Mobile Price Classification

Predizione del range di prezzo di uno smartphone in base alle sue caratteristiche

Obbiettivo del progetto di machine learning: Bob ha appena creato la sua compagnia di smartphones ma non sa come stimare il prezzo dei suoi telefoni. Ha raccolto quindi i dati e le caratteristiche di diversi telefoni in vendita e vuole che tramite un algoritmo di machine learning sia possibile individuare non il prezzo esatto ma bensì un range di prezzo in cui collocare ogni telefono in base alle sue caratteristiche specifiche.

Dataset:
Il dataset è ottenibile all'indirizzo:
https://www.kaggle.com/iabhishekofficial/mobile-price-classification

Vengono forniti due dataset, il primo è il test.csv, che sarebbe il dataset contenente i dati dei telefoni di Bob, questo lo utilizzeremo solo nella fase finale per predire la fascia di costo dei suoi telefoni. Questo dataset ha le seguenti 21 colonne:

1. id, una colonna contenente l'ID del telefono (numero crescente da 1 a 1000);
2. battery_power, l'energia totale che può contenere la batteria, misurata in mAh;
3. blue, una variabile booleana che indica se ha il bluetooth o no (valore 0 o 1);
4. clock_speed, la velocità di esecuzione delle istruzioni del microprocessore;
5. dual_sim, variabile che indica se il telefono supporta il dual sim o no;
6. fc, indica i mega pixel della fotocamera frontale
7. four_g, indica se ha il 4G o no
8. int_memory, indica i Gigabyte di memoria interna
9. m_dep, indica lo spessore del telefono in centimetri
10. mobile_wt, indica il peso del telefono
11. n_cores, indica il numero di cores del processore
12. pc, indica i megapixel della fotocamera primaria
13. px_height, indica la risoluzione dei pixel in altezza
14. px_width, indica la risoluzione dei pixel in larghezza
15. ram, indica la dimensione della ram in Megabytes
16. sc_h, indica l'altezza dello schermo in centimetri
17. sc_w, indica la larghezza dello schermo in centimetri
18. talk_time, indica quanto tempo durerà la batteria con una carica piena in chiamata
19. three_g, variabile booleana che indica se ha o meno il 3G

20. touch_screen, variabile booleana che indica se il telefono è fornito di touchscreen
21. wifi, variabile booleana che indica se il telefono ha il wifi o no.

Il secondo dataset fornito si chiama train.csv e contiene i dati dei telefoni con cui allenare i nostri modelli di machine learning. Anch'esso è costituito da 21 colonne che sono esattamente come quelle del dataset test.csv con la sola differenza che in train.csv non vi è la colonna id all'inizio e la 21-esima colonna è la colonna "price_range" che contiene le variabili target che per ogni telefono indicano se è di fascia di prezzo bassa (valore 0), media (valore 1), alta (valore 2) o molto alta (valore 3).
Avendo una variabile target da predire utilizzeremo un tipo di machine learning supervisionato.

Come visionabile sul sito, si nota subito che nel dataset la maggior parte delle features dei telefoni sono variabili quantitative ma vi sono anche delle variabili categoriche (vero o falso/presente o non presente) che ci dicono se un telefono è fornito o meno di un certo componente.

Data Analysis & Visualization

Per l'analisi dei dati, abbiamo utilizzato le seguenti librerie Python:

1. Numpy;
2. Pandas;
3. Seaborn;
4. Matplotlib (modulo Pyplot);

Come in ogni progetto di machine learning, siamo partiti analizzando i dati presenti all'interno dei dataset, per capirne la tipologia, quanto fossero affidabili, se vi fossero valori nulli e soprattutto abbiamo cercato di capire quale fossero le features che più pesavano nella individuazione della classe target.

L'intero processo di analisi dei dati e visualizzazione si trova nel file "*DataAnalysis&Visualization.py*" (in alternativa vi è lo stesso file con estensione *.ipynb* di tipo Python Notebook)

Per iniziare abbiamo scaricato i due dataset in locale come file cvs e li abbiamo importati e tradotti utilizzando la funzione di Pandas *pd.read_csv*.

Sempre con Pandas abbiamo controllato se vi fossero dei valori nulli o non validi all'interno dei nostri dataset e con grande piacere abbiamo scoperto che non vi erano valori nulli in nessuno dei due dataset.

Abbiamo poi visualizzato le prime cinque righe di entrambi i dataset per dare un primo controllo ai dati presenti ed è stata subito notata la presenza di alcune variabili categoriche, di una colonna *id* omettibile nel dataset di test e che la colonna target nel dataset di train era l'unica colonna contenente variabili qualitative.
Il resto delle colonne conteneva invece dati di tipo quantitativo.

