Relazione

January 23, 2019

Table of Contents

1 Progetto Google-Revenue Kaggle

---

corso: web-intelligence
autore: bernardi riccardo
matricola: 864018

2 Glossario

- lightgbm = in alcuni casi indica l' albero di regressione, è esplicito rispetto al contesto
- features = colonne o dimensioni del dataset
- transactionRevenue = target deprecato, ora predittore
- totalTransactionRevenue = target odierno
- sampling = mescolamento
- training = addestramento di un regressore
- EDA = analisi esplorativa dei dati, prima analisi
- regressore = predittore
- LB = leaderboard di kaggle per questa competizione
- RMSE = errore nella predizione/validazione
- score = minimo/basso RMSE, minimo errore, posizione alta in LB

3 Asserzioni sui dati

Una volta osservati i dati si può notare che ci sono molte colonne che non sono utilizzabili, queste sono quelle che sono presenti solo in uno dei due set, in tal caso non servono perchè un modello trainato su quelle features non verrà sfruttato nel test visto che non sono presenti lì e viceversa non saranno utili perchè non possono essere trainate. Altre colonne non usabili sono quelle

diverse dall' ID che però sono univoche, non aggiungono informazioni poichè sono aleatorie e non vi si può dedurre un pattern. Le colonne che hanno un solo valore inoltre sono ugualmente da scartare poichè hanno una colonna costante che non aggiunge informazione poichè tutte le righe lo possiedono. Si vuole poi fare una importante osservazione sugli utenti e cioè che un utente avente più entries non sempre abbia un comportamento coerente. Si risolverà il comportamento incoerente degli utenti mediando i valori numerici poichè diminuisco la varianza e poi si prenderà la moda dei valori categoriali. Questo perchè fare la media di dati categoriali non permetterebbe poi di avere una funzione inversa che mappa un reale su dei numeri interi. Ha senso prendere l' elemento più ripetuto poichè in tal caso con un semplice dict poi si può mappare dall' intero alla categoria. Si assumerà perciò che un utente possa avere a volte uno smartphone a volte un altro(per esempio). Per migliorare quindi, dopo queste assunzioni, lo score si è deciso di droppare le colonne costanti, con valori unici e le colonne che non siano presenti in entrambi i datasets a parte la colonna del target. I dati non presenti vengono assunti a 0.0 poichè è coerente e consistente che sia così, i customers che hanno nan in una colonna significa che un certo campo numerico ha valore 0, in un campo categoriale invece mettiamo 0 per comodità e poi verrà valutato attraverso la moda quindi se è un valore outlier verrà mascherato dalla robustezza di questo funzionale statistico.

4 Data Load and Parsing

4.1 Introduzione

La parte di data loading ma soprattutto parsing dopo l' aggiornamento del dataset, avvenuto per un data leakage, è diventata una parte core di questa analisi poichè l' encoding delle informazioni è diventato sempre più difficoltoso a causa di una maggiore densità e innestatezza dei dati. Si è iniziato osservando che il training set conteneva json da normalizzare e parsare come nel precedente dataset ma in aggiunta erano comparse due nuove mastodontiche colonne che comprendevano liste di jsons(in realtà liste di dicts che potevano contenere altre liste e altri dicts, ma ad uno sguardo iniziale si poteva incappare nell' errore di vederli appunto come jsons). Il solo unfolding di queste ultime 2 colonne ha richiesto molta struttura e codice(~200 righe, in python!) poichè non era assolutamente pensabile nè di droppare tutte le colonne "dense" nè di unfoldarle a mano, si è perciò optato per costruire delle funzioni che facessero auto-discovering di liste o dicts all interno delle colonne ma in maniera ricorsiva poichè una volta aperta una colonna e aggiunta a destra bisognava ri-discoverare nell' intero dataset se era possibile continuare ad unfoldare liste e/o dicts(in realtà non è implementato ricorsivamente a causa del fatto che python non regge bene questo paradigma, vedremo che vengono usati cicli while e vengono posti vincoli sull' unfolding). Una volta costruite le funzioni di auto-discovering si è però notato che mancava qualcosa: alcune colonne venivano riconosciute come lists e/o dicts ma altre no... Il problema è stato fixato usando 2 try...catch... innestati che discoverano le colonne controllando il tipo con eval() oppure senza poichè si è notato che una volta unfoldato un dict viene rilasciato il suo valore come stringa e non come dict(nel caso di dicts innestati con anche liste innestate) e poi il problema si ripropone con liste innestate. Il problema non era risolvibile chiedendo il semplice dtypes poichè tutte le colonne più complesse di un int64 in numpy sono catalogate come object... per quanto invece riguarda chiederne il type non è sufficiente perchè appunto se è una lista encodata come stringa non verrà unfoldata quindi qui si capisce il senso del try...catch... cioè so che se chiedo l' eval di

una lista otterrò un errore e consapevolmente lo catturo poi se è una lista/dict la unfolderà, se invece è altro passerà oltre.

4.2 Unpacking e Unfolding

A questo punto si vuole introdurre la differenza tra unpacking e unfolding(nell' introduzione per semplicità non è stata trattata): nel caso di una colonna che con l auto-discovering si è scoperto contenere una lista di dicts allora si preferirà avere un solo dicts che come value contiene un array di tutti i valori che prima erano contenuti in singoli dicts con un value cioè : $dict[key- > value]_1, ..., dict[key- > value]_n - > dict[key- > value_1, ..., value_n] \forall\ key\ \in dict \in list$. Il senso di fare ciò è poi avere una interrrogazione molto molto più semplice poichè per esempio se si vuole estrarre la media del costo dei singolioggetti comprati da un utente in una sessione basterà fare $mean[dict[cost]]$ piuttosto che dover ciclare e cercare gli elementi con il rischio di vedersi throwate eccezioni se un dict solo non contiene l' elemento ricercato. Quindi l' unpacking è proprio questo, si ha la traduzione da una lista di dizionari ad un solo dizionario. L unfolding invece è un operazione simile poichè si tratta sempre della trasformazione con un aumento dell' espressività dei dati, in particolare la unfold non tratta una sola colonna come la unpack ma tratta un set di colonne che contengono solo dicts e le trasforma in opportunamente usando una ri-implementazione del costrutto funzionale map() per normalizzarle, trasformarle in un dataframe (ogni colonna) e poi aggiungerle al dataframe di partenza avendo cura di non lasciare dati obsoleti indietro(droppando le colonne vecchie e lasciando le nuove), tutto ciò avviene in maniera molto molto efficiente poichè si usano sempre colonne di dataframe che vengono modificate in maniera parallela e non si passa mai nulla per copia e non si scende mai di livello presso strutture dati a prestazioni peggiori, in questa maniera la modifica e il merge delle colonne avviene in tempo praticamente costante perchè si sfrutta il passaggio per reference delle colonne di un dataframe, il risultato è che una volta caricato dall' HD il dataframe poi si ha un ambiente che permette tante operazioni in tempo quasi nullo.

4.3 Conclusioni

Vengono aggiunti vari controlli , caricate le colonne che sono effettivamente jsons e inoltre vengono droppate colonne uniche e colonne costanti, il tutto per dare all utente finale una sola ed unica funzione che permette di essere chiamata con pochi parametri e fa già di per sè un lavoro mastodontico di prima pulizia dei dati ma con eleganza perchè lo fa in maniera trasparente.

