Università degli studi di
Milano-Bicocca

Advanced Machine Learning

Progetto finale

Un aiuto per Airbnb:

Approcci di Machine Learning per prevedere la destinazione dei nuovi iscritti statunitensi

Autori:
Dario Carolla - 807547 - d.carolla@campus.unimib.it
Matteo Gaverini - 808101 - m.gaverini1@campus.unimib.it
Paolo Mariani - 800307 - p.mariani20@campus.unimib.it

Giugno 2020

Sommario

Il seguente report ha il fine di illustrare gli approcci che hanno portato alla risoluzione del problema di classificazione multiclasse proposto da Airbnb tramite una challenge su Kaggle. Avendo a disposizione alcune informazioni degli utenti e le relative sessioni di utilizzo del portale Airbnb, si desidera prevedere il Paese in cui i nuovi iscritti statunitensi effettueranno la loro prima prenotazione. Per la risoluzione verranno presentati approcci *Deep Learning*, *Ensemble Learning* oltre a modelli classici di apprendimento automatico, verrà svolta un'ottimizzazione dei principali iperparametri tramite *AutoML*, infine verrà effettuato un confronto di tutti i modelli realizzati e, tramite *cross-validation*, verrà decretato l'approccio migliore.

1 Introduzione

Airbnb è una piattaforma digitale che consente ai privati di affittare, per brevi periodi, camere o appartamenti. Fondato nel 2007 da tre ragazzi statunitensi, oggi il portale conta alloggi in oltre 100.000 città in più di 220 Paesi del mondo [1]. Il successo e la crescita economica di tale piattaforma è legato ad una serie di fattori tra cui la capacità di creare valore dai dati generati dagli utenti attraverso la *Data Science* [2]. Le ricerche effettuate sul portale ed il tempo trascorso su una pagina web sono dati che possono essere utilizzati dall'azienda non solo per capire le preferenze di un utente, ma soprattutto per decretare se una persona effettuerà una prenotazione tramite Airbnb. Per questo motivo le tracce digitali lasciate dagli utenti, se usate ed interpretate nel modo corretto, possono rappresentare un vantaggio competitivo per l'impresa. Il progetto prevede di determinare il Paese in cui i nuovi iscritti statunitensi ad Airbnb effettueranno la loro prima prenotazione, le cui mete possibili sono incluse in una lista di 11 destinazioni. Il task da risolvere è quindi una classificazione multiclasse dove però, a differenza di tutti gli altri problemi classici di questo tipo, non si restituisce come outcome una sola label ma una top 5 delle destinazioni più probabili ordinate in modo decrescente secondo il loro valore di probabilità. Per risolvere questo problema si sono adottati diversi approcci di *Machine Learning*: innanzitutto si è definita una rete neurale secondo le tecniche di *Deep Learning*, dopodiché, al fine di confrontare il risultato ottenuto, si sono prima realizzati dei modelli classici di apprendimento automatico come *Random Forest* e poi adottati approcci

di *Ensemble Learning*. Una volta ottenuti tutti i modelli e ottimizzato i principali iperparametri di ciascuno attraverso *AutoML*, si è proceduto alla loro validazione utilizzando la tecnica della *cross-validation* al fine di decretare il classificatore migliore.

N.B. Tutte le immagini citate nel report sono contenute nell'appendice.

2 Dataset

Il dataset, messo a disposizione da Airbnb per una competition su Kaggle, è costituito da sei diversi file in formato .csv contenenti informazioni relative agli utenti statunitensi che effettuano determinate azioni sul sito (ricerche mete, invio richieste di prenotazione etc.) ed info descrittive per ogni meta. Per risolvere il task in esame, si è deciso di non utilizzare tutti i file poichè tre di questi risultano essere superflui e poco informativi; il primo **countries.csv** contiene solamente delle informazioni riguardo alle mete come posizione latitudine, longitudine e superficie quadrata, il secondo **age_gender_bkts.csv**, include statistiche descrittive riguardo la popolazione di ogni destinazione suddivisa per range di età, infine l'ultimo **sample_submission.csv**, definisce il formato che devono avere i risultati per essere poi validati su Kaggle. I file utilizzati invece sono tre: **train_users.csv**, **test_users.csv** e **sessions.csv**. Il primo contiene, oltre alla variabile target *country_destination*, attributi descrittivi riguardanti gli utenti che hanno già effettuato la loro prima prenotazione (età, sesso, tipo dispositivo utilizzato etc.). Il secondo include le stesse feature del file precedente (ad eccezione di *date_first_booking* e *country_destination*) relative però ai nuovi utenti iscritti ad Airbnb cioè quelli che non hanno ancora prenotato. Infine l'ultimo file contiene tutti i log riguardanti le sessioni di utilizzo del sito da parte degli utenti che si sono registrati dopo il 01-01-2014.

2.1 Analisi e Preprocessing per train_users.csv e test_users.csv

I due file contengono rispettivamente 213451 e 62096 osservazioni. Tali csv sono stati uniti in un'unica struttura dati chiamata ***all_data*** per eseguire il preprocessing descritto in questa sezione. Di seguito si presentano gli attributi di ***all_data***:

