

UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Ingegneria dell'Informazione ed
Elettrica e Matematica Applicata



Statistical Data Analysis
PROJECT

Ammendola Giovanni:	0622701167
Petrone Vincenzo:	0622701134
Valitutto Andrea:	0622701366
Ventre Salvatore:	0622701343

Anno Accademico 2019-2020

INDICE

Introduzione	1
Dataset	1
Preprocessing	2
Funzioni di utilità	2
Rinominazione colonne	3
Inserimento colonna	3
Cambiamento del formato della data	3
Rimozione e modifica delle colonne	4
Fusione dei dataset	4
Sostituzione dei dati mancanti	5
Progetto Python	7
Preprocessing	7
Analisi preliminare	7
Bilanciamento del dataset	7
Parametri statistici dei dati	8
Generazione di un dataset sintetico	8
Naïve Bayes	9
Applicazione del classificatore Naïve Bayes sul dataset sintetico	9
Applicazione del classificatore Naïve Bayes sul dataset reale	11
Utilizzo di due feature	11
Utilizzo di una feature per volta	12
Utilizzo di coppie di feature	12
Utilizzo di tutte le feature disponibili	13
KNN	13
Applicazione del classificatore KNN sul dataset sintetico	14
Spark	19
Applicazione del classificatore KNN sul dataset reale	19
Utilizzo di due feature	20
Utilizzo di una feature per volta	21
Utilizzo di tutte le feature disponibili	21
Logistic Regression	23
Applicazione del classificatore Logistic Regression sul dataset sintetico	23
Spark	26
Applicazione del classificatore Logistic Regression sul dataset reale	28
Utilizzo di due feature	29
Utilizzo una feature per volta	29
Utilizzo di coppie di feature	30
Utilizzo di tutte le feature disponibili	30

Naïve Kernel	31
Applicazione del classificatore Naïve Kernel sul dataset sintetico	31
Spark	33
Applicazione del classificatore Naïve Kernel sul dataset reale	33
Utilizzo di due feature	33
Utilizzo una feature per volta	33
Utilizzo di tutte le feature disponibili	34
Progetto R	35
Regessione Multipla	35
Metodi di Ricampionamento statistico	39
Validation Set	40
K-Fold Cross Validation	40
Bootstrap	41
Subset Selection	42
Best Subset Selection	42
Stepwise Selection	46
Forward Stepwise Selection	46
Backward Stepwise Selection	46
Considerazioni finali	47
Regessione con Regolarizzazione	48
Ridge	48
Analisi e osservazioni	49
LASSO	50
Analisi e osservazioni	51
PCR	52
Analisi e Osservazioni	53
PLS	54
Analisi e Osservazioni	54

1. Introduzione

Per la realizzazione del progetto di Statistical Data Analysis è stata richiesta l'applicazione delle diverse tecniche di analisi dei dati studiate durante il corso. Sono stati, dunque, esplorati e confrontati i diversi metodi, motivando la scelta sulla base di considerazioni teoriche ed evidenze sperimentali.

Il progetto si compone di due parti: la prima affronta i diversi approcci sulla classificazione, mentre la seconda si concentra sulle tecniche di regressione. Il tutto è stato realizzato sulla base di un dataset opportunamente scelto durante la prima fase del progetto. In ciascuna delle due parti, vengono mostrati grafici diagnostici e tabelle riassuntive per valutare la bontà delle scelte attuate.

1.1. Dataset

La fase iniziale del progetto è stata caratterizzata dallo studio di un dataset, la cui scelta è stata completamente libera, ma con il solo obbligo di concentrare maggiormente l'attenzione su dataset inerenti al percorso di studi di Artificial Intelligence. Inoltre, considerando le indicazioni di progetto, che vedono l'applicazione sia di metodologie di regressione che di classificazione, l'analisi ha portato a scegliere un dataset che permetesse di affrontare entrambe le tecniche.

Il contenuto del dataset riguarda le informazioni sulle principali linee aeree internazionali. Le informazioni di cui parliamo sono: compagnia aerea, recensione da parte del viaggiatore, votazione per i differenti servizi offerti, una votazione complessiva tenendo in considerazione tutti i servizi presenti e, infine, un'indicazione sulla raccomandazione del volo.

In particolare, è stata sfruttata la piattaforma [Kaggle](#), una comunità online di data science che consente di trovare e pubblicare set di dati. Tra i diversi dataset esplorati, ne sono stati scelti due in base ai requisiti che mostravano e alle nostre esigenze di progetto. Entrambi contengono le informazioni precedentemente citate con lievi differenze su alcuni aspetti. L'idea di base è stata quella di aggregare i due dataset, in modo da incrementare il numero di dati a disposizione. Però, è stato notato che la matrice dei dati contenuti presenta delle voci uguali per entrambi, spingendoci ad una fase successiva di ripulitura dei dati duplicati.

Di seguito, vengono mostrati i set di dati iniziali:

Dataset 1: composto da 65948 righe e 17 colonne. Reperibilità: [Kaggle](#)

- **airline:** nome della compagnia aerea;
- **overall:** voto (su un punteggio da 1 a 10) finale dell'utente sulla base dei servizi offerti dalla compagnia aerea per un determinato volo;
- **author:** autore che ha recensito il volo;
- **review_data:** data in cui è stata rilasciata la recensione;
- **customer_review:** feedback del cliente, in formato testuale, sul volo;
- **aircraft:** modello di aereo;

- **traveller_type**: tipo di viaggiatore (Business, Family Leisure, Solo Leisure, Couple Leisure);
- **cabin**: classe di viaggio (Economy, Business, First, Premium);
- **route**: luogo di partenza e destinazione del volo;
- **date_flown**: data di volo;
- **seat_comfort**: comodità del sedile (valutazione da 1 a 5);
- **cabin_service**: servizio del personale a bordo (valutazione da 1 a 5);
- **food_bev**: servizio di ristorazione a bordo (valutazione da 1 a 5);
- **entertainment**: servizio di intrattenimento a bordo (valutazione da 1 a 5);
- **ground_service**: servizio a terra messo a disposizione dalla rispettiva linea aerea (valutazione da 1 a 5);
- **value_for_money**: rapporto qualità-prezzo (valutazione da 1 a 5).
- **recommended**: dichiarazione sulla raccomandazione del volo ('sì' o 'no').

Dataset 2: composto da 41456 righe e 20 colonne. Reperibilità: [GitHub](#)

Le colonne aggiuntive rispetto al primo dataset sono:

- **wifi_connectivity_rating**: servizio di connessione ad internet fornito a bordo (valutazione da 1 a 5);
- **link**: link della recensione;
- **title**: titolo della recensione;
- **autor_country**: provenienza del cliente che ha recensito il viaggio.

L'unica colonna mancante rispetto al primo dataset è date_flown.

1.2. Preprocessing

Dunque, la fase successiva è stata dedicata alla pre-elaborazione dei dati, fondamentale per il prosieguo del progetto. Infatti, i metodi di raccolta dei dati sono spesso controllati in modo impreciso, dando vita a valori fuori range, valori mancanti, ecc. Eseguire una tecnica di regressione o classificazione su dataset che non sono stati attentamente esaminati, può produrre risultati fuorvianti. Pertanto, il controllo sulla rappresentazione e sulla qualità dei dati è fondamentale prima di eseguire qualsiasi operazione sui dati, influendo sul modo in cui vengono interpretati i risultati finali.

Fatta questa premessa, attraverso *RStudio*, un ambiente di sviluppo per elaborazione statica e grafica, sono state effettuate diverse tappe per migliorare la qualità dei dati a disposizione.

1.2.1. Funzioni di utilità

Per facilitare le operazioni di analisi del dataset, il progetto ha visto l'uso di due funzioni di base, allo scopo di evitare la riscrittura delle stesse ad ogni utilizzo. Le funzioni sono state definite all'interno del file `preprocessing.R` e sono:

- `na.omit.unique`: acquisisce una feature di un set di dati, e rimuove ogni valore NA, restituendo solo i valori di funzionalità univoci.

- `rename`: rinomina una feature con un nuovo valore, acquisendolo in ingresso, insieme al vecchio valore che verrà rinominato. Restituisce il nuovo set di dati aggiornato.

Un altro aspetto da evidenziare riguarda l'esecuzione dello script `sql.R`, al cui interno sono presenti delle istruzioni attinenti all'utilizzo del pacchetto `RSQLite`. Questo è stato impiegato per l'esecuzione di una query, necessaria a preparare il dataset alle successive operazioni di preprocessing. Nel dettaglio, dopo aver creato un effimero database in memoria, quest'ultimo è stato popolato con i record presenti nel dataset in locale. In seguito, è stata realizzata una query per selezionare tutti i record che rispettassero la seguente condizione: stesso nome di un autore con una stessa recensione, ma che avessero la stessa compagnia aerea rappresentata con nome diverso (ad esempio “Adria Airways” e “adria-airways”).

Il risultato della query è stato utile perché mostrava quali righe del dataset completo fossero uguali al netto di una denominazione diversa della compagnia aerea.

```
q = dbGetQuery(conn,
  'SELECT m1.*
   FROM mer m1 JOIN mer m2
   WHERE m1.author = m2.author
   AND m1.customer_review = m2.customer_review
   AND m1.airline != m2.airline')
```

1.2.2. Rinominazione colonne

Dopo aver caricato i due dataset, la prima operazione è stata quella di rinominare tutte le colonne dei due set di dati per avere congruenza tra le osservazioni considerate per le diverse stime che sono state compiute. In particolare, sono state rinominate tutte le colonne del secondo dataset con i nomi utilizzati nel primo, ad eccezione di `cabin` (primo dataset) e `cabin flown` (secondo dataset) in “`class`” .

Questa operazione di rinominazione è servita per poter unificare le parti discordanti tra i due dataset.

1.2.3. Inserimento colonna

Come descritto in precedenza, il confronto tra i due set di dati è caratterizzato dalla non presenza della colonna `wifi_connectivity` nel primo, che ne ha obbligato l'aggiunta. In particolare, l'aggregazione di questa colonna, durante questa fase, è stata eseguita andando ad inserire una voce “NA”, sinonimo di assenza di dato, per ogni riga del primo dataset.

1.2.4. Cambiamento del formato della data

Con il fissato obiettivo di voler aggregare le righe presenti nei due dataset e, successivamente, individuare le osservazioni uguali, l'attenzione è stata spostata su una serie di operazioni atte ad omogeneizzare le diverse entità. La prima operazione è stata la modifica

sul formato della data, descrivente l'entità **review_data**. Infatti, il primo dataset presentava un formato della data *giorno-mese-anno* tipicamente inglese (ad esempio 8th May 2019) che differiva dal secondo dataset, che rappresentava il tempo con *anno-mese-giorno* facendo riferimento alla convenzione internazionale ISO 8601 (ad esempio 2015-04-10). Per questo motivo, il primo dataset ha subito un adeguamento sul formato della data relativa al secondo. Il risultato ottenuto, presentava un formato *anno-mese-giorno* su ambedue i set di dati.

1.2.5. Rimozione e modifica delle colonne

Successivamente, il bilanciamento sui dati ha previsto la sostituzione dell'entità **recommended** con valori numerici di 0 e 1, in modifica rispettivamente alle stringhe “no” e “yes” che la contrassegnavano precedentemente. Vista anche la disparità sul numero di colonne, è stata effettuata una rimozione delle entità ritenute non significative ai fini del progetto. In particolare nel primo dataset è stata eliminata la colonna **date_flown**, mentre nel secondo sono state eliminate le voci **link**, **title** e **autor_country**, che rappresentavano oggetti irrilevanti. Prima dell'unione dei due set di dati, l'ultima operazione è stata quella di ordinare le colonne secondo una certa disposizione, per ottenere una determinata coerenza che ne consentisse l'aggregazione.

1.2.6. Fusione dei dataset

Avere un set di dati di grandi dimensioni è cruciale per le prestazioni del modello di apprendimento. Questa consapevolezza ha portato al raggiungimento dell'obiettivo fissato durante la prima fase di scelta dei due dataset, ovvero alla fusione dei dati. La conseguenza, però, è stata quella di avere un dataset finale con record di dati sporchi, contrassegnati dalla esistenza di righe identiche sulle voci che formano il dataset.

Per rendere il set di dati coerente, è stata eseguita l'identificazione delle parti che presentavano entità doppie, procedendo alla pulizia di quest'ultime. In particolare, dopo aver verificato che i nomi che contraddistinguono le diverse voci fossero esattamente gli stessi, sono stati rimossi tutti i duplicati, raggiungendo la dimensione totale di 79.576 righe ed un totale di 17 colonne.

Tutte gli interventi precedentemente realizzati sono stati indispensabili per il “merge” dei due set di dati, ma l'applicazione del dataset finale, nel contesto del progetto di analisi dei dati statistici, ha imposto l'eliminazione di altre colonne che sono state ritenute non vincolanti nella messa in atto delle tecniche di regressione e classificazione. Le colonne rimosse sono: **author**, **review_date**, **customer_review**, **aircraft** e **route**, riducendo a 12 il numero delle variabili.

Il dataset finale è così composto:

airline	traveller type	class	overall	seat comfort	cabin service
---------	----------------	-------	---------	--------------	---------------

food beverage	entertain_ ment	ground service	wifi connectivity	value for money	recommen_ ded
------------------	--------------------	-------------------	----------------------	--------------------	------------------

1.2.7. Sostituzione dei dati mancanti

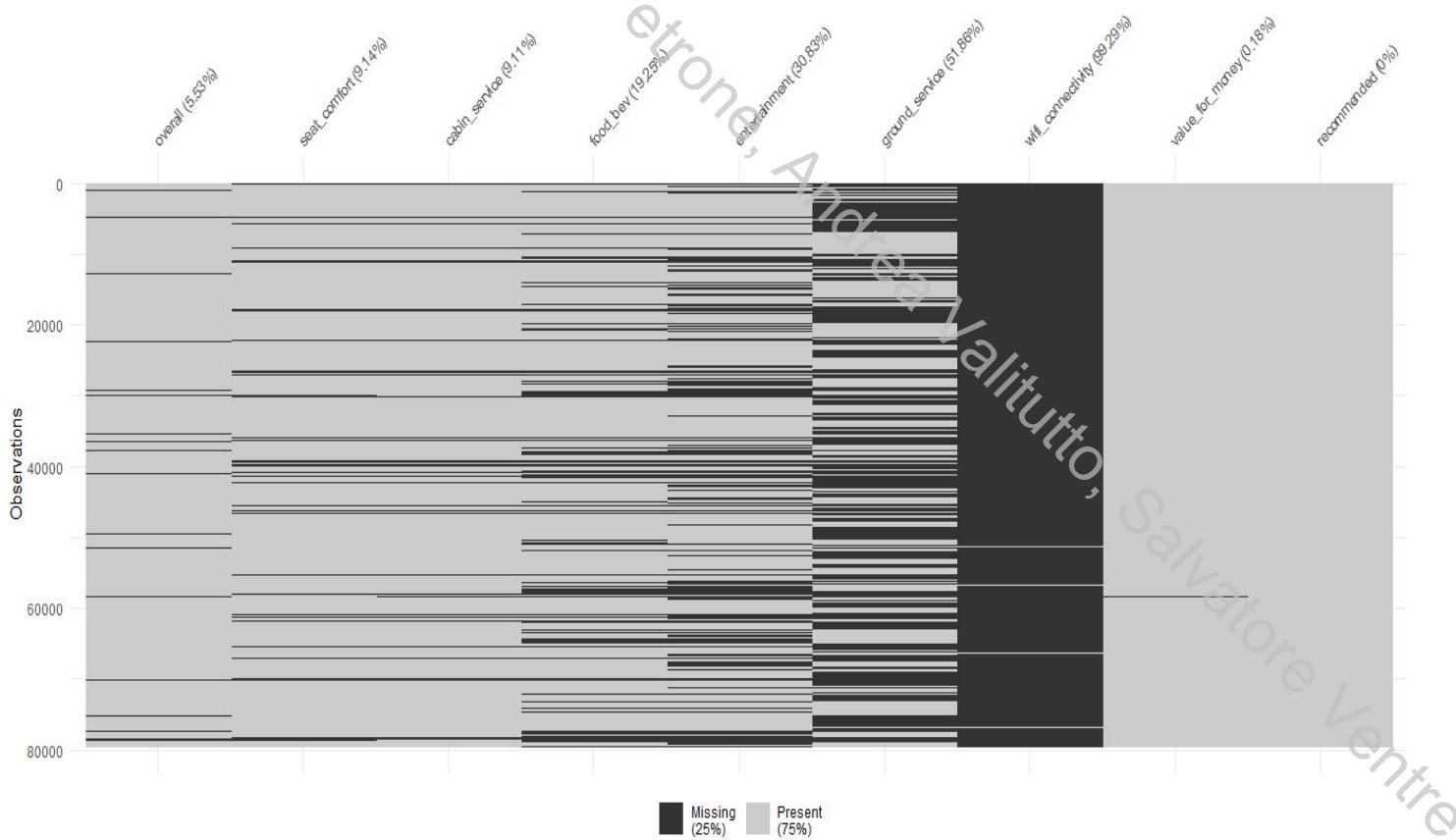
Le stime effettuate attraverso l'utilizzo dei tool di sviluppo, prendono in considerazione soltanto i record del dataset in cui tutti i suoi attributi sono completi. Poiché nel set di dati abbiamo solamente 507 record completi, l'ultimo intento è stato quello di voler migliorare ulteriormente le osservazioni presenti nel dataset, aggiungendo nei campi mancanti delle informazioni correlate ai valori della riga. Prima di iniziare l'analisi dei dati, è bene controllare la quantità di dati mancanti per constatare il numero di record che possono distorcere il risultato finale. Il dataset in questione era caratterizzato da valori numerici uguali allo zero (0) per tutte quelle voci assenti nei diversi campi di valutazione. Siccome questi valori coprivano gran parte del dataset, sono stati presi in esame tre differenti approcci per la gestione dei dati mancanti:

- **uso di casi completi:** rappresenta la soluzione più semplice e può essere raggiunta limitando l'analisi all'insieme di record pienamente osservati. Il vantaggio di questa soluzione è che può essere implementata molto facilmente, ma tuttavia ha come conseguenza quella di perdere una notevole quantità di dati con ripercussioni sulla precisione degli stimatori. Il dataset in questione, con una soluzione di questo tipo, sarebbe ridotto a 507 righe.
- **rimozione dei predittori con molti dati mancanti:** questa soluzione risulta utile nel caso in cui la maggior parte dei dati mancanti sia concentrata in un predittore. Osservando il dataset, il predittore con molti dati mancanti è rappresentato da **wifi_connectivity**, dunque l'uso di questo metodo avrebbe imposto l'eliminazione solo di questa colonna.
- **sostituzione per i valori mancanti:** l'idea è di sostituire le osservazioni mancanti sulla risposta o i predittori con valori artificiali che tentano di preservare la struttura del set di dati:
 - è possibile usare un modello predittivo per prevedere la risposta mancante, quindi creare un nuovo set di dati completamente osservato contenente le previsioni, anziché i valori mancanti. Infine rivalutare il modello predittivo in questo set di dati espanso.
 - sostituire i dati mancanti con la media campionaria dei casi osservati (nel caso di variabili quantitative). Questo rappresenta il metodo più sofisticato e si basa sull'uso di modelli predittivi (contenuti ad esempio nel pacchetto `mice`). Questo approccio è interessante se i dati contengono molte voci NA sparse nei diversi predittori, rendendo inefficiente, ad esempio, l'analisi dei dati completi.

L’approccio utilizzato è stato l’ultimo illustrato, con l’impiego del pacchetto MICE (Multivariate Imputation via Chained Equations), usato comunemente dagli utenti *R*. Il procedimento MICE presume che i dati mancanti siano casuali (MAR, *Missing at Random*), il che significa che la probabilità che manchi un valore dipende solo dal valore osservato. Per predire i valori mancanti, nel nostro caso, è stato utilizzato il PMM (*Predictive Mean Matching*), ovvero un metodo di imputazione con lo scopo di ridurre la distorsione su un set di variabili numeriche.

Nella figura successiva, è visibile in colorazione scura l’insieme di record mancanti che contraddistinguono l’intero dataset. Questo insieme rappresenta il 25% del totale ed è senz’altro determinante ai fini del progetto. Applicando la trasformazione PMM, il risultato raggiunto ha visto un dataset pienamente osservato, caratterizzato da assenza di valori nulli, grossolani o incompleti.

L’ultimo step di questa fase di preprocessing è stato quello di creare un nuovo file “.csv” contenente tutti i dati sopra citati. Il file contenente il dataset completo è stato chiamato preprocessed_complete.csv e da questo momento in poi è stato preso in considerazione questo set per effettuare tutte le successive analisi di regressione e classificazione.



2. Progetto Python

In questa sezione verranno analizzati i risultati ottenuti tramite l'applicazione di diversi metodi di classificazione. Per lo sviluppo di questa parte, è stato utilizzato l'ambiente Colab, reso disponibile da Google. Tale ambiente supporta l'esecuzione di codice Python, un linguaggio che mette a disposizione diverse librerie dedicate all'analisi statistica.

L'obiettivo preposto è stato quello di predire e stimare il parametro (o *classe*) **recommended** (rappresentata da uno 0 per i voli recensiti come non consigliati e da un 1 per i voli recensiti come consigliati) utilizzando come parametri di classificazione tutti quelli risultanti dalla fase di preprocessing dei dati, ovvero: **seat_comfort**, **cabin_service**, **food_bev**, **entertainment**, **ground_service**, **wifi_connectivity**, **value_for_money**.

Per la classificazione, sono stati utilizzati i metodi analizzati durante il corso, quali **Naïve Bayes**, **KNN (K-Nearest Neighbors)**, **Logistic Regression** e **Naïve Kernel**. Per gli ultimi tre, è stato utilizzato anche il framework Apache Spark il quale, in generale, permette di parallelizzare dei compiti. Ciò ha richiesto un'implementazione manuale dei suddetti algoritmi, le quali funzioni sono state appunto parallelizzate attraverso Spark.