5 Preprocessing

5.1 Introduzione

Dopo una prima pulizia dei dati viene la parte più importante cioè la strategia per rendere i dati il più possibile espressivi attraverso opportune codifiche. Sceglieremo, non essendoci dati ordinali, di encodare tramite one-hot-encoding i dati categoriali. Questa operazione è molto costosa in termini computazionali e spaziali poichè ogni inefficienza ha un costo esponenziale, per esempio inizialmente si facevano più dataframes che poi venivano ri-arrangiati droppando colonne fino al raggiungimento delle colonne ricercate mentre ora si punta a fare un merge tra il dataframe iniziale, il nuovo dataframe dei dati categoriali categoriali encodato e poi droppando le colonne

encodate, questa operazione avviene ovviamente in tempo costante perchè è solo il costo di mettere un poiter dall' ultima colonna di un dataframe che punta alla prima colonna(leggasi indirizzo di memoria in 0) del secondo dataframe. Si fa notare quindi l' utilizzo efficiente dei dataframes. Il vantaggio infatti dei dataframe pandas è la possibilità di essere interrogati con colonne, indici, array di colonne, interrogazioni simil-sql e la grande efficienza con la quale si possono unire più dataframes orizzontalmente o verticalmente in tempo praticamente costante poichè in realtà si lavora a basso livello con pointers. Si fa notare inoltre l' utilizzo di pd.Series invece che di list poichè dopo attenti tests si è notato essere molto più efficienti. L' utilizzo poi delle matrici numpy è motivato dall' efficienza dei calcoli e la semplicità con la quale si può passare una matrice ad un dataframe, questo passaggio è piccolo ma lo si può notare direttamente nell' encoding delle colonne, è una parte minuscola ma cruciale e averla implementata così è stato importantissimo per ottenere efficienza. Una volta finito l' encoding si passa ad un altra fase anch' essa molto importante cioè vengono uniti gli utenti che hanno più istanze. Si decide di trasformare il dataframe iniziale in 2 diversi dataframes in cui si fa rispettivamente la media dei valori numerici e la moda dei valori categoriali encodati. In tutto ciò, che potrebbe sembrare un operazione trasparente in realtà(ed effettivamente dal lato utente lo è) ci sono diversi problemi di corruzione dei dati dovuti anche al fatto che viene modificato l' indice dei due dataframes. Perdendo l' indice originale anticipo già che si avrebbero problemi nelle successive parti del codice che lavorano appunto con l' indice numerico predefinito perciò si è preparata una procedura generica per resettarlo in maniera efficace. A questo punto is aggiungevano colonne che si pensa possano essere significative come giorno/mese/anno/hr/weekday nel vecchio dataset ma nel nuovo in realtà non lo si fa poichè molto di questi dati vengono già dati gratuitamente. Vedremo alla fine dell' analisi che effettivamente risulteranno utili ma questo lo si può immaginare anche a monte poichè dei consumatori possono seguire un pattern per scegliere il giorno in cui comprare(il giorno successivo al giorno di paga). A questo punto il dataset viene diviso in customers che hanno speso e quelli che non lo hanno fatto poichè poi dividendo in dev e val poi vogliamo mantenere la proporzione.

5.2 Funzione encode_cats

Questa funzione rappresenta il cuore dell' encoding e la sua forza, come in tutte le altre funzioni di questo notebook, è fornire un interfaccia generica, potente, efficinete e trasparente. Da notare infatti il parametro target che permette di definire parametricamente le colonne che *non* si vuole modificare. Da notare inoltre che la funzione prende il train e il test e li encoda in contemporanea per mantenerne la coerenza : $(train[cols] \cap test[cols]) - targetValue = \emptyset$ dove targetValue è il valore che vogliamo predire alla fine. La funzione encoda correttamente i valori ma non soltanto... dopo l' encoding si perdono i nomi delle colonne a causa della trasformazione in numpy.matrix ma non è detto che la prima colonna del test encoded sia la stessa del train encoded, sembra complicato ma il fatto è che un volta che spacchetto una colonna e la faccio diventare 50 colonne se quella colonna conteneva un solo valore diverso nel test bè allora avrò colonne diverse e magari pure un ordine diverso nelle colonne, ricordandomi anche che non so più i nomi delle nuove colonne. Il problema quindi viene risolto droppando le colonne diverse tra i due nuovi datasets encodati e sfruttando il fatto che i nomi delle colonne possono essere restituiti dall' oggetto che ha prodotto l' encoding(non è così semplice perchè anche i nomi sono una matrice quindi si passa per un flattening intermedio).

5.3 Funzione group_me

Questa seconda funzione viene subito dopo l' encoding e punta a rendere i dati, già ben preparati, più parlanti rispetto al singolo utente che è il nostro target. Si decide di fare la media dei dati numerici poichè non abbiamo un encoding su quelli ma abbiamo solo bisogno di un range in cui farli variare e si prende la moda dei dati categoriali, cioè si prende per ogni utente il valore che appare più volte poichè non servirebbero a nulla dei valori coninui se stiamo usando valori discreti: non verrebbero mappati! Il calcolo della moda inoltre è un punto dolente dell' intero processo poichè richiede più tempo del previsto(infatti si penserà ad un ottimizzazione o ad una sostituzione con un funzionale simile ma più efficiente come la media che purtroppo restituirà valori continui ma questo fortunatamente viene sopportato da un albero di regressione) questo perchè richiede di aggregare tutti i dati e calcolare la frequenza di ogni valore per poi scegliere il valore con più ripetizioni.

5.4 Conclusioni

L' ultima parte di meno importanza ma comunque necessaria è la divisione del dataset in dev e val del train per poter allenare un regressore lineare e un albero di regressione, qui inoltre si fa stratified-sampling dividendo il dataset per spesa cioè i compratori vengono divisi dai visitatori non compratori in maniera da prendere la stessa percentuale da tutti e due i gruppi per poter mantenere il rapporto dell' 1% di cui parla la legge di Pareto. In questa sezione vengono fatte poi operazioni minime di preparazione del dataset scegliendo le colonne di training, la percentuale di righe per training e test e poco altro.

6 Training e Scoring

6.1 Introduzione

Sono stati fatti vari tentativi per ottenere un buon punteggio e si è iniziato prevedendo la media della spesa degli utenti, poi si è passati a prevedere con una regressione lineare che però correttamente otteneva lo stesso score di un file zero-filled e alla fine si è compreso che la scelta migliore trattandosi di un task di regressione che poteva però contare su tanti dati e di diversa natura e non solo numerici sarebbe stato usare alberi di regressione. Si è inoltre tentato di cercare clusters attraverso i dati ma si è notato che comparando il revenue con la feature hits(che ha massima correlazione) si ottiene un insieme globulare di punti nello zero(disponibili le immagini pubblicamente sul mio account plotly "riccardobernardi"). Inoltre trattandosi di un task di regressione l' idea di fare clustering è stata abbandonata poichè non abbastanza orientata al risultato(erano già state fatte molte EDA su kaggle). Il training avviene attraverso un framework molto diffuso nelle competizioni di ML chiamato LightGBM, viene scelto questo framework perchè ben documentato, potente, coerente coi tipi di dati e mantenuto da un ente che punta su questo framework. Il training avviene (per ora) solo attraverso un dataset di dev cioè di training e di val cioè di validation (ma non si esclude di aggiungere una k-fold validation oppure di usare cv già incluso nell' engine di LGBM) che vengono estratti dal training iniziale mantenendo la proporzione dell' 1% dei compratori. Questo perchè ci potrebbe essere un alto rischio di mandare in val solo zeri e di avere un modello che predice solo zeri. Tutto ciò si chiama stratified sampling.

