Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica per il Management

Demand Forecasting mediante algoritmi di boosting: una valutazione sperimentale

Relatore:
Chiar.mo Prof.
MARCO DI FELICE

Correlatore:
Dott. MICHAEL ROMOLO

Presentata da:
MIRIAM CRUDELINI

Sessione II
Anno Accademico 2018/2019

Ai miei genitori

Introduzione

Nell'epoca in cui viviamo, grazie ai dispositivi a nostra disposizione, ognuno di noi è produttore di una grande mole di dati, all'interno dei quali sono racchiuse importanti informazioni.

Il processo di analisi ed estrazione della conoscenza sta diventando sempre più attraente agli occhi delle aziende, perchè queste informazioni possono rivelarsi utili e generare vantaggio competitivo.

Un'importante utilizzo della conoscenza estratta dai dati si ha nel *demand forecasting*. Il demand forecasting è il processo di previsione della domanda futura di un prodotto o servizio offerto, e permette di ottimizzare le decisioni aziendali, ridurre i rischi, gestire le scorte, pianificare le vendite ed effettuare tante altre valutazioni interne all'azienda o sul mercato.

In questa tesi verranno analizzate alcune metodologie per effettuare previsioni sulla domanda di un prodotto, al fine di creare uno strumento di supporto per le aziende. Per il raggiungimento di tale obiettivo mi concentrerò solamente su una tipologia di algoritmi spesso utilizzati in questo ambito.

Il progetto è stato svolto all'interno della società Iconsulting Spa, presso la quale ho effettuato il tirocinio.

Iconsulting Spa è una società di consulenza informatica fondata nel 2001 da ex-ricercatori. È specializzata in sistemi a supporto delle decisioni e si occupa di progetti di Business Intelligence, Data Warehouse, Big data e Performance management.

Iconsulting supporta il cliente, aiutandolo a comprendere le proprie performance e migliorarle attraverso le opportune decisioni.

Per lo svolgimento del progetto di tesi, sono stata inserita in un team che attualmente segue un'azienda che produce occhialeria di lusso. Il nome di tale azienda verrà omesso per motivi di privacy, quindi mi riferirò ad essa in modo generico.

Il cliente gestisce una start-up nel settore occhialeria in ambito "Fashion Retail". Tale start-up è nata da una holding multinazionale responsabile delle vendite di prodotti. Per massimizzare lo sviluppo dei propri marchi, l'azienda ha deciso di internalizzare la catena del valore per le attività dedicate al settore dell'occhialeria riguardanti la gestione delle vendite.

Due volte l'anno l'azienda organizza un evento al quale partecipano gli agenti di vendita di tutte le sue organizzazioni commerciali. L'obiettivo è quello di presentare i prodotti che verranno messi in vendita durante la stagione successiva. Le due stagioni di vendita sono "Primavera/Estate" e "Autunno/Inverno".
Durante l'evento vengono mostrati prodotti che possono essere nuove produzioni oppure vecchi modelli. Gli agenti di vendita hanno la possibilità di osservare i prodotti ed effettuare una previsione di vendita su essi.
Concluso l'evento, tali previsioni verranno utilizzate per determinare la quantità iniziale di produzione dell'organizzazione.

L'obiettivo del progetto è quello di fornire uno strumento che sia di supporto agli agenti di vendita, in modo che possano effettuare delle previsioni su base scientifica oltre che esperienziale.
Per il raggiungimento di tale scopo sono stati proposti e valutati tre algoritmi di machine learning basati sul boosting.

Per migliorare le prestazioni è stata implementata una fase iniziale di ottimizzazione dei modelli che si è sviluppata in due modi.
Il primo approccio seguito è stata l'ottimizzazione manuale, in cui sono stati

scelti autonomamente i valori dei parametri e inseriti in una matrice. Su essa è stato poi eseguito il training e, dopo aver confrontato i risultati, sono stati selezionati i valori ottimali.
Nel secondo metodo applicato si è eseguita un'ottimizzazione guidata, in cui si è definito un valore minimo e uno massimo per i parametri scelti. Il risultato ottenuto da questa ottimizzazione è la combinazione dei valori ottimali.
I modelli costruiti sono stati addestrati sull'apposito set di dati e successivamente testati su un altro insieme. Infine, i risultati ottenuti sono stati riportati e analizzati.

Di seguito presenterò la struttura della tesi, facendo una panoramica generale dei capitoli di cui si compone.

Nel primo capitolo verranno presentati i concetti fondamentali utili a comprendere il contesto in cui si inserisce l'elaborato. Ci sarà quindi un'introduzione al Machine Learning, al Data Mining e ai modelli di apprendimento.
Nel secondo capitolo verranno illustrati gli strumenti usati per lo sviluppo del progetto: l'ambiente di programmazione e le relative librerie.
Nel capitolo successivo verrà fatta una panoramica sullo svolgimento del processo di previsione.
Nel quarto capitolo si entrerà nel dettaglio del lavoro svolto, dalla fase di pre-processing al test dei modelli implementati.
Infine, nell'ultimo capitolo saranno riportati e analizzati i risultati ottenuti con l'utilizzo dei modelli applicati.

Indice

Introduzione **i**

1 Stato dell'arte **1**

1.1 Machine Learning 1
1.2 Storia del machine learning 2
1.3 Data Mining 3
1.4 Modelli di apprendimento 5
1.4.1 Supervised learning 5
1.4.2 Unsupervised learning 8
1.5 Ensemble Learning 10
1.5.1 Bagging 11
1.5.2 Boosting 12
1.5.3 Random Forest 14
1.5.4 Stacking 14
1.6 Demand forecasting 15
1.6.1 Altri casi 17

2 Strumenti **21**

2.1 R 21
2.2 Caret 24
2.3 Gbm 25
2.4 Xgboost 27
2.5 Mlr 28

3 Progettazione **31**
3.1 Il dataset 31
3.2 Pre-processing 32
3.2.1 Anova 33
3.2.2 Boruta 33
3.3 Training e Test 33

4 Implementazione **37**
4.1 Pre-processing 37
4.2 Training 39
4.2.1 Prima modalità di ottimizzazione 39
4.2.2 Seconda modalità di ottimizzazione 45

5 Risultati ottenuti **51**
5.1 I parametri 51
5.2 Accuratezza 53

Conclusioni **59**

Bibliografia **61**

Elenco delle figure

1.1 Processo KDD 5
1.2 Bagging 12
1.3 Boosting 13
1.4 Stacking 15
1.5 Demand Forecasting 16

2.1 R 22
2.2 CUI R 23
2.3 RStudio 24

5.1 RMSE modelli 53
5.2 RMSE modelli 54
5.3 RMSE modelli Italia 55
5.4 RMSE modelli Cina 55
5.5 R^2 modelli Italia 57
5.6 R^2 modelli Cina 57

Elenco delle tabelle

5.1 Ottimizzazione manuale xgboost e gblinear 52
5.2 Ottimizzazione manuale gbm 52
5.3 Ottimizzazione guidata xgboost e gblinear 52
5.4 Ottimizzazione guidata gbm 53
5.5 Riassunto risultati 58

Capitolo 1

Stato dell'arte

1.1 Machine Learning

Il *machine learning* è il settore dell'intelligenza artificiale che si occupa dello sviluppo di sistemi in grado di apprendere da dati, piuttosto che attraverso la programmazione esplicita.[1]

Il principale obiettivo del machine learning è rendere le macchine intelligenti, cioè capaci di imparare dalle esperienze e, in base a queste, prendere decisioni o modificare il proprio comportamento.
Una delle più recenti definizioni di machine learning è quella di Tom M. Mitchell, professore dell'Università di Carnegie Mellon:

> "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." [2]

Traducendo: "Si dice che un programma impara da una certa esperienza E rispetto a una classe di compiti T ottenendo una performance P, se la sua perfomance nel realizzare i compiti T, misurata dalla performance P, migliora con l'esperienza E.[1]"

[1] *E, T, P: Task, Experience, Performance.*

Quindi un programma è in grado di apprendere dall'esperienza se, nel realizzare un compito, le sue prestazioni migliorano durante il periodo di tempo in cui si ripete la stessa attività.