Visualizzando la forma del dataset abbiamo notato che il dataset di train conteneva 2000 righe e 21 colonne (inclusa quella target) mentre quello di test conteneva 1000 righe e 21 colonne (di cui una era la colonna id).

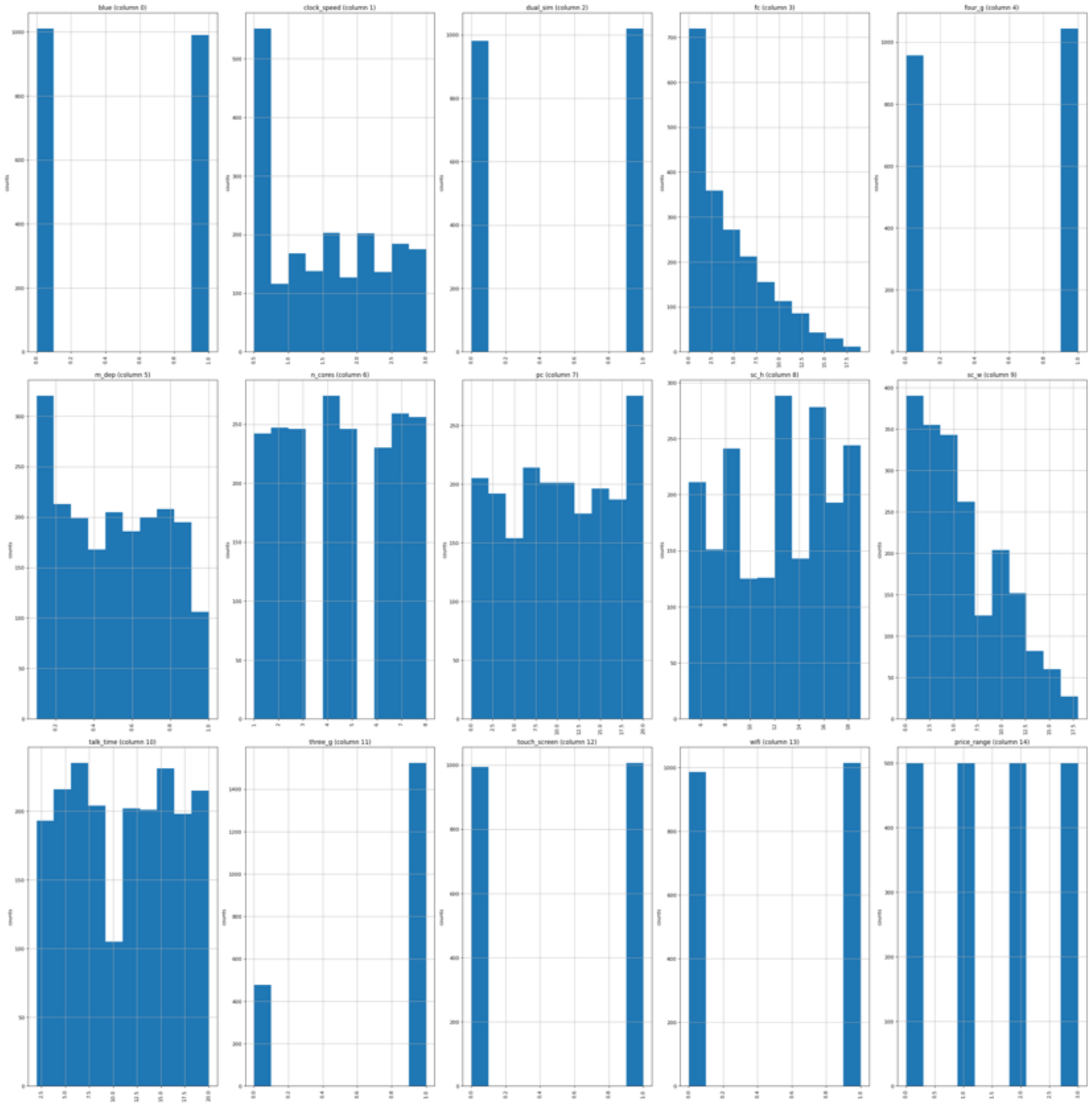
Abbiamo poi contato il numero di telefoni presenti nel dataset di train per ogni fascia di prezzo ed essendo 500 per ogni fascia suddivisi in quattro fasce, abbiamo affermato che il nostro dataset era perfettamente bilanciato.