6.2 Lin-reg

Per poter dare dimostrazione delle scelte fatte si vuole dare la possibilità di testare una predizione fatta con LinReg piuttosto che con un albero di regressione. La regressione lineare è stata scelta perchè intrinsecamente il task è di regressione cioè richiede che la predizione sia un valore continuo. Alla fine la regressione sarà solo usata come cartina tornasole/benchmark per verificare che l' albero di regressione superi la regressione lineare, il che è molto verosimile. La regressione lineare è implementata usando la libreria scikit-learn. Alla fine comunque si potrà notare che la regressione lineare funzionerà egregiamente, tanto da superare addirittura gli alberi di regressione per casi speciali. Si fa però notare che questo accade solo nel nuovo dataset mentre invece nel vecchio dataset l' albero di regressione non aveva pari.

6.3 LightGBM

La conoscenza viene alla fine estratta tramite questa libreria sviluappata da microsoft che implementa alberi di regressione cioè alberi che dividono in maniera da aumentare l' information gain o diminuire l' impurità, ecco di seguito le possibili implementazioni spiegate:

Gini Index:

Si calcola l' impurità dei dati in ingresso

$$Gini(D) = 1 - \sum (p_i)^2$$

Si calcola l' impurità per ogni valore A della feature presa in considerazione e poi si ripeterà per ogni feature nel dataset di training. Viene simulato lo split binario e si calcola se sia conveniente per ogni feature così da scegliere la feature che migliora l' impurità.

$$Gini_A(D) = \frac{|D_1|}{|D|} \cdot Gini(D_1) + \frac{|D_2|}{|D|} \cdot Gini(D_2)$$

Si calcola poi l' impurità dello split simulato rispettivamente al non-split:

$$Gini(D) - Gini_A(D_j)$$

E questo quindi è un valore da rendere minimo per scegliere il miglior split per creare un nodo che sia localmente ottimo e cioè che abbia buone probabilità di creare una foglia pura.

oppure

Information Gain di Shannon

Si calcola l' entropia per il dataset non splittato

$$Info(D) = -\sum (p_i) \cdot log_2(p_i)$$

poi si calcola l' entropia per ogni valore A della feature presa in considerazione e come prima poi si vorrà ripeterlo per ogni feature per poi decidere quale feature determina il migliore split

$$Info_A(D) = \frac{|D_j|}{|D|} \cdot Info(D_j)$$

Si calcola poi il gain rispetto al non-split del dataset

$$Gain(A) = Info(D) - Info_A(D)$$

Ed in questo caso sarà un valore da massimizzare perchè diversamente dal gini poichè qui abbiamo il gain e non l' impurità

inoltre

Gain Ratio:

$$SplitInfo_A(D) = -\sum \frac{|D_j|}{|D|} \cdot log_2(\frac{|D_j|}{|D|})$$

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo_A(D)}$$

Il vantaggio di usare un albero di regressione è che i dati vengono divisi in sotto-nodi a sx o dx rispetto alle formule qui sopra per semplicemente rendere minimo l' rmse, in questa maniera saremo sicuri che dopo un certo numero di iterazioni avremo una granularità molto fine nella scelta dei valori di previsione. E' infatti dimostrato che continuando a dividere in maniera iterativa si otterrà il miglior valore possibile nonostante sia una scelta greedy. La varianza dei dati viene migliorata per esempio mediando i valori tra le foglie o tra i fari alberi nel caso di una random forest. Per quanto invece il bias esso viene ridotto usando il metodo del boosting cioè re-samplando i dati più rari per ottenere un migliore train su di essi. Bisogna inoltre guardarsi dall' overfitting contro il quale si è più volte andati incontro con questo dataset ma lo si è evitato utilizzando il metodo del bagging e usando random forests.

6.4 Conclusioni

Si sono implementati metodi per il tracking e il monitoring dei risultati su file csv in maniera da poter decidere i metodi migliori da utilizzare dopo un po di tentativi. Si è inoltre prodotto codice che tenta iterativamente tutte le configurazioni ragionevoli in un certo range di valori per poter plottare un grafico che indichi qual' è il miglior valore da scegliere per una certa feature

7 Scelte

Durante lo svolgimento del progetto sono state fatte scelte che si desidera puntualizzare qui

7.1 (No) EDA

Si sceglie di non dare nella versione finale nessuno sketch di EDA per due motivi : il primo è che in realtà l' EDA è stato fatto ma quando il codice è diventato preponderante non era più possibile runnare EDA e codice nello stesso file perchè la grafica provoca rallentamenti e rendeva difficile il debug visivamente, l' EDA quindi è disponibile in parte presso il mio account di plotly pubblicamente ("riccardobernardi") e in maggior parte nelle versioni precedenti di questo kernel e nel file "extra.py" in questa cartella. Il secondo motivo è che ad un certo punto dello sviluppo si è ritenuto che l' EDA fosse troppo semplicistico e non fosse abbastanza fare analisi colonna per colonna trovandosi a lavorare con centinaia di colonne. Si fa notare che comunque tutti gli EDA sufficienti a farsi un ottima idea del dataset sono disponibili pubblicamente sui kernel di kaggle di questa competizione.

7.2 transactionRevenue e totalTransactionRevenue

Si è deciso in di seguire i consigli del docente nel togliere il transactionRevenue e prevedere il totalTransactionRevenue senza quel dato, si vuole puntualizzare che in questa maniera il risultato risulta ben peggiore rispetto alla LB di Kaggle in cui si ha ottenuto un ottimo score di **0.1227** per questo motivo si lascia la possbilità di attivare o meno la funzionalità che elimina transactionRevenue all inizio del kernel kaggle-v6.0 per avere l' ebbrezza di un punteggio molto molto alto(rmse molto basso). Detto ciò si è assolutamente d' accordo sul fatto che eliminare transactionRevenue sia la cosa più corretta da fare poichè non si può contare sul fatto che poi col nuovo test sia ancora una colonna disponibile(transactionRevenue è molto predittiva per totalTransactionRevenue e lo si può vedere dal grafico della feat importance alla fine del kernel principale)

EDIT:Da un recente comunicato di Kaggle hanno dichiarato il campo transactionRevenue deprecato

7.3 Inefficienza della moda

Si prevede di fare una funzione meno sematicamente significativa della moda come la media o una approssimazione della moda per raggruppare i valori categorici di ogni utente a seguito del fatto che allo stato attuale rappresenta il collo di bottiglia di tutto il progetto in termini di tempo, persino meno performante del OneHotEncoding.

7.4 Rischio di overfit con productSKU

Si è deciso di non utilizzare i codici prodotti per migliorare le previsioni poichè l' elaborazione risulterebbe complessa a fronte di un possibile overfit ma non si esclude di poter trovare una maniera di estrapolare conoscenza dagli stessi in altre maniere con magari delle euristiche sulle tipologie di prodotti

7.5 Molte feature sintetiche vs poche features reali

Si è scelto, ascoltando i consigli del docente, ad un certo punto dell' analisi del dataset, di creare features sintetiche piuttosto che di analizzare singolarmente la moltitudine di dati presenti. Si è fatto ciò poichè in questo corso si vuole privilegiare il fatto che i pattern sui dati escano spontaneamente e non si hanno sempre delle asserzioni sulle distribuzioni dei dati.