La necessità di ricorrere al machine learning nasce dal fatto che prevedere a priori l'intero set di possibili comportamenti in base all'input è complesso da descrivere in un linguaggio di programmazione. Per questo motivo vengono usati algoritmi diversi che consentono agli elaboratori di evolvere il proprio comportamento basandosi sui dati per migliorare, descrivere e prevedere i risultati. In generale, un algoritmo di machine learning parte da un set di dati su cui si allena e in seguito, apportando le giuste modifiche, permette di ottenere un modello accurato.

La continua presenza dei sistemi informativi nelle nostre vite, e la mole di dati che vengono creati in questo modo, rendono necessarie le tecnologie adatte a gestire lo storage e ad estrarre conoscenza dai dati stessi.
Per questo motivo, il machine learning sta diventando sempre più un aspetto centrale all'interno di organizzazioni di sviluppo, al fine di poter sfruttare i dati ed acquisire informazioni su essi.
Utilizzando il giusto modello, in breve tempo è possibile effettuare delle analisi esplorative, per capire la realtà attuale dell'impresa, oppure delle analisi predittive, per prevedere e anticipare i cambiamenti.

1.2 Storia del machine learning

Le prime sperimentazioni per la realizzazione di macchine in grado di apprendere in modo automatico risalgono all'inizio degli anni '50, quando gli studiosi cominciarono a pensare di utilizzare metodi statistici per le macchine, in modo da renderle in grado di prendere decisioni, tenendo conto delle probabilità di accadimento di un evento.

Il primo a parlare di intelligenza artificiale fu Alan Turnig, il quale nel 1950 scrisse un articolo intitolato "Computing machinery and intelligence" in cui descriveva il "Test di Turing". Turing era convinto che fosse possibile

seguire gli schemi del cervello umano per sviluppare un'intelligenza artificiale. Questo articolo fu la base dei successivi studi sull'intelligenza artificiale. Pochi anni dopo, nel 1952, Arthur Samuel scrisse il primo programma computer learning: si tratta del gioco della dama per IBM.
Il 1956 viene considerato l'anno in cui è nato il campo dell'Intelligenza Artificiale (AI) quando, durante la conferenza di Dartmouth, viene coniato il termine per indicare questo nuovo settore. [3]
Negli anni successivi vengono scritti i primi algoritmi di mappatura di percorsi, mentre nel 1990 nascono i primi algoritmi di analisi di grandi quantità di dati, capaci anche di trarre conclusioni successive all'analisi. Questo rappresenta il passaggio da un approccio basato sulla conoscenza, ad uno basato sui dati.

Nei primi anni del 2000 il campo del machine learning si allarga a nuove tecniche e nel 2006 nasce il termine *"deep learning"* per indicare le nuove architetture di reti neurali, in grado di apprendere più dei modelli precedenti. Gli anni successivi sono anni in cui il machine learning resta in continuo aggiornamento, grazie anche alle innovazioni portate dalle grandi società come Google e Facebook. Quest'ultima per esempio, nel 2014 sviluppa un algoritmo per il riconoscimento delle persone, con la stessa precisone di un essere umano.

Oggi stiamo vivendo una terza fase di intelligenza artificiale, basata sulla continua disponibilità di grandi modi di dati e sulla conseguente necessità di analizzarli e studiarli.

1.3 Data Mining

L'applicazione delle tecniche di machine learning ai big data rientra nel campo del *data mining*.

Con il termine *big data* si intendono grandi moli di dati, non strutturati, continuamente disponibili e non analizzabili attraverso le metodologie tradizionali. [2] Questi dati condividono le seguenti caratteristiche:

- **Volume**
- **Velocità**, relativa all'estrazione dei dati, che avviene in tempo reale
- **Varietà**, si fa riferimento alla provenienza dei dati
- **Veridicità**, nel senso di esattezza nel rappresentare la realtà

Il data mining nasce dall'unione della statistica, del machine learning e dell'artificial intelligence, e rappresenta il processo di esplorazione e analisi dei big data, per la ricerca di relazioni fra essi e per l'estrazione di informazioni al fine di individuare pattern[2] utili e significativi[3].
Le tecninche utilizzate nel data mining per l'estrazione delle conoscenze, possono essere di due tipi:

- **Verification-oriented**, quando il sistema si deve occupare di verificare le ipotesi generate da una sorgente esterna;
- **Discovery-oriented**, quando il sistema identifica autonomamente nuovi pattern, partendo dai dati.

È opportuno sapere che il data mining è una parte del processo di Knowledge Discovery in Databases (KDD)[4], il quale è composto da:

1. **Data selection**, comprende la rimozione dei dati non correlati tra loro, in modo tale da ridurre i tempi di analisi;
2. **Data pre-processing**, rimozione dei dati che potrebbero causare problemi o incongruenze;
3. **Data trasformation**, fase in cui i dati vengono trasformati in formati adatti alle tecninche di data mining;

[2] *Pattern: indica una struttura, un modello o una rappresentazione sintetica dei dati, ed è il risultato dell'estrazione.*

[3] *Un pattern è significativo se è comprensibile, potenzialmente utile, valido sui dati con un certo grado di confidenza ed è non noto a priori.*

[4] *KDD: è il processo di automatico di esplorazione dei dati, che ha lo scopo di estrarre nuove informazioni da essi.*

4. **Data mining**, prevede l'analisi al fine di individuare pattern utili;
5. **Evaluation**, fase finale in cui vengono interpretati i risultati ottenuti.

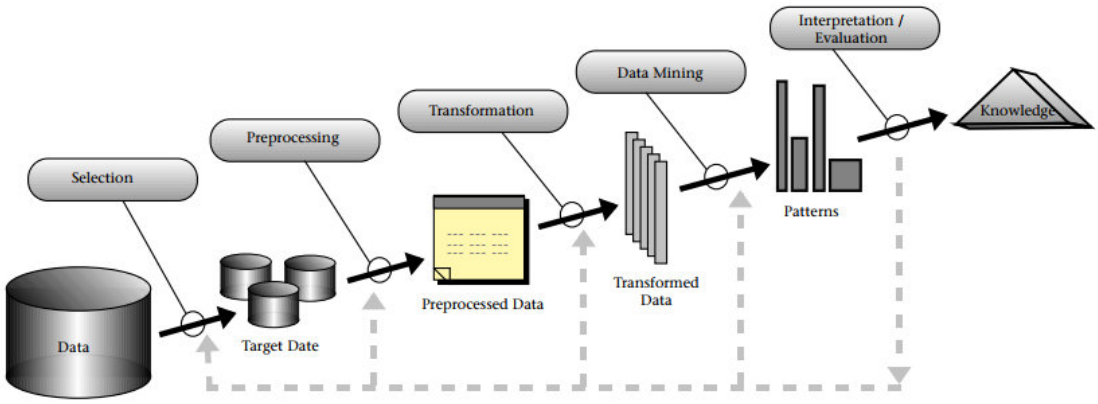

Figura 1.1: Processo KDD

1.4 Modelli di apprendimento

La scelta di un algoritmo dipende fortemente dal tipo di dati che stiamo analizzando e dalla loro struttura, perciò è necessario capire in quale categoria rientra il problema che si sta affrontando.
Esistono principalmente due tipologie di apprendimento all'interno del machine learning:

- Supervised learning
- Unsupervised learning

1.4.1 Supervised learning

Come detto in precedenza, il supervised learning, o apprendimento supervisionato, è una delle principali tipologie di metodi di machine learning.

L'obiettivo dei metodi di apprendimento supervisionato è individuare una funzione da un set di dati di addestramento, detto training set. Il training set è composto da coppie, ognuna delle quali è formata da un input e un output. La funzione da estrapolare rappresenta la relazione presente tra input e output, e deve permettere di attribuire il corretto output per ogni nuovo input non presente nel training set. In altre parole, dato un insieme composto da coppie input-output, si cerca di trovare un modo per mappare correttamente i risultati y partendo da un insieme in input x.

Ogni dato in input, x_i, rappresenta un attributo, o feature, e può essere semplicemente un attributo numerico oppure un oggetto dalla struttura più complessa, come un'immagine. L'output, y_i, rappresenta la variabile di risposta e può essere di ogni tipo, ma la maggior parte dei metodi assume che sia una variabile categorica o numerica. Quando y_i è una variabile categorica, si parla di classificazione, mentre quando è un valore scalare si parla di regressione, utilizzata soprattutto per capire la correlazione fra le variabili.[4]

Un problema che si può riscontrare è che un modello che viene allenato su determinato training set, poi non sia in grado di generalizzare il modello appreso su un set di dati più ampio. In questo caso, si ha una situazione di *overfitting*, ossia il modello risulta essere molto preciso per il training set ma potrebbe non essere applicabile a grandi serie di dati sconosciuti. Per evitare questo problema, il testing va eseguito utilizzando dati imprevisti o sconosciuti, al fine di valutare la precisione del modello di previsione.