Spesso per capire più approfonditamente le variabili presenti nel dataset è doveroso eseguire alcune analisi statistiche di base sui dati.
Ho quindi utilizzato la funzione *describe* su entrambi i set di dati, che ha restituito per ogni colonna, il conteggio delle istanze, la media, la deviazione standard, il minimo, il massimo e i percentili venticinquesimo, cinquantesimo e settantacinquesimo.

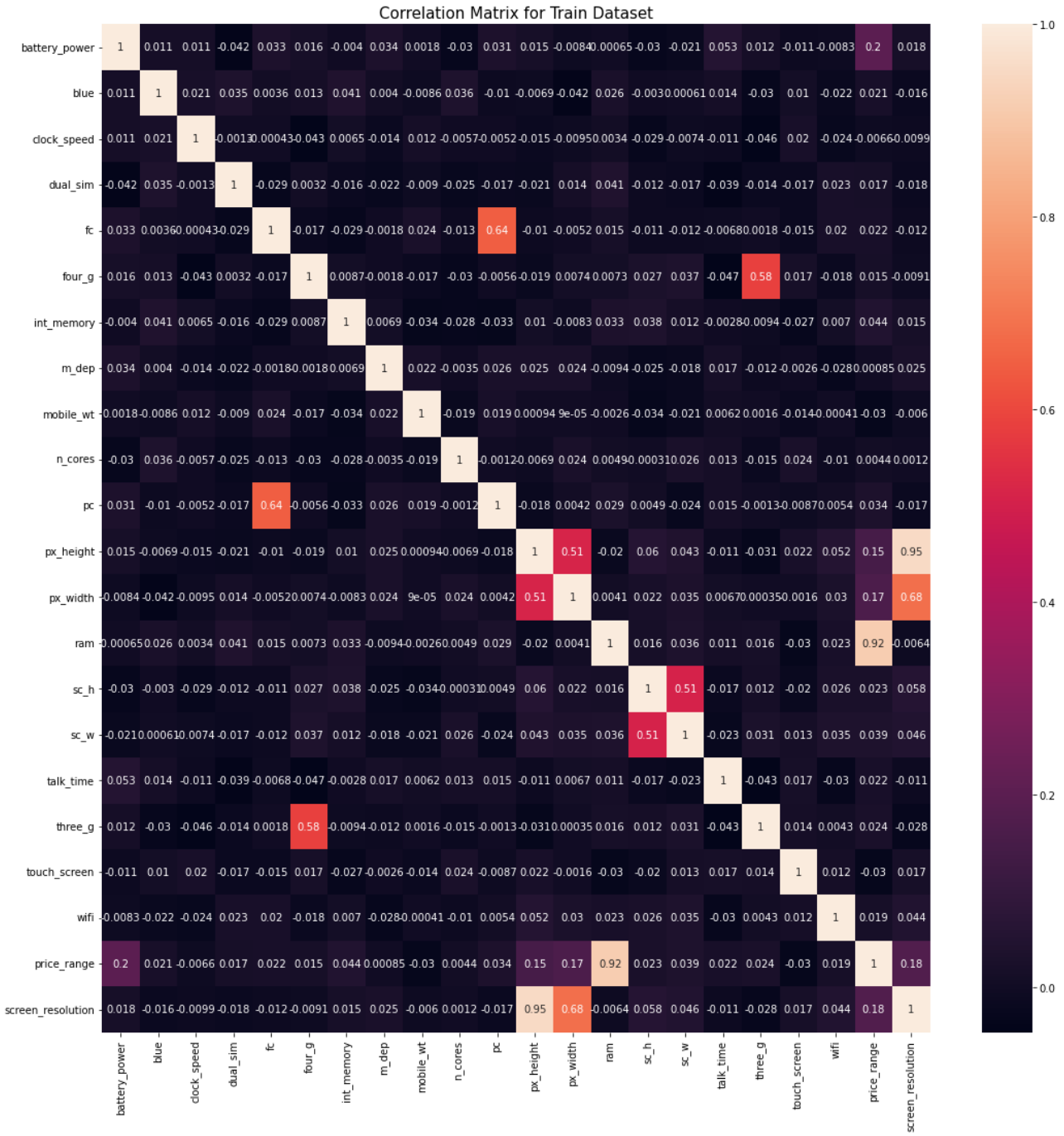
Visualizzare i dati è un ulteriore tecnica efficace per avere un'idea di come sono distribuiti, per fare questo abbiamo utilizzato e modificato a dovere una funzione che ci permette di creare diversi istogrammi per ogni colonna del dataset, inserendo automaticamente le etichette lungo l'asse delle x e delle y, tra cui il titolo e anche i valori minimo e massimo che può assumere ogni variabile.