7.6 Cross-validation

Si è scelto(per ora ma non si esclude di aggiungerlo) di non aggiungere la cross-validazione/k-fold-validazione dei dati puntando maggiormente sullo sviluppo di features più importanti poichè già di per sè l' elaborazione del dataset è piuttosto pesante e i dati non mancano. Avendo tanti dati si può pensare che basti solo prendere qualche decina di migliaia di righe in più per ottenere un migliore score.

8 Features aggiuntive

Si vuole ora dare una breve lista sulle features disponibili in questa toolbox:

- si può fare un rapido ed efficace tuning dei parametri modificandoli all' inizio del notebook, il vantaggio di farlo lì è oltre all' accentramento di tutti i parametri più importanti anche il fatto che quei parametri verranno dumpati insieme allo score ottenuto su un file e letti dalla tabella qui sotto per avere subito idea del miglioramento o meno
- utilizzo di pandas
- è disponibile una progressbar che viene già usata ma può essere istanziata in qualsiasi momento anche lato utente
- è disponibile una funzione chiamata cc(parameters) che è già presente in parti del codice e in maniera silente performa una forte verifica sulla correttezza e consistenza del dataset, questo permette di ottenere subito un errore appena una funzione che modifica il dataset crea inconsistenze, in tale maniera si eviteranno errori a cascata
- viene utilizzata l' API di kaggle per submittare perciò non si fa mai la fatica di muovere il mouse
- tutte le variabili che regolano il notebook sono in una dashboard iniziale contenuti in un dict chiamato parameters che istanzia le coppie (k:v) all inizio del programma inserendole nelle variabili, il vantaggio è automatizzare la stampa dei parametri insieme allo score nel file tests.csv che compare qui sotto
- le operazioni più costose hanno %time davanti così da poter sapere dopo un po di tentativi il tempo che si aspetterà
- l' operazione più costosa (group_me) può essere approssimata passando da "mode" a "mean"
- le funzioni più importanti o costose danno un output minimo, le meno importanti e in tempo $\Theta(1)$ non danno output a meno che non lavorino sul filesystem
- viene data la possibilità di salvare il dataset su disco nel caso in cui si usi la moda poichè è un operazione che impiega molto tempo mentre in tutti gli altri casi si consiglia di runnare normalmente il kernel poichè media/mediana/moda_approssimata hanno ottima complessità
- testing iterativo dei parametri per scegliere il migliore valore per ciascuno di essi(per ora solo n_leaves ma assolutamente scalabile a piacere con l' aggiunta di un for che cicla la lista di parametri da testare per migliorare lo score)

9 Features future

- Si vorrebbe innanzitutto fare un merge del test e del train poichè dopo il data-leakage essi sono entrambi datasets di test ma per ora non lo si è fatto poichè già il training viene fatto su una piccola parte del dataset train
- pca sui numerics
- aggiungere weekday
- cascading = fare n predittori specializzati per cose diverse su set diversi
- probabilmente le azioni di un utente dipende da quello che ha fatto al massimo nelle 2 settimane precedenti

10 I numeri di questa competizione

- Score committato : 0.1227
- numero di submissions totali : 120 in circa 2 mesi(tempo effettivamente disponibile) -> una media di circa 2 al giorno, notare che il numero di submissions è limitato a 5 al giorno perciò l' impegno è stato costante
- 6 versioni del kernel per un unica competizione(verrà consegnata solo la versione definitiva ma sono disponibili anche le altre)
- 6 moduli separati di funzioni per un totale di più di 1200 righe di codice in totale(solo nei moduli)
- 44 commits in 27 giorni, poco meno di 2 commits al giorno(in continua aggiunta, inoltre il github non è stato fatto da subito ma solo alla versione 6 poichè ritenuta matura)
- 2 settimane per adattare il nuovo dataset al vecchio kernel. Il must era mantenere più informazioni possibili e scartare quasi nulla.
- 168hr il tempo impiegato in questo progetto, la stima è fatta per difetto e seguendo l' idea del problema di fermi: ogni 5 submissions si pensa che la prima costi 7hr(nella versione 6 del kernel cioè quest' ultima il kernel veniva preparato e le submissions venivano fatte il giorno dopo quindi ben più di 7hr) e che le successive vengano gratuitamente perciò $\frac{120}{5} \cdot 7hr = 24 \cdot 7 = 168$
- 4, il livello del mio account su kaggle cioè competition contributor.

11 Considerazioni

Si è notato che la regressione lineare è più efficiente e più rapida di quanto testato inizialmente nei kernel di versioni precedenti che non godevano di OHE e addirittura supera LightGBM in caso di dataset ridotti cioè inferiori a 100000 righe, questo probabilmente perchè fintanto che il dataset è ridotto e ci sono molti zeri la regressione lineare indovina sempre puntando tutto a 0 mentre l' albero di regressione con pochi dati si può confondere, tanto più si allarga il dataset tanto più la regressione lineare riduce la sua precisione. Inoltre introducendo transactionRevenue l' albero di regressione supera di molto la regressione lineare anche per datasets piccoli. Inoltre se si aggiunge il transaction revenue LightGBM riesce molto meglio della regressione lineare.

12 Dati raccolti

Avvertenze:

- nella tabella qui sotto trovare il numero di righe nel test a 500 non è un errore! essendo questi dati interni e non spediti a kaggle posso diminuire le righe di test(cioè da prevedere) per potermi concentrare su ottenere dati nel train. I dati raccolti perciò sono rispetto al validation.
- il final_score è calcolato senza il transactionRevenue come consigliato dal docente ma è stato inserito un parametro per sfruttare o meno questo vantaggio
- i parametri vengono scritti in maniera automatica sul file perciò si avrà cura di tenere a mente che se il "tipo" == "lin_reg" i parametri che regolano LightGBM non devono essere valutati

Considerazioni:

- Si è voluto sperimentare come l' accuratezza migliorasse all' aumentare dei dati, non solo per migliorare il punteggio ma per controllare che la LGN valesse o meglio per avere riscontro della convergenza degli algoritmi usati. Se così non fosse stato allora avremmo scoperto errori nella programmazione.
- Si è notato che col nuovo dataset si è in grado di fare previsioni migliori che non col dataset precedente fornito da kaggle(disponibili i kernel di versione precedente a questa a dimostrarlo, score >= 1.57 circa)
- In maniera abbastanza inasspettata si è scoperto che la regressione lineare dà uno score migliore di un albero di regressione nel caso in cui la transactionRevenue venga esclusa e il dataset sia piccolo, è facile immaginare sia così poichè il dataset ha molti zeri.
- Si fa notare che dove $|dataset| \geqslant 10000$ si preferisce usare la media per i dati categoriali invece della moda poichè con un dataset di 10000 righe la moda impiega 34min di computazione e per un dataset maggiore sarebbe impraticabile. E' perciò un approssimazione non voluta ma obbligata, detto ciò il risultato è comunque molto valido, sia con la moda su un dataset piccolo sia con la media con un dataset grande e questo è supportato da fatto che nonostante la media non sia significativa per dati categoriale questo viene comunque accettato da un albero di regressione(cambieranno soltanto i valori nei nodi)

```
In [1]: import pandas as pd
        df = pd.read_csv("tests.csv")

In [2]: pd.read_csv("tests.csv")
```