I modelli di apprendimento supervisionato hanno ampia applicabilità a una varietà di problemi aziendali, tra cui rilevamento di frodi, soluzioni di raccomandazione, riconoscimento vocale o analisi del rischio. I più importanti metodi di supervised learning sono la classificazione e la regressione.

Classificazione

Essendo uno dei metodi di supervised learning, la classificazione necessita di dati già inclusi in determinate classi per costruire su essi un modello che permetterà di classificare nuovi dati.

Nell'ambito dell'apprendimento supervisionato, l'obiettivo della classificazione è quello di ottenere un modello capace di ordinare correttamente i dati ricevuti in input, in delle classi.

L'attività di classificazione si compone di tre parti: training, test e prediction. Nella prima fase viene costruito un modello a partire dall'insieme di dati già classificati, ricevuti in input. Successivamente, nella fase di test, viene utilizzato il modello precedentemente allenato su un insieme di dati la cui classificazione non è nota, e viene verificata in questo modo l'accuratezza del modello. Infine, dopo aver apportato eventuali modifiche, attraverso il modello viene effettuala la previsione sui dati da classificare.

Regressione

L'approccio della classificazione è simile a quello della regressione, ma quest'ultima si applica quando le variabili da prevedere sono di tipo quantitativo, quindi $x_i \in \mathbb{R}$ e $y_i \in \mathbb{R}$. L'idea alla base della regressione è quella di poter approssimare la funzione di distribuzione che ha generato le osservazioni y_i. La previsione è verificata dallo scarto di errore che viene generato dal modello, rispetto ai valori reali.

La regressione mette in relazione una o più variabili dipendenti (x_i) con una variabile indipendente (y_i), quest'ultima sarà espressa in funzione delle variabili indipendenti e un termine di errore.

Esempi di applicazioni dell'analisi di regressione per affrontare problemi reali sono:

- Prevedere il prezzo del mercato azionario, date le attuali condizioni di mercato e altre possibili informazioni;
- Prevedere l'età di un utente che guarda uno specifico video su Youtube;
- Prevedere la temperatura all'interno di un edificio utilizzando i dati metereologici, l'ora e altri dati.

Alberi decisionali

Gli alberi decisonali rappresentano un metodo usato per costruire un modello di previsione. In base alla natura della variabile da prevedere si distinguono alberi di classificazione e alberi di regressione.
Un albero decisionale è composto da:

- **Nodi** che contengono i nomi delle variabili indipendenti;
- **Archi** che sono etichettati con i possibili valori delle variabili indipendenti;
- **Foglie** che rappresentano le classi, cioè una collezione di osservazioni raggruppate in base ai valori di una variabile indipendente, e sono unite ai nodi tramite gli archi.

Partendo dal nodo radice, le osservazioni vengono attribuite ai nodi successivi sulla base di regole di ripartizione che vengono dedotte dalle caratteristiche dei dati.

1.4.2 Unsupervised learning

Un'altra tipologia di metodi di machine learning è l'unsupervised learning, o apprendimento non supervisionato. Questi algoritmi rappresentano la scelta migliore quando il problema da affrontare riguarda dati non classificati, ossia dati le cui classi non sono note a priori, ma devono essere apprese automaticamente.

L'obiettivo dell'apprendimento non supervisionato è quello di individuare strutture significative nei dati ricevuti in input, così da poterle ricondurre a modelli che permettano di effettuare previsioni, in modo accurato, sui dati futuri.

Le tecniche di apprendimento non supervisionato lavorano confrontando i dati e cercando similarità o differenze fra essi, senza averne una conoscenza preliminare come invece accade nell'apprendimento supervisionato. Questi algoritmi si rivelano molto efficienti con elementi di tipo numerico, dato che

possono utilizzare tutte le tecniche derivate dalla statistica, ma risultano meno efficienti se applicati a dati non numerici.
I principali algoritmi di apprendimento non supervisionato sono il clustering e le regole di associazione.

Clustering

Essendo un algoritmo di apprendimento non supervisionato, il clustering riceve in input un insieme di dati non classificato e permette di ottenere dei set di dati raggruppati in base a caratteristiche simili. Le tecniche di clustering si basano quindi su misure relative alla somiglianza tra gli elementi. La somiglianza può essere espressa anche in termini di distanza in uno spazio multidimensionale, e quindi l'appartenenza o meno ad un insieme dipende da quanto l'elemento preso in esame è distante dall'insieme stesso.

Il clustering può essere top-down, quando si parte da un cluster[5] che viene suddiviso ricorsivamente in gruppi, oppure bottom-up quando ogni oggetto del dataset è un cluster e viene poi unito agli altri simili. In entrambi i casi è necessario stabilire il numero di cluster che si vuole ottenere e il raggiungimento di tale numero rappresenta la condizione di arresto.

Il clustering ha numerose applicazioni pratiche, come il trattamento delle immagini e la ricerca di mercato, ma può essere utile anche nella classificazione dei clienti in base a caratteristiche comuni.

Regole di associazione

Le regole di associazione rappresentano un'altra tecnica di apprendimento non supervisionato, il cui scopo è la ricerca di correlazioni in un insieme di dati. Queste regole sono uno strumento efficace per individuare le relazioni di dipendenza che sono presenti tra gli attributi del dataset, senza avere conoscenze a priori su tali legami.

Si distinguono diversi tipi di regole di associazione, principalmente in base al tipo di attributi che viene analizzato.

[5] *Cluster: dall'inglese significa gruppo.*

Si parla quindi di regole di associazione tradizionali, quando si utilizzano attributi categorici, quantitative quando sono definite su attributi con valori continui e regole di multilivello che vengono definite su gerarchie di generalizzazioni.
Un semplice esempio di regola di associazione è: "Se è presente l'attributo x, allora sarà presente anche l'attributo y".
I campi in cui vengono applicate le regole di associazione sono numerosi, ma l'ambito in cui sono utilizzate tradizionalmente è quello marketing ed in particolare dei recommendation system.

1.5 Ensemble Learning

Il concetto di *ensemble learning* richiama l'utilizzo di diversi modelli, uniti in un modo tale da riuscire a massimizzare le prestazioni usando i punti di forza di ognuno di essi, e limitando le relative debolezze.

I metodi di apprendimento ensemble sfruttano i modelli di apprendimento deboli, i quali utilizzati singolarmente non raggiungono un buon livello di accuratezza, ma combinati fra loro possono generalizzare bene creando un modello forte. In questo contesto, con il termine *"weak learner"* si intende un modello capace di generare delle risposte leggermente migliori di quelle che si otterrebbero in modo casuale, mentre con *"strong learner"* si indica un modello che tende ad avvicinarsi a un modello ideale, ed è capace di risolvere anche i problemi tipici dei modelli tradizionali, come l'overfitting[6].
Ogni modello utilizzato restituisce delle previsioni che possono essere corrette o meno. L'errore di predizione, in un problema di apprendimento supervisionato, è dato da: $prediction\_error = bias^2 + varianza$.

Le due componenti dell'errore sono quindi accuratezza e varianza. L'obiettivo dei modelli che utilizzano le metodologie ensemble è ottenere una

[6] *Overfit: dall'inglese significa sovradattamento, in questo contesto si può avere quando un modello si specializza eccessivamente su set di dati da non saper generalizzare nuovi dati in input.*

predizione attendibile, riducendo varianza e distorsione.
Spesso può esserci un trade-off fra queste due componenti: i classificatori con bassa accuratezza tendono ad avere un'alta varianza, e viceversa.

Per migliorare l'accuratezza del modello si suppone che i vari classificatori commettano errori diversi su ciascun campione, ma anche che concordino sulla loro corretta classificazione. La media degli output del classificatore riduce l'errore calcolando la media dei componenti dell'errore. La combinazione degli output data da un modello, non porta necessariamente a una performance di classificazione sicuramente migliore rispetto a quella data da altri, ma riduce la probabilità di scegliere un classificatore con prestazioni scadenti.