- *date_account_created*: data di iscrizione dell'utente al portale, a partire dal 01-01-2010 fino al 30-09-2014.
- *timestamp_first_active*: timestamp che rappresenta la data in cui l'utente effettua la sua prima attività su Airbnb, può assumere valori antecedenti a *date_account_created* poichè un utente può effettuare un'interazione con il portale ancora prima di iscriversi.
- *date_first_booking*: data in cui avviene la prima prenotazione dell'utente, eliminata poichè presente solo nel file *train_users.csv*.
- *gender*: genere dell'utente, i valori che può assumere sono "MALE", "FEMALE", "OTHER" e "-unknown-", osservabili nella Figura 6.
- *age*: età dell'utente; la sua distribuzione è rappresentata nella Figura 4.
- *signup_method*, *signup_flow* e *signup_app*: il primo attributo rappresenta la piattaforma attraverso il quale l'utente si iscrive ad Airbnb, il secondo indica la pagina da cui un utente si è registrato infine l'ultimo attributo corrisponde al sistema operativo usato nel momento dell'iscrizione.
- *language*: lingua predefinita dall'utente sul portale; quasi la totalità assume valore "en" poichè i dati sono relativi ad utenti statunitensi.
- *affiliate_channel*, *affiliate_provider* e *first_affiliate_tracked*: la prima feature rappresenta il tipo di marketing effettuato attraverso il quale Airbnb è riuscita a raggiungere l'utente, la seconda l'advertiser che ha effettuato quella tipologia di marketing infine l'ultima feature indica l'attività di marketing con cui l'utente entra in contatto per primo con il portale. Si osserva la distribuzione di *first_affiliate_tracked* rispetto a *affiliate_channel* nella Figura 7.
- *first_device_type* e *first_browser*: il primo attributo indica il dispositivo utilizzato dall'utente durante la sua prima attività mentre il secondo il browser che ha usato la prima volta. La distribuzione dei valori del tipo di dispositivo si osserva nella Figura 8.
- *country_destination*: variabile target, può assumere 12 valori diversi tra cui 10 codici ISO 3166-1 alpha-2 [3] che rappresentano le mete di destinazione (Italia, Francia, Canada, Gran Bretagna, Spagna, Germania, Australia, Portogallo, Olanda e USA) e 2 valori che sono "NDF"

e "other", il primo indica la mancata prenotazione dell'utente al primo utilizzo del sito mentre il secondo rappresenta che l'utente ha prenotato in un Paese che non è incluso nella lista delle 10 nazioni. Si osserva la distribuzione nella Figura 9.

Per ciò che concerne il preprocessing, sono state effettuate due operazioni: *variable trasformation* e *missing replacement*. La prima è stata svolta sulle variabili *date_account_created* e *timestamp_first_active*; tale operazione prevede l'estrazione del giorno, mese, anno di ciascuna data e la creazione di due variabili derivate per feature (*first_active_month_sin*, *first_active_month_cos*, *account_creation_month_sin* e *account_creation_month_cos*) che tengano conto sia della stagionalità delle iscrizioni che quella delle prime attività degli utenti. A tal fine si sono utilizzate le funzioni seno e coseno. La seconda operazione invece, è stata svolta sulle feature *first_affiliate_tracked* ed *age*; tale task prevede l'applicazione di due diverse tecniche di *missing replacement*: moda condizionata e costante globale. Il primo metodo, avvenuto per la variabile *first_affiliate_tracked*, consiste nella sostituzione dei *missing values* prendendo il suo valore più frequente in relazione a quello della variabile *affiliate_channel*; si è effettuata tale operazione perchè si è riscontrato che i valori tra queste feature risultano essere dipendenti tra di loro. La seconda tecnica invece, effettuata questa volta sulla variabile *age*, prevede di sostituire tutti i *missing values* con il valore -1. All'interno di tale feature, inoltre, si è cercato di rimuovere il rumore rappresentato da alcuni outlier come ad esempio 1967 e 200. Pertanto i valori riconducibili ad una data di nascita, sono stati sostituiti con l'età attuale al momento dell'iscrizione mentre tutti quelli esterni all'intervallo [18, 100] (secondo la politica di Airbnb), sono stati trattati come *missing values*. La distribuzione della variabile dopo il preprocessing si può osservare nella Figura 5.

2.2 Analisi e Preprocessing per sessions.csv

Il dataset, rappresentato da una struttura dati chiamata ***df_sessions***, è composto da 10567737 osservazioni distinte. Dopo aver effettuato un'unione di questa struttura dati con ***all_data***, si è osservato che 140265 utenti non possiedono informazioni relative alle proprie sessioni poiché sono registrati prima del 01-01-2014. Tutti i valori mancanti sono stati perciò sostituiti con il valore -1. Di seguito si presentano gli attributi di ***df_sessions***:

- *action*, *action_type* e *action_detail*: la prima feature indica l'azione effettuata sul sito dall'utente, la seconda il tipo di azione infine l'ultima feature rappresenta i dettagli dell'azione effettuata.
- *device_type*: device utilizzato dall'utente per compiere l'azione.
- *secs_elapsed*: secondi impiegati dall'utente per svolgere l'azione corrispondente.

Per quanto riguarda il preprocessing, sono state effettuate due operazioni: *missing replacement* e creazione variabili aggiuntive. La prima è avvenuta su tutte le variabili, esclusa *device_type*, usando tre tecniche diverse: imputazione manuale, moda condizionata e mediana. L'imputazione manuale, applicata sulla variabilie *action*, prevede di sostituire i *missing values* con "message"; si è effettuata questa operazione perché, riscontrando la dipendenza tra le feature *action* ed *action_type*, è emerso che per ogni valore nullo di *action* è associato un valore "message_post" in *action_type* ed ogni volta che è presente tale valore, l'azione è sempre un "message". La seconda tecnica, effettuata sulla variabile *action_type*, consiste nel sostituire i *missing values* con la moda che l'attributo assume in relazione a specifici valori di *action*. L'ultima tecnica, effettuata questa volta sulla variabile *secs_elapsed*, prevede invece di sostituire i valori mancanti con la mediana. La seconda operazione di preprocessing, fondamentale per il progetto, prevede la creazione di 15 feature ulteriori che sostituiranno quelle viste precedentemente. Le nuove variabili sono così definite:

- *number_actions*: numero di azioni svolte da ogni utente.
- *favourite_** e *last_**: la prima feature rappresenta il valore più frequente contenuto in una determinata variabile, la seconda feature invece, rappresenta l'ultima informazione presente in ordine cronologico (* indica che la creazione è avvenuta per diversi attributi: *favourite_** per tutti gli attributi eccetto *secs_elapsed* mentre il secondo per *action*, *action_type* e *action_detail*).
- *number_favourite_action*: numero di volte in cui l'utente ha effettuato l'azione più frequente.
- *translate*: variabile binaria, assume valore 1 quando il valore di *action* è 'ajax_google_translate_*' ovvero l'utente chiede una traduzione di ciò che legge.

- *total/min/max/mean/stdev+_secs_elapsed*: la prima feature indica il tempo totale delle azioni effettuate dall'utente; la seconda e la terza la durata minima e massima di un'azione, infine la quarta e la quinta, la media e la deviazione standard delle durate delle azioni effettuate da ciascun iscritto.
- *diff_mean*: scarto tra il tempo totale delle azioni effettuate da un utente e la media delle durate delle azioni di tutti gli iscritti.