Out[2]:

	max_new_feat	commit	n_leaves	feature_fraction	bagging_fraction	\
0	50	0	100	0.90	0.80	
1	50	0	100	0.90	0.80	
2	50	0	8192	0.80	0.95	
3	50	0	8192	0.40	0.25	
4	50	0	100	0.70	0.50	
5	50	0	200	0.99	0.99	
6	50	0	8192	0.40	0.25	
7	50	0	1000	0.99	0.99	
8	50	0	8192	0.99	0.99	

9	500	0	100	0.99	0.99
10	500	0	100	0.99	0.99
11	500	0	8192	0.99	0.99
12	500	0	100	0.99	0.99
13	500	0	100	0.99	0.99
14	500	0	8192	0.99	0.99
15	500	0	8192	0.99	0.99
16	500	0	100	0.99	0.99
17	500	0	100	0.99	0.99
18	500	0	8192	0.99	0.99
19	500	0	100	0.99	0.99
20	500	0	100	0.99	0.99
21	500	0	8192	0.99	0.99
22	500	0	4096	0.99	0.99
23	500	0	4096	0.99	0.99
24	500	0	2048	0.99	0.99
25	500	0	1024	0.99	0.99
26	500	0	1024	0.99	0.99
27	500	0	2048	0.99	0.99
28	500	0	1024	0.99	0.99
29	500	0	512	0.99	0.99
..	...	...	...	...	...
167	500	0	32	0.99	0.99
168	500	0	16	0.99	0.99
169	500	0	32	0.99	0.99
170	500	0	64	0.99	0.99
171	500	0	128	0.99	0.99
172	500	0	256	0.99	0.99
173	500	0	512	0.99	0.99
174	500	0	16	0.99	0.99
175	500	0	32	0.99	0.99
176	500	0	64	0.99	0.99
177	500	0	128	0.99	0.99
178	500	0	256	0.99	0.99
179	500	0	512	0.99	0.99
180	500	0	16	0.99	0.99
181	500	0	17	0.99	0.99
182	500	0	18	0.99	0.99
183	500	0	19	0.99	0.99
184	500	0	20	0.99	0.99
185	500	0	21	0.99	0.99
186	500	0	22	0.99	0.99
187	500	0	23	0.99	0.99
188	500	0	24	0.99	0.99
189	500	0	25	0.99	0.99
190	500	0	26	0.99	0.99
191	500	0	27	0.99	0.99
192	500	0	28	0.99	0.99

193	500	0	29	0.99	0.99
194	500	0	30	0.99	0.99
195	500	0	31	0.99	0.99
196	500	0	32	0.99	0.99

	learn_rate	train_rows	test_rows	test_also_lin_reg	bagging_freq	\
0	0.004	100000	500	1	1	
1	0.004	100000	500	1	1	
2	0.004	100000	500	1	20	
3	0.004	100000	500	1	1	
4	0.004	100000	500	1	1	
5	0.004	100000	500	1	1	
6	0.004	100000	500	1	1	
7	0.004	100000	500	1	1	
8	0.004	100000	500	1	20	
9	0.004	100000	500	1	1	
10	0.004	100000	500	1	1	
11	0.004	100000	500	1	20	
12	0.004	10000	500	1	1	
13	0.004	10000	500	1	1	
14	0.004	10000	500	1	20	
15	0.004	10000	500	1	20	
16	0.004	10000	500	1	1	
17	0.004	10000	500	1	1	
18	0.004	10000	500	1	20	
19	0.004	100000	500	1	1	
20	0.004	100000	500	1	1	
21	0.004	100000	500	1	20	
22	0.004	100000	500	1	20	
23	0.004	100000	500	1	20	
24	0.004	100000	500	1	20	
25	0.004	100000	500	1	1	
26	0.004	100000	500	1	1	
27	0.004	100000	500	1	20	
28	0.004	100000	500	1	20	
29	0.004	100000	500	1	20	
..	...	...	...	...	...	
167	0.004	10000	500	1	20	
168	0.004	10000	500	1	20	
169	0.004	10000	500	1	20	
170	0.004	10000	500	1	20	
171	0.004	10000	500	1	20	
172	0.004	10000	500	1	20	
173	0.004	10000	500	1	20	
174	0.004	10000	500	1	20	
175	0.004	10000	500	1	20	
176	0.004	10000	500	1	20	
177	0.004	10000	500	1	20	

178	0.004	10000	500	1	20
179	0.004	10000	500	1	20
180	0.004	10000	500	1	20
181	0.004	10000	500	1	20
182	0.004	10000	500	1	20
183	0.004	10000	500	1	20
184	0.004	10000	500	1	20
185	0.004	10000	500	1	20
186	0.004	10000	500	1	20
187	0.004	10000	500	1	20
188	0.004	10000	500	1	20
189	0.004	10000	500	1	20
190	0.004	10000	500	1	20
191	0.004	10000	500	1	20
192	0.004	10000	500	1	20
193	0.004	10000	500	1	20
194	0.004	10000	500	1	20
195	0.004	10000	500	1	20
196	0.004	10000	500	1	20

	transactionRevenue	percentage	grouping_mode_cats	final_score	\
0	0	18	mode	1.276707	
1	0	18	mode	1.517199	
2	0	18	mode	1.530116	
3	0	18	mode	1.536360	
4	0	18	mode	1.506410	
5	0	18	mode	1.534567	
6	0	18	mode	1.536360	
7	0	18	mode	1.534808	
8	0	18	mode	1.536360	
9	1	18	mode	1.056175	
10	1	18	mode	0.034409	
11	1	18	mode	0.145676	
12	0	18	mode	1.092369	
13	0	18	mode	1.002389	
14	0	18	mode	1.002389	
15	0	18	mode	1.055687	
16	0	18	mean	1.222760	
17	0	18	mean	1.639784	
18	0	18	mean	1.824215	
19	0	18	mean	1.287251	
20	0	18	mean	1.525817	
21	0	18	mean	1.646259	
22	0	18	mean	1.646259	
23	0	18	mean	1.646259	
24	0	18	mean	1.646259	
25	0	18	mean	1.282734	
26	0	18	mean	1.532604	

27	0	18	mean	1.665875
28	0	18	mean	1.665875
29	0	18	mean	1.665875
..	...	...	...	...
167	0	18	mode	1.006207
168	0	18	mode	1.010126
169	0	18	mode	1.010126
170	0	18	mode	1.010126
171	0	18	mode	1.010126
172	0	18	mode	1.010126
173	0	18	mode	1.010126
174	0	18	mode	1.010126
175	0	18	mode	1.010126
176	0	18	mode	1.010126
177	0	18	mode	1.010126
178	0	18	mode	1.010126
179	0	18	mode	1.010126
180	0	18	mode	1.010126
181	0	18	mode	1.010126
182	0	18	mode	1.010126
183	0	18	mode	1.010126
184	0	18	mode	1.010126
185	0	18	mode	1.010126
186	0	18	mode	1.010126
187	0	18	mode	1.010126
188	0	18	mode	1.010126
189	0	18	mode	1.010126
190	0	18	mode	1.010126
191	0	18	mode	1.010126
192	0	18	mode	1.010126
193	0	18	mode	1.010126
194	0	18	mode	1.010126
195	0	18	mode	1.010126
196	0	18	mode	1.010126

	min_child_samples	type
0	-1	lin_reg
1	-1	LightGBM
2	30	LightGBM_rf
3	10	LightGBM_rf
4	-1	LightGBM
5	-1	LightGBM
6	10	LightGBM_rf
7	-1	LightGBM
8	10	LightGBM_rf
9	-1	lin_reg
10	-1	LightGBM
11	10	LightGBM_rf

