	sex	0.066568
10	capital.gain	0.094299
11	capital.loss	0.003555
12	hours.per.week	0.063642
13	native.country	0.018714

Figura 8: Score calcolato tramite `SelectKBest`

3.4 Feature Scaling

Come ultima parte del preprocessing normalizzo i valori delle feature tramite un'operazione di scaling, nello specifico scelgo di utilizzare la funzione `MinMaxScaler` (`sklearn`). `MinMaxScaler` risulta vantaggioso perchè a differenza ad esempio dell'utilizzo dello *z-score* ci garantisce che tutte le feature saranno comprese in uno stesso range di valori (tra 0 e 1). Allo stesso tempo però è un approccio che non è in grado di gestire bene gli outlier (a differenza di uno scaling che utilizza lo *z-score*) ma in questo caso specifico dove ho già affrontato tale problema mi è sembrato l'approccio migliore tra i due. L'operazione di estrazione di massimo e minimo viene fatta sul *Training set* e i valori vengono poi riutilizzati per fare anche lo scaling del *Test set*.

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	race	sex	capital.gain	hours.per.week
0	-1.112175	3.0	-1.815589	11.0	9	0	7.0	1.0	4.0	0.0	-0.297114	40
1	-1.578018	3.0	-0.676981	1.0	7	1	9.0	3.0	1.0	0.0	-0.297114	40
2	-0.707154	5.0	0.428719	15.0	10	0	13.0	1.0	4.0	1.0	-0.297114	40
3	1.394699	3.0	0.504372	11.0	9	0	11.0	4.0	4.0	0.0	-0.297114	30
4	-1.112175	3.0	-1.084744	11.0	9	0	11.0	3.0	4.0	0.0	-0.297114	32
...	...	...	...	...	...	...	...	...	...	...	...	...
46713	0.818901	1.494220	1.033560	12.0	14	1	0.741329	0.0	4.0	1.0	-0.297114	40
46714	0.004428	3.000000	-1.857279	11.0	9	1	6.000000	0.0	4.0	1.0	3.365733	40
46715	0.054978	3.000000	-0.713522	12.0	14	1	3.000000	0.0	4.0	1.0	-0.297114	40
46716	1.110568	5.000000	0.549693	12.0	14	1	11.000000	0.0	4.0	1.0	3.365729	45
46717	0.542763	5.438417	-0.320801	12.0	14	1	3.000000	0.0	4.0	1.0	-0.297114	70

Figura 9: Prima dello scaling

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	race	sex	capital.gain	hours.per.week
0	0.185940	0.428571	0.132528	0.733333	0.533333	0.0	0.538462	0.2	1.000000	0.0	0.0	0.428571
1	0.086257	0.428571	0.329528	0.066667	0.400000	1.0	0.692308	0.6	0.000000	0.0	0.0	0.428571
2	0.272609	0.714286	0.520834	1.000000	0.600000	0.0	1.000000	0.2	1.000000	1.0	0.0	0.428571
3	0.722373	0.428571	0.533924	0.733333	0.533333	0.0	0.846154	0.8	1.000000	0.0	0.0	0.318681
4	0.185940	0.428571	0.258978	0.733333	0.533333	0.0	0.846154	0.6	1.000000	0.0	0.0	0.340659
...	...	...	...	...	...	...	...	...	...	...	...	...
46713	0.599161	0.213460	0.625483	0.800000	0.866667	1.0	0.057025	0.0	1.0	1.0	0.000000	0.428571
46714	0.424876	0.428571	0.125315	0.733333	0.533333	1.0	0.461538	0.0	1.0	1.0	0.999995	0.428571
46715	0.435693	0.428571	0.323206	0.800000	0.866667	1.0	0.230769	0.0	1.0	1.0	0.000000	0.428571
46716	0.661573	0.714286	0.541765	0.800000	0.866667	1.0	0.846154	0.0	1.0	1.0	0.999994	0.483516
46717	0.540071	0.776917	0.391154	0.800000	0.866667	1.0	0.230769	0.0	1.0	1.0	0.000000	0.758242

Figura 10: Dopo lo scaling

4 Modeling

A questo punto del progetto è necessario selezionare un modello. Per farlo svolgo due operazioni, un confronto tra i modelli base (Cross validation) e successivamente un Fine Tuning dei parametri di quelli più promettenti (Randomized Search e Grid Search). Inoltre effettuerò un'operazione simile anche per un modello di Deep Learning sfruttando la libreria Keras, per confrontarlo con i vari algoritmi di Machine Learning e osservare come si comporta e chi risulta migliore in questo specifico task.