Esistono molti metodi ensemble che si differenziano principalmente nella manipolazione dei dati. I più diffusi sono:

- Bagging
- Boosting
- Stacking
- Random Forest

I primi modelli ensemble risalgono agli anni '90, ma non esiste uno studio specifico che possa aiutare un utente a scegliere il meta-classificatore migliore per le proprie esigenze. Attualmente sono utilizzati spesso per affrontare una varietà di problemi di apprendimento, come la selezione delle caratteristiche.

1.5.1 Bagging

Il *bagging* è un meta-classificatore che viene ideato da Leo Breiman nel 1994. [5] Il funzionamento alla base consiste nel raccogliere le previsioni effettuate dai weak learner, confrontarle e ottenere un'unica previsione.

La peculiarità del bagging è nel metodo in cui vengono selezionati i training set dei singolo alberi. Per effettuare il campionamento viene utilizzato

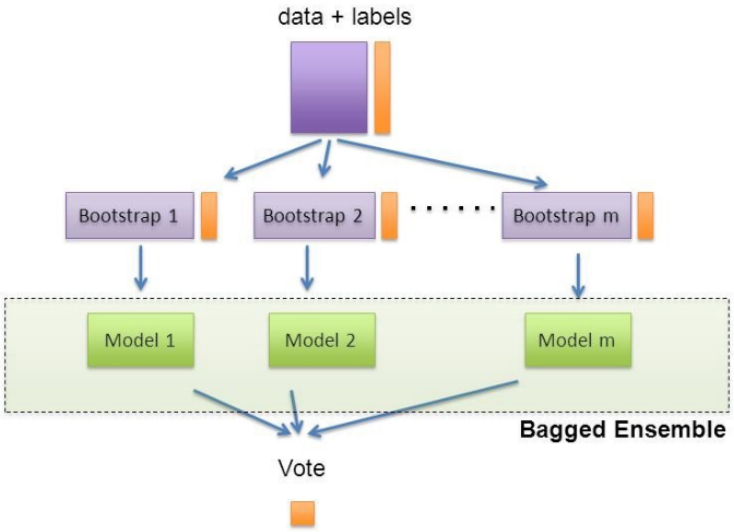

Figura 1.2: Bagging

bootstrap[7] per creare l'insieme di dati su cui allenare i classificatori di base. Per produrre ogni training set sull'insieme di allenamento originale viene effettuato un campionamento casuale con ripetizione, ossia vengono selezionate in modo casuale un certo numero di variabili in modo da aumentare la diversità degli alberi che vengono prodotti.

Un'altra particolarità di questo metodo è che risulta più performante quando i learner di base non sono particolarmente stabili, e sono quindi molto sensibili alle variazioni del training set. Invece, in presenza di learner stabili le previsioni potrebbero peggiorare.

1.5.2 Boosting

Il *boosting* è una tecnica di machine learning ideata nel 1995 da Robert Schapire e che rientra nella categoria dell'apprendimento ensemble.
L'idea alla base del boosting è quella di unire classificatori deboli (weak

[7] *Bootstrap: tecnica statistica di ricampionamento per approssimare la distribuzione campionaria di uno stimatore.*

learner) al fine di ottenere un classificatore che abbia una migliore accuratezza rispetto ai precedenti. Il funzionamento è simile a quello di un algoritmo per costruire alberi decisionali, ma in questo caso l'elaborazione si ripete per più iterazioni, creando in modo adattivo la successione di classificatori deboli. Proprio questa caratteristica distingue il boosting da altri approcci ensemble.

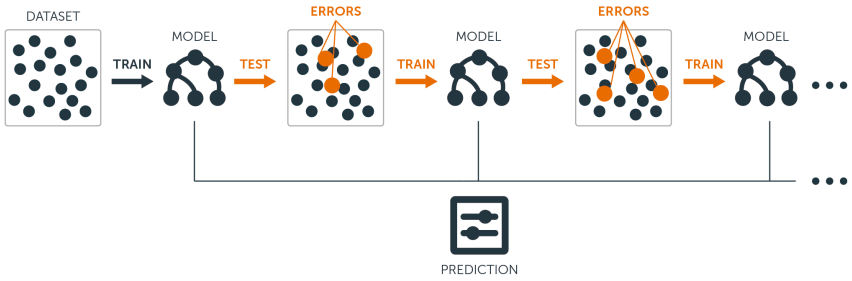

Figura 1.3: Boosting

Il dataset che viene fornito in input verrà chiamato training set, ed è composto da un certo numero di oggetti i quali hanno tutti lo stesso peso, in termini di importanza. A partire da questi elementi viene costruito il primo albero decisionale, che corrisponde alla prima ipotesi di previsione. Successivamente, l'algoritmo verifica l'esattezza dei risultati ottenuti e aumenta l'importanza dei risultati sbagliati. Il training set non sarà più uguale a quello iniziale, perchè gli elementi avranno pesi diversi.

In seguito, l'algoritmo costruisce un secondo albero decisionale prendendo in input il set di dati pesato ottenuto precedentemente. A questo punto, i risultati ottenuti verranno nuovamente verificati e, come in precedenza, verrà attribuito un peso maggiore ai risultati sbagliati. Il ciclo si ripeterà n volte, costruendo n alberi decisionali diversi e migliorando la previsione iterativamente.

Con il boosting ci si concentra maggiormente sull'insieme di dati che non si riesce a classificare in modo corretto. Questo significa che il numero di alberi diventa un parametro molto importante, soprattutto per il trade-off

che si crea fra varianza e distorsione, perchè un numero troppo elevato può portare a overfitting.

Una delle implementazioni più diffuse del boosting è Adaboost[8], in cui l'output del meta-classificatore è dato dalla somma pesata delle predizioni dei singoli modelli e in questo senso viene definito adattivo.

1.5.3 Random Forest

Il *random forest* è una metodologia ensemble avanzata da Tin Kam Ho nel 1995, e recentemente estesa da Leo Breiman.
L'originale algoritmo di Ho prevede che, a partire dal set di dati di allenamento, vengano costruiti un elevato numero di alberi decisionali ognuno dei quali genererà una previsione. La previsione che apparirà più volte fra gli alberi decisionali, verrà considerata come output finale.

Nella versione di Breiman, random forest viene esteso con bagging, utilizzando bootstrap come metodo di campionamento e conferendo così maggior robustezza e accuratezza al modello.

1.5.4 Stacking

Lo *stacking* è una metodologia ensemble proposta da Wolpert nel 1992, che si differenzia dai metodi visti precedentemente introducendo un ulteriore classificatore.
Mentre le tecniche viste si basano sulla media per combinare i modelli di apprendimento, lo stacking invece sostituisce la media con un ulteriore modello di apprendimento. L'idea è quella di valutare l'affidabilità degli output ed è utilizzato per combinare alberi prodotti con algoritmi diversi.

Con lo stacking i learners di base stimeranno il primo livello di output basandosi sul set di dati originali, successivamente la previsione ottenuta viene usata come input per il classificatore di secondo livello.

[8] Adaboost: acronimo di Adaptvie Boosting.

Questo modello è utilizzato per ottenere maggiore precisione nella generalizzazione.

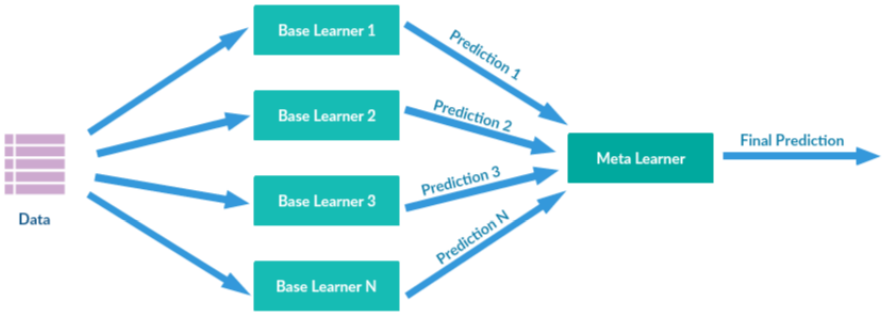

Figura 1.4: Stacking

1.6 Demand forecasting

A livello aziendale, un importante applicazione del data mining si ha nel demand forecasting.
Con *demand forecasting* si intende il processo di previsione della domanda futura di un prodotto o servizio offerto da un'organizzazione, utilizzando i dati storici delle vendite, al fine di ottimizzare le decisioni aziendali.
Le organizzazioni si trovano spesso ad affrontare rischi interni ed esterni a essa, di conseguenza la maggior parte delle decisioni aziendali vengono prese in condizioni di incertezza e rischio. Il demand forecasting serve a ridurre questi i rischi, fornendo una stima di previsione della domanda di un prodotto, determinando in anticipo il risultato più probabile.