Una volta effettuate tutte le operazioni di preprocessing, si è deciso di eliminare gli attributi correlati tra di loro in modo da non incorrere nel problema della *course of dimensionality*. Per far ciò si è effettuata un'analisi di correlazione realizzando due heatmap, una relativa alla struttura dati ***all_data*** (si veda Figura 10) mentre l'altra a ***df_sessions*** (si veda Figura 11). Analizzando il primo plot, si evidenzia che *date_account_created* e *timestamp_first_active* sono fortemente correlate. Si è deciso quindi di eliminare la feature *timestamp_first_active* (compreso tutte le variabili ad essa associate) in quanto *date_account_created*, risulta molto più significativa perché un utente, prima di prenotare, deve registrarsi ad Airbnb. Oltre a questo si è deciso di rimuovere anche *account_creation_month* perché correlata con *account_creation_month_sin*. Analizzando invece la seconda heatmap, emerge che *total_secs_elapsed*, *number_favourite_action_detail* e *number_favourite_action_type* risultano essere estremamente correlate con le altre variabili pertanto si è deciso di eliminarle. Una volta rimosse tutte le feature correlate, si sono effettuate altre due operazioni affinché il classificatore *Neural Network* non subisca gli effetti negativi di *exploding gradient* e apprendimento lento o instabile; tali task risultano essere *one-hot encoding* e *normalization*. La prima operazione è avvenuta solamente sulle feature categoriche che assumessero meno di 10 valori possibili; le altre invece, sono state codificate con un *Label Encoding*. Si è deciso di compiere due operazioni separate per uno stesso tipo di feature affinché non aumenti la complessità del dataset. La *normalization* invece, si è effettuata su tutte le variabili numeriche applicando per ognuna un *min-max scaling*.

3 Approccio Metodologico

Per risolvere il problema, si è innazitutto realizzata una *Fully Connected Neural Network*, dopodiché, al fine di confrontare il risultato ottenuto, si sono

implementati altri 4 modelli. I primi due, appartenenti al mondo del *Machine Learning* classico, risultano essere *Random Forest* e *Knn*, gli altri invece sono un *Decision Tree Gradient Boosting* chiamato *XGBoost* e un *Ensemble* che include tutti gli approcci definiti in precedenza. Tutti i modelli sono stati allenati sulla porzione di dati dedicata al training (**train_users.csv**), la quale è stata a sua volta suddivisa in 80% *training set* e 20% *test set*, entrambe stratificate rispetto alla variabile target. Si è effettuato un partizionamento stratificato perchè, come è visibile nella Figura 9, il dataset risulta essere fortemente sbilanciato in quanto la maggior parte dei valori è "NDF" oppure "US". I classificatori realizzati meno *Ensemble*, sono stati poi sottoposti all'ottimizzazione dei principali iperparametri tramite *AutoML*, un processo che permette di ottimizzare una funzione obiettivo determinando la combinazione ideale degli iperparametri [4]. Nel caso in esame, la funzione obiettivo corrisponde a massimizzare la metrica di valutazione utilizzata in tutti i modelli (richiesto dalla competition Kaggle) ovvero la *NDCG@k* con $k = 5$. Tale metrica, il cui valore è compreso nell'intervallo $[0.0, 1.0]$, è calcolata nel seguente modo:

$$DCG_k = \sum_{i=1}^{k} \frac{2^{rel_i} - 1}{log_2(i+1)},$$

$$nDCG_k = \frac{DCG_k}{IDCG_k}$$

dove rel_i è la rilevanza del risultato in posizione i.

Per effettuare *AutoML* è stata utilizzata la libreria *pyGPGO* [5], attraverso la quale viene effettuata l'ottimizzazione bayesiana [6]. Tale processo iterativo prevede l'utilizzo di un modello surrogato che, dopo essere stato addestrato con delle configurazioni randomiche iniziali, ottimizza la funzione obiettivo corrispondente attraverso una funzione di acquisizione fino a quando non raggiunge un criterio d'arresto. Per quanto riguarda l'ottimizzazione effettuata, si è utilizzato come modello surrogato il *Gaussian Process* mentre come funzione di acquisizione l'*Expected Improvement*; considerando invece il *budget*, per ogni classificatore si sono effettuate 25 iterazioni incluse le 5 iniziali randomiche. Una volta ottenute le configurazioni migliori per ogni modello, si è proceduto alla loro valutazione tramite *10-fold cross-validation* in modo da decretare il classificatore migliore.

3.1 Fully Connected Neural Network

L'approccio principale per risolvere il problema è stato realizzare una *Fully Connected Neural Network* (FCNN) con il framework *Keras* [7] per sfruttare le tecniche di apprendimento *Deep Learning*. La struttura della rete prevede 5 layer di cui 3 nascosti; si sono scelti 3 strati nascosti perchè, tramite una ricerca empirica, si è rivelata essere la migliore combinazione per il task in esame. Per quanto riguarda l'ampiezza del layer di input, si è scelto un numero di neuroni pari a quello delle feature del dataset cioè 54 mentre per il layer di output si sono definiti 12 neuroni (essendo il task da risolvere un problema di classificazione multiclasse). Considerando le funzioni di attivazione, per tutti i layer eccetto quello di output si è utilizzata la $ReLU$ in quanto, come suggerisce la letteratura, risulta essere la migliore per i problemi di classificazione. Per l'ultimo strato invece, si è utilizzata la *softmax*, poiché l'obiettivo è trovare una distribuzione di probabilità per ogni classe a cui l'osservazione può appartenere. Oltre a questo, su tutti i layer meno quello finale, si è applicato il *kernel_initializer* [8] di tipo *he_uniform* che imposta i pesi iniziali del modello con una distribuzione uniforme della varianza. Considerando la *loss*, la scelta è ricaduta sulla *categorical crossentropy* in quanto misura la distanza tra la distribuzione di probabilità generata dalla rete e quella vera delle label; in questo modo, minimizzando la *loss*, si può trovare una distribuzione il più possibile vicina a quella reale delle label. Per poter valutare le prestazioni della rete durante la fase di training, oltre al valore della *loss*, è stata monitorata anche la *top5 accuracy* in quanto il framework *Keras* non permette di utilizzare la metrica *NDCG@k*. Per evitare *overfitting* della rete si sono applicate due tecniche di regolarizzazione: *Dropout* ed *Early Stopping*. Il primo, applicato su ogni strato, prevede di disattivare una percentuale di neuroni durante l'addestramento della rete; tale percentuale è definita dall'iperparametro *dropout rate*. Il secondo invece, consiste nel trovare il numero esatto di *epochs* entro le quali la *loss* non peggiori fissato un numero massimo di 200 epoche. Per quanto riguarda l'ottimizzatore, si è scelto *adam* poiché i risultati ottenuti in termini di prestazioni/tempo si sono rivelati i migliori tra tutti gli ottimizzatori testati. Uno dei principali problemi nella creazione di una rete neurale risulta definire l'ampiezza dei hidden layer; per questo motivo il numero di neuroni di ciascuno è stato ottimizzato con *AutoML* fornendo un intervallo possibile di valori che va da 1 al numero di feature del dataset. Inoltre, si è deciso di ottimizzare altri due iperparametri ovvero *dropout rate* e *learning rate*. Per il primo l'intervallo di