4.1 Cross validation

Ho deciso di testare i seguenti modelli:

- Logistic Regression
- Decision Tree Classifier
- Random Forest Classifier
- Ada Boost Classifier
- Gradient Boosting Classifier
- XGBoost Classifier

Sfrutto la funzione `RepeatedStratifiedKFold` (`sklearn`) per creare un insieme di 10 coppie train-validation su cui alleno e valido i sopracitati. I risultati in *Figura 11* sono la media degli score dell'operazione di cross validation, utlizzando come metodo di valutazione *f1-score*, dato che non mi sarà possibile utilizzare l'accuratezza a causa del dataset sbilanciato.

```
LogisticRegression() Score: 0.800 (0.006)
DecisionTreeClassifier() Score: 0.858 (0.005)
RandomForestClassifier() Score: 0.885 (0.005)
AdaBoostClassifier() Score: 0.852 (0.005)
GradientBoostingClassifier() Score: 0.869 (0.005)
XGBClassifier() Score: 0.905 (0.004)
```

Figura 11: Risultati utilizzando la media degli score dell'operazione stratified k fold cross validation

4.2 Fine Tuning dei parametri

Partendo dai risultati precedenti scelgo di lavorare su *Random Forest* e *XGBoost* e li sottopongo quindi alla funzione `RandomizedSearchCV` con una repeated k fold cross validation di 3 fold, 2 ripetizioni e 100 iterazioni partendo dai parametri in figura.

```
random_parameters = [
    # Parametri RandomForest
    {
        "bootstrap": [True],
        "max_depth": np.arange(1, 20),
        "max_features": np.arange(1, 8),
        "min_samples_leaf": np.arange(1, 10),
        "min_samples_split": np.arange(2, 20),
        "n_estimators": np.arange(50, 150, 10),
    },
    # Parametri XGB
    {
        "n_estimators": np.arange(50, 150, 10),
        "criterion": ["gini", "entropty"],
        "max_depth": np.arange(1, 15),
        "max_features": ["int", "float", "auto", "log2"],
    },
]
```

```
grid_parameters = [
    # Parametri RandomForest
    {
        "bootstrap": [True],
        "max_depth": np.arange(17, 21),
        "max_features": np.arange(1, 5),
        "min_samples_leaf": np.arange(1, 5),
        "min_samples_split": np.arange(15, 19),
        "n_estimators": np.arange(100, 121, 10),
    },
    #  Parametri XGB
    {
        "n_estimators": np.arange(90, 111, 10),
        "criterion": ["gini", "entropty"],
        "max_depth": np.arange(5, 9),
        "max_features": ["log2", "auto"],
    },
]
```

Figura 12: Parametri provati, con *Randomized Search* a sinistra, con *Grid Search* a destra

Ora usando come punto di partenza i parametri migliori trovati con il `RandomizedSearch` creo un intervallo di parametri per il `GridSearchCV`. Ottengo infine i due modelli migliori possibili.

```
Grid Search RandomForestClassifier()
      {
      'bootstrap': True,
      'max_depth': 19,
      'max_features': 7,
      'min_samples_leaf': 1,
      'min_samples_split': 15,
      'n_estimators': 110
      }
Score: 0.9251643781510652
```

```
Grid SearchXGBClassifier()
      {

      'criterion': 'gini',
      'max_depth': 6,
      'max_features': 'log2',
      'n_estimators': 100

      }
Score: 0.9243476179075891
```

Figura 13: Risultati dell'operazione di `GridSearchCV`

`RandomForest` si dimostra leggermente superiore e di conseguenza sarà il modello definitivo che utilizzerò.

Effettuo ora `RandomizedSearchCV` su un modello sequenziale creato con la libreria `Keras`, questo è possibile grazie al wrapper `KerasClassifier` fornito dalla libreria `scikeras`.
Per lo sviluppo della struttura della rete non esistono delle regole precise ma bensì solo alcune linee guida. Ho seguito alcuni principi condivisi praticamente dall'intera community di Machine Learning, cioè:

- Mantenere la rete semplice
- Numero di neuroni negli hidden layer compreso tra input e output
- Numero di neuroni negli hidden layer decrescente
- Piccolo numero di hidden layer, tra 1 e 5

```
Random Search KerasClassifier()
      {
      'optimizer__learning_rate': 0.889,
      'optimizer': 'Adam',
      'epochs': 20,
      'batch_size': 1
      }

      Accuracy: 0.828878836411215
```

Figura 14: Risultato di *Randomized Search* sulla rete neurale

Di conseguenza la rete neurale è composta da input layer, 3 hidden layer e output layer. I neuroni in ogni layer sono 12-9-5-3-1.
Dati gli scarsi risultati e l'onerosità computazionale (e temporale) di effettuare un'operazione di `GridSearchCV`, ho predisposto il codice per quest'ultima ma non l'ho eseguito.

5 Conclusioni

Per concludere sarà necessario verificare che il modello migliore trovato (*XGBoost* basandoci puramente sui risultati del *Grid Search*) si comporti correttamente quando gli vengono sottoposte osservazioni mai viste prima.