I motivi che spingono le organizzazioni ad attuare un processo di demand forecasting possono essere molteplici, come la politica dei prezzi, la politica di produzione, la pianificazione finanziaria delle vendite e del marketing, la gestione delle scorte, l'ingresso in un nuovo mercato o la formulazione di startegie. In base ai requisiti specifici di un'azienda, è possibile sviluppare il modello di previsione della domanda più appropriato.

Il processo di demand forecasting si basa sull'analisi della domanda passata del prodotto preso in considerazione, nelle attuali condizioni del mercato. Questo processo può essere utile solamente se vengono effettuati tutti i passaggi:

1. Definizione dello scopo della previsione che si vuole effettuare;
2. Scelta della prospettiva temporale per cui effettuare la previsione, considerando che potrebbero cambiare i fattori che influenzano la domanda;
3. Scelta del metodo di previsione della domanda, che varia da un'organizzazione all'altra in base all'obiettivo, al periodo di tempo e ai requisiti;
4. Raccolta dei dati;
5. Stima e interpretazione dei risultati

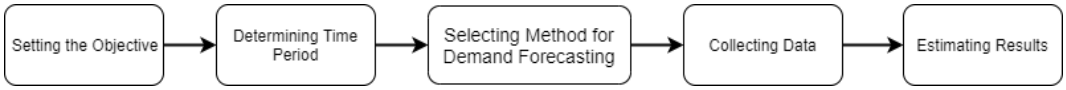

Figura 1.5: Processo di Demand Forecasting

In generale si individuano tre orizzonti temporali che organizzano i metodi di previsione in base al periodo di riferimento: lungo termine, medio termine e breve termine.
Le previsioni a lungo termine si riferiscono a un periodo di tempo che va da 1 a 5 anni, sono più generiche e anche meno affidabili. Le previsioni a medio termine considerano un arco temporale che va da 2 mesi a 1 anno. Infine, le previoni a breve termine sono quelle che arrivano ad un periodo massimo di 2 mesi e sono solitamente le più precise e attendibili.

Il processo di previsione della domanda può essere influenzato da numerosi fattori, come:

- Tipo di bene, un bene nuovo e un bene già esistente sul mercato influenzeranno la previsione in modo diverso;
- Concorrenza del mercato;
- Prezzo del bene;
- Livello di tecnologia;

Altri fattori che influenzano la previsione della domanda, sono quelli legati al periodo, al livello (livello macro, livello industriale, livello aziendale) e alla natura (specifica o generale) della previsione.

Fra le tecniche messe a disposione dal data mining, non esiste un metodo con più probabilità di successo di altri. Questo avviene soprattutto perchè ci sono molte variabili da tenere in considerazione, relative all'azienda, al mercato e al periodo. Inoltre, scegliere l'algoritmo giusto non è sufficiente, in quanto occorre anche individuare la giusta configurazione di iperparametri relativi all'algoritmo scelto.

Nell'ambito del demand forecasting sono state condotte molte ricerche per individuare i metodi migliori.
I metodi usati tradizionalmente si basavano su approcci di previsione di serie temporali, limitando l'utilizzo di dati a quelli relativi ai periodi di tempo correlati con la domanda. Oggi invece si fa grande uso dei modelli di data mining e machine learning.
I metodi più applicati sono quelli di deep learning (DL), impiegati anche in molti campi di riconoscimento di immagini. Un'altra metodologia molto usata è quella degli ensemble learning, dove possono essere combinate le previsioni di modelli diversi.

1.6.1 Altri casi

Ci sono molte tecniche di data mining che possono essere applicate per al demand forecasting. Di seguito presenterò brevemente alcuni casi di studio

il cui obiettivo è la previsione della domanda, illustrando anche le relative soluzioni adottate.

1 - Approccio deep learning

La novità introdotta in questo caso è che vengono costruiti nove diversi metodi di serie temporali, un algortimo di supporto di regressione vettoriale (SVR) e un modello di previsione della domanda basato su deep learning[9]. Questi modelli vengono poi combinati insieme, attraverso un metodo che ricorda la strategia boosting ensemble.

Un'altra particolarità di questo lavoro è l'adattamento della strategia di boosting al modello di previsione della domanda. La combinazione di queste tre tecniche ha garantito il miglioramento dell'accuratezza. La scelta finale del sistema proposto, si basa sui migliori algoritmi della settimana che guadagnano più peso. In questo modo, le previsioni risultano più affidabili.
Questa soluzione è stata implementata e testata su dati di vita reale del mercato.

I risultati dell'esperimento indicano che il sistema proposto presenta notevoli miglioramenti dell'accuratezza per il processo di previsione della domanda rispetto ai singoli modelli di previsione. [8]

2 - SkuBrain

Un altro caso interessante che ho scelto di riportare è SkuBrain.
SkuBrain è una piattaforma il cui scopo è quello di ottimizzare l'inventario dell'utente, aiutandolo a decidere la quantità di stock da acquistare. Questo avviene attraverso la previsione della domanda e la pianificazione dell'inventario. [9]

SkuBrain usa oltre 30 diversi algoritmi di previsione. Alcuni di questi sono:

[9] *Deep Learning: tecnica di apprendimento automatico che applica architetture di reti neurali profonde per risolvere vari problemi complessi. Il deep learning viene applicato in campi come riconoscimento di oggetti visivi e riconoscimento vocale.*

- Media, media mobile e media ponderata
- Exponential smoothing, adatto per la previsione di dati senza trend, è come una media mobile ponderata in cui i pesi diminuiscono all'aumentare dei dati;
- ARIMA[10], sono modelli che mirano a descrivere le corellazioni dei dati, tenendo conto della stagionalità, delle medie mobili e della ponderazione;
- Domanda intermittente di Croston, usata in situazioni in cui sono presenti pochi dati.

3 - Domanda intermittente

Nell'ultimo caso riportato viene illustrata una soluzione adatta quando si verifica la presenza di una domanda intermittente.

La domanda intermittente si verifica casualmente, con valori variabili e molti periodi con domanda nulla.
Sono state sviluppate tecniche di previsione della domanda intermittente, le quali tengono conto delle particolari caratteristiche di questo tipo di domanda.

Oltre alle tecniche tradizionali e ai metodi specializzati, il data mining offre un'alternativa migliore alla previsione della domanda intermittente.
In presenza di questo tipo di domanda, può essere una buona soluzione applicare un modello basato su reti neurali artificiali, un algoritmo di supporto di regressione vettoriale (SVR), modelli basati su alberi decisionali e il modello di Croston.

I risultati ottenuti da questo studio hanno dimostrato che l'uso dei metodi di data mining sono maggiormente utili. Fra questi il support vector machine risulta essere il modello più appropriato, poichè è quello che ha riportato l'errore minore. [10]

[10] *ARIMA: acronimo di Autoregressive Integrated Moving Average*

Capitolo 2

Strumenti

In questo capitolo verranno illustrati gli strumenti utilizzati durante lo svolgimento del progetto.
In particolare parlerò di R, Rstudio, e delle principali librerie usate nello sviluppo del progetto di tesi, ossia caret, xgboost, gbm e mlr.

2.1 R

Per dare una definizione di che cos'è R, si riporta la descrizione dall'homepage del progetto:

> "R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS." [11]

Qui R viene definito come un software per l'elaborazione statistica e grafica, funzionante su diverse piattaforme.
Più avanti, all'interno dello stesso sito, è scritto:

> "R is a language and environment for statistical computing and graphics." [12]

R quindi, oltre ad essere un ambiente di sviluppo, è anche un linguaggio. Simile a S[11] per molti aspetti, R viene considerato come una diversa implementazione del primo.

Figura 2.1: R

R mette a disposizione un'ampia varietà di tecniche statistiche e grafiche, implementabili in modo semplice, e lasciando sempre il controllo all'utente. Questo, unito al suo essere open source, lo posiziona fra i 15 linguaggi di programmazione più popolari. La sua popolarità è dovuta anche alla disponibilità di versioni e documentazione, organizzati nel sito "CRAN"[12], che danno la possibilità di estendere le funzionalità di R in base alle esigenze, per esempio esistono moduli che permettono il collegamento con database.