2.1. Preprocessing

Prima di procedere all'analisi dei dati, è stato necessario effettuare un'operazione di normalizzazione, successiva alle operazioni di preprocessing preliminari descritte nel [capitolo precedente](#). Tale operazione è stata necessaria in quanto i valori presenti nella colonna **overall** erano in scala da 1 a 10, mentre quelli presenti nelle altre colonne erano in scala da 1 a 5. È stato quindi deciso di portare tutti i valori in una scala da 0 a 1. Segue una figura che rappresenta le prime righe del dataset.

airline	traveller_type	class	overall	seat_comfort	cabin_service	Food_bev	entertainment	ground_service	wifi_connectivity	value_for_money	recommended
adria airways	NaN	Economy	0.5	0.2	0.8	0.8	0.2	1.0	0.2	0.8	1
adria airways	NaN	Economy	0.9	0.8	1.0	0.8	0.6	0.8	0.6	0.8	1
adria airways	NaN	Economy	0.5	0.6	0.8	0.6	0.8	0.6	0.8	0.8	1
adria airways	NaN	Economy	0.7	0.8	0.8	0.8	0.6	0.6	0.8	0.8	1
adria airways	NaN	Economy	0.7	0.6	0.4	0.2	0.2	0.8	0.2	0.2	1

2.2. Analisi preliminare

Prima di applicare effettivamente i metodi di classificazione, è stata effettuata un'analisi qualitativa del dataset, per verificare che i dati fossero convincenti.

2.2.1. Bilanciamento del dataset

Come primo step è stata calcolata la frequenza delle due classi, per verificare che il dataset non fosse eccessivamente sbilanciato verso una classe. È risultato che il 50,41% dei dati appartiene alla classe 0, mentre naturalmente il restante 49,59% dei dati appartiene alla classe 1. Dunque è stato valutato che effettivamente il dataset non risulta sbilanciato.

2.2.2. Parametri statistici dei dati

Per consentire la generazione di un dataset sintetico, la cui analisi fornisce un supporto all'analisi effettuata sul dataset reale, sono stati estrapolati i parametri statistici dei dati. In particolare, sono state calcolate la media e la varianza delle feature, e la covarianza tra coppie di feature. Questi parametri sono stati valutati sia per i dati appartenenti alla classe 0 sia per i dati appartenenti alla classe 1. Segue una figura che rappresenta la visualizzazione della media e la varianza delle feature **overall** e **value_for_money** sia per i voli raccomandati sia per i voli non raccomandati. Come ci può aspettare, queste feature assumono valori più alti nelle recensioni di voli raccomandati.

```
For recommended airlines, the overall mean is 0.8334 with variance 0.0233
For not recommended airlines, the overall mean is 0.2150 with variance 0.0231

For recommended airlines, the value_for_money mean is 0.8618 with variance 0.0237
For not recommended airlines, the value_for_money mean is 0.3431 with variance 0.0368
```

Segue una figura che rappresenta la matrice di covarianza tra le feature.

	overall	seat_comfort	cabin_service	food_bev	entertainment	ground_service	wifi_connectivity	value_for_money
overall	0.118813	0.074305	0.085989	0.077593	0.068729	0.089433	0.054163	0.095058
seat_comfort	0.074305	0.078800	0.058519	0.055453	0.053168	0.058202	0.042551	0.064142
cabin_service	0.085989	0.058519	0.095642	0.066581	0.054562	0.067194	0.045567	0.071542
food_bev	0.077593	0.055453	0.066581	0.085243	0.056798	0.058861	0.044595	0.065849
entertainment	0.068729	0.053168	0.054562	0.056798	0.088913	0.053782	0.058417	0.058154
ground_service	0.089433	0.058202	0.067194	0.058861	0.053782	0.100955	0.047282	0.074927
wifi_connectivity	0.054163	0.042551	0.045567	0.044595	0.058417	0.047282	0.085062	0.047567
value_for_money	0.095058	0.064142	0.071542	0.065849	0.058154	0.074927	0.047567	0.097611

I dettagli sull'effettiva generazione del dataset sintetico sono riportati nel [paragrafo successivo](#).

2.3. Generazione di un dataset sintetico

Per l'analisi dei dati, è stato dapprima utilizzato un approccio basato su un dataset sintetico, al fine di valutare (anche visivamente) la distribuzione dei dati all'interno del dataset. Esso è stato particolarmente utile in quanto i dati non sono distribuiti in maniera continua, bensì in maniera discreta; infatti, i valori di **overall** sono del tipo $\{0.1, 0.2, \dots, 1.0\}$, mentre tutti gli altri sono del tipo $\{0.2, 0.4, 0.6, 0.8, 1.0\}$.

Il dataset sintetico è stato ricavato da solo due campi del dataset reale. I campi selezionati possono essere modificati cambiando il valore assegnato alle variabili nel codice sorgente. Di default, sono selezionate le colonne **overall** e **value_for_money**. Per generare tale dataset si è supposto che la distribuzione dei valori seguisse una multivariata normale; sono stati quindi calcolati i valori di media, varianza e covarianza a partire dal dataset reale come descritto nel [paragrafo precedente](#). Utilizzando il metodo `multivariate_normal` messo a disposizione dalla libreria `numpy`, vengono generati alcuni sample secondo tale

distribuzione. Poiché i dati del dataset reale sono discreti, non è stato possibile rappresentarli in maniera chiara, poiché erano presenti tutte le possibili coppie di valori associati ad una classe (per fare un esempio banale, il punto di coordinate (**overall**, **value_for_money**)=(0.5, 0.6) può appartenere sia alla classe 0 sia alla classe 1). Ciò rendeva particolarmente difficile apprezzare la distribuzione dei dati. Utilizzando un dataset sintetico continuo, invece, è stato possibile graficare la distribuzione dei dati rendendone immediata la comprensione.

2.4. Naïve Bayes

Il classificatore Naïve Bayes utilizza il teorema di Bayes per calcolare la probabilità di osservare una classe date le osservazioni delle feature, vale a dire la *probabilità a posteriori*; dopodiché applica il criterio di *massima probabilità a posteriori*, abbreviato come MAP (Maximum A Posteriori), vale a dire che è scelta quella classe che appunto massimizza la probabilità a posteriori.

In particolare, il classificatore suppone che le feature siano tra loro indipendenti, dunque la distribuzione congiunta delle feature condizionata all'osservazione di una classe è calcolata come il prodotto delle distribuzioni marginali delle singole feature.

In sostanza, dunque, il classificatore Naïve Bayes è caratterizzato dall'*ipotesi di indipendenza*.

2.4.1. Applicazione del classificatore Naïve Bayes sul dataset sintetico

Il primo metodo di classificazione utilizzato è stato il classificatore Naïve Bayes. Per verificare che l'ipotesi di indipendenza fosse adatta ai dati (nel caso contrario gli errori di classificazione sarebbero stati molto frequenti), sono state graficate le distribuzioni relative alle feature **overall** e **value_for_money** per entrambe le classi. Il primo grafico rappresenta la distribuzione delle due feature sul dataset sintetico, mentre il secondo la distribuzione in cui si è data per vera l'ipotesi di indipendenza.

È possibile notare come le due distribuzioni siano facilmente separabili in entrambi i casi e che la correlazione tra le feature **overall** e **value_for_money** non è estremamente rilevante: infatti, la distribuzione in accordo all'ipotesi di indipendenza non differisce molto dalla distribuzione secondo i parametri statistici reali. In entrambi i grafici, la curva sul lato destro rappresenta la distribuzione dei sample di classe 1, mentre sulla sinistra quella dei sample di classe 0. Si nota che per bassi valori di **overall** e **value_for_money**, la classe **recommended** è pari a 0, mentre per alti valori, tale classe è 1.

Si fa inoltre notare che, osservando la matrice di covarianza, affermare che l'ipotesi di indipendenza tra le feature possa essere assunta come vera non rappresenta un'eccessiva forzatura, dato che il valore di covarianza tra coppie di feature è sempre inferiore a 0.1. In particolare, la covarianza tra le due feature scelte, **overall** e **value_for_money**, è la più alta di tutte, vale a dire 0,095058.

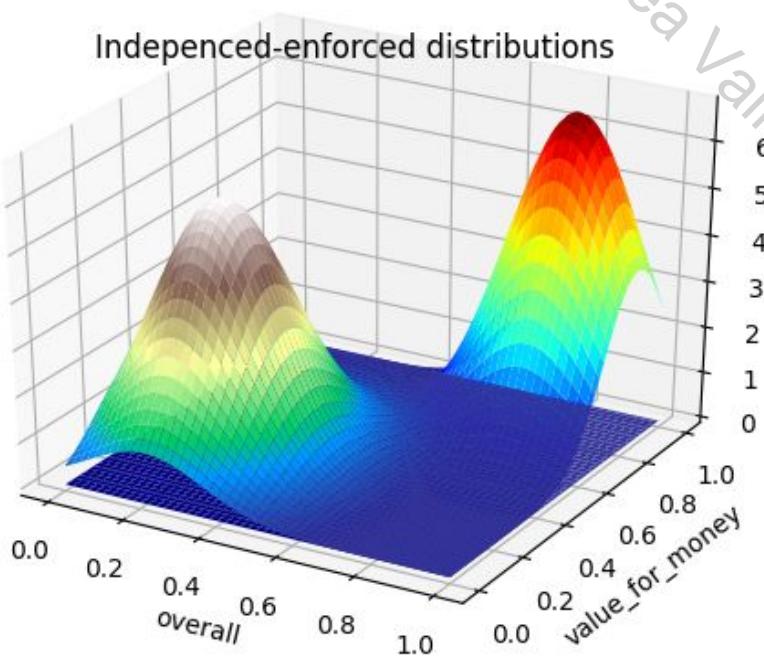
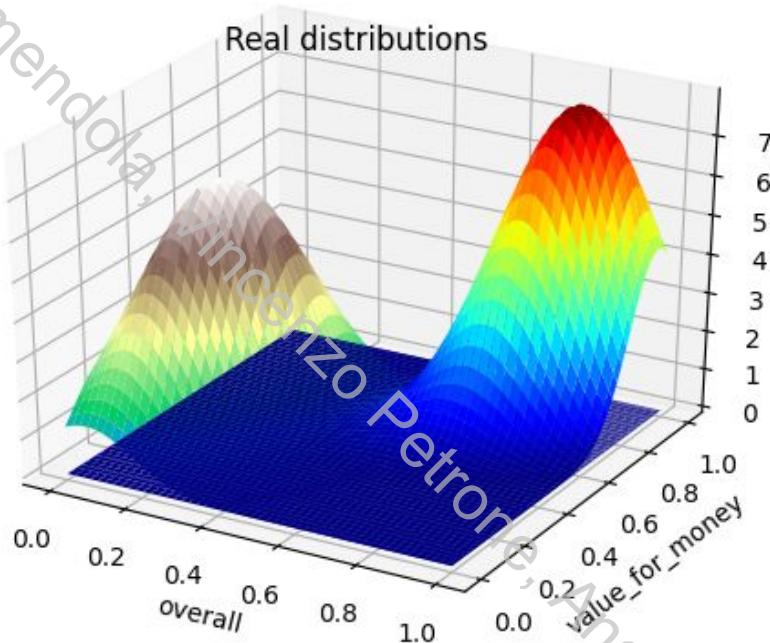
La paletta di colori utilizzata per rappresentare la classe 0 è la seguente:



La paletta di colori utilizzata per rappresentare la classe 1 è la seguente:



Seguono due figure che rappresentano le distribuzioni sopra descritte.



In questo caso, la percentuale di corretta classificazione è stata del 97,81%, come riporta la figura seguente:

```
Performances...  
  
Confusion matrix:  
[[11829  306]  
 [ 216 11522]]  
  
Classifier performance metrics:  
      precision    recall   f1-score   support  
0.0          0.98     0.97     0.98     12135  
1.0          0.97     0.98     0.98     11738  
accuracy                           0.98     23873  
macro avg       0.98     0.98     0.98     23873  
weighted avg    0.98     0.98     0.98     23873  
  
final performance: 97.81%
```

2.4.2. Applicazione del classificatore Naïve Bayes sul dataset reale

Il classificatore Naïve Bayes, la cui implementazione è disponibile nella libreria di Python `sklearn`, è stato applicato al dataset reale in più modalità:

- Con le sole due feature **overall** e **value_for_money**
- Una feature per volta
- Con tutte le possibili coppie di feature
- Con tutte le feature disponibili

Utilizzo di due feature

Utilizzando la porzione del dataset reale composta unicamente dalle feature **overall** e **value_for_money** i risultati sono stati leggermente inferiori rispetto a quelli riscontrati utilizzando il dataset sintetico, come riporta la figura seguente:

```

Performances...

Confusion matrix:
[[11455  541]
 [ 499 11378]]

Classifier performance metrics:
      precision    recall   f1-score   support
0        0.96     0.95     0.96    11996
1        0.95     0.96     0.96    11877

accuracy                           0.96    23873
macro avg      0.96     0.96     0.96    23873
weighted avg    0.96     0.96     0.96    23873

final performance: 95.64%

```

Utilizzo di una feature per volta

Al fine di comparare l'impatto di ogni feature sulla percentuale di corretta classificazione, il classificatore Naïve Bayes è stato testato ciclicamente sulla porzione del dataset composta da ciascuna singola feature, ottenendo i risultati riportati nella tabella seguente:

overall	seat_comfort	cabin_service	food_bev	entertainment	ground_service	wifi_connectivity	value_for_money
95.33%	82.30%	86.08%	81.62%	7.03%	84.48%	73.90%	91.01%

Come era prevedibile, si può notare come i parametri più significativi per la valutazione di un volo come raccomandato o meno sono quelli di **overall** e **value_for_money**, mentre quelli che forniscono meno precisione sono **wifi_connectivity** e **entertainment**.

Utilizzo di coppie di feature

Come ulteriore analisi, il classificatore Naïve Bayes è stato testato ciclicamente su tutte le possibili coppie di feature, ottenendo i risultati riportati nelle tabelle seguenti:

overall + seat_comfort	overall + cabin_service	overall + food_bev	overall + entertainment
94.85%	95.00%	95.07%	94.64%

overall + ground_service	overall + wifi_connectivity	overall + value_for_money	seat_comfort + cabin_service
94.64%	95.20%	95.53%	89.10%

seat_comfort + food_bev	seat_comfort + entertainment	seat_comfort + ground_service	seat_comfort + wifi_connectivity
86.31%	83.51%	88.25%	84.96%

seat_comfort + value_for_money	cabin_service + food_bev	cabin_service + entertainment	cabin_service + ground_service
91.78%	87.53%	87.17%	89.64%

cabin_service + wifi_connectivity	cabin_service + value_for_money	food_bev + entertainment	food_bev + ground_service
87.37%	92.45%	83.92%	88.97%

food_bev + wifi_connectivity	food_bev + value_for_money	entertainment + ground_service	entertainment + wifi_connectivity
85.30%	91.72%	87.32%	79.48%

entertainment + value_for_money	ground_service + wifi_connectivity	ground_service + value_for_money	wifi_connectivity + value_for_money
91.50%	87.20%	91.87%	91.53%

Come ulteriore conferma, il risultato più alto è stato ottenuto utilizzando congiuntamente le feature **overall** e **value_for_money**.

Utilizzo di tutte le feature disponibili

Come ultima analisi, il classificatore Naïve Bayes è stato testato utilizzando congiuntamente tutte le feature disponibili, con il seguente risultato:

```
Performances ...

Confusion matrix:
[[11521  562]
 [ 575 11215]]

Classifier performance metrics:
      precision    recall   f1-score   support
          0         0.95     0.95     0.95     12083
          1         0.95     0.95     0.95     11790
   accuracy                           0.95     23873
  macro avg       0.95     0.95     0.95     23873
weighted avg     0.95     0.95     0.95     23873

final performance: 95.24%
```

Si nota come i risultati sono leggermente inferiori rispetto a quelli ottenuti utilizzando le due uniche feature **overall** e **value_for_money**.

2.5. KNN

Il classificatore K-Nearest Neighbor sfrutta come idea di base il fatto che dati appartenenti ad una stessa classe siano raggruppati tra loro e divisi da quelli appartenenti ad altre classi. Dunque, un nuovo dato x verrà assegnato alla classe più frequente tra i K dati più vicini a x . Quindi, un *iperparametro* di questo classificatore è appunto il numero di vicini K da utilizzare nell'algoritmo.

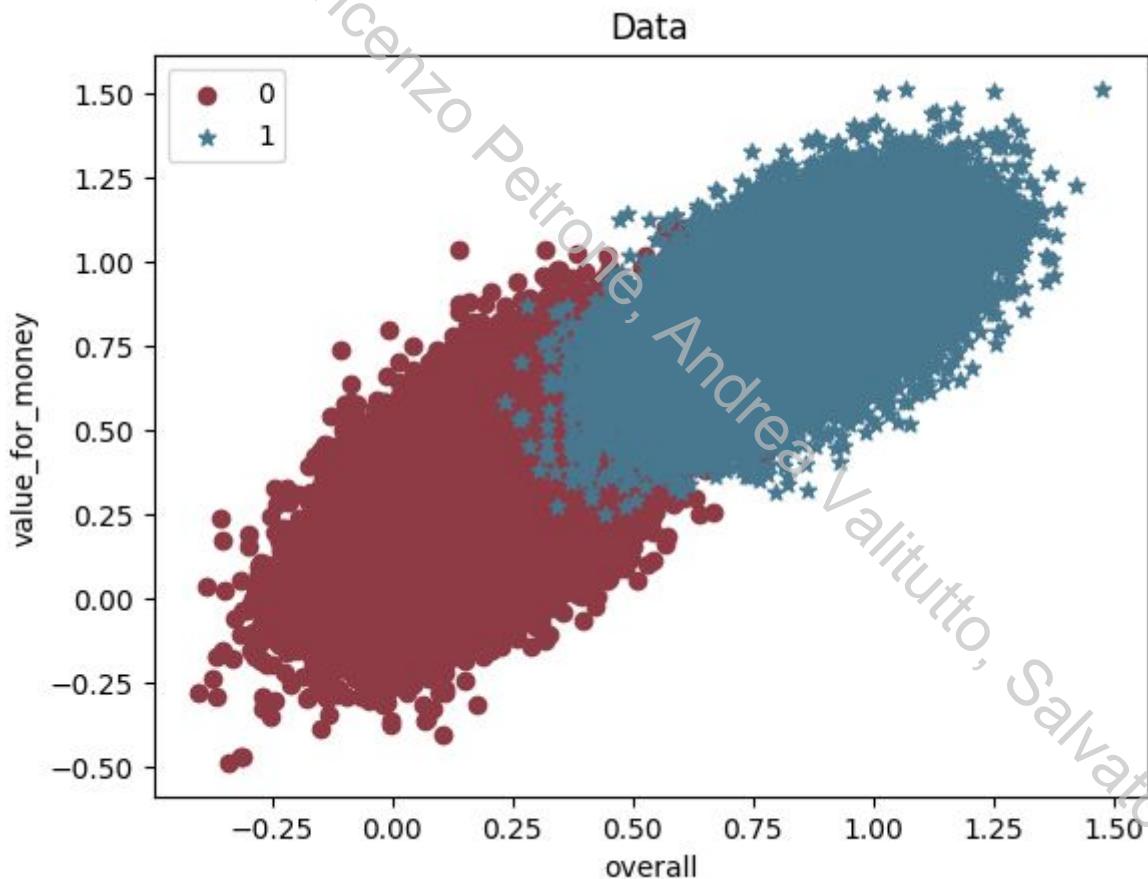
La definizione di distanza varia a seconda dei casi e scelte possibili sono la distanza euclidea (vale a dire quella in particolare utilizzata in questa analisi), la distanza di Minkowski (generalizzazione della distanza euclidea) o la distanza di Manhattan. Dopo aver ottenuto tali distanze, il nuovo sample viene assegnato alla classe più frequente tra i K elementi più vicini.