5.1 Test

Configuro un'istanza di `XGBoost` con i parametri trovati, la alleno sul *Training set* e infine vado a verificarne le capacità di generalizzazione sul *Test set*, rimasto intoccato per tutto il progetto.

```
---------- XGBoost ----------
------------------------------------------------------------
Confusion Matrix
[[6432  648]
 [ 576 1365]]
------------------------------------------------------------
Classification Report
              precision    recall  f1-score   support

           0       0.92      0.91      0.91      7080
           1       0.68      0.70      0.69      1941

    accuracy                           0.86      9021
   macro avg       0.80      0.81      0.80      9021
weighted avg       0.87      0.86      0.87      9021
```

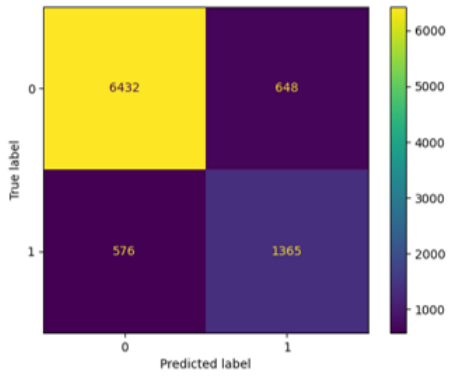

Figura 15: Risultati ottenuti con XGBoost sul Test set

5.2 Considerazioni

Arrivati in fondo al progetto non mi sembra di essere riuscito a colmare il problema del bilanciamento del dataset, infatti sul valore ">50k" (cioè la categoria "1") le predizioni del modello sono parecchio errate.

Appendices

A RandomForest

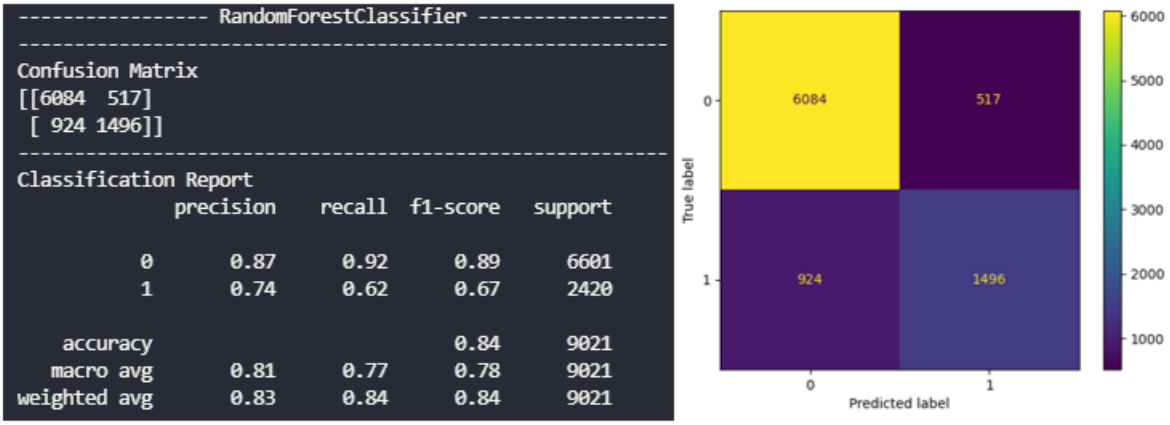

Figura 16: Risultati ottenuti con RandomForest sul Test set

Dopo i risultati ottenuti con `XGBoost` ho deciso di fare il testing anche con `RandomForest` utilizzando i suoi parametri migliori (*Figura 13*). I risultati in *Figura 16* ci riportano a ciò che ho detto nelle considerazioni finali.

B Confronto

Per curiosità mi sono chiesto se tutto il lavoro che ho svolto sul dataset nella fase di *Preprocessing* sia servito a qualcosa. Ho deciso quindi di dare in pasto sia a `RandomForest` sia a `XGBoost` il dataset di partenza (l'unica operazione che ho dovuto effettuare è stato l'encoding delle variabili e del label).

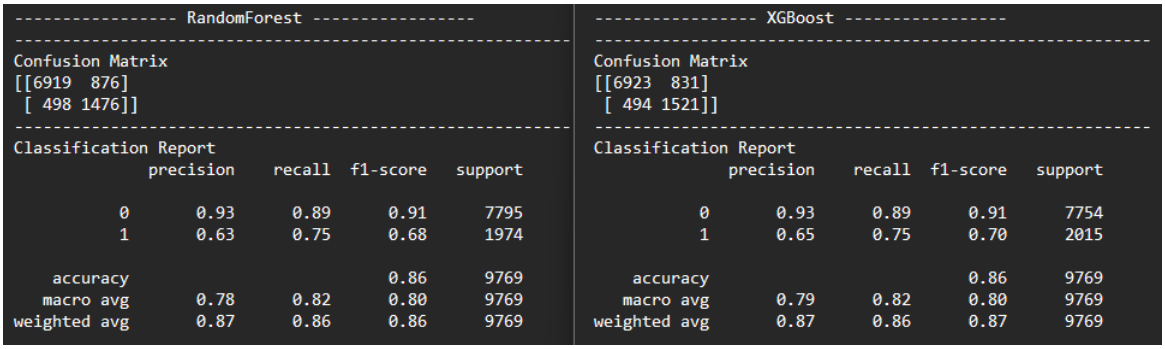

Figura 17: Risultati ottenuti dai due modelli sul dataset non processato

A quanto sembra la risposta é no, questo non va molto a mio favore ma onestamente non avrei saputo fare di meglio sul dataset.