valori possibili è $[0.05, 0.40]$, mentre per l'altro $[0.001, 0.010]$. L'addestramento della rete neurale è stato effettuato utilizzando dei *batch* di dimensione 128 mescolati ad ogni epoca per garantire l'indipendenza l'uno dall'altro. Per ciò che concerne il *validation set*, si è utilizzato il 20% del *training set* il cui sottoinsieme però, non viene utilizzato per addestrare il modello ma per valutare la rete durante l'aggiornamento dei pesi dei neuroni. La struttura ottima ricavata dal processo di ottimizzazione *AutoML* è la seguente:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_126 (Dense)            (None, 54)                2970
_________________________________________________________________
dropout_101 (Dropout)        (None, 54)                0
_________________________________________________________________
dense_127 (Dense)            (None, 30)                1650
_________________________________________________________________
dropout_102 (Dropout)        (None, 30)                0
_________________________________________________________________
dense_128 (Dense)            (None, 16)                496
_________________________________________________________________
dropout_103 (Dropout)        (None, 16)                0
_________________________________________________________________
dense_129 (Dense)            (None, 3)                 51
_________________________________________________________________
dropout_104 (Dropout)        (None, 3)                 0
_________________________________________________________________
dense_130 (Dense)            (None, 12)                48
=================================================================
Total params: 5,215
Trainable params: 5,215
Non-trainable params: 0
_________________________________________________________________
time: 203 ms
```

Figura 1: Struttura FCNN

Considerando gli altri iperparametri ottimizzati, si evince che il valore ottimo per *dropout rate*, valido per ogni layer, è 0.10 mentre per *learning rate* è 0.004. Di seguito viene mostrato il grafico dell'andamento della *loss* e della *top5 accuracy* durante la fase di addestramento e valutazione del modello migliore.

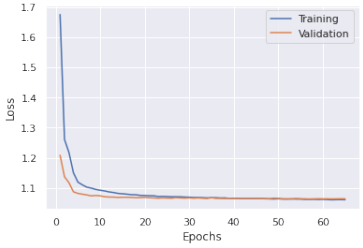

Figura 2: Andamento funzione loss

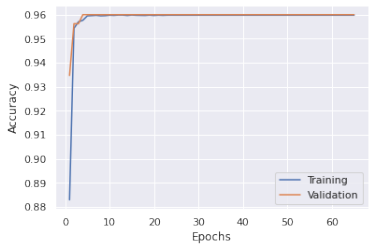

Figura 3: Andamento top5 accuracy

È dunque possibile osservare come, tramite le tecniche di regolarizzazione utilizzate, il modello non sia soggetto né a *overfitting* né a *underfitting* raggiungendo così una buona capacità di generalizzazione.

3.2 XGBoost classifier