Tale metodo di classificazione è piuttosto oneroso da un punto di vista computazionale e richiede la regolazione dell'*iperparametro* K. A tal fine è stata eseguita una partizione del dataset in 3 sezioni disgiunte:

- Training Set: partizione dalla quale estrarre i K elementi più vicini a un nuovo sample
- Test Set: per la valutazione finale delle performance in termini di percentuale di corretta classificazione
- Validation Set: per valutare il miglior parametro K
 - In particolare, il K migliore è quel K tale per cui l'errore quadratico medio (*mean squared error*, MSE) sulle stime dei dati appartenenti al validation set è minore

2.5.1 Applicazione del classificatore KNN sul dataset sintetico

Anche in questo caso, prima di procedere alla fase di fitting del classificatore, è stata mostrata graficamente la distribuzione dei sample su di un grafico di tipo scatter:



In questo grafico relativo al dataset sintetico, viene mostrato come i punti del dataset sono distribuiti rispetto ai parametri **overall** e **value_for_money**. I punti in rosso rappresentano gli elementi appartenenti alla classe 0, mentre quelli in blu sono gli elementi appartenenti alla classe 1. Si nota come le due classi sono sovrapposte solo per pochi sample (considerando

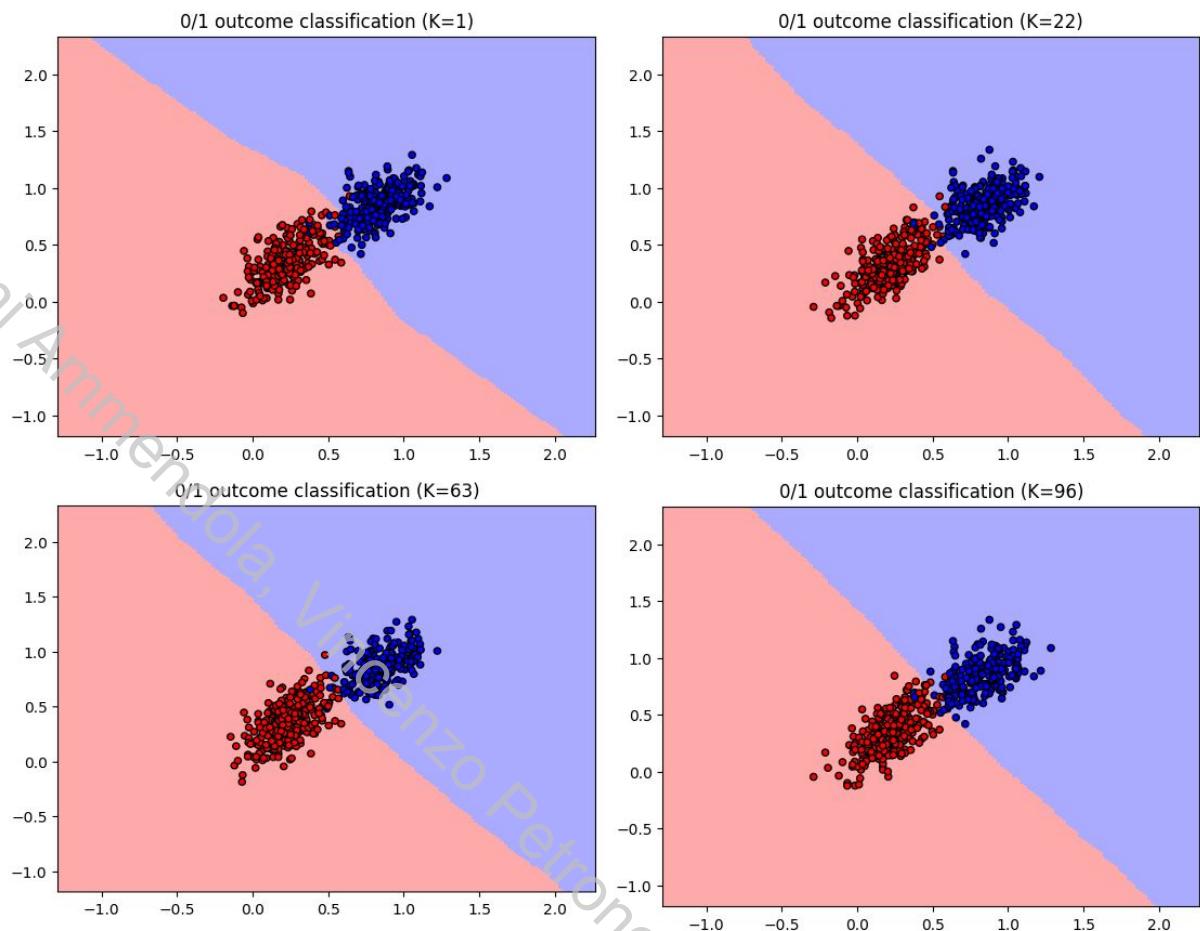
che la dimensione del dataset è di circa 80.000 elementi) e, come sarà evidenziato dopo, le due classi sono facilmente separabili (al netto di un basso errore di classificazione).

Per il fit del modello KNN sono state create due funzioni principali:

- una funzione dedicata alla regolazione dell'iperparametro K, la quale prende in ingresso, oltre al dataset, il numero di valori di K da testare e il relativo range
- una funzione dedicata al fit del modello che prende in ingresso, oltre al dataset, il valore di K, passato ciclicamente dalla funzione dedicata al tuning

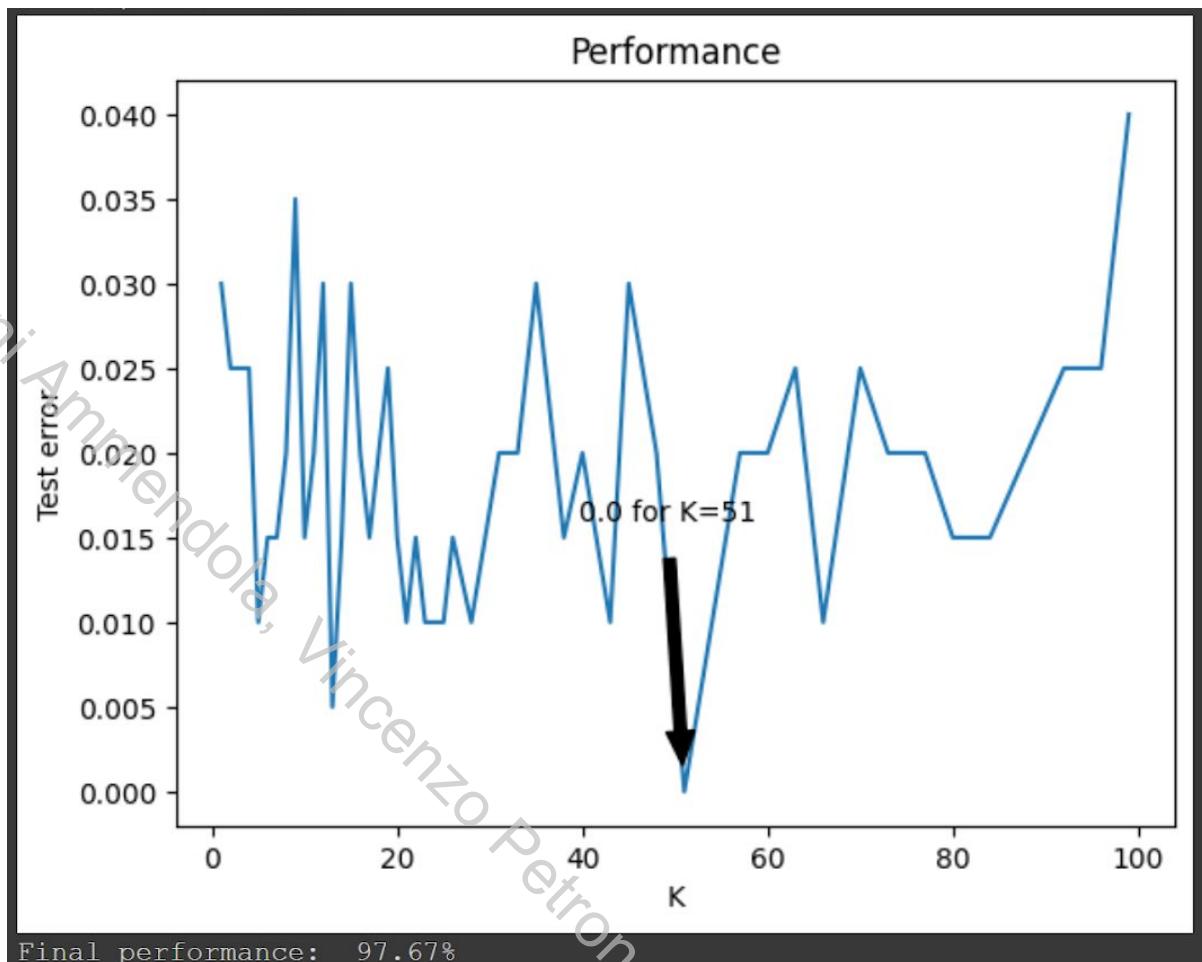
La vera fase di training del modello avviene nella funzione di fitting, dove vengono utilizzate le funzioni messe a disposizione dalla libreria `sklearn`. Tale funzione, inoltre, consente di utilizzare anche il framework Spark, che consente di parallelizzare il calcolo e la ricerca degli elementi più vicini. Tale framework verrà analizzato nel [paragrafo successivo](#).

Come già detto, il calcolo dei K elementi più vicini e quindi il fit e test del modello KNN è un'operazione computazionalmente complessa. Inoltre, per testare le capacità del modello implementato manualmente al fine di essere utilizzato con Spark, ogni operazione viene effettuata due volte (calcolo della distanza, trovare i K elementi più vicini, calcolo della classe più frequente). Dunque, per ridurre i tempi di elaborazione e per avere una visualizzazione più chiara, solo per il dataset sintetico si è deciso di utilizzare un set ridotto di 1.000 elementi, che permette comunque di avere un'idea generale del comportamento dell'algoritmo KNN. Si riportano di seguito parte dei risultati ottenuti da una regolazione del parametro K per 50 valori da 1 a 100 (per brevità vengono riportati solo alcuni esempi):



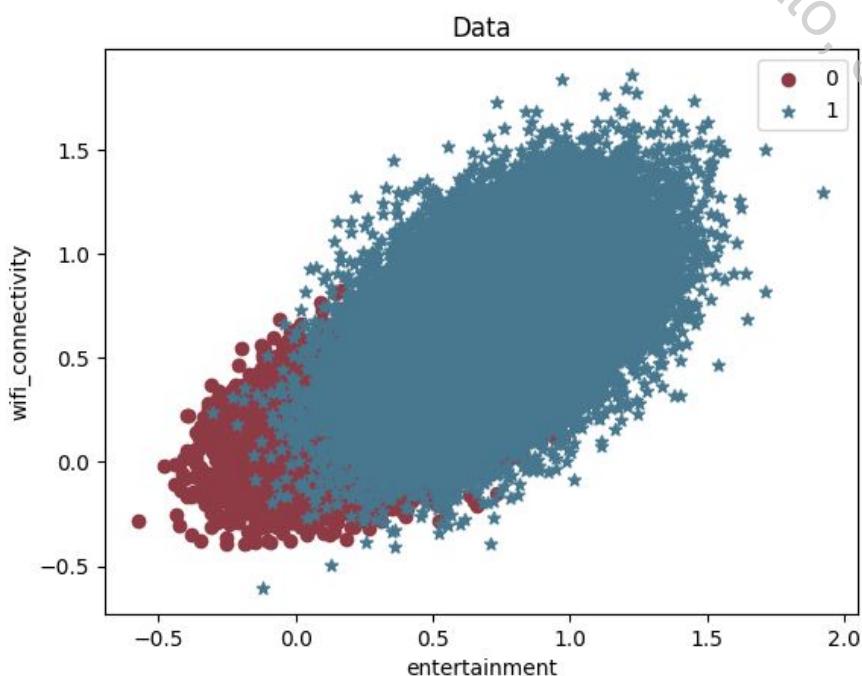
È possibile notare come per valori piccoli di K l'algoritmo vada a seguire molto precisamente la divisione tra le due nuvole di punti. Per valori di K più grandi, invece, si nota una divisione molto più approssimata, tendente ad una divisione lineare delle due classi.

In tutti i casi, comunque, si può notare che il modello riesce comunque a separare correttamente le due regioni. Infatti, oltre ai grafici delle regioni di decisione, lo script grafica anche l'andamento delle prestazioni del modello, in base al valore dell'MSE, indicando quale sia il valore di K che restituisce il risultato migliore. Di seguito viene riportato tale grafico.

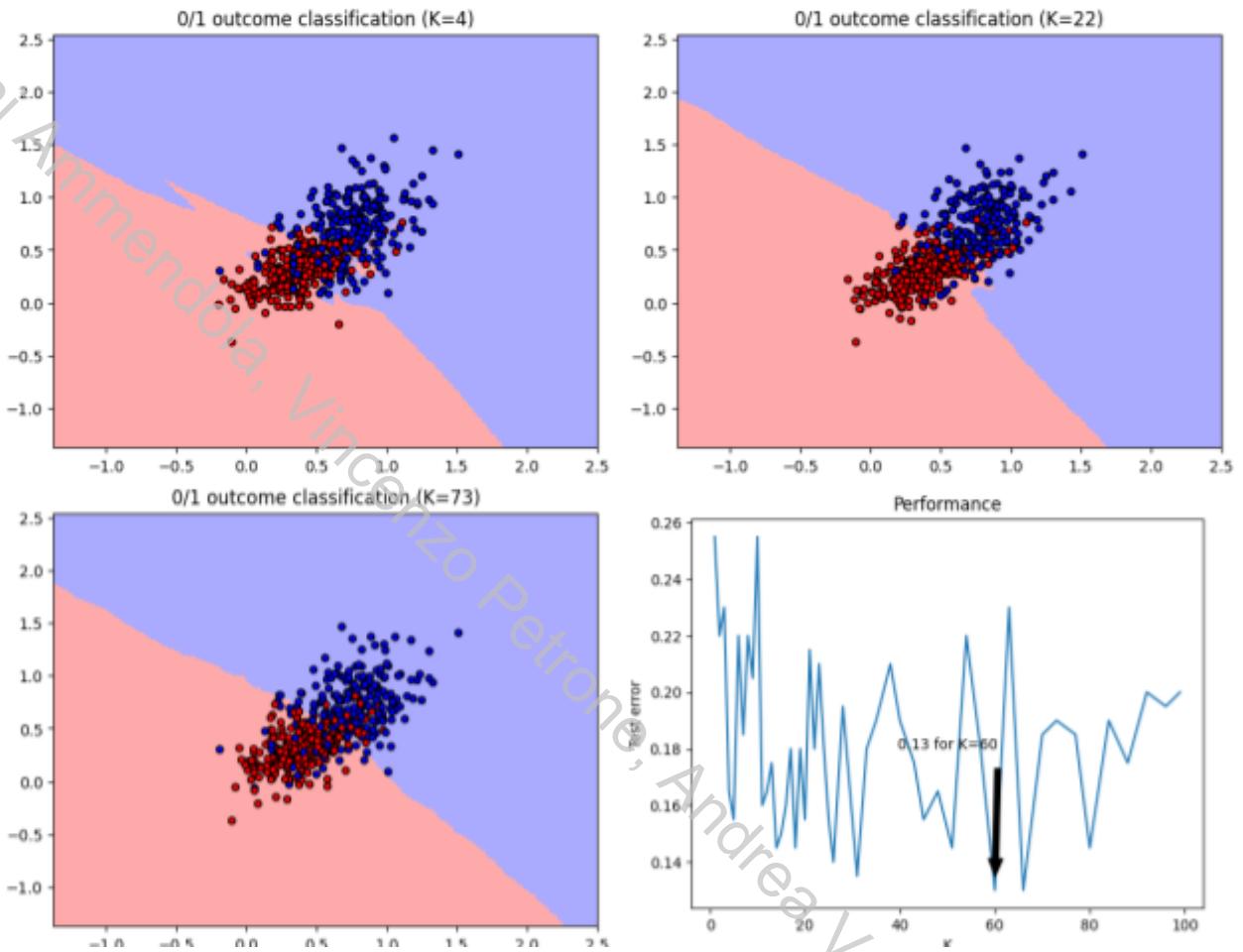


Si fa notare che le variazioni di MSE tra i diversi valori di K non sono così elevate. Infatti, il test error oscilla intorno al valore 0,02, raggiungendo picchi massimi di 0,035 e azzerandosi per K pari a 51. Sono stati eseguiti più test con dataset diversi, ed i risultati ottenuti sono in linea con questo grafico, eccetto per leggere variazioni statistiche.

Tale discorso, però, non è del tutto valido per altre coppie di feature, come, ad esempio, **wifi_connectivity** ed **entertainment**. Infatti, graficando uno scatter plot delle due classi in funzione di questi parametri, le due regioni diventano molto sovrapposte.



Ovviamente, tale sovrapposizione rende più complessa la divisione delle due nuvole di punti, causando molti più errori durante la classificazione. Vengono ora riportati alcuni modelli di KNN con valori di K diversi e l'andamento dell'errore in funzione di K:



Si può notare come l'errore sia più alto rispetto al caso precedente proprio a causa della sovrapposizione dei punti.

Al termine di questa analisi sintetica, è possibile trarre due importanti conclusioni:

- L'algoritmo KNN è adatto a risolvere questo problema di classificazione restituendo buoni risultati sul database sintetico, specialmente su alcune coppie di features facilmente separabili.
- L'implementazione manuale dell'algoritmo è corretta e del tutto simile all'implementazione nelle librerie `sklearn`, ad esclusione di minori scelte implementative. Infatti, nel caso in cui i risultati ottenuti dalle due implementazioni fossero diversi, si sarebbe dovuto verificare un errore di esecuzione.

2.5.1.1. Spark

Per poter utilizzare correttamente il framework Spark è stato necessario implementare nuovamente le funzioni utilizzate dal KNN, quali quella del calcolo della distanza e della determinazione dei K elementi più vicini. Come funzione di distanza è stato deciso di implementare quella euclidea, mentre la classe di un nuovo sample x viene assegnata secondo quella più frequente tra i K elementi più vicini a x , senza pesare ogni classe in base alla distanza rispetto a x . Tali scelte si sono rispecchiate anche nel fit tramite le funzioni di libreria. Infatti, per poter apprezzare la bontà degli algoritmi implementati, è stato deciso di utilizzare la stessa funzione per il calcolo della distanza e lo stesso metodo di determinazione della classe. Questo ha permesso di essere sicuri del fatto che i due modelli restituiscano gli stessi risultati. Infatti, all'interno della funzione di fit è presente un controllo sui risultati che interrompe l'esecuzione del codice nel caso in cui i risultati siano diversi. Durante i numerosi test sul dataset sintetico, tale evento non è stato mai riscontrato. La parte di codice che gestisce il fit tramite Spark effettua anche alcune operazioni preliminari per il corretto funzionamento del meccanismo di parallelizzazione, come la parallelizzazione del dataset e il broadcast dell'iperparametro K e dei punti del Test Set.

Quando ogni cluster ha terminato il calcolo delle distanze tra i sample della propria partizione e l'elemento x e dopo aver restituito i K elementi più vicini, viene effettuata una nuova parallelizzazione dei risultati per il calcolo della classe più frequente. Dato che il problema di classificazione è binario e che le due classi sono espresse come 0 o come 1, è stato sufficiente eseguire la somma delle classi associate ai K elementi. La somma ottenuta viene quindi confrontata con il valore K: se tale somma è maggiore di $K/2$, x viene assegnato alla classe 1; altrimenti, x viene assegnato alla classe 0

2.5.2. Applicazione del classificatore KNN sul dataset reale

Visti i risultati ottenuti, il classificatore KNN è stato applicato anche al dataset reale utilizzando la sola implementazione di libreria. Infatti, data l'elevata quantità di dati, eseguire il fit più volte sugli stessi dati e con gli stessi parametri avrebbe richiesto una quantità di tempo eccessiva. Avendo quindi dimostrato tramite il dataset sintetico che l'implementazione parallelizzata tramite Spark è completamente funzionante e corretta e, tenendo in considerazione che tale esecuzione parallela non avrebbe portato alcun beneficio, si è deciso di utilizzare esclusivamente l'implementazione di libreria. Si fa comunque notare, che eseguendo il fit del KNN utilizzando anche Spark, il programma potrebbe restituire risultati differenti. Come già scritto, tale evento non si è mai verificato con il dataset sintetico, ma è stato riscontrato più volte con il dataset reale. Se si considera il fatto che il dataset reale è discreto, si può ipotizzare che le diverse scelte implementative possano causare tali divergenze. Si è ipotizzato che durante il calcolo delle distanze, più punti di classi diverse possano accavallarsi, ovvero avere le stesse feature, e quindi restituire la stessa distanza. In tal caso, potrebbe verificarsi che l'algoritmo di libreria scelga un sample di una classe nella

determinazione della classe più frequente, mentre l'implementazione manuale vada a selezionare l'altro sample appartenente ad una classe diversa.

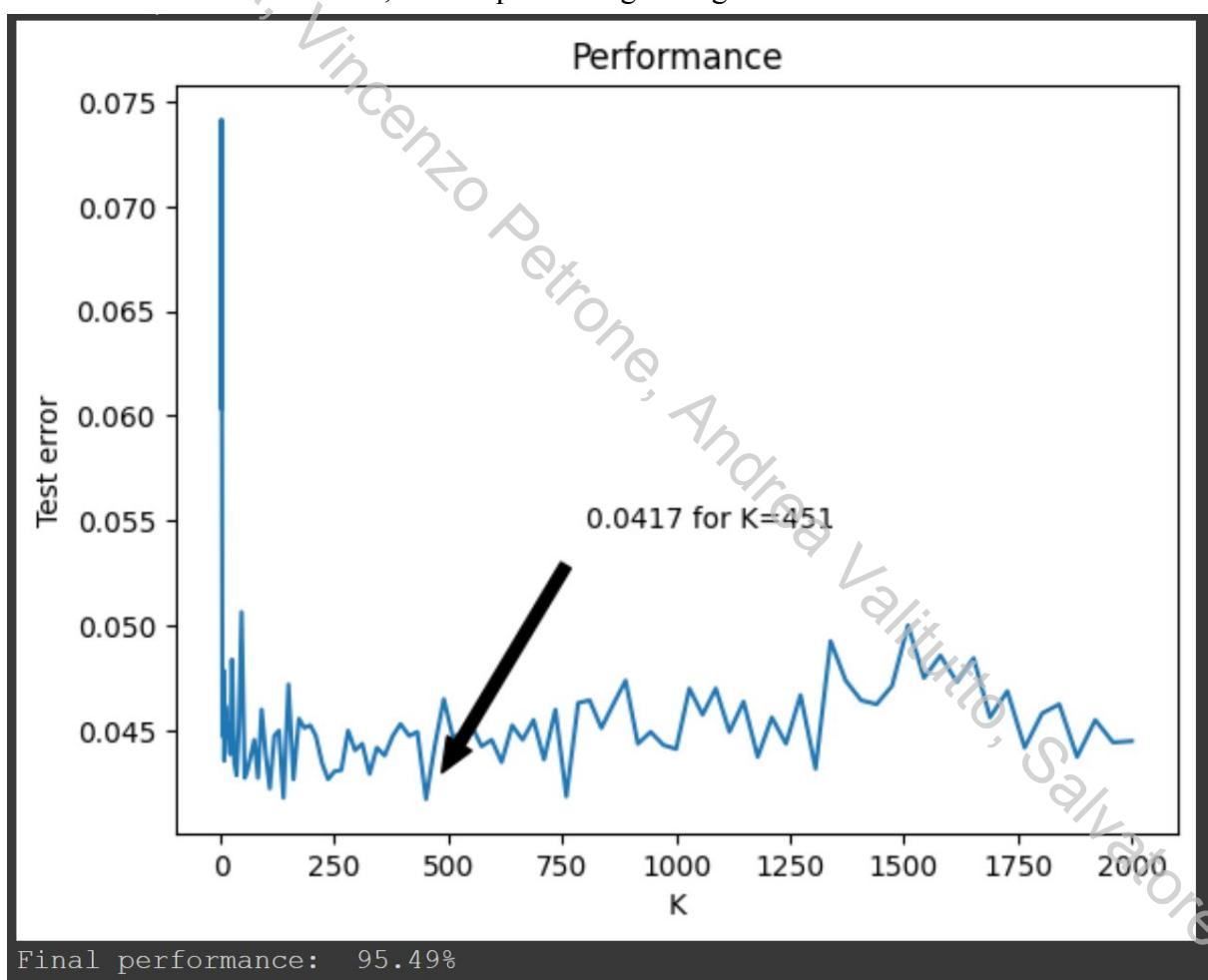
Sono stati provati diversi approcci:

- Con le sole due feature **overall** e **value_for_money**
- Una feature per volta
- Con tutte le feature disponibili

In questo caso non sono state testate tutte le possibili coppie di feature: dato che l'operazione di fit, tuning e test è particolarmente complessa, ciò avrebbe richiesto molte ore di elaborazione.