Per motivi di visualizzazione ho escluso le colonne che contengono più di 50 valori diversi.

Ho pensato poi di fondere due variabili, ovvero *px_height* e *px_width* che sarebbero il numero di pixel in altezza e i pixel in larghezza che se moltiplicati ci restituiscono la risoluzione dello schermo, creando una nuova colonna chiamata *screen_resolution*.

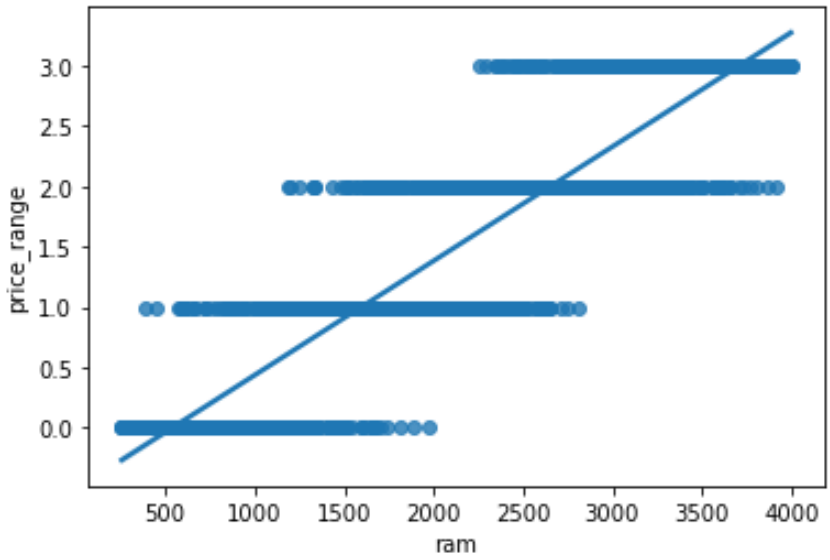


La matrice di correlazione di Pearson è una tabella che indica il coefficiente di coolegamento tra fattori. Ogni cella della tabella mostra la connessione tra due variabili.
Per visualizzare la correlazione tra le varie features e la varibile target ho computato la matrice di correlazione di Pearson sul dataset di training, ed ottenuto la seguente tabella.

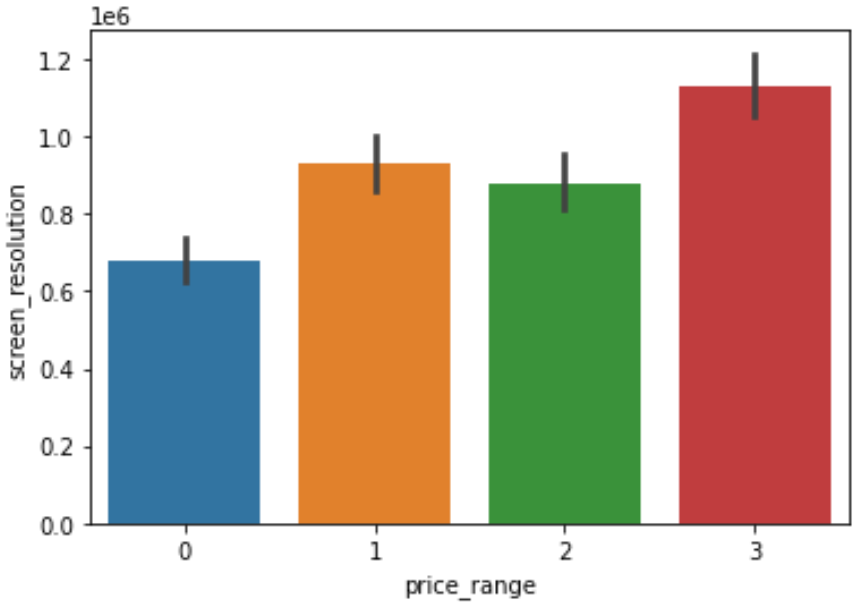


Come possiamo notare vi è un ottima correlazione tra la fascia di prezzo e la dimensione della memoria ram. Con la variabile target vi è anche una certa correlazione con la potenza della batteria e con la risoluzione dello schermo.