Il progetto R nasce attorno alla metà degli anni '90 (la prima versione è stata rilasciata nel 1993), i primi sviluppatori furono Robert Gentleman e Ross Ihaka, ricercatori dell'università di Auckland. L'obiettivo di questo progetto era di fornire un ambiente statistico gratuito e di qualità.

Oggi R è il software di riferimento per le analisi statistiche in ambito accademico e negli ultimi anni anche in quello aziendale, soprattutto perchè consente di creare istruzioni personalizzate attraverso l'apposita console.
R mette a disposizione tecniche statistiche e grafiche, permettendo quindi il calcolo, la manipolazione dei dati e la visualizzazione. Proprio R Development Core Team lo definisce come una suite integrata di funzionalità [12], perchè classificarlo come ambiente è restrittivo.

[11] *S: è un linguaggio object oriented non opensource, sviluppato da John Chambers e alcuni colleghi, nei Bell Laboratories, verso la fine degli anni '70.*

[12] *CRAN: acronimo di Comprehensive R Archive Network.*

R non mette a disposizione una vera interfaccia grafica, infatti è un ambiente basato su un'interfaccia utente a carattere (CUI), ma per una maggiore semplicità d'uso può essere integrato con una delle numerose interfacce grafiche disponibili. L'assenza di un'interfaccia grafica può essere visto come un punto di debolezza, infatti le istruzioni e le funzioni devono essere immesse dalla linea di comando.
Un altro punto di debolezza di R, è la possibilità che alcune funzioni statistiche non siano implementate. Essendo un ambiente open source, è possibile dare una soluzione al problema scrivendo il codice mancante, anche se questo richiede una maggiore padronanza del linguaggio.

Parlando invece dei vantaggi che R offre, si può dire che è una suite all'avanguardia, infatti solitamente si possono trovare librerie che implementano le ultime scoperte statistiche. Un altro punto di forza è che permette di effettuare calcoli su vettori, matrici e operazioni più articolate, gestire i modelli statistici dai più semplici ai più complessi e realizzare grafici di ogni tipo. Inoltre è anche un ambiente molto versatile, in quanto dà la possibilità di creare strumenti personali e di usufruire di quelli già pronti (attraverso dei parametri di default). Infine, R mette a disposizione una funzione di "help" utile per consultare le informazioni relative ai comandi, conferendo così una maggior completezza al software.

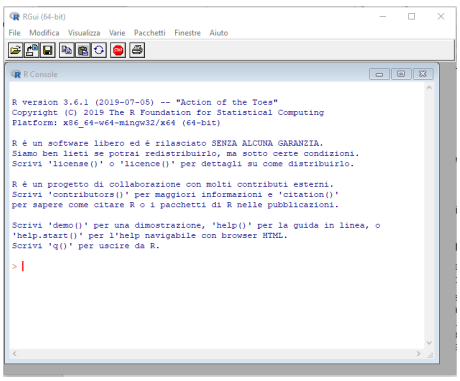

Figura 2.2: Interfaccia utente a caretteri di R

R può essere integrato con RStudio, un ambiente di sviluppo integrato gratuito e scaricabile da internet[13] con interfaccia grafica (GUI).
RStudio include una console che supporta l'evidenziazione della sintassi, il completamento del codice e il rientro intelligente. Permette anche di eseguire il codice direttamente dall'editor, prevede la gestione di più directory di lavoro ed è presente un debugger per diagnosticare e correggere gli errori.
RStudio mette a disposizione anche molte altre funzionalità consentendo a chiunque di poter utilizzare R in modo semplice.

Figura 2.3: RStudio

2.2 Caret

Una delle principali librerie open source utilizzate nel progetto è caret.
Caret[14] è un pacchetto per R che mette a disposizione delle funzioni per semplificare il processo di creazione dei modelli predittivi, per problemi di classificazione e regressione.

Caret utilizza altre 30 librerie che non carica all'avvio ma al momento di effettivo utilizzo, presupponendo però che siano installati e in caso contrario viene richiesta l'installazione.
In particolare, caret contiene funzioni per la suddivisione dei dati, il pre-processing, la selezione delle variabili, l'ottimizzazione del modello attraverso il ricampionamento, la stima dell'importanza delle variabili e altre funzionalità. [14]
Di seguito introdurrò di alcune funzioni di caret utilizzate in questa tesi.

[13] *https://rstudio.com/products/rstudio/*
[14] *Caret: acronimo di Classification And REgression Training.*

Una delle funzioni principali che caret mette a disposizione è train, la quale può essere utilizzata per valutare la regolazione dei parametri di un modello in base alle prestazioni, per scegliere il modello ottimale e per stimare le prestazioni su un set di allenamento. Ogni fase di questo processo può essere personalizzata cambiando i parametri della funzione che non sono tutti obbligatori e, se non vengono impostati dall'utente, R considera i valori di default.
Alcuni parametri che permettono di personalizzare in modo significativo il modello, modificandoli in base alle necessità, sono:

- method, per indicare quale modello di classificazione o regressione utilizzare
- trainControl, per modificare il metodo di ricampionamento
- metric, per impostare la metrica di valutazione

Oltre a questi, possono essere modificati anche altri parametri che consentono di personalizzare il processo.

Altra funzione molto utile è createDataPartition, che permette di dividere un unico set di dati in due parti in base alla percentuale impostata, creando così un trainSet e un testSet. La partizione avverrà in modo casuale.

Un'ulteriore funzione utilizzata spesso è dummyVars, che permette di ottenere un set di variabili fittizie. L'output di questa funzione può essere utilizzato per esempio come argomento di predict. Questa funzione non fa parte del pacchetto caret, ma è comunque utilizzata spesso perchè consente di ottenere delle predizioni di valori, partendo da un set di dati. L'unico argomento obbligatorio della funzione è data, in cui indicare il dataset su cui generare le variabili.

2.3 Gbm

Per l'applicazione di algoritmi di boosting sono disponibili molte librerie open source per R. Fra queste le più utilizzate sono gbm e xgboost. La prima,

gbm[15], è un'implementazione dell'algoritmo AdaBoost di Freud e Schapire, e del modello Gradient Boosting Machine di Friedman, mentre la seconda è in realtà una particolare implementazione di gbm.
Gbm, come tutti gli algoritmi di Boosting, crea dei modelli in sequenza e attribuisce un peso alle istanze, dando un peso maggiore alle previsioni sbagliate. In questo modo viene migliorata l'accuratezza del modello, riducendo l'errore iterativamente.

La principale funzione messa a disposizioni in questa libreria è gbm, che applica l'algoritmo al set di dati specificato. Alcuni parametri che è possibile indicare sono:

- distribution, dove si può indicare il nome della distribuzione da utilizzare, di default Gbm proverà a utilizzarne una che ritiene adatta;
- n.trees, il numero totale di alberi che equivale al numero di iterazioni;
- shrinkage, parametro di restringimento che viene applicato a ciascun albero, detto anche tasso di apprendimento (learning rate). Per un tasso di apprendimento minore è necessario un maggior numero di alberi;
- interaction.depth, la profondità massima di ciascun albero;
- n.minobsinnode ossia il numero minimo di osservazioni nei nodi finali degli alberi;
- cv.folds, indica il numero di validazioni incrociate da eseguire e calcola una stima dell'errore di generalizzazione.

Un'altra funzione utilizzata è gbm.perf, che permette di stimare il numero ottimale di alberi indicando anche il metodo per effettuare la stima. Il risultato è quindi il numero di iterazioni che minimizza l'errore. Questa funzione si rivela utile soprattutto in fase di ottimizzazione dei parametri.

[15] *Gbm: acronimo di Generalized Boosted Regression Models.*

2.4 Xgboost

Xgboost[16] è una libreria open source che permette di costruire un modello e fare previsioni, implementando il framework gradient boosting di Friedman (2000 e 2001).
Questa libreria nasce come estensione di gbm, proprio per potenziarla, in modo da rendere le previsioni più accurate e con tempi minori. All'inizio era semplicemente un'applicazione da terminale, successivamente con l'aumentare del successo sono stati creati i pacchetti per R, Phyton, Java e altri linguaggi. Ora ha integrazioni sia con scikit-learn (Python) che con caret.

La principale differenza fra questa libreria e la precedente è che xgboost utilizza una formalizzazione del modello più regolare, per cercare di controllare l'overfitting.