Utilizzo di due feature

Utilizzando la porzione del dataset reale composta unicamente dalle feature **overall** e **value_for_money** i risultati sono stati leggermente inferiori rispetto a quelli riscontrati utilizzando il dataset sintetico, come riporta la figura seguente:



Si nota che per un numero relativamente piccolo di elementi considerati (il dataset reale è composto da circa 80.000 sample) l'errore sul Test Set è di circa due punti percentuali maggiore rispetto all'errore sul Test Set sintetico. La performance finale del KNN sul dataset reale completo e sulle sole feature **overall** e **value_for_money** è del 95,49%, in linea con quanto atteso dall'andamento del test error a seconda del miglior parametro K.

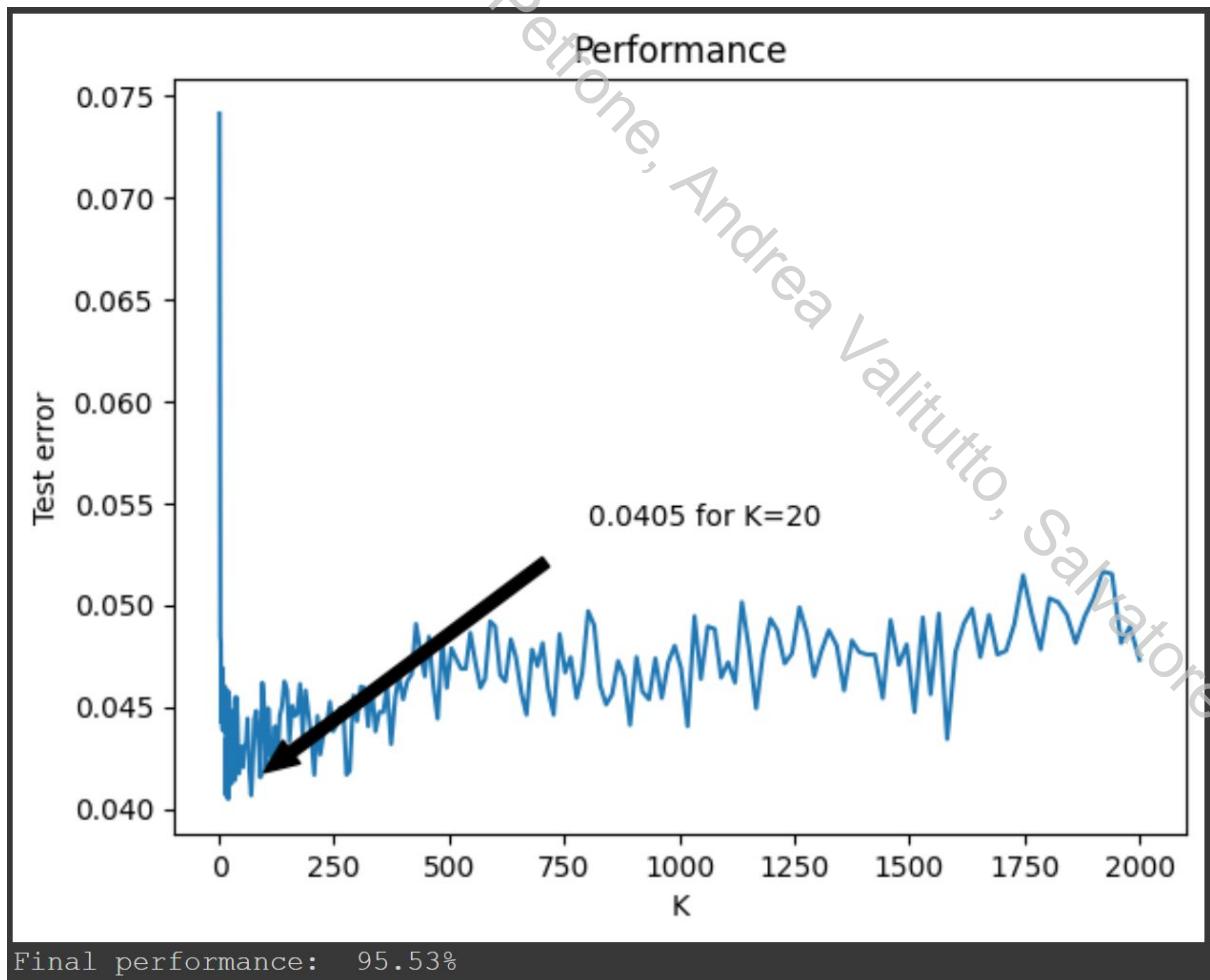
Utilizzo di una feature per volta

Al fine di verificare quali siano le feature che maggiormente caratterizzano gli elementi di una classe, è stato deciso di effettuare un fit su ogni singola feature presente nel dataset. Per brevità verranno riportati solo i risultati finali, escludendo i grafici dell'andamento del test error a seconda del parametro K per ogni feature. Si nota che le feature che offrono una migliore precisione per la predizione siano proprio **overall** e **value_for_money**, in accordo con gli altri metodi provati. I valori di **wifi_connectivity** ed **entertainment**, invece, offrono risultati peggiori, come già osservato nel metodo precedente. A causa della diversa distribuzione dei dati, i valori di K siano molto diversi per ogni feature.

	overall	seat_comfort	cabin_service	food_bev	entertainment	ground_service	wifi_connectivity	value_for_money
K	108	141	311	1241	1306	640	1339	99
MSE	4.17%	17.30%	13.30%	17.20%	21.80%	15.10%	24.40%	8.56%
Score	95.46%	81.90%	85.85%	81.80%	77.13%	84.09%	74.69%	90.75%

Utilizzo di tutte le feature disponibili

Al termine dell'analisi sulle singole feature, l'algoritmo KNN è stato eseguito su tutto il dataset, sfruttando tutte le feature disponibili, ottenendo i seguenti risultati:



Si nota come l'errore sia ridotto per valori piccoli di K e aumenta all'aumentare di K. Mentre le oscillazioni tra valori di K vicini sono dovute alla variabilità dei dati, l'andamento crescente dell'errore ha una natura diversa: con valori di K troppo grandi, si vanno a considerare troppi elementi. Nel caso in cui il nuovo elemento sia circondato da altri sample della sua classe vera, questo verrà correttamente classificato. Se invece il nuovo elemento si trova nell'intersezione tra le diverse nuvole di punti, considerando molti elementi, si vanno a considerare anche quelli dell'altra classe.

2.6. Logistic Regression

Il classificatore binario Logistic Regression si basa nel calcolare la probabilità che un dato appartenga a una classe attraverso la *funzione sigmoide* $\sigma(x) = \frac{1}{1+e^{-x}}$, la funzione inversa della *funzione logit* $\text{logit}(x) = \log(\frac{x}{1-x})$. Assumendo di rappresentare la classe y con 0 o 1, la probabilità che un dato x appartenga a y , sia $P[y|x]$, è $\frac{e^{w^T x}}{1+e^{w^T x}} = \sigma(w^T x)$ se $y = 1$, $\frac{1}{1+e^{w^T x}} = 1 - \sigma(w^T x)$ se $y = 0$, dove w è un vettore di parametri da apprendere.

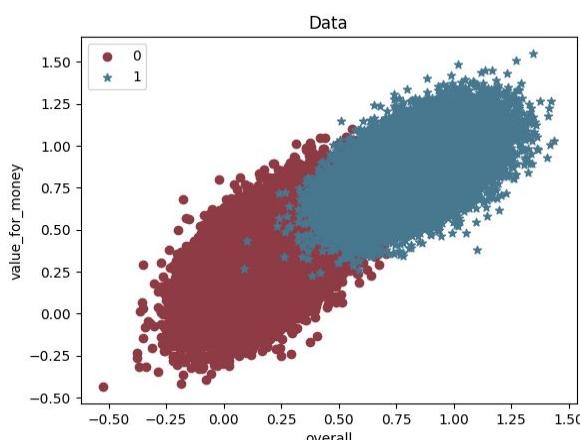
In altre parole, un dato x è classificato come appartenente alla classe 1 se $w^T x > 0$ (in tal caso $P[1|x] = \frac{e^{w^T x}}{1+e^{w^T x}} > \frac{1}{2}$ e $\sigma(w^T x) > \frac{1}{2}$), è classificato invece come appartenente alla classe 0 se $w^T x < 0$ (in tal caso $P[0|x] = \frac{1}{1+e^{w^T x}} > \frac{1}{2}$ e $\sigma(w^T x) < \frac{1}{2}$).

Dunque, questo classificatore è caratterizzato dal vettore di parametri w che deve essere appreso attraverso la minimizzazione di una *funzione di costo* $J(w) = -\frac{1}{N} \sum_{i=1}^N [y_i \cdot \log(P[1|x_i]) + (1-y_i) \cdot \log(P[0|x_i])]$, che può essere anche scritta nella forma $J(w) = -\frac{1}{N} \sum_{i=1}^N [y_i \cdot \log(\sigma(w^T x_i)) + (1-y_i) \cdot \log(1-\sigma(w^T x_i))]$.

Per l'apprendimento di w viene eseguito un algoritmo noto con il nome di *discesa del gradiente*, che consiste nell'aggiornare w ciclicamente secondo una formula del tipo $w = w - \alpha \cdot \frac{d}{dw} J(w)$, dove $\frac{d}{dw} J(w) = \sum_{i=1}^N [(\sigma(w^T x_i) - y_i) \cdot x_i]$ è appunto il gradiente di $J(w)$ e α è un iperparametro chiamato *learning rate* o *step size*. L'algoritmo di discesa del gradiente tenta appunto di minimizzare $J(w)$ calcolando w in funzione della direzione negativa di $\frac{d}{dw} J(w)$.

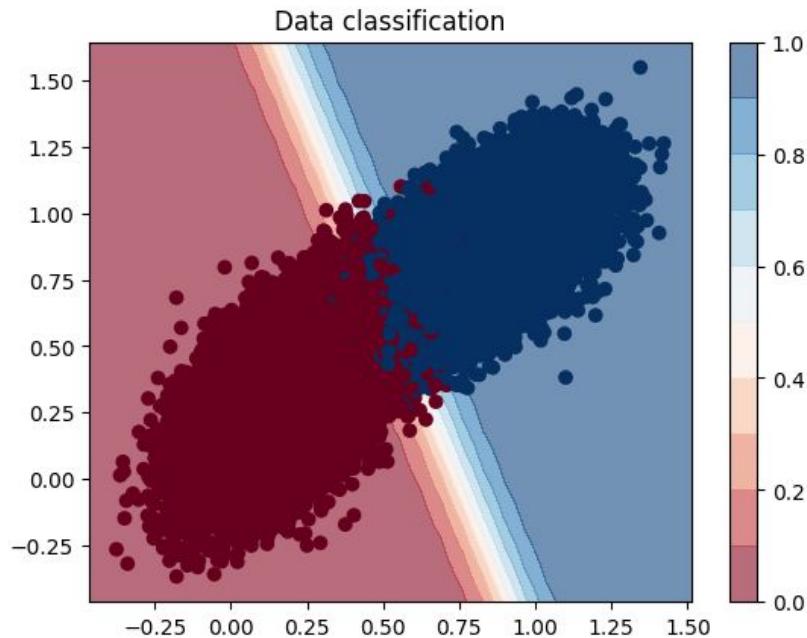
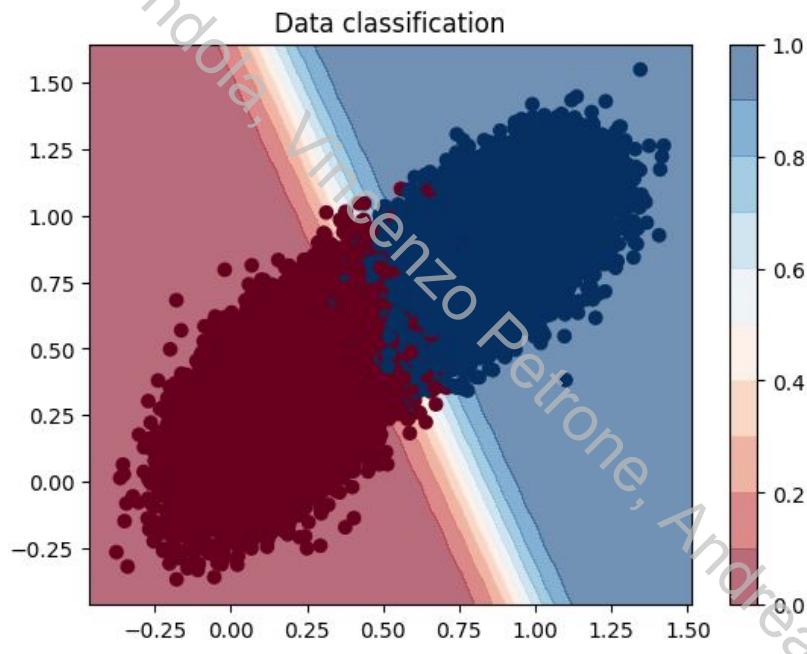
2.6.1. Applicazione del classificatore Logistic Regression sul dataset sintetico

Prima di applicare il classificatore Logistic Regression sul dataset sintetico, si mostra la sua distribuzione su uno scatterplot, già mostrato in precedenza:



Successivamente, è stato eseguito l'algoritmo di classificazione secondo il classificatore Logistic Regression implementato nella libreria `sklearn`; il risultato della classificazione è stato poi paragonato al risultato ottenuto eseguendo l'algoritmo di classificazione implementato manualmente attraverso il framework `Spark`.

Seguono due figure: la prima come risultato dell'applicazione dell'algoritmo tramite `Spark`, la seconda come risultato dell'applicazione dell'algoritmo di libreria. Tali figure rappresentano le regioni di classificazione, distinte attraverso 10 livelli di probabilità: le regioni dai colori tendenti al rosso sono le regioni classificate come 0 (dunque queste regioni sono costituite da quei punti x tali per cui $\sigma(w^T x) < \frac{1}{2}$), mentre le regioni dai colori tendenti al blu sono le regioni classificate come 1 (dunque queste regioni sono costituite da quei punti x tali per cui $\sigma(w^T x) > \frac{1}{2}$).



Ricordando che il dataset sintetico è composto unicamente dalle feature **overall** e **value_for_money**, i risultati dell'algoritmo manuale su Spark sono una percentuale di corretta classificazione del 98,13%, mentre i risultati dell'algoritmo di libreria sono mostrati nella figura seguente:

```
Performances...

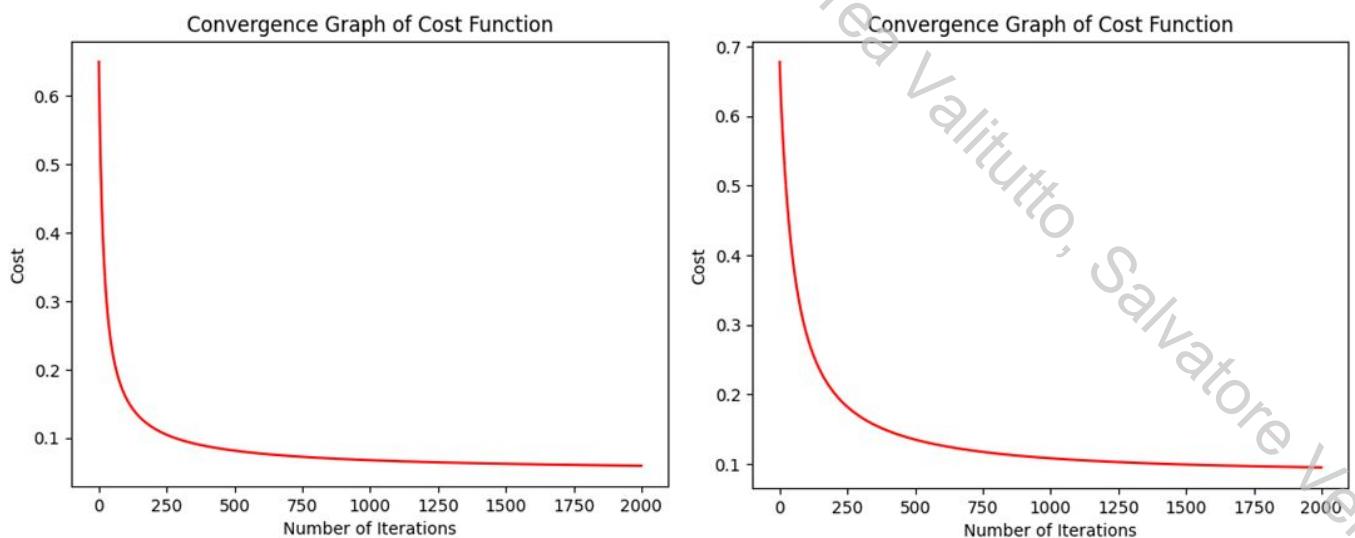
Confusion matrix:
[[11806  240]
 [ 196 11631]]

Classifier performance metrics:
      precision    recall    f1-score   support
0         0.98     0.98     0.98     12046
1         0.98     0.98     0.98     11827

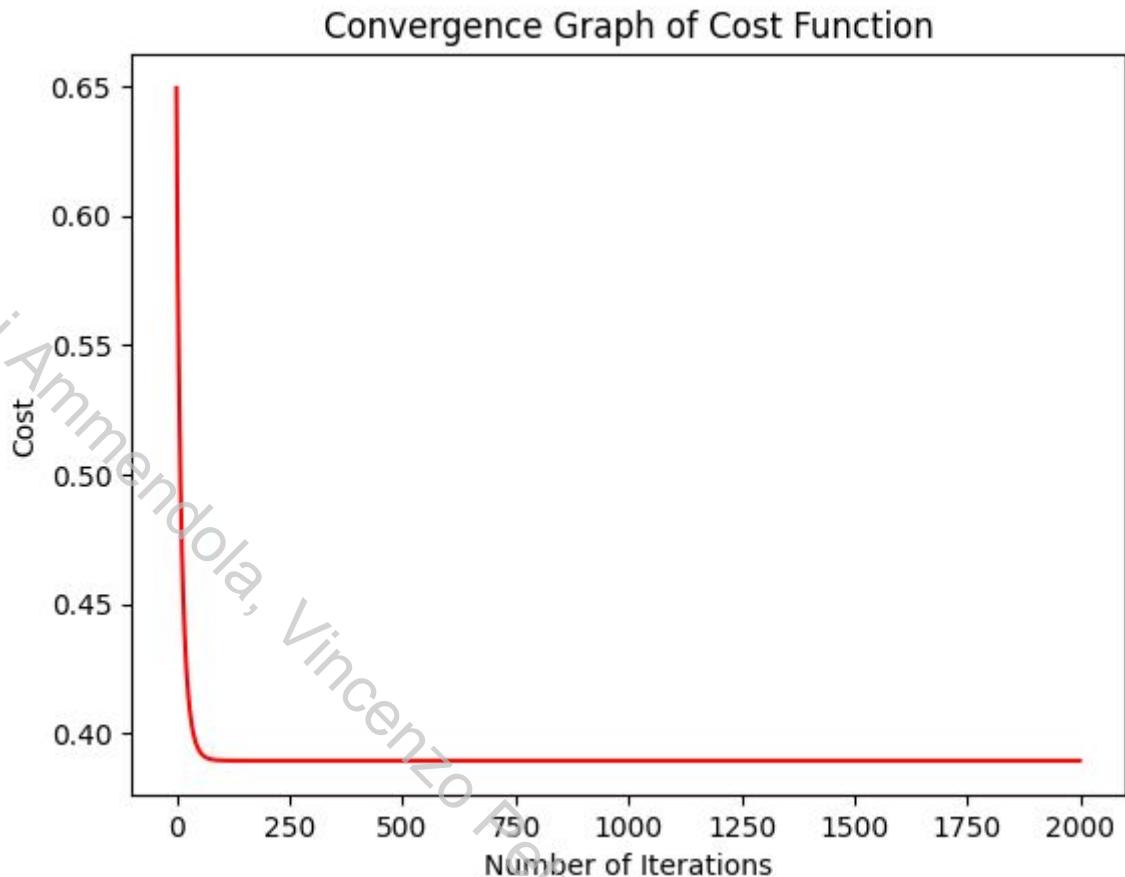
accuracy                           0.98
macro avg       0.98     0.98     0.98     23873
weighted avg    0.98     0.98     0.98     23873

final performance: 98.17%
```

È stata inoltre valutata la convergenza della funzione di costo al variare dei parametri learning rate e momentum. Di seguito sono mostrate due figure: la prima mostra il grafico di convergenza della funzione di costo per *learning_rate* = 0.9 e *momentum* = 0.001, mentre la seconda mostra il grafico di convergenza della funzione di costo per *learning_rate* = 0.3 e *momentum* = 0.05.