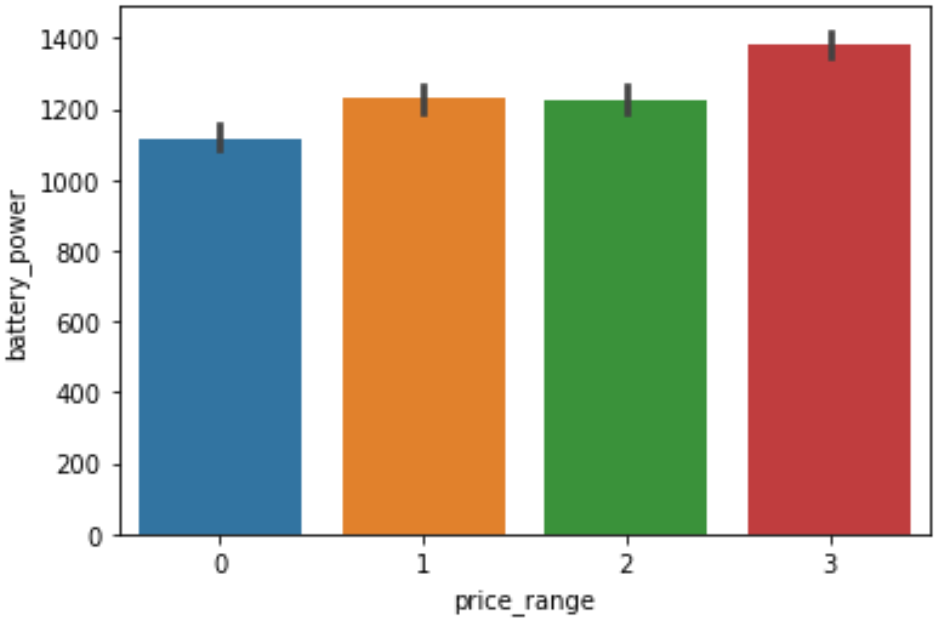
Questa correlazione è visualizzabile anche nel seguente grafico *regplot*:



La correlazione tra prezzo e risoluzione dello schermo è invece meno evidente e non possiamo dire con certezza che vi sia effettivamente una correlazione tra risoluzione dello schermo e prezzo del telefono, in quanto tra fascia alta e fascia media questa relazione è inversa, come vediamo nel seguente grafico a barre:

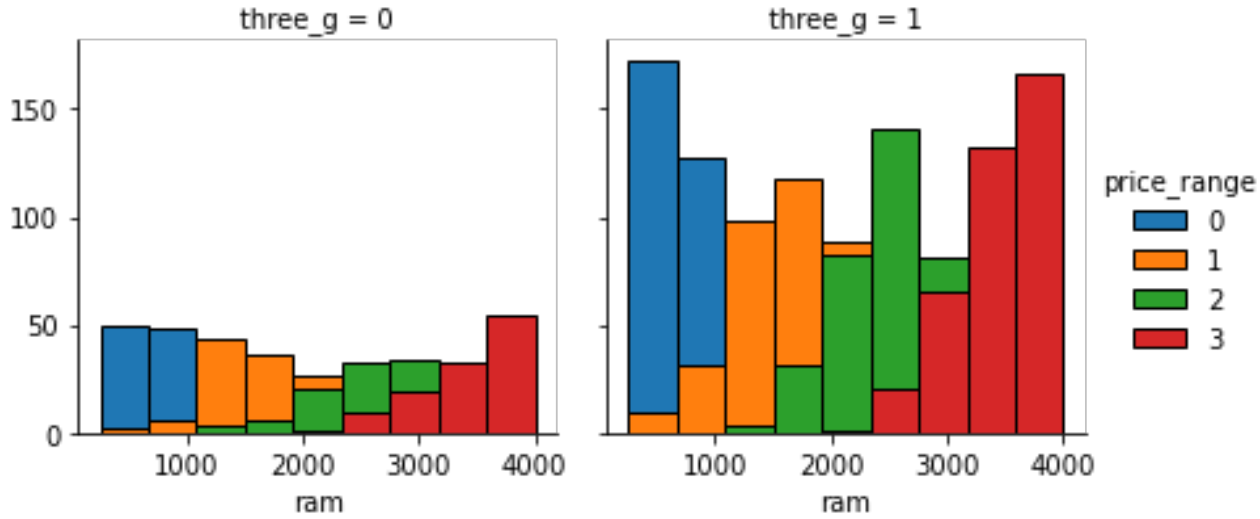


Vi è anche una piccola correlazione tra il prezzo e la capacità della batteria da quanto possiamo notare nel seguente grafico, anche se non è così netta quanto la dimensione della ram.