Il secondo classificatore risulta essere *XGBoost classifier*, un modello che implementa un *gradient boosted decision tree*. Questo classificatore prevede di realizzare un certo numero di alberi decisionali in sequenza, in modo tale che ogni *decision tree* complementi quelli precedentemente costruiti [9]. Si è deciso di applicare un algoritmo di *gradient boosting*, implementato con la libreria *XGBoost* [10], in quanto è conosciuto per la sua efficacia nel risolvere problemi di classificazione caratterizzati dalla presenza di dataset complessi, come nel caso in esame. Gli iperparametri ottimizzati per questo modello risultano essere cinque: *gamma*, *learning rate*, *alpha*, numero *decision tree* e profondità massima di ciascun albero. Il primo parametro rappresenta la riduzione minima della *loss* (*reg:linear*) affiché venga effettuato un'ulteriore creazione di un ramo nell'albero; l'intervallo stabilito in cui si individua il valore ottimo è $[0.0, 2.0]$. Il secondo parametro, *learning rate*, indica la velocità con cui il modello si adatta al problema; l'intervallo definito in cui si decreta la configurazione migliore è $[0.0, 1.0]$. Il terzo parametro rappresenta *L1 regularization*, il cui valore ottimo si individua tra 0.0 e 1.0. Gli ultimi due iperparametri definiscono la struttura del classificatore, per il primo l'intervallo definito è $[2, 10]$ mentre per l'altro $[3, 10]$. Il risultato migliore ottenuto è un *XGBoost classifier* costituito da 7 alberi con profondità massima pari a 7, *gamma* impostato a 2.0, *learning rate* a 0.37 e *alpha* settato a 0.47.

3.3 Random Forest

Il terzo classificatore scelto è *Random Forest*, un modello costituito da un insieme di *decision tree*. Questo classificatore presenta la stessa struttura di *XGBoost* con l'unica differenza che gli alberi sono indipendenti l'uno dall'altro: ogni albero non predice una label basandosi sui risultati precedentemente ottenuti dagli altri modelli, ma genera una previsione in maniera indipendente [9]. Gli iperparametri ottimizzati risultano essere numero *decision tree* e loro profondità massima; per il primo l'intervallo definito in cui si individua il valore ottimo è $[100, 500]$, per l'altro invece $[5, 30]$. Il risultato ottenuto dal

processo di *tuning*, risulta essere un *Random Forest* costituito da 452 alberi con profondità massima pari a 14.

3.4 K-Nearest Neighbors

Il quarto modello risulta essere *K-Nearest Neighbor*, un algoritmo che, seppur nella sua semplicità, raggiunge risultati molto buoni nei problemi di classificazione. Rispetto a tutti gli altri modelli visti, in questo classificatore si è ottimizzato un solo iperparametro che è k; esso rappresenta il numero di vicini da considerare per assegnare l'etichetta finale ad ogni istanza [11]. Per definire la regione che include tutti i vicini si è utilizzata come misura di distanza la *minkowski distance*. Considerando l'ottimizzazione, si è definito un range compreso tra 50 e 200 per individuare la configurazione migliore di k che risulta essere 117.

3.5 Ensemble Model