Come si nota, per un valore maggiore del learning rate, la funzione di costo converge più velocemente.



Quest'ultima figura invece mostra come, con $learning_rate = 0.9$ e $momentum = 0.05$ la funzione di costo converge più velocemente, ma ad un valore molto superiore rispetto a quanto accade con i parametri definiti precedentemente.

2.6.1.1. Spark

Per implementare il classificatore binario Logistic Regression tramite Spark è stato ovviamente necessario definire, come accennato nell'[introduzione teorica](#) di questo capitolo:

- una funzione per calcolare il gradiente della funzione di costo rispetto ai parametri e ai dati nel Training Set
- la funzione di costo stessa rispetto ai parametri e ai dati nel Training Set
- l'algoritmo di discesa del gradiente rispetto agli iperparametri:
 - learning rate (o step size)
 - *momento*, ulteriore iperparametro utile per velocizzare l'apprendimento
- la funzione di classificazione rispetto ai parametri calcolati tramite l'algoritmo di discesa del gradiente e ai dati nel Test Set
- la funzione sigmoide, utile per i calcoli già citati

```

def compute_gradient(yx_train):
    """Returns the gradient on a single point in the training set.
    global w_br
    x = yx_train[1:]
    y = -1 + 2 * yx_train[0]

    w_dot_x = w_br.value @ x.T
    gradient = -y * (1 - sigmoid(y * w_dot_x)) * x
    return gradient

```

Come mostra la figura precedente, la funzione di calcolo del gradiente prende in ingresso un punto del Training Set del tipo yx , dove y è la classe assegnata al dato x . Poiché per $y \in \{0, 1\}$ il gradiente è $(\sigma(w^T x) - y) \cdot x$ per un dato x , si dimostra che la sua implementazione all'interno di questa funzione, vale a dire $-y \cdot (1 - \sigma(y \cdot w^T x)) \cdot x$, è del tutto equivalente.

- Per $y = 0$, il gradiente è $(\sigma(w^T x) - y) \cdot x = \sigma(w^T x) \cdot x$
 - All'interno della funzione, y viene riassegnato come -1 , dunque $-y \cdot (1 - \sigma(y \cdot w^T x)) \cdot x = (1 - \sigma(-w^T x)) \cdot x = \sigma(w^T x) \cdot x$
 - Si noti infatti che $(1 - \sigma(-z)) = \sigma(z) \forall z \in \mathbb{R}$
- Per $y = 1$, il gradiente è $(\sigma(w^T x) - y) \cdot x = (\sigma(w^T x) - 1) \cdot x$
 - Dunque all'interno della funzione il gradiente viene calcolato come $-y \cdot (1 - \sigma(y \cdot w^T x)) \cdot x = -1 \cdot (1 - \sigma(w^T x)) \cdot x = (\sigma(w^T x) - 1) \cdot x$

Dunque l'implementazione rispetta effettivamente la formulazione teorica.

```

def compute_cost(X, Y, w):
    """Returns the cost value, computed with the respect to the dataset X, the current weight w
    and the actual labels Y"""
    h = sigmoid(X @ w)
    epsilon = 1e-5
    cost = (1 / len(Y)) * (((-Y).T @ np.log(h + epsilon)) - ((1 - Y).T @ np.log(1 - h + epsilon)))
    return cost

```

Come mostra la figura precedente, la funzione di costo prende in ingresso i dati X e le rispettive classi Y , dopodiché calcola il valore della funzione di costo secondo la formulazione teorica $J(w) = -\frac{1}{N} \sum_{i=1}^N [y_i \cdot \log((w^T x_i)) + (1 - y_i) \cdot \log(1 - (w^T x_i))]$.

```

def gradient_descent(YY_XX_train, w, learning_rate, momentum, iterations):
    """Performs the gradient descent algorithm. Returns the cost history and
    the optimal parameters."""
    global w_br

    RDD_train = sc.parallelize(YY_XX_train).cache()

    cost_history = np.zeros((iterations,1))

    for i in range(iterations):
        gradient = RDD_train.map(compute_gradient).reduce(add)
        w = w - learning_rate * (gradient/(N//2) + momentum * w)

        cost_history[i] = compute_cost(YY_XX_train[:, 1:],
                                      YY_XX_train[:, 0].reshape(len(YY_XX_train), 1),
                                      w.reshape(len(w), 1))
        w_br = sc.broadcast(w)

    return cost_history, w

```

La figura precedente mostra l'implementazione dell'algoritmo del gradiente sul Training Set completo. Esso viene prima parallelizzato in un RDD tramite Spark, successivamente viene eseguita la funzione di calcolo del gradiente per tutti i punti nel Training Set all'interno di ogni partizione dell'RDD, dopodiché i risultati vengono sommati per il calcolo del gradiente sull'intero Training Set. Quindi il vettore dei parametri viene aggiornato in funzione del gradiente stesso e, come accennato precedentemente, del learning rate e del momento. Infine viene calcolato il nuovo valore della funzione di costo.

```

def predict(x, w):
    """Returns the predicted label for x, given the p
    return np.round(sigmoid(x @ w.reshape(len(w), 1)))

```

La figura precedente mostra l'implementazione della funzione di predizione, che approssima il valore di $(w^T x)$ a 0 o 1.

```

def sigmoid(x):
    """Sigmoid function in range
    return 1 / (1 + np.exp(-x))

```

L'ultima funzione implementa semplicemente la funzione sigmoide $(x) = \frac{1}{1+e^{-x}}$.

2.6.2. Applicazione del classificatore Logistic Regression sul dataset reale

Seguendo un ragionamento simile fatto per il classificatore KNN, visti i risultati ottenuti, il classificatore Logistic Regression è stato applicato anche al dataset reale utilizzando la sola implementazione di libreria. Avendo infatti dimostrato tramite l'applicazione del classificatore sul dataset sintetico che l'implementazione parallelizzata tramite Spark è completamente funzionante e corretta, in quanto porta agli stessi risultati degli algoritmi di

libreria, e tenendo in considerazione che tale esecuzione parallela non avrebbe portato alcun beneficio, si è deciso di utilizzare esclusivamente l'implementazione di libreria perché l'esecuzione tramite Spark avrebbe richiesto una quantità di tempo eccessiva.

Il classificatore è stato testato su diverse porzioni del dataset:

- Con le sole due feature **overall** e **value_for_money**
- Una feature per volta
- Con tutte le possibili coppie di feature
- Con tutte le feature disponibili

In generale, i risultati sono stati leggermente superiori di quelli riscontrati utilizzando il classificatore Naïve Bayes.

Utilizzo di due feature

Utilizzando la porzione del dataset reale composta unicamente dalle feature **overall** e **value_for_money**, i risultati sono stati leggermente inferiori rispetto a quelli riscontrati utilizzando il dataset sintetico, come riporta la figura seguente:

```
Performances...

Confusion matrix:
[[11569  522]
 [ 485 11297]]

Classifier performance metrics:
      precision    recall   f1-score   support
          0         0.96     0.96     0.96    12091
          1         0.96     0.96     0.96    11782
   accuracy                           0.96    23873
  macro avg       0.96     0.96     0.96    23873
weighted avg       0.96     0.96     0.96    23873

final performance: 95.78%
```

Utilizzo una feature per volta

Al fine di confermare l'impatto di ogni feature sulla percentuale di corretta classificazione, il classificatore Logistic Regression è stato testato ciclicamente sulla porzione del dataset composta da ciascuna singola feature, ottenendo i risultati riportati nella tabella seguente:

overall	seat_comfort	cabin_service	food_bev	entertainment	ground_service	wifi_connectivity	value_for_money
95.42%	82.07%	86.12%	82.26%	77.84%	84.38%	73.81%	91.03%

Come già riscontrato in precedenza, si può notare come i parametri più significativi per la valutazione di un volo come raccomandato o meno sono quelli di **overall** e **value_for_money**, mentre quelli che forniscono meno precisione sono **wifi_connectivity** e **entertainment**.

Utilizzo di coppie di feature

Il classificatore Logistic Regression è stato testato ciclicamente su tutte le possibili coppie di feature, ottenendo i risultati riportati nelle tabelle seguenti:

overall + seat_comfort	overall + cabin_service	overall + food_bev	overall + entertainment
95.49%	95.56%	95.49%	95.25%

overall + ground_service	overall + wifi_connectivity	overall + value_for_money	seat_comfort + cabin_service
95.36%	95.65%	95.53%	89.01%

seat_comfort + food_bev	seat_comfort + entertainment	seat_comfort + ground_service	seat_comfort + wifi_connectivity
86.65%	84.07%	88.27%	84.45%

seat_comfort + value_for_money	cabin_service + food_bev	cabin_service + entertainment	cabin_service + ground_service
91.82%	87.47%	87.27%	89.57%

cabin_service + wifi_connectivity	cabin_service + value_for_money	food_bev + entertainment	food_bev + ground_service
86.45%	92.84%	84.05%	88.82%

food_bev + wifi_connectivity	food_bev + value_for_money	entertainment + ground_service	entertainment + wifi_connectivity
84.92%	91.74%	88.90%	79.22%

entertainment + value_for_money	ground_service + wifi_connectivity	ground_service + value_for_money	wifi_connectivity + value_for_money
91.67%	86.33%	92.00%	91.72%

Come ulteriore conferma, il risultato più alto è stato ottenuto utilizzando congiuntamente le feature **overall e value_for_money**.

Utilizzo di tutte le feature disponibili

Infine, il classificatore Logistic Regression è stato testato utilizzando congiuntamente tutte le feature disponibili, con il seguente risultato:

```
Performances...

Confusion matrix:
[[11374  502]
 [ 480 11517]]

Classifier performance metrics:
      precision    recall  f1-score   support
          0       0.96     0.96     0.96    11876
          1       0.96     0.96     0.96    11997

      accuracy                           0.96    23873
     macro avg       0.96     0.96     0.96    23873
  weighted avg       0.96     0.96     0.96    23873

final performance: 95.89%
```

2.7. Naïve Kernel

Il classificatore Naïve Kernel sfrutta come idea di base la stessa del KNN, ovvero che gli elementi di una stessa classe siano raggruppati e divisi dagli altri. La differenza sta nel fatto che non vengono considerati i K elementi più vicini, ma tutti quelli che hanno una distanza minore di una certa soglia, sia tale soglia h . Un nuovo dato x verrà assegnato alla classe più frequente tra i dati la cui distanza da x è minore o uguale a h . Quindi, un *iperparametro* di questo classificatore è appunto la soglia di distanza h da utilizzare nell'algoritmo. Anche in questo caso, la definizione di distanza può variare: nel caso in esame, è stata scelta nuovamente la distanza euclidea.

Dopo aver ottenuto tali distanze, vengono considerate le classi dei sample vicini e calcolata la classe più frequente.

Tale metodo di classificazione è piuttosto oneroso da un punto di vista computazionale e richiede la regolazione dell'*iperparametro* h . A tal fine è stata eseguita una partizione del dataset in 3 sezioni disgiunte:

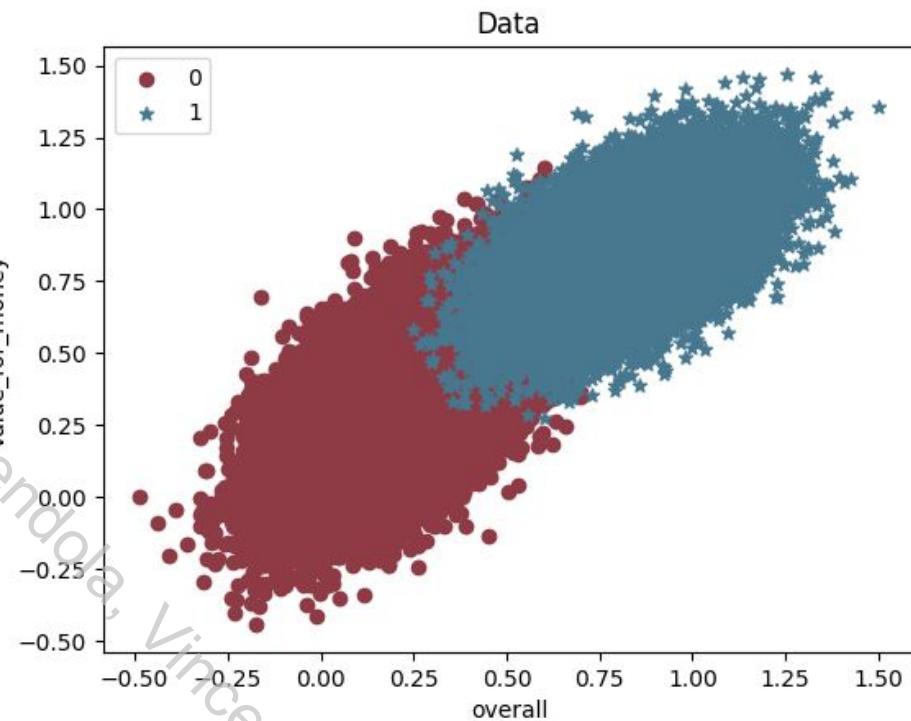
- Training Set: partizione dalla quale estrarre gli elementi la cui distanza è inferiore ad h
- Test Set: per la valutazione finale delle performance in termini di percentuale di corretta classificazione
- Validation Set: per valutare il miglior parametro h
 - In particolare, il valore h migliore è quell' h tale per cui l'MSE sulle stime dei dati appartenenti al Validation Set è minore

Dato che le librerie standard non mettono a disposizione dei metodi per l'esecuzione dell'algoritmo Naïve Kernel, è stato deciso di implementarlo manualmente. Questa ha scelta ha reso possibile anche l'utilizzo del framework Spark per la parallelizzazione del calcolo delle distanze.

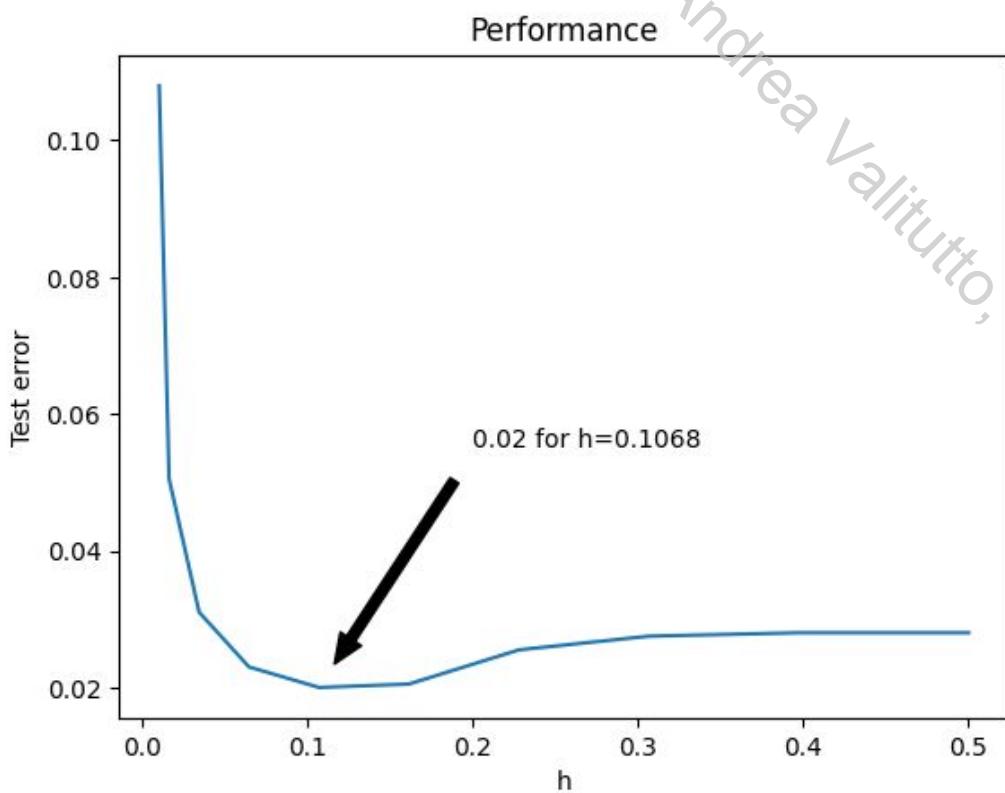
Tale algoritmo, come già detto, è molto simile al KNN, quindi è stato sufficiente modificare il metodo di aggregazione dei risultati, ovvero non viene più selezionato un numero fisso di sample, che quindi devono essere ordinati in base alla distanza, ma vengono selezionati tutti quelli vicini. Quindi è sufficiente, una volta calcolate tutte le distanze, calcolare la classe più frequente.

2.7.1. Applicazione del classificatore Naïve Kernel sul dataset sintetico

Così come per gli altri modelli, anche in questo caso si è deciso di effettuare alcune verifiche preliminari con un dataset sintetico. I risultati attesi da tale metodo sono in linea con quanto ottenuto finora con gli altri metodi, eccetto per piccole variazioni. Anche in questo caso, viene mostrata la distribuzione dei dati al variare delle feature **overall** e **value_for_money**.



Come nel caso dei metodi precedenti, il modello Naïve Kernel è stato applicato al dataset sintetico in una sola modalità, ovvero utilizzando solo una coppia di feature, **overall** e **value_for_money**. I risultati del modello Naïve Kernel su questa coppia di feature sono in linea con quelli ottenuti finora: infatti, la percentuale di corretta classificazione è del 98,2%. Segue una figura che mostra l'MSE al variare di h , e la selezione del miglior valore di h .



2.7.1.1. Spark

Come già accennato, l'implementazione del Naïve Kernel è stata effettuata direttamente tramite il framework Spark, rendendo possibile la parallelizzazione su più cluster nel caso in cui il software venga eseguito su più macchine. Per ogni partizione del dataset distribuito, viene quindi effettuato un filtraggio che consiste nel selezionare tutti i sample la cui distanza è inferiore alla soglia h , mandata in broadcast tra tutti i cluster. Per selezionare la classe, viene effettuata la stessa operazione del KNN, vale a dire che la somma delle label delle classi viene confrontata con la metà del numero di sample. Se tale somma è maggiore della metà del numero di vicini in un dato raggio, allora la classe assegnata è la classe 1; in caso contrario, viene assegnata la classe 0.

2.7.2. Applicazione del classificatore Naïve Kernel sul dataset reale

Dopo una veloce analisi del dataset sintetico, si è deciso di procedere all'analisi del dataset reale utilizzando il modello Naïve Kernel. Sono stati provati anche in questo caso diversi approcci:

- Con le sole due feature **overall** e **value_for_money**
- Una feature per volta
- Con tutte le feature disponibili

Data la complessità del calcolo dell'iperparametro h e della valutazione delle performance, si è deciso di utilizzare un dataset di dimensioni ridotte.

Utilizzo di due feature

Utilizzando una porzione del dataset reale composta unicamente dalle feature **overall** e **value_for_money** i risultati sono stati leggermente inferiori rispetto a quelli riscontrati utilizzando il dataset sintetico, vale a dire del 94,87%:

Utilizzo una feature per volta

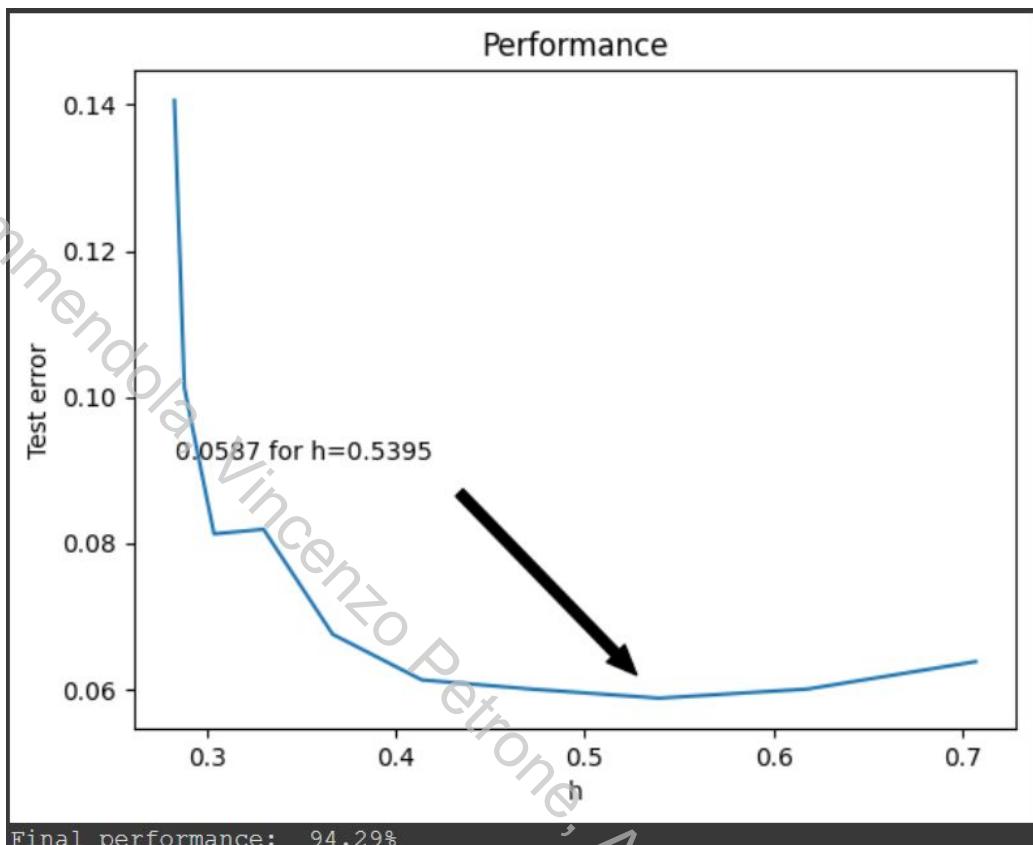
Anche con questo modello, si è deciso di analizzare i dati tramite una feature per volta. I risultati sono sempre in linea con quanto osservato finora:

- Per le feature **overall** e **value_for_money**, il modello riesce a stimare con una buona precisione la classe.
- Utilizzando altre feature, tale precisione si riduce, arrivando al minimo con le feature **wifi_connectivity** ed **entertainment**

Final performance: 88.00%								
	overall	seat_comfort	cabin_service	food_bev	entertainment	ground_service	wifi_connectivity	value_for_money
K	0.3666	0.2828	0.3666	0.3666	0.2881	0.2881	0.3666	0.2828
MSE	5.50%	17.25%	13.00%	14.50%	18.50%	13.75%	23.00%	9.75%
Score	93.17%	80.50%	84.33%	80.67%	75.17%	83.83%	78.17%	88.00%

Utilizzo di tutte le feature disponibili

Al termine dell'analisi sulle singole feature, l'algoritmo del Naïve Kernel è stato eseguito su tutto il dataset, sfruttando tutte le feature disponibili, ottenendo i seguenti risultati:



Si fa notare che, come nei casi precedenti, l'MSE tende ad aumentare se h si avvicina a 1, poiché in questo caso si tende a considerare più vicini del necessario, esattamente come accadeva nel caso del KNN all'aumentare di K .