Abbiamo quindi continuato raccogliendo le variabili categoriche all'interno di una matrice chiamata *categorical_var* e aggiunto anche una colonna *price_range_qualitative* traducendo per chiarezza i valori 0, 1, 2 e 3 del range di prezzo in low, medium, high e very high.

Abbiamo quindi raggruppato le variabili categoriche in base alla fascia di prezzo per evidenziare eventuali correlazioni con il prezzo. Per visualizzarle abbiamo in seguito inserito il tutto in dei grafici *facetgrid*, ponendo sull'asse delle x la ram che, come abbiamo visto, mostra un'ottima correlazione con il prezzo e lungo la y la variabile categorica selezionata. Lo spettro dei colori sarà riferito al range di prezzo.
Nella seguente relazione ho riportato solamente quello del 3G che ha mostrato una correlazione maggiore, i grafici restanti sono consultabili nella cartella images.



Possiamo ora dire di avere abbastanza informazioni sui dati e possiamo proseguire con la parte di pre-processing e valutazione dei modelli per trovare quello che meglio esegue il task richiesto.

Preprocessing and Model Comparison

Per la parte di preprocessing e comparazione dei modelli, abbiamo utilizzato le seguenti librerie e metodi:

- Numpy
- Pandas
- Matplotlib (pyplot)
- Sklearn:
    - feature selection, utilizzando:
        - SelectKBest
        - chi2
    - Preprocessing
    - Model selection
        - train_test_split (per separare il dataset in training/validation e test set)
        - KFold
    - Metrics:
        - Accuracy score
        - F1 score
        - Confusion matrix
    - Linear model (nello specifico la logistic regression)
    - Tree (Utilizzando il Decision Tree Classifier e *plot tree* per graficare gli alberi)
    - Svm (utilizzando SVC)
    - Ensemble, con i seguenti classificatori:
        - Random Forest Classifier
        - Bagging Classifier
        - Adaptive Boosting Classifier
        - Gradient Boosting Classifier
- XGBoost (con l'utilizzo dell'Extreme Gradient Boosting Classifier)
- Tensorflow:
    - Keras:
        - Optimizers
        - Models
        - Utils (nello specifico il metodo to_categorical)
        - Layers

Tutto il codice relativo a questa parte è visualizzabile nel file "*Preprocessing& ModelComparison.py" (o .ipynb)*

All'interno del file ho creato alcune funzioni per ridurre le ripetizioni nel codice.
La prima funzione che ho creato, si chiama *score* e ci permette di usare tre metriche, quelle che ho ritenuto più efficaci, per valutare la bontà predittiva dei nostri modelli. La prima metrica è l'accuracy score che è definito come il rapporto tra il numero di predizioni corrette e il numero totale di predizioni, poi ho scelto l'F1 score, che è la media armonica tra la precision e il recall e infine la matrice di confusione che ci permette di visualizzare lungo la diagonale della matrice il numero di predizioni corrette e nelle altre celle le predizioni sbagliate.
La funzione prende in ingresso la colonna delle y di test, la colonna delle y predette e la specifica average scelta per l'F1 score e ritorna in uscita i due valori di affidabilità della predizione e la matrice di confusione.

La seconda funzione utilizzata è la KFoldCrossValidation che prende in ingresso il modello su cui eseguire il training, il dataset X e i valori delle y da predire, crea quindi uno split in 10 fold del dataset dividendolo in train e validation set ed esegue l'allenamento sui 10 diversi dataset. Questo approccio l'ho ritenuto necessario dato che le dimensioni del nostro dataset erano abbastanza ridotte, soprattutto per minimizzare la varianza tra i risultati. Ad ogni iterazione vengono anche calcolate le capacità predittive con la funzione score e salvate in tre liste. Terminato poi l'allenamento viene fatta la media dei valori contenuti nelle liste e viene riportato il valore medio delle varie predizioni.

L'ultima funzione che ho creato in questo file si chiama findBestFeatures e permette prendendo in ingresso il modello con cui eseguire la predizione, l'X di training e la y di training di trovare il numero di features che permettono la maggior accuratezza nella predizione e successivamente ritorna un grafico con una retta che interseca l'accuratezza (asse delle y) e il numero di variabili utilizzate (asse delle x).

Dopo aver creato queste funzioni, ho ripreso i dataset e li ho elaborati come fatto precedentemente nella fase di data analysis e ho cercato tramite l'utilizzo della funzione SelectKBest (usando il chi quadro come funzione di score e un k=2) le migliori features del nostro dataset e le ho riportate in una tabella.