Le funzioni utilizzate di questa libreria sono xgb.train e xgb.cv.
La prima permette di effettuare l'addestramento sull'insieme di dati applicando xgboost. In generale, questa libreria può essere utilizza sia con modelli lineari sia con modelli ad albero.
Nella funzione xgb.train è possibile indicare il tipo di booster da usare: gblinear o gbtree. Altri parametri di configurazione del modello sono eta, che indica il tasso di apprendimento, max_depth cioè la massima profondità di ogni albero, subsample che consente di indicare il numero di sottocampioni per costruire gli alberi e rendere il calcolo più breve, nround per specificare il massimo numero di iterazioni da eseguire.
Un parametro molto utile è early_stopping_rounds, dove è possibile specificare il numero di iterazioni dopo il quale interrompere l'allenamento, se le prestazioni non migliorano.
Sono presenti anche altri parametri di configurazione che variano in base al booster scelto.

L'altra funzione, xgb.cv, viene utilizzata per eseguire una cross-validation sul modello, restituendo il numero di iterazioni che permette di ottenere l'er-

[16] *Xgboost: acronimo di eXtreme Gradient Boosting.*

rore minore. Oltre al dataset, si possono indicare il massimo numero di iterazioni da eseguire, il numero di sottocampioni in cui dividere l'insieme e la metrica di valutazione da utilizzare.
Il dataset viene diviso in modo casuale in sottocampioni della stessa dimensione. Ogni volta che la procedura viene ripetuta, un campione viene conservato per la convalida, mentre gli altri vengono utilizzati per l'addestramento. Tutte le osservazioni sono utilizzate sia per la validazione che per l'addestramento. Dopo aver effettuato tutte le iterazioni viene restituito il risultato.

2.5 Mlr

Mlr[17] è una libreria che mette a disposizione molte tecniche di regressione e classificazione, metodi per l'ottimizzazione dei parametri e per il ricampionamento. Prevede inoltre estensioni dei learners di base, con operazioni comuni nel machine learning, e la possibilità di eseguire operazioni in parallelo.

Questa libreria è stata utilizzata per effettuare in modo preciso e rapido l'ottimizzazione dei parametri nei modelli di boosting. Le funzioni implementate sono state utili quindi per costruire il classificatore, il modello, per eseguire il tuning e per individuare la combinazione che minimizza l'errore.
Alcune delle funzioni appartenenti a questa libreria e usate nello svolgimento del progetto sono: makeClassifTask, makeLearner, tuneParams.
Esempio:

```
xgb_learner <- makeLearner("classif.xgboost",
    predict.type = "response")

params <- makeParamSet(
    makeDiscreteParam("booster", values = "gbtree"),
    makeIntegerParam("max_depth", lower = 2, upper = 15),
```

[17] *Mlr: acronimo di Machine Learning in R.*

```
        makeNumericParam("eta", lower = 0.01, upper = 0.08)
    )

    myTune <- tuneParams(
        learner = xgb_learner,
        task = trainSetDummy,
        par.set = params,
        show.info = FALSE
    )
```

Nel precedente esempio viene utilizzata la funzione makeLearner per creare il modello xgboost.
Successivamente attraverso makeParamSet vengono modificati i paramentri di default, impostando quelli relativi al modello che desiderato.
Infine tuneParams esegue l'ottimizzazione individuando la combinazione che consente di ottenere risultati migliori.

Capitolo 3

Progettazione

Il seguente capitolo ha la funzione di illustrare e spiegare i passaggi che sono stati eseguiti nel progetto.
L'obiettivo del progetto è quello di ottenere un modello che sia capece di valutare i dati a disposizione e sulla base di questi effettuare previsioni in modo accurato.
Per il raggiungimento di tale scopo sono stati proposti e valutati tre algoritmi di machine learning basati sul boosting.

3.1 Il dataset

I dataset a disposizione sono due, riferiti a mercati differenti: mercato italiano e mercato cinese. La loro struttura è profondamente diversa in quanto il primo, si riferisce a un mercato basato sulle piccole boutique e sulla vendita al dettaglio, mentre il mercato cinese è basato su una vendita all'ingrosso. Di conseguenza, in quest'ultimo dataset sono presenti molti outlier[18] che tenderanno quindi a influenzare i risultati. Il dataset italiano invece, tolto qualche outlier, risulta essere molto più costante.

[18] *Outlier: termine usato in statistica per definire, in un insieme di osservazioni, un valore anomalo chiaramente distante dalle altre osservazioni disponibili.*

I modelli di previsione e le analisi effettuate sono state eseguite su entrambi i dataset.

In generale, ogni riga del dataset rappresenta un occhiale, mentre ogni colonna rappresenta una caratteristica relativa all'oggetto.
La variabile su cui si vuole effettuare la previsione è la quatità netta di ordinato (indicato con "Net.Order... Quantity"), relativa al singolo oggetto all'interno del mercato preso in considerazione.

Nei due dataset, le caratteristiche sono le stesse, **139 variabili**, mentre cambia il numero di osservazioni. Il dataset relativo al mercato italiano è composto da **87772 record**, l'altro dataset è composto invece da **51436 record**.
Nelle successive fasi del processo, entrambi gli insiemi di dati verranno ridotti.

3.2 Pre-processing

Per poter effettuare delle analisi e delle previsioni attendibili sui dati, è necessaria una fase di data cleaning, ossia di pulizia dei dati.
Pulire il dataset è utile per rimuovere tutti i valori che possono influenzare in modo sbagliato i risultati, come gli outlier o variabili fittizie. Nel pre-processare i dati, sono stati rimossi:

- Rumori, quindi errori e outliers
- Dati mancanti, presenti all'interno del dataset come "NA", "Not Defined" oppure "Missing"
- Dati inconsistenti, come codici non rilevanti ai fini dell'analisi

In seguito è spesso utile effettuare una selezione delle variabili che hanno maggiore influenza sui risultati. Questo ridurrà ulteriormente la dimensione del dataset, ma renderà le previsioni più attendibili.
Esistono molti algoritmi che possono essere di aiuto in questa fase, in questo progetto di tesi sono stati applicati Anova e Boruta.

3.2.1 Anova

Il primo algoritmo applicato è ANOVA[19], con cui è stata effettuata l'analisi della varianza. Questa analisi permette di testare le differenze tra le medie campionarie, per far questo si prendono in considerazione le rispettive varianze. La varianza fornisce la misura della variabilità dei valori assunti dalla variabile stessa, misura quindi quanto questi valori si discostano quadraticamente dalla media aritmetica o dal valore atteso.

3.2.2 Boruta

Dopo l'analisi della varianza, è stato applicato al dataset un algoritmo che prende il nome di Boruta. Questo algoritmo è utile per la selezione delle variabili rilevanti ed usa random forest come metodo di classificazione.
In primo luogo viene duplicato il dataset e vengono mischiati i valori delle copie duplicate per rimuovere le correlazioni con la variabile dipendente. Successivamente viene eseguito il classificatore random forest, e viene calcolata l'importanza di ogni variabile. Infine, confronta l'importanza di ogni variabile con la massima importanza delle variabili duplicate: se risulta minore, la variabile non viene considerata rilevante, in caso contrario, sì.
Dopo un numero finito di iterazioni si ottiene l'insieme di variabili più significative all'interno del dataset, ossia le variabili indipendenti che hanno maggiore influenza sulla variabile dipendente.

3.3 Training e Test

Il training rappresenta la parte centrale del processo, in quanto vengono ricercati e generalizzati i pattern partendo dai dati che descrivono l'evento che si sta analizzando.
Per ottenere un set di dati su cui effettuare l'addestramento dei modelli, il dataset iniziale viene diviso in due. Queste due partizioni devono essere

[19] *ANOVA: acronimo di Analysis of Variance.*

estratte casualmente, in modo da ottenere campioni rappresentativi. Inoltre, per ottenere i due insiemi, il dataset non viene diviso in modo uguale, ma è preferibile destinare un maggior numero di record al set di allenamento, così che i modelli possano apprendere quanto più possibile. Un rischio a cui si può andare incontro durante il training è l'overfitting. Esso si verifica quando il modello su cui si sta effettuando l'addestramento, si specializza eccessivamente sull'insieme di dati, adattandosi così a caratteristiche specifiche del trainSet. Di conseguenza aumenteranno le prestazioni sul set di allenamento, ma diminuiranno sul set di test.