3. Progetto R

In questa sezione vengono affrontati ed esplorati tutti i metodi di regressione, applicando le conoscenze acquisite. Per lo sviluppo di questa parte, è stato utilizzato il software di analisi dei dati *RStudio* e, in particolare, una serie di librerie che forniscono le funzioni necessarie allo svolgimento delle operazioni, con lo scopo di stimare un'eventuale relazione esistente tra la variabile dipendente e le variabili indipendenti.

Con il susseguirsi dei vari metodi, sono state messe in evidenza le informazioni principali che suggerivano le decisioni sulle ulteriori applicazioni da mettere in atto.

L'obiettivo fissato è stato quello di stimare la variazione di **overall**, la nostra variabile dipendente, in relazione ai differenti predittori del dataset, ovvero:

- variabile dipendente: **overall**
- variabili indipendenti: **recommended**, **seat_comfort**, **cabin_service**, **food_bev**, **entertainment**, **ground_service**, **wifi_connectivity**, **value_for_money**.

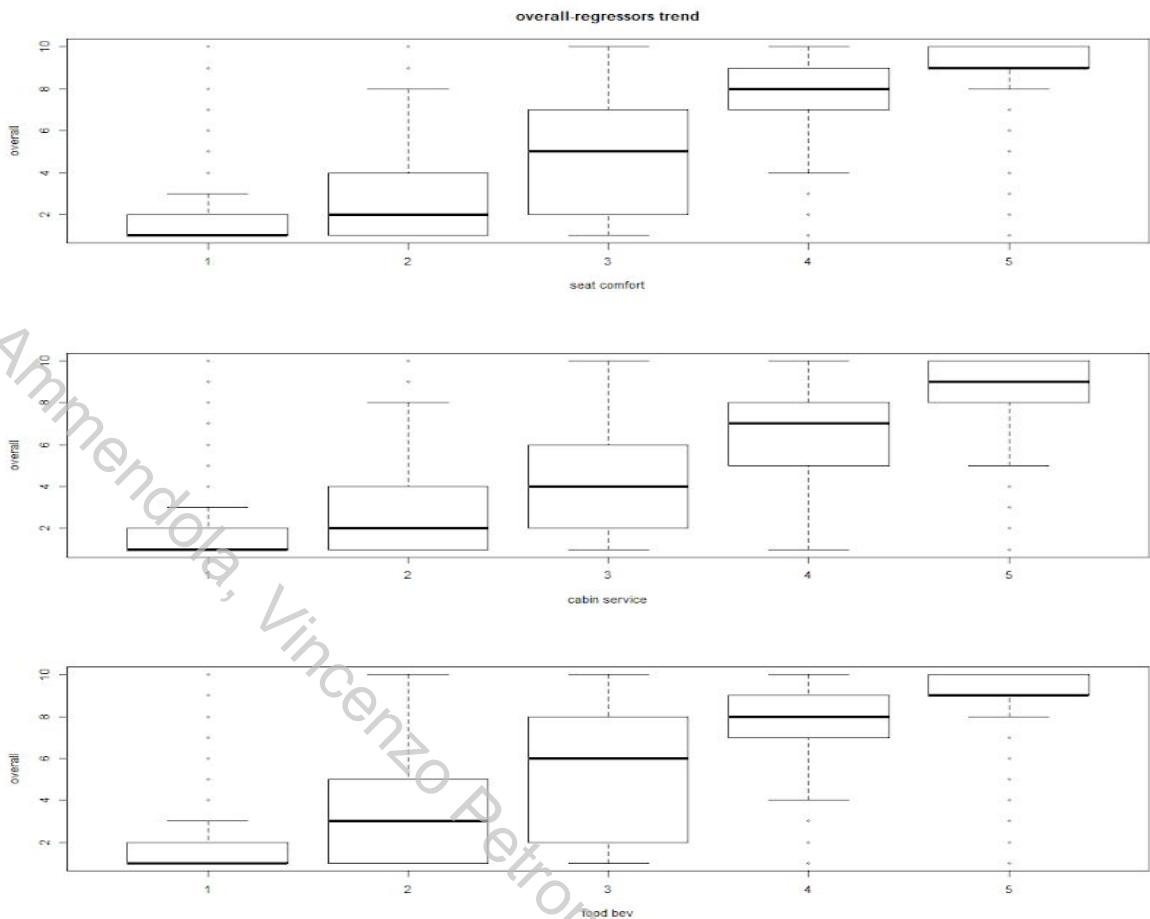
Da questo elenco si evince come tre record del dataset, ossia **airline**, **traveller_type** e **class**, non siano stati presi in considerazione in quanto sterili ai fini della stima di **overall**.

3.1. Regressione Multipla

La regressione è quella tecnica statistica utilizzata per studiare le relazioni che intercorrono tra due o più caratteri statistici. A differenza della regressione semplice, che analizza la relazione tra una variabile Y e un preditore X, la regressione multipla descrive la connessione tra la variabile Y e più predittori: X_1, X_2, X_3 , ecc.

Strettamente legata alla regressione è il concetto di **correlazione**, infatti si suppone che più variabili X (nella regressione multipla), assumano valori determinati e si cerca la relazione che lega la variabile Y alle prime. In altre parole, si cerca di stabilire un legame funzionale tra le variabili (del tipo $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2$). Il tipo di regressione analizzata è la **regressione dei minimi quadrati**, che ha come obiettivo quello di trovare la miglior retta interpolatrice dei punti del piano (retta di regressione dei minimi quadrati).

Il primo step per poter costruire il modello migliore che interpreta i dati, è stato quello di identificare la tendenza tra la risposta del sistema (**overall**) e tutti i predittori che condizionano la risposta. Per fare ciò sono stati rappresentati, graficamente, la dipendenza tra overall e tutti i suoi regressori in modo tale da capire l'andamento della risposta al variare dei regressori.



Come si evince dai grafici, la tendenza tra **overall** e i regressori presi in considerazione (**seat_comfort**, **cabin_service** e **food_bev**) non è lineare. Un ragionamento analogo è stato effettuato anche per tutti gli altri regressori non presenti nel grafico.

La conseguenza di tale risultato ha spinto alla realizzazione di diverse trasformazioni non lineari dei nostri predittori: trasformazione quadratica, cubica, di ordine 4 e, infine, logaritmica. Per ognuna, i risultati prodotti sono stati valutati e confrontati tra di loro, per capire quale modello interpretasse nel miglior modo possibile i dati.

In primo luogo, ogni modello è stato scomposto in modo da capire quali regressori fossero significativi per la stima di **overall**, attraverso la funzione `summary(model)` in R. Così facendo, per ogni regressore abbiamo ottenuto la stima dei coefficienti, lo standard error, t-value e p-value. L'output prodotto per ogni modello riporta anche il “*Residual Standard Error*” (RSS), “*Multiple R-squared*”, “*Adjusted R-squared*” e “*F-statistic*”.

Successivamente, è stata fatta un'analisi sull'collinearità presente nel modello in questione. Utilizzando la funzione `vif(lm.fit)` in R, è stato possibile analizzare il VIF (Variance Inflation Factor) e, dal risultato ottenuto, si evince che i valori del VIF sono inferiori a 5, ad eccezione di **value_for_money** a 6 per i modelli con trasformazione polinomiale. In ogni caso, nessun valore supera 10, la soglia definita come indice massimo per la collinearità:

```
> car::vif(fit4)
              GVIF  Df GVIF^(1/(2*Df))
recommended      4.739402  1    2.177017
poly(seat_comfort, 4) 3.590440  4    1.173258
poly(cabin_service, 4) 4.228088  4    1.197479
poly(food_bev, 4)     4.022134  4    1.190028
poly(entertainment, 4) 2.883404  4    1.141532
poly(ground_service, 4) 3.374096  4    1.164179
poly(wifi_connectivity, 4) 2.808968  4    1.137806
poly(value_for_money, 4) 6.559297  4    1.265048
```

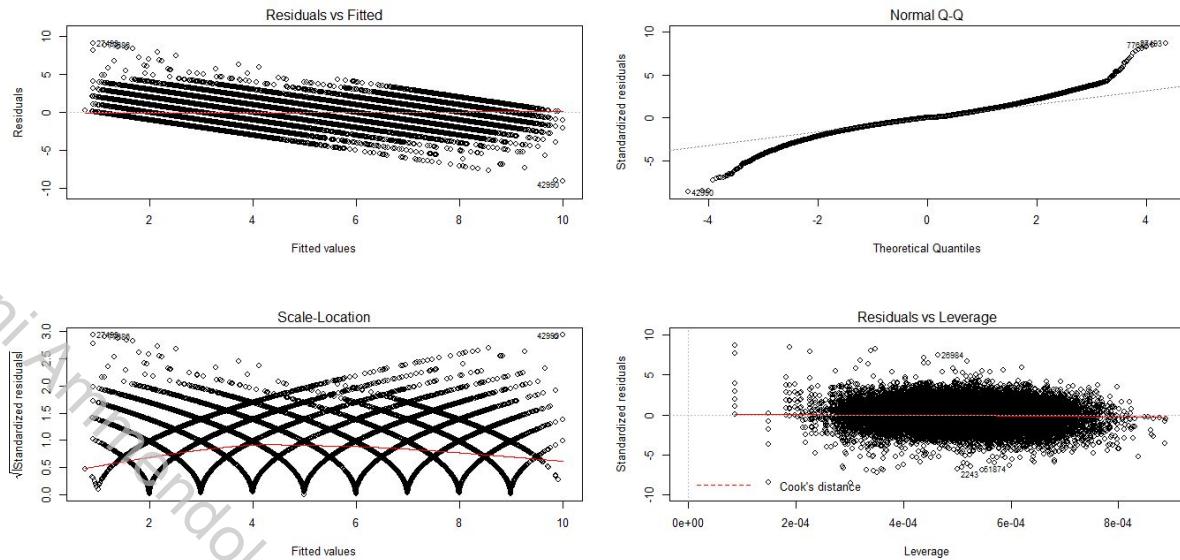
La funzione `anova` (`model1, model2`) ha permesso di confrontare i diversi modelli anche in termini di RSS e di capire quale fosse quello migliore tra tutti nell'interpretare i dati. Al termine di questa indagine, dopo aver esaminato tutte le trasformazioni dei regressori sopra citati, si è concluso che il modello che ha prodotto i migliori risultati, in relazione a tutti i parametri che forniscono le funzioni utilizzate, è stato quello con una trasformazione di ordine 4 dei regressori, come suggerito dall'output di `anova` mostrato in figura:

Analysis of Variance Table

```
Model 1: overall ~ recommended + poly(seat_comfort, 3) + poly(cabin_service,
3) + poly(food_bev, 3) + poly(entertainment, 3) + poly(ground_service,
3) + poly(wifi_connectivity, 3) + poly(value_for_money, 3)
Model 2: overall ~ recommended + poly(seat_comfort, 4) + poly(cabin_service,
4) + poly(food_bev, 4) + poly(entertainment, 4) + poly(ground_service,
4) + poly(wifi_connectivity, 4) + poly(value_for_money, 4)
Res.Df   RSS Df Sum of Sq    F Pr(>F)
1 79553 87754
2 79546 87640  7    114.27 14.817 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

A dare supporto ai risultati ottenuti, ci sono i grafici che dimostrano coerenza per quanto riguarda l'analisi della tendenza (non lineare) tra "overall" e i suoi regressori.

Di default R permette di plottare l'output del fit "`lm()`" fornendo alcuni grafici utili per l'apprendimento del modello utilizzato. Nella figura sottostante viene mostrata l'analisi relativa ad un modello con una trasformazione di ordine quattro dei predittori:



- *Grafico in alto a sinistra:* residui in funzione dei valori \hat{Y} . Sono presenti più linee di rappresentazione dei valori perchè, avendo valori discreti, viene mostrata una linea per ogni possibile risposta. Per risposta si intende ogni possibile valore che può assumere overall e, di conseguenza, essendo compreso da 1 a 10, sono mostrate 10 linee;
- *Grafico in alto a destra:* mostra se i residui possono essere spiegati (approssimabili) da una distribuzione normale. Il Normal Q-Q plot calcola i quantili di tutti i residui (anche tra punti di diversa distribuzione). Nel caso della regressione, prende i residui standardizzati, ne calcola i quantili e li confronta con i quantili teorici ipotizzando che i nostri dati vengano fuori da una distribuzione normale. Vuol dire che se i residui standardizzati sono ben interpretabili in termini di distribuzione normale, si troveranno allineati lungo la retta.
- *Grafico in basso a sinistra:* valori di \hat{Y} in funzione dei residui standardizzati. Grafico molto utile per individuare possibili **outliers**.
- *Grafico in basso a destra:* viene utilizzata la distanza di Cook. Utile per capire se ci stanno punti ad elevata influenza (“High Leverage Points”).

Infine, sono stati visualizzati quali fossero i predittori più e meno significativi nel calcolo della variabile indipendente. Una veloce analisi qualitativa ha ottenuto come risultato che, tra i predittori meno significativi figurassero **entertainment**, **wifi_connectivity**, **food_bev** e **seat_comfort**. L’immagine seguente riporta un esempio della significatività dei predittori sul modello di grado 4.

coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	4.039990	0.008855	456.260	< 2e-16 ***
recommended	2.382418	0.016202	147.049	< 2e-16 ***
poly(seat_comfort, 4)1	80.197151	1.714849	46.766	< 2e-16 ***
poly(seat_comfort, 4)2	10.467691	1.172241	8.930	< 2e-16 ***
poly(seat_comfort, 4)3	-0.903590	1.087648	-0.831	0.40610
poly(seat_comfort, 4)4	-6.203441	1.056802	-5.870	4.37e-09 ***
poly(cabin_service, 4)1	121.074678	1.853455	65.324	< 2e-16 ***
poly(cabin_service, 4)2	22.933709	1.187773	19.308	< 2e-16 ***
poly(cabin_service, 4)3	6.930331	1.086816	6.377	1.82e-10 ***
poly(cabin_service, 4)4	-4.993480	1.056648	-4.726	2.30e-06 ***
poly(food_bev, 4)1	69.995053	1.790512	39.092	< 2e-16 ***
poly(food_bev, 4)2	14.019276	1.188120	11.800	< 2e-16 ***
poly(food_bev, 4)3	-0.281704	1.085940	-0.259	0.79532
poly(food_bev, 4)4	-3.311297	1.055235	-3.138	0.00170 **
poly(entertainment, 4)1	48.513183	1.619982	29.947	< 2e-16 ***
poly(entertainment, 4)2	3.068035	1.137910	2.696	0.00701 **
poly(entertainment, 4)3	2.366611	1.061798	2.229	0.02583 *
poly(entertainment, 4)4	-2.320536	1.053434	-2.203	0.02761 *
poly(ground_service, 4)1	167.295629	1.752951	95.437	< 2e-16 ***
poly(ground_service, 4)2	-5.637883	1.133313	-4.975	6.55e-07 ***
poly(ground_service, 4)3	4.181108	1.068695	3.912	9.15e-05 ***
poly(ground_service, 4)4	-4.773979	1.052140	-4.537	5.70e-06 ***
poly(wifi_connectivity, 4)1	-17.621380	1.600336	-11.011	< 2e-16 ***
poly(wifi_connectivity, 4)2	7.814279	1.129980	6.915	4.70e-12 ***
poly(wifi_connectivity, 4)3	-2.034234	1.063157	-1.913	0.05570 .
poly(wifi_connectivity, 4)4	-0.822053	1.065209	-0.772	0.44028
poly(value_for_money, 4)1	234.162317	2.277116	102.833	< 2e-16 ***
poly(value_for_money, 4)2	-7.965631	1.172366	-6.794	1.09e-11 ***
poly(value_for_money, 4)3	-8.351883	1.138266	-7.337	2.20e-13 ***
poly(value_for_money, 4)4	0.375848	1.056655	0.356	0.72207

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1				

Residual standard error: 1.05 on 79546 degrees of freedom
 Multiple R-squared: 0.9071, Adjusted R-squared: 0.9071
 F-statistic: 2.678e+04 on 29 and 79546 DF, p-value: < 2.2e-16

3.2. Metodi di Ricampionamento statistico

I metodi di ricampionamento statistici utilizzati per testare i modelli realizzati, sono: l'approccio K-fold Cross Validation e Bootstrap. Queste tecniche vengono utilizzate per effettuare il *fit* di un modello su una parte di dati riservati per il *training* e un'altra parte di dati per il *test* del modello ottenuto. L'obiettivo è sempre quello di trovare il modello ottimale, che si comporti in maniera soddisfacente sia sul set di allenamento sia sul set di test. Un'indicazione indispensabile sulla risposta del modello, viene fornita dal *MSE* (*Mean Square Error*), che rappresenta la discrepanza tra i valori dei dati osservati ed i valori dei dati stimati. Per tutti i metodi citati, andremo a esaminare l'MSE stimato sul test set da tutti modelli, confrontando i risultati ottenuti.

Un'osservazione va tenuta in considerazione per i modelli più complessi. Infatti, man mano che la complessità aumenta, l'MSE sul training set diminuisce. Questo avviene perché la complessità del modello aiuta ad apprendere, in maggior misura, tutte le variazioni che

contraddistinguono i dati di addestramento. Di contro, la conseguenza porta una sovradattamento del modello sui dati di prova ed un malfunzionamento sui campioni di test. Questo fenomeno è noto come *overfitting*.

3.2.1. Validation Set

L'approccio del set di validazione è un semplice metodo di divisione dei dati per l'addestramento e il test. I dati sono divisi in due parti: la prima parte viene utilizzata per addestrare il modello, la seconda divisione dei dati viene utilizzata per testarlo.

Nel caso in esame, il criterio usato è quello di assegnare un numero di campioni pari a 39788 per entrambe la parti.

I valori ottenuti sono riprodotti nella tabella seguente:

Transformation	MSE
Linear	1.111269
Polynomial-2	1.095797
Polynomial-3	1.094223
Polynomial-4	1.093318
Logarithmic	1.257818

3.2.2. K-Fold Cross Validation

Il metodo K-Fold Cross Validation elimina molti svantaggi del metodo del set di validazione. Principalmente, fa un ottimo lavoro assicurando che il bias non penetri nelle prestazioni del modello. Lo fa allenandosi e testando su ciascuno dei sottogruppi in cui vengono divisi i dati. Il numero di sottogruppi (*folds*) scelto per l'analisi è stato di dieci, ed ognuno caratterizzato da una serie casuale di punti dati. Questo numero indica anche la quantità di iterazioni che vengono eseguite per l'addestramento ed il test dei sottogruppi. Le prestazioni complessive del modello vengono calcolate in base all'errore medio in tutte le iterazioni. I valori ottenuti sono riprodotti nella tabella seguente:

Transformation	MSE
Linear	1.120662
Polynomial-2	1.105010
Polynomial-3	1.103584
Polynomial-4	1.102515

I risultati sull'MSE ottenuti sono simili ma leggermente superiori a quelli ottenuti tramite l'approccio Validation Set. Poiché in questo caso i modelli sono stati testati su più set di test differenti, questi ultimi risultati sono da considerarsi più attendibili.

3.2.3. Bootstrap

Un altro metodo di ricampionamento dei dati è quello bootstrap. Bootstrap è un metodo statistico flessibile e potente che può essere utilizzato per quantificare l'incertezza associata ad uno stimatore. La stima generale del bootstrap è la media delle stime ottenute da ciascuna stima del campione bootstrap e avrà una varianza inferiore rispetto a un meccanismo di suddivisione generale del train-test.

Nel progetto viene usato tale approccio per esaminare la discrepanza che esiste tra la stima dello *standard error* dei coefficienti effettuata da bootstrap, e quella effettuata dai modelli di regressione multipla di ogni trasformazione.

Dunque, in questo caso, la misura dell'affidabilità del metodo è stata valutata attraverso un numero variabile di valori, che dipende dal grado del polinomio esaminato.

Ad esempio, vengono mostrati gli *standard error* relativi ai coefficienti della trasformazione lineare:

Coefficients	Standard Error difference
Intercepts	0.0004636287
recommended	-0.006179144
seat_comfort	-0.0006253075
cabin_service	-0.0009786729
food_bev	-0.000399543
entertainment	-0.0005200939
ground_service	-0.001155898
wifi_connectivity	-0.0002902589
value_for_money	-0.001460509

Una piccola differenza nella stima dello standard error è sinonimo di una buona interpretabilità dei dati da parte del modello scelto.

3.3. Subset Selection

La Subset Selection è il processo di selezione di un sottoinsieme di predittori da utilizzare nella costruzione del modello. In questa sezione vengono considerati alcuni dei metodi per la selezione del modello migliore: Best Subset Selection, Forward Stepwise Selection e Backward Stepwise Selection.

Tutti questi metodi sono stati applicati per le diverse trasformazioni, già precedentemente descritte, e sono stati commentati i risultati ottenuti con grafici di supporto.

3.3.1. Best Subset Selection

La funzione `regsubset()` in R, presente nel pacchetto `leaps`, può essere utilizzata per identificare diversi modelli migliori di dimensioni diverse. È necessario specificare il parametro opzionale `nvmax`, che rappresenta il numero di predittori da incorporare nel modello.

La funzione `summary()` riporta il miglior set di variabili per ogni dimensione del modello.

		recommended	seat_comfort	cabin_service	food_bev	entertainment
1	(1)	"*"	" "	" "	" "	" "
2	(1)	"*"	" "	" "	" "	" "
3	(1)	"*"	" "	" "	" "	" "
4	(1)	"*"	" "	"*"	" "	" "
5	(1)	"*"	"*"	"*"	" "	" "
6	(1)	"*"	"*"	"*"	"*"	" "
7	(1)	"*"	"*"	"*"	"*"	"*"
8	(1)	"*"	"*"	"*"	"*"	"*"

		ground_service	wifi_connectivity	value_for_money
1	(1)	" "	" "	" "
2	(1)	" "	" "	"*"
3	(1)	"*"	" "	"*"
4	(1)	"*"	" "	"*"
5	(1)	"*"	" "	"*"
6	(1)	"*"	" "	"*"
7	(1)	"*"	" "	"*"
8	(1)	"*"	"*"	"*"

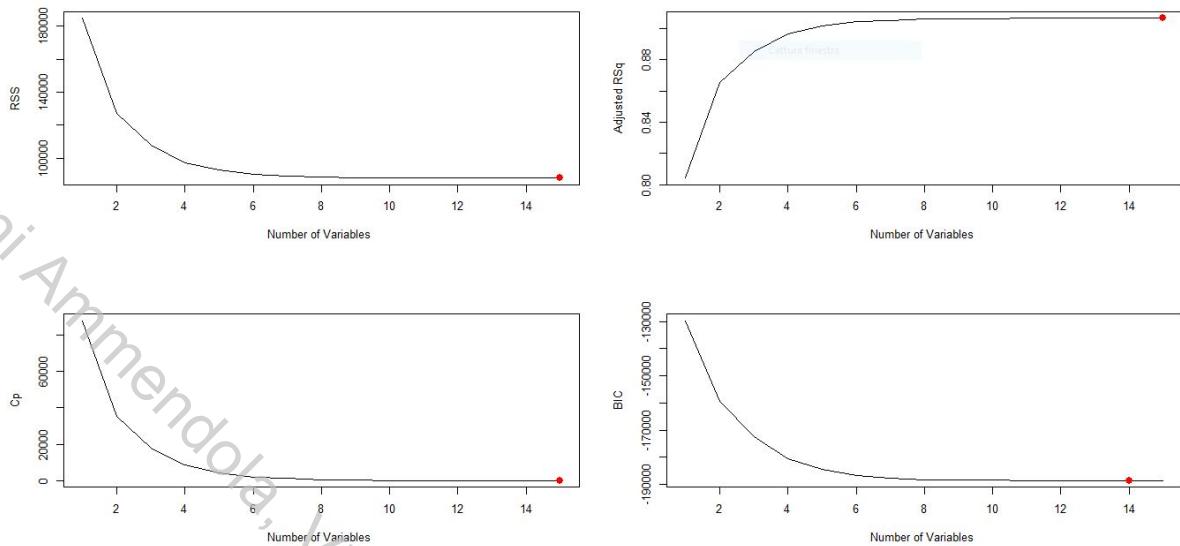
Dall'output sopra, un asterisco specifica che una data variabile è inclusa nel modello corrispondente.

Come viene scelto il modello migliore? Per rispondere a tale quesito, sono necessarie alcune metriche o strategie statistiche per confrontare le prestazioni complessive dei modelli e scegliere quello migliore. E' necessario stimare l'errore di previsione di ciascun modello e selezionare quello con l'errore minimo.

I criteri di selezione del modello sono: R^2 , Cp e BIC. La funzione `summary()` restituisce alcune metriche (Adjusted R^2 , Cp, BIC), permettendoci di identificare il modello migliore complessivo; per "migliore" si intende un modello che massimizza R^2 e minimizza l'errore di previsione (RSS, Cp, BIC). Non esiste un'unica soluzione corretta per la selezione dei modelli, ciascuno di questi criteri porterà a modelli leggermente diversi. Infatti nel caso in esame, per ogni metrica presa in considerazione, abbiamo ottenuto modelli diversi per le trasformazioni non lineari.

Di seguito verranno mostrati, graficamente, i modelli migliori rispetto alle metriche sopracitate.

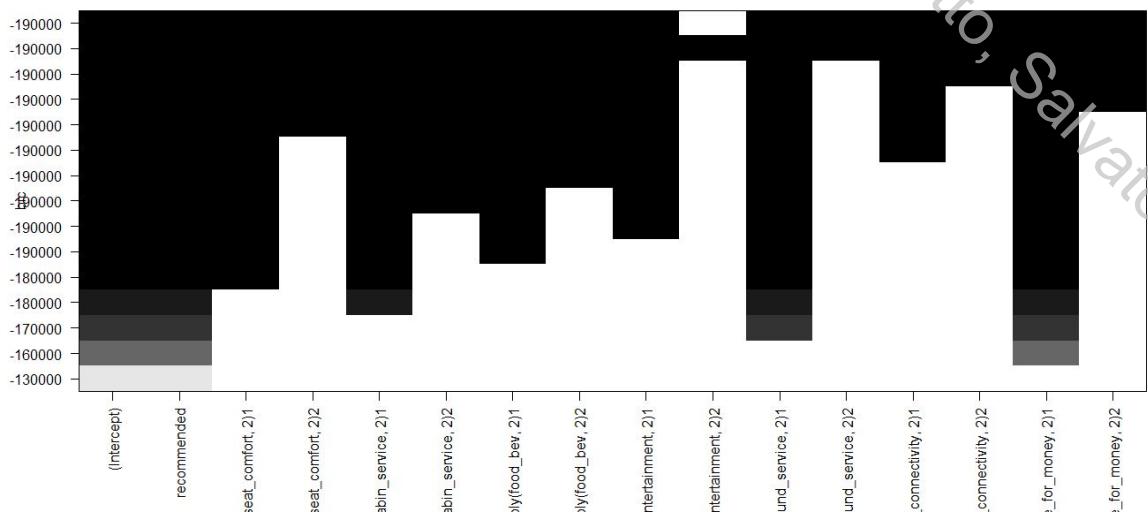
Trasformazione polinomiale di secondo grado



- *Grafico in alto a sinistra:* mostra l'RSS all'aumentare dei predittori all'interno del modello (min a 15 predittori);
- *Grafico in alto a destra:* mostra l'Adjusted R² all'aumentare dei predittori (max a 15 predittori);
- *Grafico in basso a sinistra:* mostra il Cp all'aumentare dei predittori (min a 15 predittori);
- *Grafico in basso a destra:* mostra il BIC all'aumentare dei predittori (min a 14 predittori).

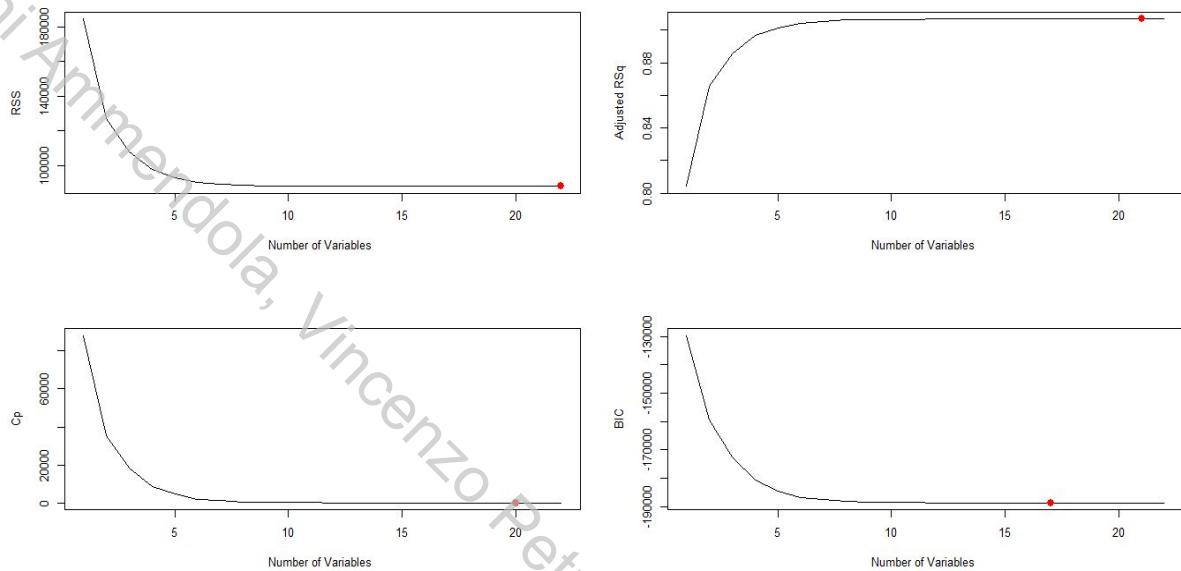
Come possiamo vedere, abbiamo modelli diversi: mentre per il BIC abbiamo il modello migliore a 14 predittori, per le altre metriche il modello migliore è quello a 15 predittori.

Per capire quale predittore è stato eliminato per minimizzare il BIC, è stata utilizzata una rappresentazione di questo tipo:



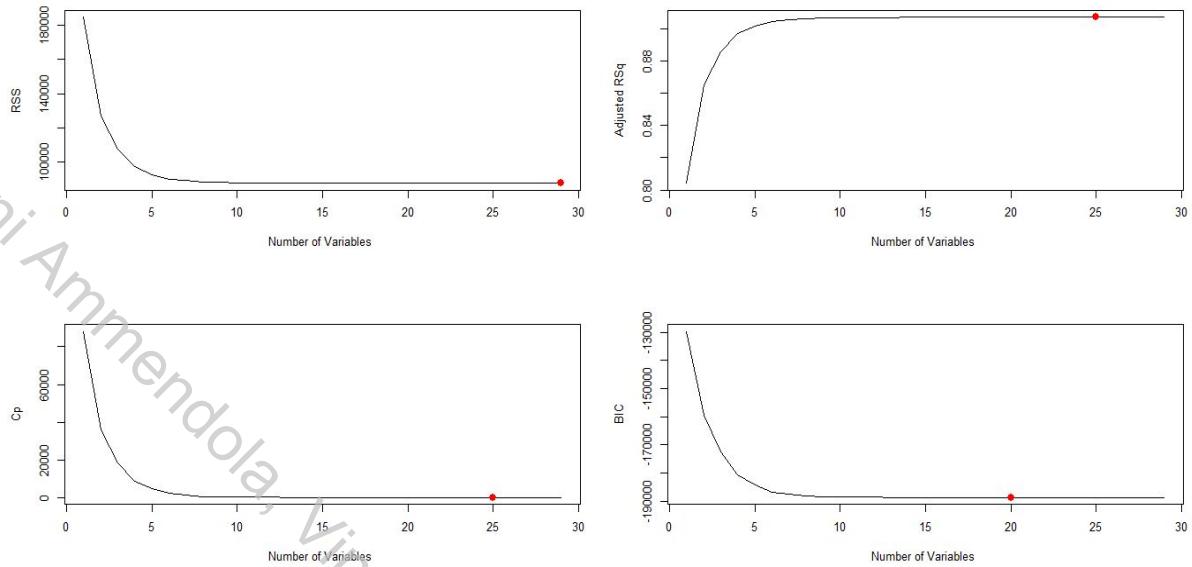
Il miglior modello che minimizza il BIC si ottiene considerando la prima riga del grafico (partendo da sopra). Ogni quadrato nero indica che il predittore corrispondente va inserito nel modello.

Trasformazione polinomiale di terzo grado



- RSS : min a 22 predittori;
- $Adjusted R^2$: max 21 predittori;
- BIC : min a 17 predittori;
- Cp : min a 20 predittori.

Trasformazione di quarto grado



- *RSS*: min a 29 predittori;
- *Adjusted R²*: max a 25 predittori;
- *BIC*: min a 20 predittori;
- *Cp*: min a 25 predittori.

3.3.2. Stepwise Selection

La Stepwise Selection (*regressione graduale*) è un metodo per adattare i modelli di regressione in cui la scelta delle variabili predittive viene effettuata mediante una procedura automatica. Ad ogni passaggio, viene considerata una variabile per l'aggiunta (*forward*) o la sottrazione (*backward*), dall'insieme di tutte le variabili.

Forward Stepwise Selection

Tale approccio prevede di iniziare con una sola variabile nel modello, di testare l'aggiunta di ogni variabile utilizzando un criterio di adattamento del modello scelto, aggiungendo la variabile. Se l'inclusione di quest'ultima fornisce un miglioramento significativo dell'adattamento, la variabile viene aggiunta e il processo viene iterato fin quando nessuna variabile aggiunta ottimizzerà il modello.

Backward Stepwise Selection

Questa tecnica impone di iniziare con tutte le variabili in questione, di testare la cancellazione di ogni variabile utilizzando un criterio, ed eventualmente eliminare la variabile. Se la perdita di quest'ultima determina un deterioramento insignificante del modello, la variabile viene eliminata e il processo viene iterato fin quando ulteriori variabili rimosse non produrranno perdite significative.

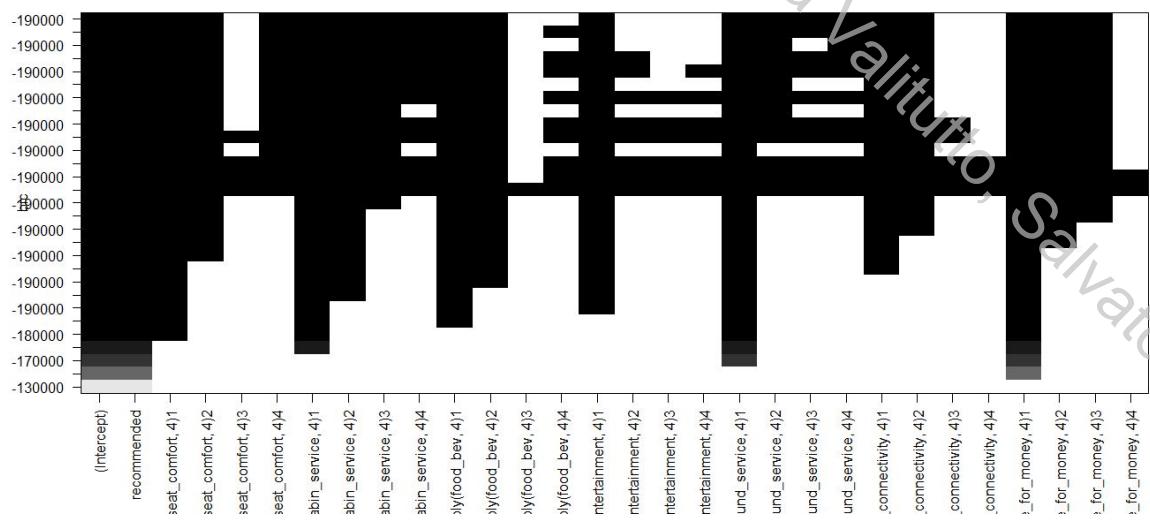
3.3.3. Considerazioni finali

Tutta questa premessa sul meccanismo che sta dietro la generazione di modelli mediante questi due approcci ci serve per mettere in evidenza un risultato fondamentale. Infatti, la risposta prodotta da queste due tecniche, in termini di valori parametrici, è la stessa di quella ottenuta attraverso la **best subset selection**. Questo fa capire come non ci sia una discordanza, nonostante l'algoritmo utilizzato per la ricerca del modello migliore sia diverso. Per valutare la prestazione del modello migliore, trovato da queste tre tecniche, è stato utilizzato il *Validation Set Approach*. E' stato realizzato il fit sul training set e poi stimato l'MSE sul test set. Il modello migliore che fornisce il minimo MSE è quello con trasformazione di ordine quattro dei predittori.

Transformation	Predictors considered	Min MSE
Linear	8	1.106853
Polynomial-2	14	1.090422
Polynomial-3	22	1.089150
Polynomial-4	28	1.087877

L'MSE minimo si ha con un modello di 28 predittori su un totale di 29 predittori.

Constatato ciò, il successivo lavoro di confronto è stato svolto prendendo in considerazione la trasformazione del quarto ordine, andandola a particolarizzare sul parametro BIC. Difatti, il valore del numero dei predittori con questo parametro risulta minimo:



Come conferma di quanto affermato [precedentemente](#), i predittori non considerati da questi approcci di subset selection sono **seat_comfort**, **food_bev**, **entertainment** e **wifi_connectivity**.

Questa analisi è stata necessaria per costruire un successivo modello, in accordo ai grafici che mostrano l'andamento di RSS, Adjusted R², Cp e BIC al variare del numero di predittori. Seguendo la *one-standard-error-rule*, il modello selezionato è quello che minimizza il BIC, in quanto è quello che prevede il minor numero di predittori (20). Per un numero di predittori superiori a 20, gli altri parametri non migliorano in maniera significativa; dunque, anche se gli altri parametri suggeriscono di selezionare un modello più complesso, a parità di performance è preferibile selezionare il modello più semplice, al fine di prevenire problemi di overfitting dovuti ad un'eccessiva complessità del modello. Tale modello è stato comparato con un altro, descritto ancora da una trasformazione di grado quattro, ma avente l'insieme completo dei predittori. Il confronto è stato misurato con anova (model1, model2), producendo il seguente output:

```

Model 1: overall ~ recommended + poly(seat_comfort, 2) + I(seat_comfort^4) +
          poly(cabin_service, 4) + poly(food_beve, 2) + entertainment +
          poly(ground_service, 4) + poly(wifi_connectivity, 2) + poly(value_for_money,
          3)
Model 2: overall ~ recommended + poly(seat_comfort, 4) + poly(cabin_service,
          4) + poly(food_beve, 4) + poly(entertainment, 4) + poly(ground_service,
          4) + poly(wifi_connectivity, 4) + poly(value_for_money, 4)
Res.Df   RSS Df Sum of Sq    F    Pr(>F)
1 79555 87714
2 79546 87640  9    74.282 7.4913 4.922e-11 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

Come si può notare, la differenza esistente tra le due versioni dei modelli è minima, come giustifica il parametro **Residual Sum of Square**, che varia di appena 74 punti.

3.4. Regressione con Regolarizzazione

In un modello di machine learning, la regolarizzazione riduce significativamente la varianza del modello, senza un sostanziale aumento del suo Bias. Attraverso l'introduzione di un parametro λ , si punta a ridurre i coefficienti del modello in modo da ridurre la varianza e quindi l'accuratezza stessa. Le due tecniche di regolarizzazioni affrontate in questo progetto sono Ridge e LASSO. Anche in questo caso il modo di lavorare è stato lo stesso adoperato per la Subset Selection: analizzare i risultati prodotti dai modelli di diverse trasformazioni e confrontarli per valutare quale abbia un “peso” migliore. Inoltre, data l'influenza di λ , si è preferito usare la regola del *cross-validation* per stimarla in maniera più affidabile, invece di usare ben precisi valori arbitrari. In questo modo, è stato possibile calcolare il miglior valore di λ , in modo da ottenere risultati più vantaggiosi.

3.4.1. Ridge

Con regressione Ridge ci si riferisce ad un modello di regressione lineare i cui coefficienti non sono stimati dal metodo dei *minimi quadrati* (OLS), ma da un altro stimatore, chiamato **stimatore ridge**, che possiede bias, ma ha una varianza inferiore rispetto allo stimatore OLS.