12	-1	lin_reg
13	-1	LightGBM
14	10	LightGBM_rf
15	10	LightGBM_rf
16	-1	lin_reg
17	-1	LightGBM
18	10	LightGBM_rf
19	-1	lin_reg
20	-1	LightGBM
21	10	LightGBM_rf
22	10	LightGBM_rf
23	10	LightGBM_rf
24	10	LightGBM_rf
25	-1	lin_reg
26	-1	LightGBM
27	10	LightGBM_rf
28	10	LightGBM_rf
29	10	LightGBM_rf
..	...	...
167	10	LightGBM
168	10	LightGBM
169	10	LightGBM
170	10	LightGBM
171	10	LightGBM
172	10	LightGBM
173	10	LightGBM
174	10	LightGBM
175	10	LightGBM
176	10	LightGBM
177	10	LightGBM
178	10	LightGBM
179	10	LightGBM
180	10	LightGBM
181	10	LightGBM
182	10	LightGBM
183	10	LightGBM
184	10	LightGBM
185	10	LightGBM
186	10	LightGBM
187	10	LightGBM
188	10	LightGBM
189	10	LightGBM
190	10	LightGBM
191	10	LightGBM
192	10	LightGBM
193	10	LightGBM
194	10	LightGBM
195	10	LightGBM

```
196                 10     LightGBM

[197 rows x 16 columns]
```

13 Esperimenti

Da qui in poi si è voluto sperimentare come diversi valori dei parametri potessero cambiare lo score e ovviamente poi si è reportato tutto qui sotto. Sono chiamati esperimenti poichè data un ipotesi sono stati interrogati i dati per vedere se l' ipotesi fosse supportata da essi ed ogni esperimento porta con sè una "morale" o conclusione per confermare o smentire l' ipotesi.

Si avvisa che le rappresentazioni nel grafico sono tante volte manipolate attraverso alcune traformazioni di funzioni/normalizzazioni per rendere evidente il risultato ottenuto

13.1 Esperimento 1:

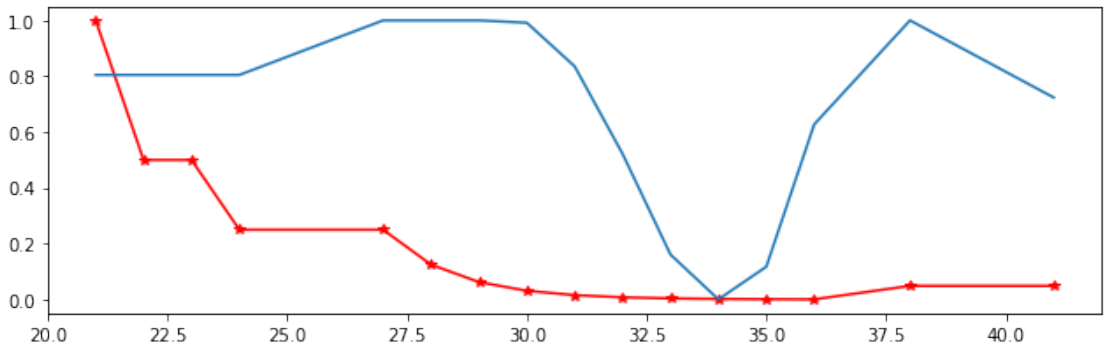
13.1.1 Aumentare le foglie in una random forest migliora lo score in virtù della teoria che ci sta dietro

Sapendo la teoria delle random forest e cioè che si vuole portare i vari alberi all' overfit per poi mediarne i risultati si è pensato che aumentando il numero di foglie potesse aumentare la precisione

Si è effettivamente notato dai dati raccolti che aumentare le foglie diminuisce l' RMSE cioè migliora lo score.

Lo si può vedere nel grafico qui sotto dove sono stati normalizzati i dati per permettere di vedere il rapporto che hanno i due. I dati sono stati scelti tra campioni simili per esempio cercando campioni con stesso numero di righe poichè quello è un fattore determinante per lo score

```
In [4]: import matplotlib.pyplot as plt
        from generic import norm

        fig, ax = plt.subplots( figsize=(9,3), tight_layout=True)

        arr = (df["type"] == "LightGBM_rf") & (df["train_rows"] == 100000) & (df["max_new_feat"
        l = df.loc[arr,"n_leaves"]
        ax.plot( norm(l) ,'r-*')
        #foglie in rosso
        #al diminuire delle foglie il punteggio cala cioè peggiora

        l = df.loc[arr,"final_score"]
        ax.plot( norm(l) )

Out[4]: [<matplotlib.lines.Line2D at 0x1a1c9fffd0>]
```




```
In [5]: arr = (df["type"] == "LightGBM_rf") & (df["train_rows"] == 100000) & (df["max_new_feat
        l = df.loc[arr,"n_leaves"]


        l = df.loc[arr,"final_score"]
        m = min(l)
        m

        df.loc[df["final_score"] == m,"n_leaves"]

Out[5]: 34    16
        Name: n_leaves, dtype: int64
```

13.1.2 Conclusione inaspettata

Aumentare le foglie è utile ma in questo caso ha provocato un punteggio più scarso. Il risultato è sicuramente di rilievo e si fornisce la scelta migliore in questo caso: la riga 34 del dataset df contiene i valori chiave per ottenere il miglior risultato nel caso di questo esperimento cioè 16 foglie

13.2 Esperimento 2

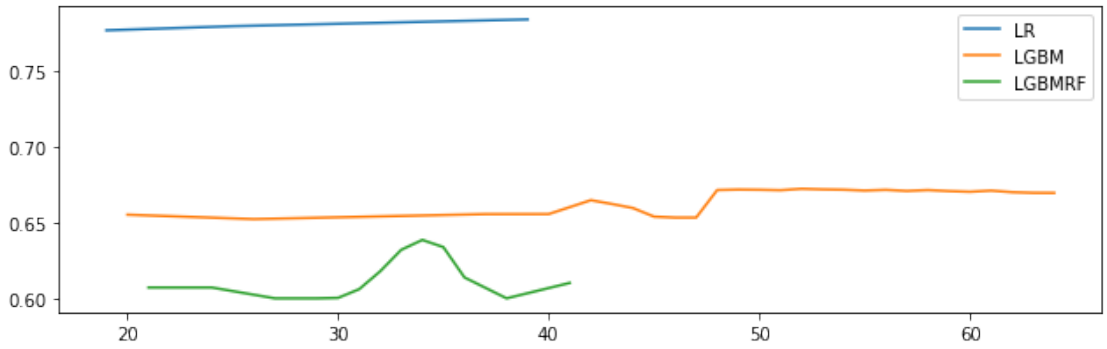
13.2.1 su piccoli dataset la LR performa meglio di LightGBM(senza transactionRevenue)