L'ultimo classificatore risulta essere *Ensemble*, un modello che implementa un paradigma chiamato *ensemble learning*. Tale paradigma consiste nel combinare più algoritmi di *Machine Learning* al fine di migliorare le performance complessive del sistema [12]. Esistono diverse tecniche che si possono utilizzare per definire un *Ensemble*, alcune semplici come *Max Voting*, altre invece più complesse come *Stacking*. Nel caso in esame è stata utilizzata la tecnica *Average Voting* per combinare tutti i modelli visti finora, già ottimizzati in termini di iperparametri: *FCNN*, *Random Forest*, *XGBoost* e *KNN*. Tale tecnica prevede che, per ogni istanza, venga effettuata la media delle probabilità di appartenenza ad ogni classe generate dai vari classificatori per produrre la previsione finale. Si è scelto di applicare il metodo *Average Voting* perché si è ritenuto il più adatto rispetto al task di classificazione da risolvere.

4 Risultati e Valutazioni

Prima di presentare i risultati, si sottolinea che, per confrontare i diversi classificatori, a parità di condizioni con esiti costanti nel tempo, si sono scelti dei *seed* garantendo la replicabilità delle soluzioni a fronte di esecuzioni diverse. Tutte le valutazioni sono state effettuate sfruttando le macchine virtuali messe a disposizione da Google tramite il servizio *Google Colab*; ogni macchina

dispone 25 GB di RAM e 68 GB di disco fisso. Di seguito vengono presentati i risultati (si veda Tabella 1) di ciascun classificatore, ottenuti sia tramite la tecnica *cross-validation* che attraverso la validazione su *Kaggle*, operazione effettuata dalla piattaforma usando un *test set* sconosciuto e non disponibile ai partecipanti alla sfida.

Classificatore	**Tempo (s)**	**Cross Validation (NDCG +/- SD)**	**Kaggle Score (NDCG)**
FCNN	1506	82.53% (+/- 0.16)	87.02%
XGBoost	842	82.85% (+/- 0.16)	87.37%
Random Forest	1557	82.81% (+/- 0.16)	87.42%
K-Nearest Neighbors	2451	81.48% (+/- 0.20)	86.05%
Ensemble	5942	82.64% (+/- 0.17)	87.14%

Tabella 1: Risultati

5 Discussione

L'obiettivo del progetto è stato individuare un classificatore, nello specifico una rete neurale *deep*, capace di garantire buone performance sia dal punto di vista dell'efficienza (uso delle risorse di *Google Colab* e tempo impiegato) che efficacia (capacità di ottenere valori paragonabili a quelli della leaderboard di *Kaggle*). Analizzando i risultati ottenuti, emergono una serie di aspetti interessanti. Innanzitutto il classificatore migliore, considerando unicamente i valori della metrica *NDCG*, risulta essere *XGBoost* e non *FCNN*; questo implica che, per il task in esame, si può raggiungere un buon risultato utilizzando unicamente un insieme di *weak learners* come i *decision tree*, anziché realizzare un modello computazionalmente pesante come una rete *deep*. Inoltre, è necessario