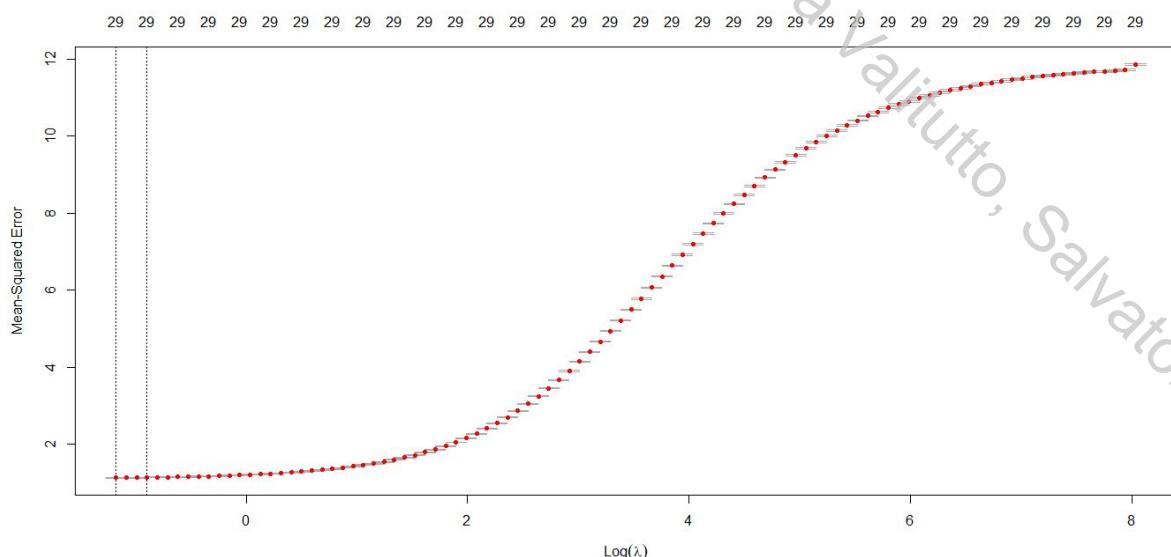
Lo stimatore ridge riduce i coefficienti di regressione, in modo che le variabili, con un contributo minore al risultato, abbiano i loro coefficienti vicini allo zero. Invece di forzarli a essere esattamente zero, vengono penalizzati con un termine chiamato **norma L2**, costringendoli così a essere piccoli in modo continuo. In questo modo, diminuiamo la complessità del modello senza eliminare nessuna variabile. L'ammontare della penalità può essere messo a punto usando una costante chiamata Lambda (λ):

$$\hat{\beta}_\lambda = \arg \min_b \sum_{i=1}^n (y_i - x_i b)^2 + \lambda \sum_{k=1}^K b_k^2$$

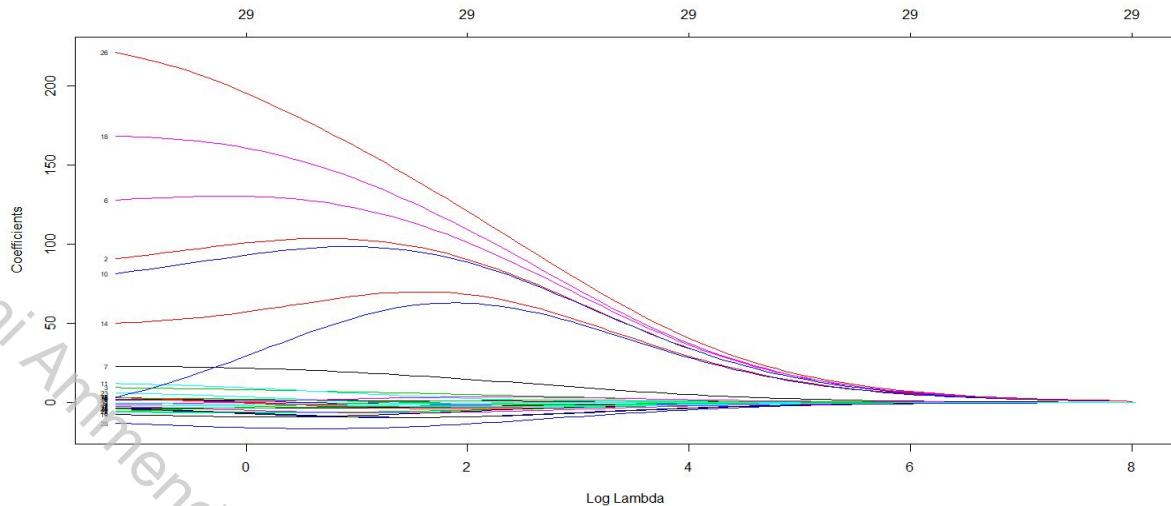
Quando $\lambda = 0$, il termine di penalità non ha alcun effetto e la regressione ridge produrrà i coefficienti minimi quadrati classici. Quando λ aumenta all'infinito, l'impatto della penalità aumenta e i coefficienti di regressione si avvicinano allo zero. La regressione Ridge è particolarmente vantaggiosa rispetto al metodo dei minimi quadrati in una situazione in cui si hanno molti dati multivariati con il numero di predittori maggiore del numero di osservazioni.

3.4.1.1. Analisi e osservazioni

Per mostrare a titolo qualitativo i risultati ottenuti, l'analisi seguente verrà commentata solo sul caso della trasformazione non lineare di quarto grado. Per poter utilizzare una regressione con regolarizzazione in *RStudio*, è necessario utilizzare la funzione `glmnet()` presente nell'omonimo package. Dapprima, è stata creata una griglia di 100 valori da 10^{-2} a 10^{10} disponibili per λ . In seguito, sono stati testati i modelli con un valore di λ arbitrario all'interno di tale griglia e, successivamente, con il miglior λ restituito dalla *cross-validation*. È stato confermato che, come atteso, all'aumentare di lambda i coefficienti sono diminuiti. Il cambiamento del valore di λ in relazione all' MSE sul test test viene rappresentato nel grafico seguente:



Per di più, un'altra osservazione da notare riguarda la variazione dei coefficienti del modello in analisi, in base al crescere di λ all'interno della griglia citata prima:



I risultati ottenuti, dopo aver effettuato il fit sul training set e valutazione dell'MSE sul test set sono i seguenti:

Transformation	MSE no regularization	MSE regularization
Linear	1.120662	1.128454
Polynomial-2	1.105010	1.11337
Polynomial-3	1.103584	1.112249
Polynomial-4	1.102515	1.111561

Quello che è emerso, considerando le risposte ricavate, è stato che il metodo Ridge non è adatto al caso in esame perché trova maggiore affidabilità nel caso in cui il numero di predittori è maggiore rispetto al numero di osservazioni, il quale risulta in contrapposizione con il problema in questione. Il leggero aumento dell'MSE è imputabile all'aumento del bias introdotto dalla regolarizzazione.

3.4.2. LASSO

LASSO (**Least Absolute Shrinkage and Selection Operator**), riduce i coefficienti di regressione verso lo zero, penalizzando il modello di regressione con un termine di penalità chiamato **norma L1**, che è la somma dei coefficienti assoluti. L'equazione di una regressione LASSO è molto simile a quella della regressione ridge:

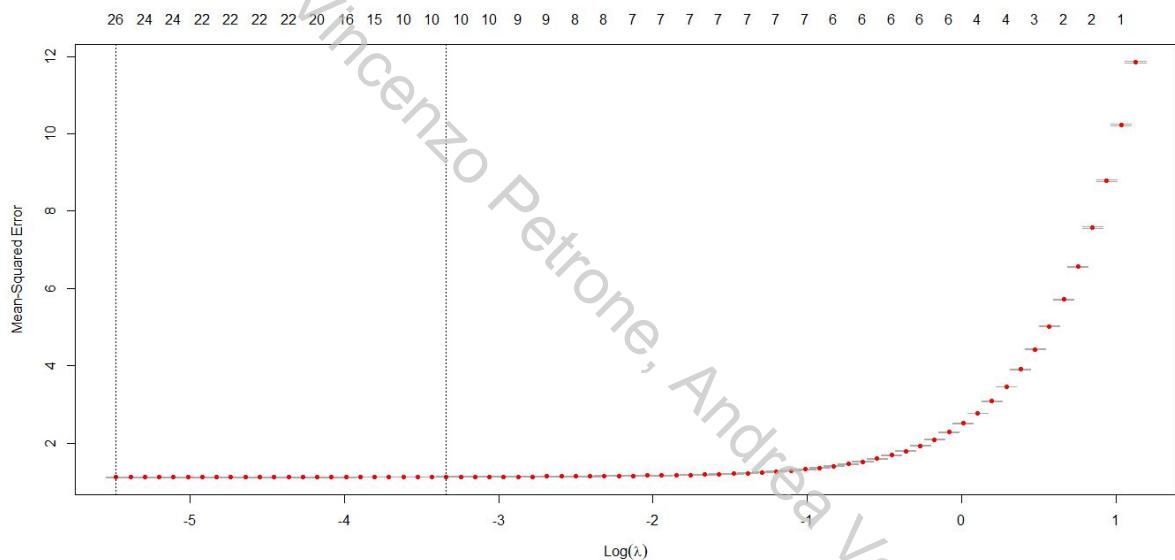
$$\hat{\beta}_\lambda = \arg \min_b \sum_{i=1}^n (y_i - x_i b)^2 + \lambda \sum_{k=1}^K |b_k|$$

La penalità ha l'effetto di forzare alcune delle stime dei coefficienti a essere esattamente uguale a zero. Ciò significa che il LASSO può anche essere visto come un'alternativa ai

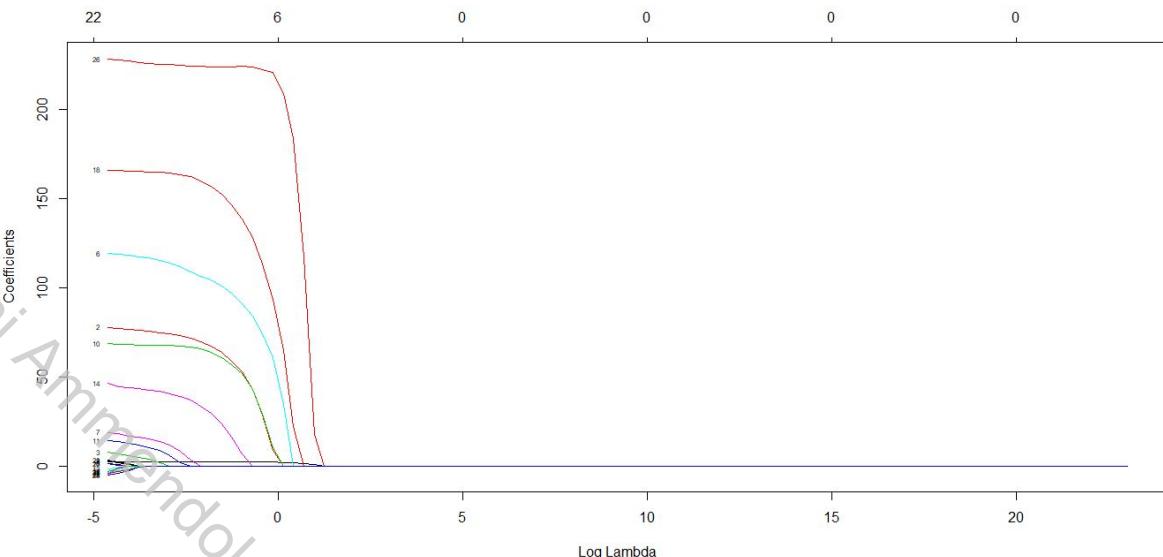
metodi di **feature selection** per eseguire la selezione delle variabili al fine di ridurre la complessità del modello. Quando λ è piccolo, il risultato è molto vicino alla stima dei minimi quadrati. All'aumentare di λ , si verifica una contrazione in modo da poter eliminare le variabili che sono a zero. Un ovvio vantaggio della regressione LASSO rispetto alla ridge, è che produce modelli più semplici e più interpretabili che incorporano solo un insieme ridotto di predittori.

3.4.2.1. Analisi e osservazioni

I dati forniti dai modelli in esame sono risultati particolarmente significativi per arrivare alla deduzione di quanto possa incidere un'analisi con regolarizzazione LASSO. E' stata utilizzata la stessa griglia di valori per λ utilizzata nella regressione Ridge, così come è stata realizzata la *cross-validation* per stimare il λ migliore. Il valore di MSE al crescere del fattore Lambda viene evidenziato dal seguente grafico:



In quest'altra rappresentazione, invece, vengono messe in risalto le curve che disegnano i coefficienti in funzione della misura di Lambda, presente sulle ascisse del grafico :



Sono stati riprodotti all'interno di una tabella i valori ottenuti dopo aver effettuato il fit sul training set, insieme alla valutazione dell'MSE sul test set:

Transformation	MSE no regularization	MSE regularization	Coefficients equal to zero
Linear	1.120662	1.112508	1
Polynomial-2	1.105010	1.097244	0
Polynomial-3	1.103584	1.096104	4
Polynomial-4	1.102515	1.095203	7

Un ultimo aspetto da mettere in risalto riguarda la differenza di valori conseguiti con i processi Ridge e LASSO. Con cognizione che Ridge non produce sostanziali risultati nel dataset in esame, LASSO, di controparte, produce modelli molto competitivi e comparabili con i modelli senza regolarizzazione, con il sostanziale beneficio di ridurre i coefficienti a zero e quindi produrre modelli molto più interpretabili.

Inoltre, si fa notare che i predittori dei quali la tecnica LASSO riduce a 0 i coefficienti sono molto simili a quelli non selezionati dalle tecniche di Subset Selection riportati [precedentemente](#).

3.5. PCR

PCR (Principal Component Regression), è una tecnica di regressione basata sull'analisi delle componenti principali (**PCA**), consiste nel costruire M componenti principali da p predittori e utilizzarle come nuovi predittori in un modello di regressione lineare. Poiché le

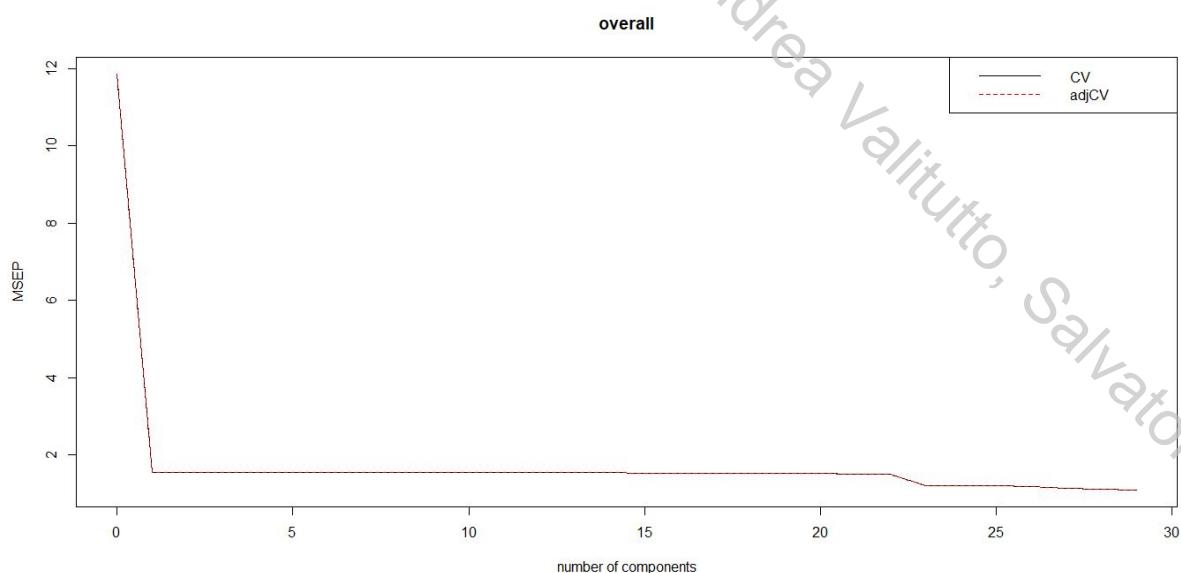
componenti principali cercano di preservare la variabilità dei predittori, l'assunzione di base, non necessariamente vera, è che questo preservi al massimo anche la relazione con Y.

Tuttavia, la PCR non è una tecnica di selezione delle variabili, poiché usa sempre tutti i predittori originali nella costruzione delle combinazioni lineari. La tecnica risulta vantaggiosa quando poche componenti riescono a riassumere una parte sostanziale della variabilità totale nei p predittori. Il parametro M di variabili da usare nella PCR è tipicamente scelto con *cross-validation*.

3.5.1. Analisi e Osservazioni

Lo sviluppo dell'analisi sul procedimento PCR ha previsto gli stessi step affrontati per i metodi precedentemente esaminati. Ovvero sono state sperimentate tutte le trasformazioni considerate in esame, partendo da quella lineare e giungendo a quella di polinomio quattro. Ancora una volta, si è tenuto traccia dei risultati prodotti di volta in volta, dedicando maggiore attenzione al confronto tra di essi. Sempre in supporto, sono stati generati dei grafici che mostrassero l'andamento dei parametri essenziali alla comprensione dell'attendibilità del modello. A dimostrazione di esempio, in seguito verranno presentati delle tabelle riassuntive e dei grafici per il modello relativo alla trasformazione di grado quattro.

Per utilizzare questa tecnica nell'ambiente di sviluppo *RStudio*, è necessario utilizzare la funzione `pcr()` presente nel package `pls`. Effettuando il `summary()` dell'oggetto restituito dalla funzione precedentemente citata, vengono mostrati due risultati principali, vale a dire “*Root MSE*” e la percentuale di varianza spiegata utilizzando n componenti. Per impostazione predefinita, la funzione `pcr()` calcola il **Root MSE**, ma è possibile specificare di mostrare l'MSE come possiamo vedere nel grafico sottostante:



Come si può notare anche dal grafico, l'MSE minimo si ottiene considerando tutti e 29 i predittori. Successivamente, è stata effettuata una regressione con le componenti principali sul training set e testato su un test set. Sono stati ricalcolati RMSE e l'MSE, ma i risultati

sono stati analoghi a quelli calcolati con la regressione semplice, poiché non c'è alcun beneficio nel ridurre il numero di componenti da considerare.

Inoltre, utilizzando una trasformazione del quarto ordine dei predittori, la percentuale della varianza spiegata sull'overall risulta essere il 90.71% considerando tutti i predittori. Per tutte le trasformazioni realizzate per la costruzione di un modello predittivo, usando la PCR i risultati sono analoghi alla regressione semplice, dunque la tecnica di riduzione della dimensionalità non è risultata efficace.

3.6. PLS

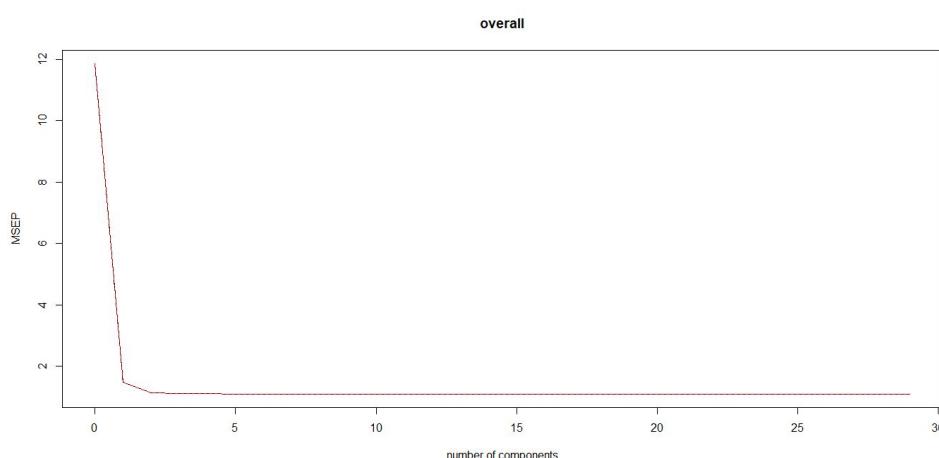
A differenza della PCR, che non considera la relazione esistente tra X_1, \dots, X_p e Y nella costruzione delle variabili Z_1, \dots, Z_M , il metodo PLS (**Partial Least Squares**) cerca di considerare questo aspetto. Dunque, la PCR può essere vista come una tecnica di statistical learning unsupervised, mentre il metodo PLS è una tecnica supervised.

3.6.1. Analisi e Osservazioni

Le osservazioni a cui si è giunti, durante il procedimento PLS, permettono di fare alcune considerazioni fondamentali. Prima di ciò, è opportuno sottolineare come l'applicazione della tecnica PLS è stata realizzata per tutte le trasformazioni in esame. A rafforzare le conseguenze espresse dai risultati, sono state realizzate delle rappresentazioni che tenessero conto delle variazioni dei singoli parametri presi in esame. Queste rappresentazioni non sono altro che tabelle e grafici riassuntivi che verranno presentati di volta in volta successivamente. Da sottolineare come, per motivi di brevità, verranno raffigurati solamente risultati caratterizzanti la trasformazione del polinomio di grado quattro.

In RStudio per poter utilizzare questa tecnica di regressione è necessario utilizzare la funzione `pls()` presente nel package `pls`. Il `summary()` prodotto è lo stesso del PCR, ovvero, è presente il *cross-validation* del Root MSE e la varianza spiegata al variare del numero di componenti. Anche in questo caso c'è un validation plot per capire il numero di componenti che minimizza l'MSE e il RMSE.

Analisi sull'intero Dataset:



Per la trasformazione di grado quattro, come è possibile vedere nel grafico, il minimo MSE viene fornito con un modello composto da 12 componenti. Mentre per le altre restanti trasformazioni, il numero di componenti coincidenti con il minimo MSE è raffigurato nella tabella sottostante:

Transformation	Components Number (min. MSE)
Linear	8 su 8
Polynomial-2	14 su 15
Polynomial-3	15 su 22
Polynomial-4	12 su 29

La successiva analisi effettuata ha previsto lo studio dei valori del PLS per un approccio che mostrava l'addestramento sulla parte del set di train e la valutazione sul set di dati di test:

Transformation	Components Number	min. MSE
Linear	7	1.111269
Polynomial-2	9	1.095787
Polynomial-3	11	1.094217
Polynomial-4	10	1.093311

Transformation	Components Number	PEV
Linear	4 su 8	90.55%
Polynomial-2	5 su 15	90.68%
Polynomial-3	7 su 22	90.70%
Polynomial-4	6 su 29	90.71%

Il vantaggio della PLS, rispetto alla PCR, è quello di riuscire a spiegare una maggiore varianza della Y con un minor numero di componenti, come mostra la tabella precedente, invece di guardare soltanto i regressori e minimizzare la proiezione dei soli regressori in uno spazio più piccolo, adesso si coinvolge anche la variabile Y. Quindi si cerca di ottimizzare la variabilità di X e della Y stessa, infatti i risultati cui si è giunti sono i seguenti:

- Il numero di predittori considerati dalla PLS è inferiore rispetto a quelli considerati dalla PCR
- La massima varianza spiegata dai predittori si raggiunge attraverso un minor numero variabili
- Per quanto riguarda l'MSE, i suoi valori sono in linea con quelli trovati con Ridge e LASSO.