Ho quindi eseguito una fase di preprocessing, in cui ho prima salvato il dataset di training *X* in uno nuovo chiamato *X_NoPrep,* quest'ultimo sarà il mio dataset senza pre-elaborazione da utilizzare per verificare l'efficacia delle mie strategie di pre-processing.

Ho quindi proceduto applicando al dataset *X* uno standard scaler che standardizza le features rimuovendo la media e scalandole a varianza unitaria.
Quindi ho creato una *X_norm* che prende il dataset di training *X* appena modificato e gli applica un MinMaxScaler che permette di normalizzare tutti i valori del dataset (per eseguire valutazioni con la chi quadro) e creato un ulteriore *X_new* che tramite la funzione SelectKbest con k = 5 e chi quadro come score function mi ha generato un nuovo dataset ridotto, di sole 5 features.

Successivamente ho creato tramite la funzione *train test split* un dataset di training, chiamato *X_train*, un dataset su cui eseguire la predizione, chiamato *X_validation,* il vettore colonna dei valori corretti di train, chiamato *y_train* e infine il vettore colonna dei risultati corretti per il dataset di validazione chiamato *y_validation.*

Lo stesso processo di train test split è stato applicato per la X_norm, creando gli stessi dataset ma con la dicitura *X_train**NoP*** ad indicare che non vi è stato applicato nessun preprocessing.

In entrambi i processi la dimensione dei datasets e dei vettori di validazione è stata impostata al 10% del totale con il parametro *stratify=y.*

Il passaggio successivo è quello che ha richiesto la maggior quantità di tempo di sviluppo del progetto, ovvero la parte di comparazione dei modelli e di fine tuning degli stessi, per ottenere le massime prestazioni possibili.

Sono stati scelti alcuni algoritmi di machine learning utilizzati comunemente nei task di classificazione che ho allenato con la funzione *kFoldCrossValidation* precedentemente presentata. Tutti i risultati raccolti sono visualizzabili nel file "*Models Comparison.txt*"

Come primo modello ho utilizzato la Regressione Logistica, che da subito ci ha dato risultati ottimi con un'accuratezza del 96,2%, un F1 score del 96,2% e soli 6,6 (media su 10 fold) elementi non classificati correttamente, il tutto con tempi di elaborazione di pochi decimi di secondo.

Come secondo modello ho utilizzato gli alberi decisionali, con massima profondità impostata a livello sei per evitare il fenomeno dell'overfitting. Purtroppo, non si sono rivelati dei predittori perfetti, infatti l'accuratezza si è fermata all'84% che è un buon risultato ma nettamente inferiore a quello della Regressione Logistica, impiegando circa un secondo e mezzo.

Un risultato migliore è stato restituito dal classificatore Random Forest con un buon risultato di accuratezza all'88,7%, ottenuto utilizzando 200 stimatori, con un tempo di training di circa 4 secondi.

Dopo la Regressione Logistica, il miglior classificatore in termini di efficienza è stata la Support Vector Machine Classifier o SVC, con un risultato di accuratezza del 95%, ottenuto allenando il classificatore sul dataset senza preprocessing, in un tempo record di 0.2 secondi.

Altri classificatori come il Bagging Classifier hanno restituito risultati di accuratezza dell'88,8% ma con un tempo di elaborazione molto elevato (circa 10 secondi).

L'Adaptive Boosting invece oltre ad impiegare in media più di otto secondi per l'elaborazione ha offerto un risultato inferiore rispetto a tutti gli altri classificatori, cioè circa l'82% di accuratezza, ma questo risultato è stato possibile solo con un fine tuning, che ha permesso di ridurre l'overfitting eseguendo il training su un dataset molto ridotto, di sole 3 features. Senza eseguire fine tuning con 20 features non ha mai superato il 65% di accuratezza, impiegando circa 2.5 secondi con fine tuning e 4 senza.

Buoni risultati sono stati restituiti con il classificatore Gradient Boosting, che utilizzando 200 stimatori, ha restituito un'accuracy del 91,2%, il tempo di elaborazione è stato di circa 20-25 secondi.

Risultati di poco migliori sono stati raggiunti dall'Extreme Gradient Boosting, utilizzando sempre 200 stimatori e come metrica di valutazione la *logloss,* con un'accuratezza del 91,5% e un tempo di calcolo di 14 secondi.