sottolineare che tra i due classificatori non è stata appurata alcuna differenza statistica, perciò non si può affermare che globalmente un modello sia migliore di un altro. Il secondo aspetto interessante che emerge è che *XGBoost* risulta essere l'unico classificatore che necessita meno di 15 minuti per completare il processo di *cross-validation*, gli altri invece, impiegano almeno più del doppio del tempo. Si può pertanto affermare che *XGBoost*, oltre ad essere il classificatore più efficace, risulta essere anche il più efficiente valutando il tempo impiegato. Considerando globalmente i risultati, è pos-

sibile asserire che tutti i classificatori ottengono delle buone performance, i valori *NDCG* sono superiori all'81%; l'unico modello che restituisce un valore leggermente inferiore rispetto a tutti gli altri risulta essere *KNN*. I motivi per cui *NDCG* assume valori alti in tutti i classificatori possono essere due. Il primo è che i modelli, influenzati dallo sbilanciamento del dataset, tendano ad includere nella top 5 delle destinazioni più probabili, i valori maggiormente rappresentati quali "NDF" oppure "US". Il secondo fattore è l'ottimizzazione effettuata con *AutoML*. Questo procedimento, molte volte ignorato a favore di un *grid search*, contribuisce in misura notevole al miglioramento delle soluzioni ottenute da un modello, soprattutto nei problemi di classificazione multiclasse. Dai risultati emerge che i valori *NDCG* ottenuti dalla validazione su *Kaggle*, sempre considerando unicamente i valori della metrica e non da un punto di vista statistico, evidenziano che il miglior classificatore sia *Random Forest* a differenza del risultato precedentemente analizzato. Oltre a ciò si evidenzia anche un miglioramento generale delle performance ottenute dai vari classificatori: tutti i valori *NDCG* sono superiori all'86%; l'unico modello che restituisce un valore leggermente inferiore rispetto a tutti gli altri risulta essere *KNN*, lo stesso classificatore da cui sono state ottenute le performance peggiori con la *cross-validation*.

6 Conclusioni

Nel progetto si sono presentate diverse metodologie per risolvere il task di classificazione proposto da Airbnb. Gli esiti ottenuti mostrano che non si può decretare un classificatore migliore. Oltre a questo si evidenzia che la rete neurale *deep*, diversamente dalle aspettative, non risulta essere il modello da cui si ottengono le performance più alte; ne consegue quindi che, adottare un approccio *Deep Learning* nel caso in esame, non si rivela la strategia migliore. In generale i risultati sono ritenuti soddisfacenti, il merito è soprattutto dovuto all'ottimizzazione effettuata con *AutoML*. A fronte degli esiti e delle analisi effettuate, si considera di poter apporre dei miglioramenti futuri. Il primo è aumentare il numero di iperparametri ottimizzati, in particolare il numero di hidden layer della FCNN attraverso *AutoML* incrementando le risorse computazionali. Il secondo invece, è migliorare la classificazione delle label più rare cercando di eliminare l'effetto dello sbilanciamento del dataset.