Inizialmente si pensava che LR non potesse in alcuna maniera superare in score LGBM ma stando ai dati su piccoli datasets(10000) la LR performa meglio. Si può pensare che LR predica sempre 0.0 che su questo dataset è il 99% dei valori del totalTransactionRevenue perciò faccia un ottimo score mentre si può pensare che LGBM non abbia abbastanza dati per fare un buon training e che fare boosting porti solo ad un forte overfit per l' ovvio basso supporto da parte dei dati. Questo fatto infatti non sarà più vero su datasets più grandi

```
In [6]: import matplotlib.pyplot as plt
```

```
        fig, ax = plt.subplots( figsize=(9,3), tight_layout=True)

        arr = (df["type"] == "lin_reg") & (df["train_rows"] == 100000) & (df["max_new_feat"] ==
        l = df.loc[arr,"final_score"]
        ax.plot( 1/l )

        arr = (df["type"] == "LightGBM") & (df["train_rows"] == 100000) & (df["max_new_feat"] =
        l = df.loc[arr,"final_score"]
        ax.plot( 1/l )

        arr = (df["type"] == "LightGBM_rf") & (df["train_rows"] == 100000) & (df["max_new_feat"
        l = df.loc[arr,"final_score"]
        ax.plot( 1/l )
        ax.legend(["LR","LGBM","LGBMRF"])
```

```
Out[6]: <matplotlib.legend.Legend at 0x1a1cc71898>
```




```
In [7]: arr = (df["train_rows"] == 100000) & (df["max_new_feat"] == 500) & (df["transactionRev

        l = df.loc[arr,"final_score"]
        m = min(l)
        m

        df.loc[df["final_score"] == m,"type"]
```

```
Out[7]: 39    lin_reg
        Name: type, dtype: object
```

13.2.2 Conclusione

L' affermazione di questo esperimento viene confermata dai dati

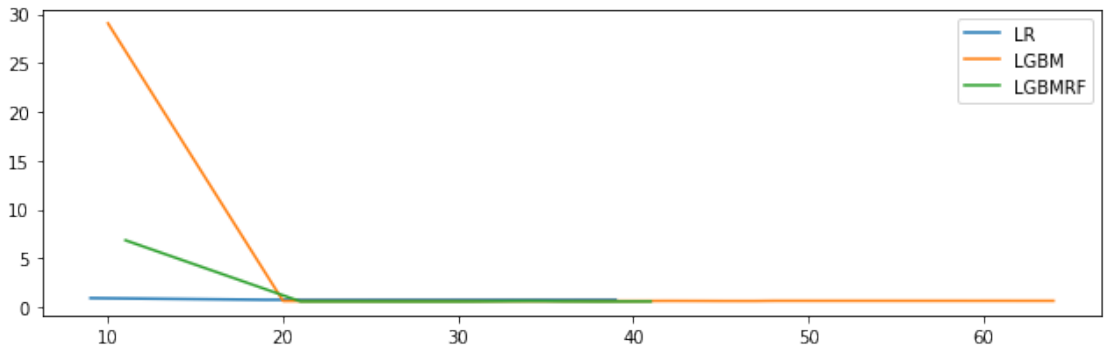
13.3 Esperimento 3

13.3.1 su piccoli dataset LightGBM con singolo albero non ha eguali(con transactionRevenue)

Inizialmente si pensava che LightGBM_rf cioè non il singolo albero ma la random forest potesse performare meglio del singolo albero ma osservando i dati si può giungere alla conclusione che LGBM a singolo albero che sfrutta il 100% delle features performa meglio sia della RF sia della LR. Si può probabilmente asserire che LGBM riesca a battere in quanto a score la RF poichè appunto il singolo albero dispone di tutte le features mentre nella RF le feats vengono divise sui vari alberi della foresta e questo porti a molti alberi con molti valori sballati ed essendo la media un funzionale non robusto questo porti ad uno score peggiore.

Si noti che a differenza degli altri esperimenti qui si è mantenuto il transactionRevenue come predittore poichè si vuole dimostrare che nella competizione di kaggle si è fatta una scelta sensata volta ad ottenere il punteggio maggiore possibile in LB

```
In [8]: import matplotlib.pyplot as plt

        fig, ax = plt.subplots( figsize=(9,3), tight_layout=True)

        arr = (df["type"] == "lin_reg") & (df["train_rows"] == 100000) & (df["max_new_feat"] ==
        l = df.loc[arr,"final_score"]
        ax.plot( 1/l )

        arr = (df["type"] == "LightGBM") & (df["train_rows"] == 100000) & (df["max_new_feat"] =
        l = df.loc[arr,"final_score"]
        ax.plot( 1/l )

        arr = (df["type"] == "LightGBM_rf") & (df["train_rows"] == 100000) & (df["max_new_feat"
        l = df.loc[arr,"final_score"]
        ax.plot( 1/l )
        ax.legend(["LR","LGBM","LGBMRF"])

Out[8]: <matplotlib.legend.Legend at 0x1a1ccb2400>
```



13.3.2 Conclusione

L' affermazione di questo esperimento viene confermata dai dati

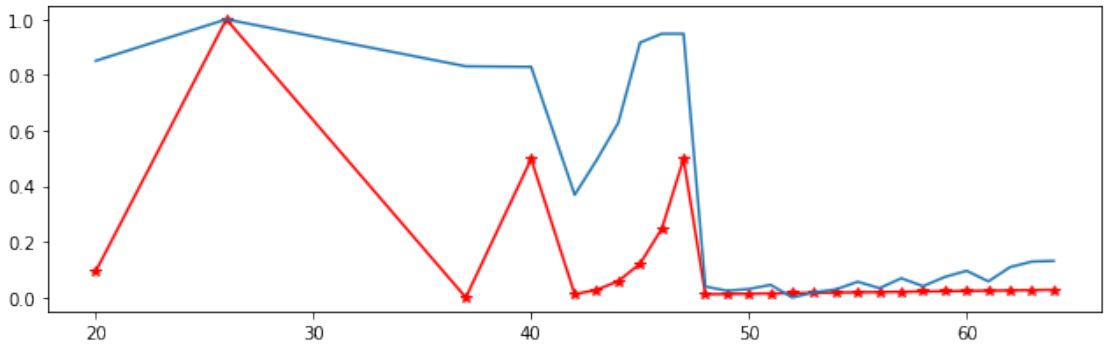
13.4 Esperimento 4

13.4.1 miglior n_leaves per LGBM

Si sta cercando il miglior valore per il numero di foglie per il singolo albero di regressione. Si cerca il valore tentando iterativamente potenze del 2 e poi selezionando il minimo rmse.

```
In [9]: import matplotlib.pyplot as plt
        from generic import norm

        fig, ax = plt.subplots( figsize=(9,3), tight_layout=True)

        arr = (df["type"] == "LightGBM") & (df["train_rows"] == 100000) & (df["max_new_feat"] =
        l = df.loc[arr,"n_leaves"]
        ax.plot( norm(l) ,'r-*')
        #foglie in rosso
        #al diminuire delle foglie il punteggio cala cioè peggiora

        l = df.loc[arr,"final_score"]
        ax.plot( norm(l) )
```

```
Out[9]: [<matplotlib.lines.Line2D at 0x1a1cddd7b8>]
```




```
In [10]: arr = (df["type"] == "LightGBM") & (df["train_rows"] == 100000) & (df["max_new_feat"]
         l = df.loc[arr,"n_leaves"]


         l = df.loc[arr,"final_score"]
         m = min(l)
         m

         df.loc[df["final_score"] == m,"n_leaves"]
```

```
Out[10]: 52    20
         Name: n_leaves, dtype: int64
```