L'ultimo modello che ho utilizzato è stata una rete neurale, che ha restituito certamente delle ottime performance di accuratezza, circa il 96,5% nel miglior risultato ottenuto, con però dei significativi ostacoli legati soprattutto alla lentezza nell'allenamento della rete (solitamente si è attestato attorno ai 18 secondi abbondanti).
La parte di fine tuning dei parametri della rete neurale e di selezione delle impostazioni di base, si è rilevata anche la più dispendiosa a livello di tempo, rispetto a tutti gli altri classificatori.
Innanzitutto, è stato necessario settare y variabile categorica, creare un *train test split* differente dai precedenti, utilizzando il dataset con le 5 migliori features ed un test set di dimensione 10% rispetto al dataset iniziale.

Il modello scelto per il task è di tipo sequenziale e gli strati sono stati tutti settati come densi. Il primo contiene 18 neuroni, come funzione di attivazione utilizza una relu e prende un input di dimensione pari a 5.
Il secondo ha sempre 18 neuroni e utilizza la funzione relu.
L'ultimo contiene quattro neuroni (pari al numero di classi che vogliamo predire) e usa come funzione di attivazione la *Softmax*.
Abbiamo inoltre fatto ricorso all'ottimizzatore *Adam.*
Utilizzando la funzione *compile*, abbiamo impostato l'ottimizzatore per il nostro modello, come metrica abbiamo fatto restituire l'accuracy per ogni epoca di training, e come funzione di loss, la più efficiente tra quelle che ho valutato è stata la categorical crossentropy.
Abbiamo quindi eseguito il fitting del nostro modello, utilizzando almeno 100 epoche per ottenere buone performance predittive, con dimensioni del *batch* pari a 64 e un *validation split* pari al 20% del dataset di training.
Ho quindi eseguito la valutazione del modello con i dataset di validazione, salvando i valori del test loss e test prediction di ogni epoca, per poi computare la predizione rispetto al dataset di validazione e rappresentare in un grafico l'andamento della loss del training e quella delle values, per permetterci quindi di eseguire più efficacemente il fine tuning del modello.

La procedura è risultata quindi lenta e macchinosa e nonostante le ottime performance predittive, la rete neurale non è stata quella che ho ritenuto la soluzione più efficiente per la risoluzione del nostro task.

Final Model Evaluation e Conclusioni

L'ultima fase del progetto riguarda la scelta del modello finale, ovvero quello che a mio parere è parso il più efficiente nel risolvere il seguente task di machine learning.
Il codice riguardante questa parte si trova nel file FinalModel.py (o .ipynb) e sono state utilizzate le seguenti librerie e metodi:

- Numpy
- Pandas
- Matplotlib (pyplot)
- Sklearn:
    - feature selection, utilizzando:
        - SelectKBest
        - chi2
    - Preprocessing
    - Model selection

        - train_test_split
        - KFold
    - Metrics:
        - Accuracy score
        - F1 score
        - Confusion matrix
    - Linear model (utilizzando la logistic regression)

Ho riscritto le funzioni create precedentemente: *score*, *findBestFeatures*, *kFoldCrossValidation*.
Ho importato i dataset di training e test, rimuovendo dal dataset di test la colonna *id* e la colonna *price_range* da quello di training, impostandola come vettore colonna y dei valori da predire.

Successivamente ho applicato preprocessing sia al dataset di training che a quello di test ed infine utilizzato la Regressione Logistica per predire la fascia di prezzo dei cellulari di Bob.
La mia scelta per questo specifico classificatore è stata motivata dal fatto che questo modello prediceva, come già visto nel file precedente, con grande accuratezza il dataset di training (96,5%), impiegando inoltre un tempo minimo (da 0.1 a 0.3 secondi).

Final results using the full training and test set:
Accuracy score: 0.97,
F1 Score:0.969877450980392,
Confusion Matrix:
[[49 1 0 0]
[1 49 0 0]
[0 2 46 2]
[0 0 0 50]]

Abbiamo infine eseguito la predizione del dataset di test, predicendo quindi il range di prezzo di ogni telefono con un'accuratezza del 97% e soli 6 telefoni ogni 100 classificati in modo errato.

In conclusione, possiamo quindi affermare che con le variabili contenute in questo dataset, un modello di classificazione relativamente semplice come la regressione logistica, offre ottimi risultati e un'accuratezza nella predizione molto alta, con tempi di elaborazione estremamente ridotti. Le sue performance si sono rilevate in questo specifico caso migliori di molti altri classificatori ben più complessi e che richiedono un tempo esponenzialmente maggiore per la fase di allenamento.