Riferimenti bibliografici

[1] Airbnb, "Airbnb, about us." [Online]. Available: https://news.airbnb.com/about-us/

[2] N. Patel, "How airbnb uses data science to improve their product and marketing." [Online]. Available: https://neilpatel.com/blog/how-airbnb-uses-data-science/

[3] ISO, "Iso 3166 country codes." [Online]. Available: https://www.iso.org/iso-3166-country-codes.html

[4] Z. Weng, "From conventional machine learning to automl," in *Journal of Physics: Conference Series*, vol. 1207, no. 1. IOP Publishing, 2019, p. 012015.

[5] J. Jiménez and J. Ginebra, "pygpgo: Bayesian optimization for python," *Journal of Open Source Software*, vol. 2, no. 19, p. 431, 2017.

[6] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in neural information processing systems*, 2012, pp. 2951–2959.

[7] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[8] K. library, "Weights initializers." [Online]. Available: https://keras.io/initializers/

[9] M. Luckner, B. Topolski, and M. Mazurek, "Application of xgboost algorithm in fingerprinting localisation task," in *IFIP International Conference on Computer Information Systems and Industrial Management*. Springer, 2017, pp. 661–671.

[10] X. library, "Xgboost documentation." [Online]. Available: https://xgboost.readthedocs.io/en/latest/index.html

[11] K. Chomboon, P. Chujai, P. Teerarassamee, K. Kerdprasop, and N. Kerdprasop, "An empirical study of distance metrics for k-nearest neighbor algorithm," in *Proceedings of the 3rd international conference on industrial application engineering*, 2015, pp. 1–6.

[12] G. Valentini and F. Masulli, "Ensembles of learning machines," in *Italian workshop on neural nets*. Springer, 2002, pp. 3–20.

7 Appendice

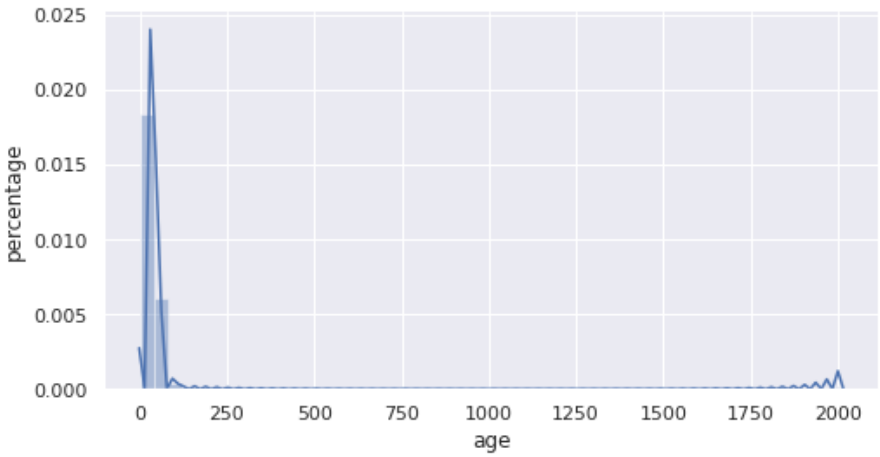

Figura 4: Distribuzione *age* prima del preprocessing

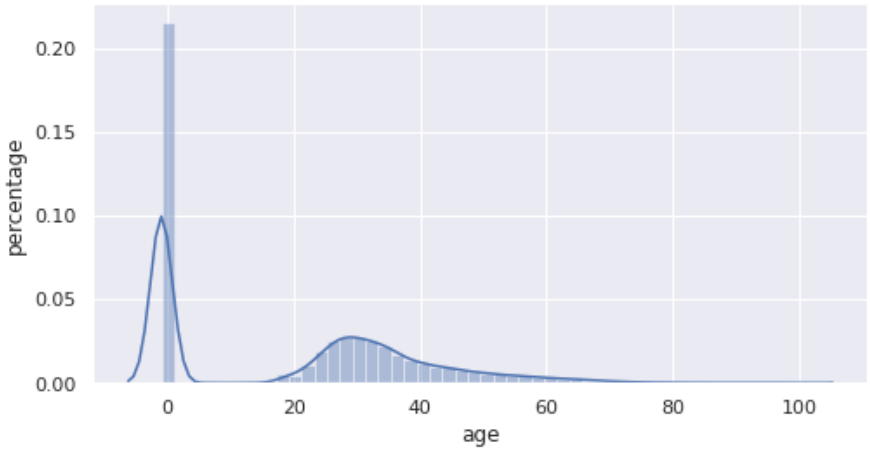

Figura 5: Distribuzione *age* dopo il preprocessing

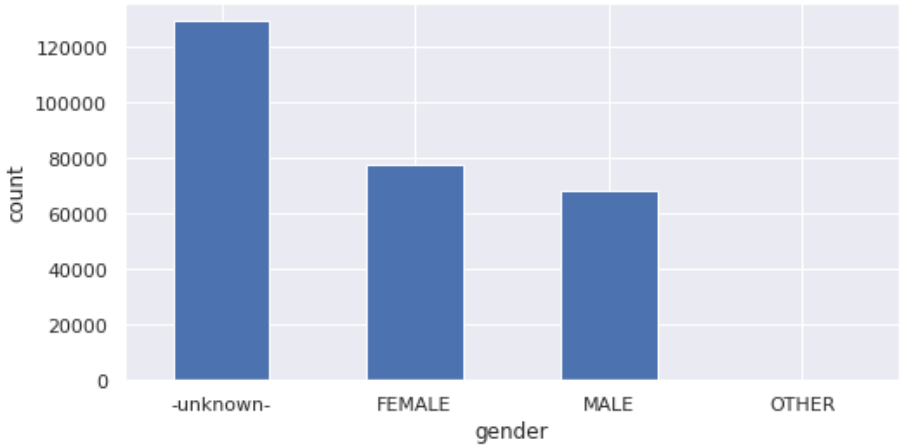

Figura 6: Distribuzione *gender*

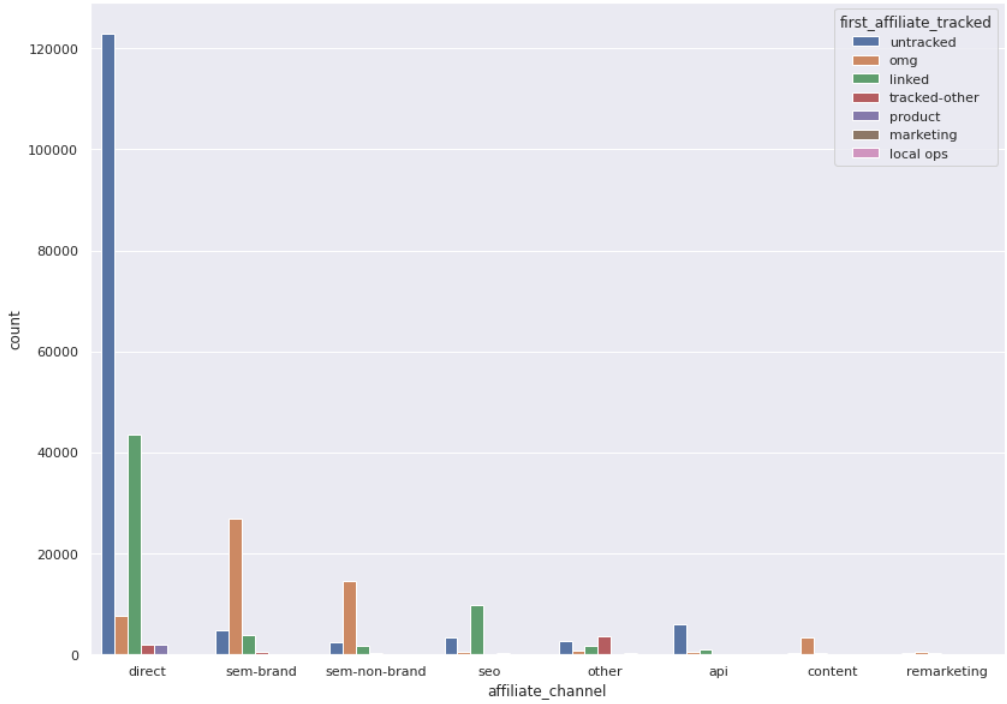

Figura 7: Distribuzione *first_affiliate_tracked* rispetto a *affiliate_channel*

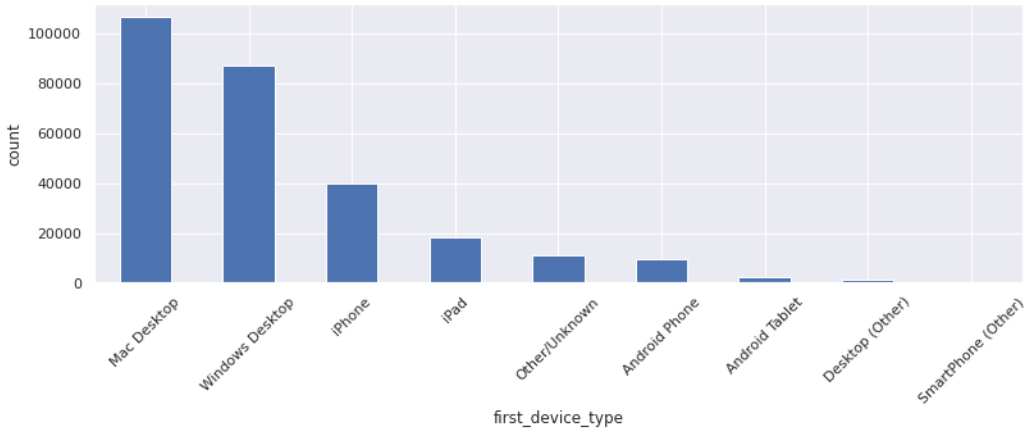

Figura 8: Distribuzione *first_device_type*

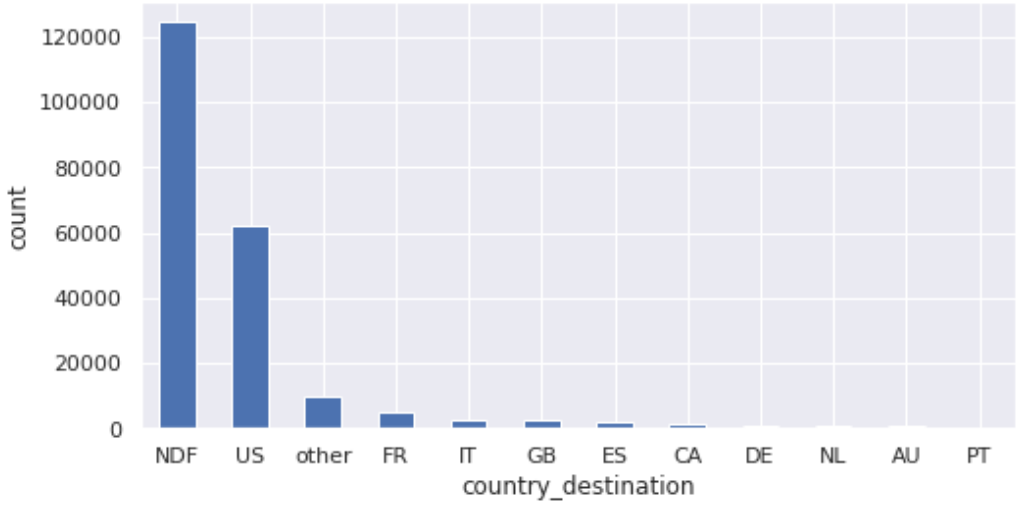

Figura 9: Distribuzione variabile target *country_destination*

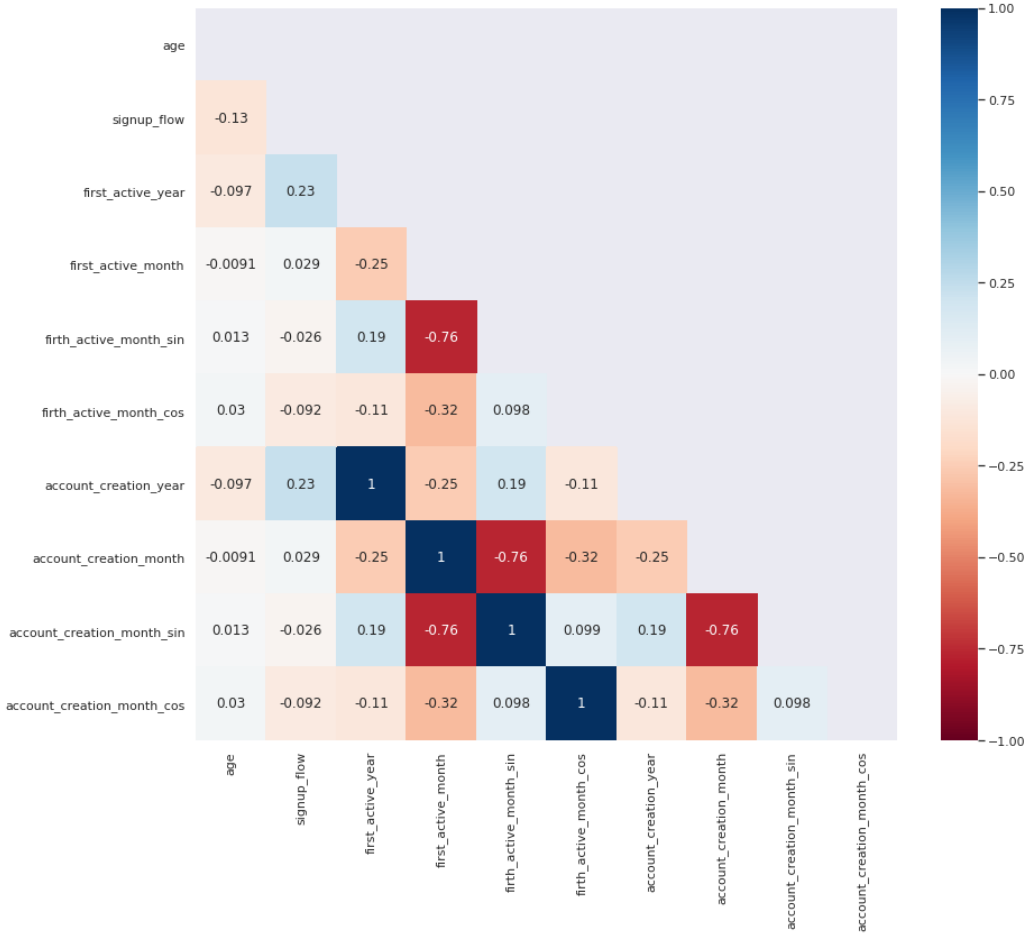

Figura 10: Analisi di correlazione delle variabili numeriche della struttura ***all_data***

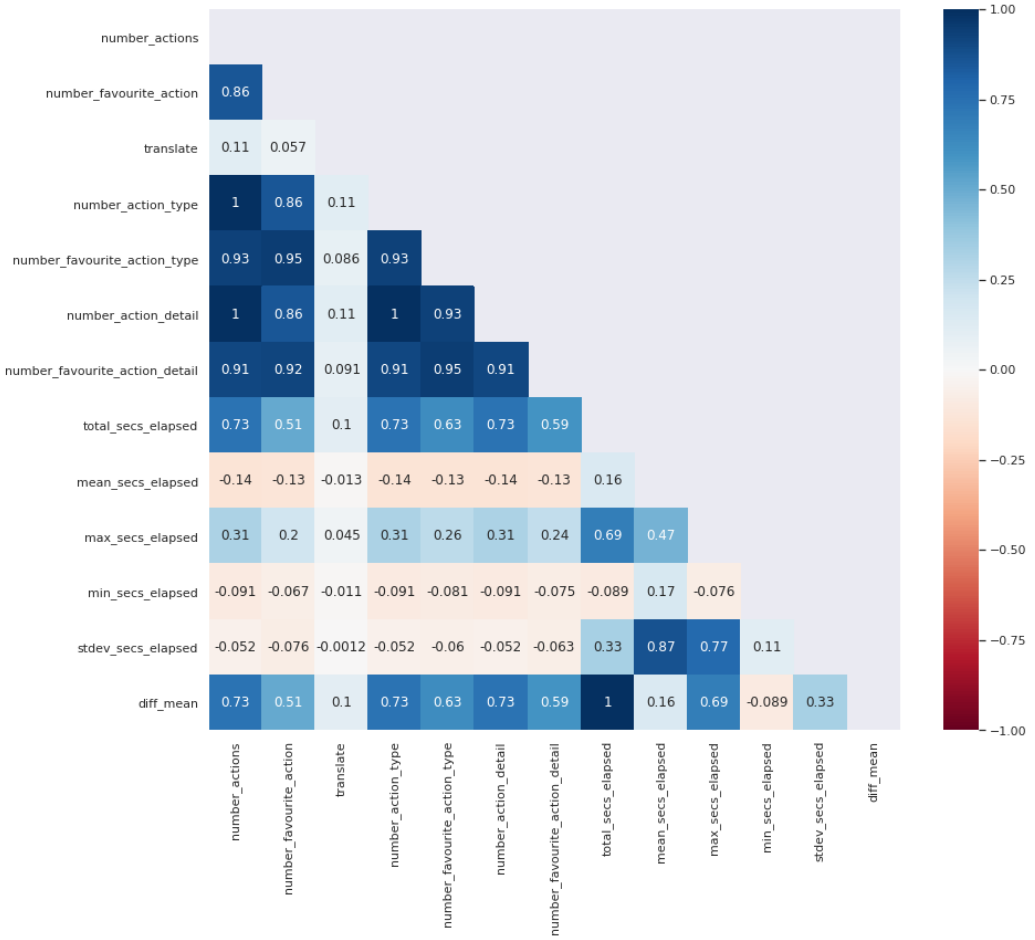

Figura 11: Analisi di correlazione delle variabili numeriche della struttura ***df_sessions***