13.4.2 Conclusione

Concludo che la scelta migliore per il numero di foglie con LGBM è 20 per l' esperimento descritto qui sopra

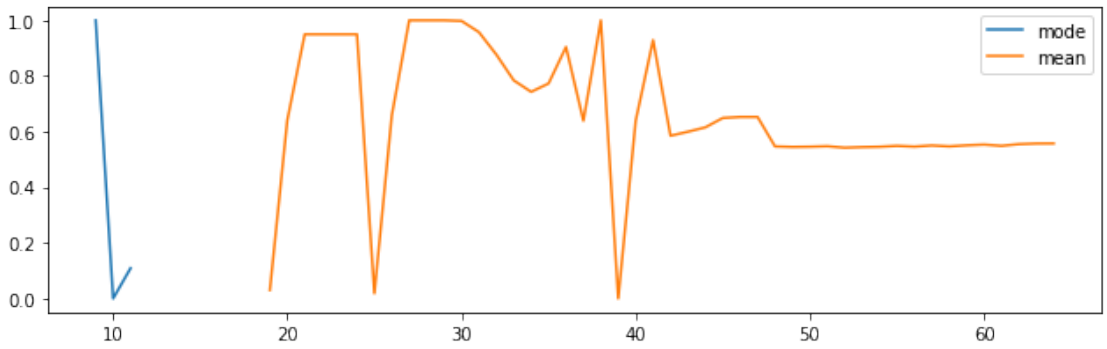
13.5 Esperimento 5

13.5.1 la moda per le cats è più potente(e sensata) della media

Si vorrebbe scegliere sempre di utilizzare la moda per i dati categoriali poichè non ha senso usare la media su valori che sono discreti(e.g.: dire che ho metà iphone e metà samsung quando entro in un negozio...), il problema è che la moda ha un costo computazionale troppo alto. Si vuole perciò dimostrare dati alla mano che usare la moda è comunque una scelta forte e ottima se solo fosse possibile computarla in maniera più rapida.

```
In [11]: import matplotlib.pyplot as plt
         from generic import norm

         fig, ax = plt.subplots( figsize=(9,3), tight_layout=True)

         arr = (df["grouping_mode_cats"] == "mode") & (df["train_rows"] == 100000) & (df["max_
         l = df.loc[arr,"final_score"]
         ax.plot( norm(l) )

         arr = (df["grouping_mode_cats"] == "mean") & (df["train_rows"] == 100000) & (df["max_
         l = df.loc[arr,"final_score"]
         ax.plot( norm(l) )

         ax.legend(["mode","mean"])
```

```
Out[11]: <matplotlib.legend.Legend at 0x1a1cf35fd0>
```



```
In [12]: arr = (df["train_rows"] == 10000) & (df["max_new_feat"] == 500) & (df["transactionRev
         l = df.loc[arr,"grouping_mode_cats"]


         l = df.loc[arr,"final_score"]
         m = min(l)
         m

         df.loc[df["final_score"] == m,"grouping_mode_cats"]

Out[12]: 13    mode
         14    mode
         Name: grouping_mode_cats, dtype: object
```

13.5.2 Conclusioni

La moda è stata testata solo su piccoli dataset(max 10000 righe) per il suo alto costo computazionale(7min su 10000righe, 34min su 100000righe senza contare il costo del caricare i dati da had, inoltre ha complessità alta poichè con l'aumentare delle righe aumenta anche la variabilità di ogni singola colonna e con il OHE si aggiungono colonne a destra). Detto ciò per piccoli datasets dà molto scarto al funzionale statistico della media in quanto a precisione.

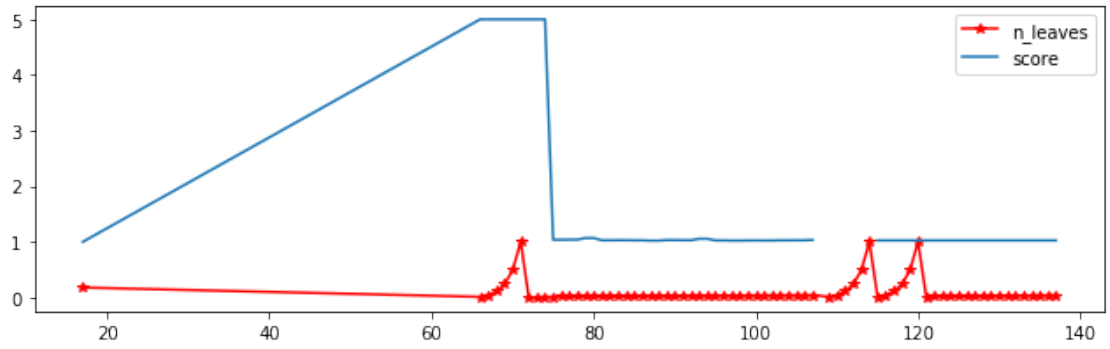
13.6 Esperimento 6

13.6.1 Aumentare le foglie di LGBM o LGBM_rf non porta miglioramento dopo un certo treshold

Si è notato che nonostante aumentare le foglie ad un albero di regressione teoricamente dovrebbe dargli più potere espressivo in realtà ad un certo punto non apporta pi alcun miglioramento poichè probabilmente viene richiesto un certo supporto per creare una foglia e l' albero nonlo crea oppure lo crea ma molti dati comunque cadono nelle foglie più votate.

```
In [13]: import matplotlib.pyplot as plt
         from generic import norm

         fig, ax = plt.subplots( figsize=(9,3), tight_layout=True)

         arr = (df["type"] == "LightGBM") & (df["train_rows"] == 10000) & (df["max_new_feat"] =
         l = df.loc[arr,"n_leaves"]
         ax.plot( norm(l) ,'r-*')
         #foglie in rosso
         #al diminuire delle foglie il punteggio cala cioè peggiora

         l = df.loc[arr,"final_score"]
         ax.plot( 1/norm(l) )

         ax.legend(["n_leaves","score"])
```

```
Out[13]: <matplotlib.legend.Legend at 0x1a1cf9fe10>
```



Vedendo il grafico non è facile capire che per un aumento delle foglie non consegue un miglioramento dello score perciò facciamo un interrogazione al dataframe volta ad estrarre il valore minimo dello score e il conseguente numero di foglie che implica quell' ottimo punteggio, noteremo che ci sono più valori diversi del parametro n_leaves, questo perchè appunto aumentare le foglie non è detto porti a un incremento dello score sopra un certo treshold

```
In [14]: arr = (df["type"] == "LightGBM") & (df["train_rows"] == 10000) & (df["max_new_feat"] =
         l = df.loc[arr,"n_leaves"]

         l = df.loc[arr,"final_score"]
         m = min(l)
         m

         df.loc[df["final_score"] == m,"n_leaves"]

Out[14]: 108    512
         109     16
         110     32
         111     64
         112    128
         113    256
         114    512
         Name: n_leaves, dtype: int64
```

13.6.2 Conclusioni

I dati supportano il fatto che oltre un certo treshold non serve a nulla aumentare le foglie, il corollario di questo fatto è che possiamo ottenere un ottimo punteggio con poche foglie perciò risparmiando computazione e spazio