Una tecnica che si può adottare per evitare il sovraddattamento è la cross-validation. Con questa tecnica il dataset viene diviso in N sottocampioni, tutti della stessa dimensione. Ad ogni ripetizione, una partizione viene conservata per la convalida, mentre le altre vengono utilizzati per l'addestramento. Dopo aver effettuato tutte le iterazioni viene restituito il risultato, ossia il valore dell'errore medio dei campioni di test.

Infine, per valutare l'accuratezza dei modelli, sono stati calcolati l'errore quadratico medio della radice, o *RMSE*, e il coefficiente di determinazione, o R^2, in quanto erano necessarie delle misure assolute e quindi valide per i tre differenti algoritmi.

L'errore quadratico medio, o *MSE*, indica la differenza quadratica media fra i valori dei dati osservati ed i valori dei dati stimati. La sua radice quadrata fornisce un altro indice statistico: l'errore quadratico medio della radice, o *RMSE*. Esso dà la misura della diffusione dei residui, cioè la distanza dei punti dati dalla linea di regressione. In altre parole, dice quanto sono concentrati i dati attorno alla retta di regressione.

$$MSE = E[(\widehat{\theta} - \theta)^2]$$

$$RMSE = \sqrt{MSE} = \sqrt{E[(\widehat{\theta} - \theta)^2]}$$

Il coefficiente di correlazione, R^2, indica una proporzione tra la variabilità dei dati e la correttezza del modello statistico utilizzato. Permette quindi di capire in che misura le variabili dipendenti spiegano la variabilità di quella

dipendente all'interno dei dati.
R^2 è un valore compreso fra 0 e 1, se è pari a 1 esiste una perfetta correlazione fra il fenomeno analizzato e il modello statitisco.

$$R^2 = 1 - \sum \frac{(y_i - \widehat{y_i})^2}{(y_i - \overline{y})^2}$$

dove: y_i sono le osservazioni, $\widehat{y_i}$ sono i valori stimati e $\overline{y}$ è la media dei valori.

Come detto precedentemente, i modelli di boosting applicati sono: Generalize Boosted Regression Model, Extreme Gradient Boosting e Gradient Boosting Linear. Questi tre modelli sono fra loro molto simili, infatti il primo è un'implementazione dell'algoritmo AdaBoost e del modello Gradient Boosting Machine, mentre gli altri due sono delle estensioni di questo.

Nella fase finale del processo i modelli vengono testati, applicando i parametri ritenuti ottimali.

Capitolo 4

Implementazione

In qusto capitolo verrà mostrato il lavoro svolto in R, facendo riferimento al processo spiegato nel precedente capitolo.
Sono stati analizzati diversi algoritmi di boosting. L'obiettivo è quello individuare un modello che, valutando i dati a disposizione, consenta di ottenere previsioni accurate.

Dapprima verrà brevemente illustrata la fase di preparazione dei dati.
Si passerà quindi al train, dove verrà eseguita l'ottimizzazione dei parametri. Il tuning avverrà seguendo due approcci: uno brute force e uno guidato. Infine verrà eseguito il test dei modelli.

4.1 Pre-processing

I dataset su cui si è svolto il lavoro sono relativi al mercato italiano, con **87772 record**, e al mercato cinese, composto da **51436 record**, mentre il numero iniziale di caratteristiche è **139** in entrambi gli insiemi.

Nella fase di pre-processing, oltre ad una pulizia dei dataset, si è verificata la correlazione fra le variabili e sono state selezionate le caratteristiche aventi una maggiore influenza sull'ordinato.
Considerando solamente le variabili più influenti (**91 caratteristiche** selezionate), sono stati ottenuti due nuovi dataset ridotti: quello relativo al mer-

cato italiano è composto ora da **12715 osservazioni**, mentre quello cinese ha **9119 osservazioni**.

Infine, sono stati applicati dei filtri specifici, alcuni richiesti dal cliente, mentre altri scelti i in base a delle considerazioni che è necessario fare trattandosi di ambito fashion.
I filtri applicati sono:

- Anno 2017 e 2018
- Collezione 'Autunno-Inverno'
- Mesi di ordinato Agosto, Settembre, Ottobre e Novembre
- Material Group Sunglasses

I due insiemi finali di dati sono composti da **6345 record** per il mercato italiano e **6481 record** per il mercato cinese.

Prima di passare al training, i dataset vengono divisi in modo da ottenere due insiemi destinati uno all'addestramento e uno al test del modello.

```
AllSet <- data[, selectedAttribute]

trainIndex <-
    createDataPartition(
        AllSet$Net.Order...Quantity,
        p = .6,
        list = FALSE,
        times = 1
    )

trainSet <- AllSet[trainIndex,]
testSet <- AllSet[-trainIndex,]
```

La suddivisione avviene in modo è casuale, in questo caso è stata effettuata attraverso la funzione createDataPartition, destinando il 60% dei record al trainSet, e il restante 40% al testSet.

4.2 Training

Nell'ambito di un progetto di demand forecasting possono essere utilizzate diverse tecniche, più o meno valide, in base ai dati che si hanno a disposizione.
In questo progetto di tesi si è scelto di applicare gli algoritmi di boosting, che fanno parte dei metodi di apprendimento ensemble, i quali vengono utilizzati spesso per affrontare una varietà di problemi di apprendimento. Per il demand forecasting gli algoritmi di boosting possono rivelarsi una buona soluzione, perchè fanno uso di alberi con poche divisioni, non molto profondi e altamente interpretabili.

Parametri come il numero di alberi o iterazioni, la velocità con cui l'apprendimento del gradiente apprende e la profondità dell'albero possono essere combinati in modo ottimale per ottenere il modello predittivo migliore. Avere un numero elevato di alberi potrebbe comportare un overfitting, per questo è necessario scegliere con attenzione i criteri di arresto per l'aumento.
Attraverso l'ottimizzazione dei parametri è possibile ottenere la combinazione che minimizza la misura di errore scelta per il modello. In questo progetto l'ottimizzazione è avvenuta seguendo due approcci differenti e utilizzando librerie distinte.

4.2.1 Prima modalità di ottimizzazione

Prima di effettuare il tuning[20], in base al modello utilizzato, è stato necessario convertire i dati in modo che potessero essere processati.

La struttura del codice è simile per il tre modelli:

[20] *Tuning: dall'inglese significa messa a punto. Nel contesto è utilizzato come sinonimo di ottimizzazione.*

1. One hot encoding, ossia la trasformazione delle variabili categoriche in una forma che possa essere fornita agli algoritmi
2. Creazione della matrice con i parametri
3. Train di tutte le combinazioni presenti nella matrice
4. Test del modello

I listati che seguono riportano le principali operazioni svolte nei tre algoritmi.

Generalized Boosted Regression Models

```
# One Hot Encoding
dmy <- dummyVars(" ~ .", data = trainSet)
trsf <- data.frame(predict(dmy, newdata = trainSet))

# Formula
groupLogit <- paste(colnames(trsf[2]), colnames(trsf[3]), sep = "+")

for (z in 4:length(trsf)) {
    groupLogit <- paste(groupLogit, colnames(trsf[z]), sep = "+")
}
form <- reformulate(groupLogit,
                    response = paste(colnames(trainSet[1])))
```

Nello script riportato viene effettuato il one hot encoding, successivamente viene creata la formula[21] che descrive il modello.
In seguito sono stati impostati i parametri da ottimizzare nel training. In questo caso viene generata una griglia formata da tutte combinazioni possibili con i parametri impostati. In questo modo i parametri potranno essere solo dei numeri finiti, questo può andare a ridurre la precisione del modello.

[21] *Per un chiarimento sui parametri relativi alle funzioni, si rimanda alla lettura del Capitolo 2.3*

```
# Set params
df_par <- expand.grid(
    shrinkage = seq(0.01, 0.1, by = 0.01),
    n.trees = seq (50, 150, by = 10),
    interaction.depth = seq(2, 10, by = 1)
    )
RMSE <- c()
R_2 <- c()
RSE <- c()

# Training
for (i in 1:nrow(df_par)) {
    gbm1 <- gbm(
        formula = form,
        distribution = "gaussian",
        data = trsf,
        n.trees = df_par$n.trees[i],
        shrinkage = df_par$shrinkage[i],
        interaction.depth = df_par$interaction.depth[i],
        cv.folds = 5,
        verbose = FALSE,
        n.cores = NULL
    )

    # Train prediction
    trainSet$y_hat <- predict(gbm1, trsf)
    RMSE[i] <- RMSE(trainSet$Net.Order...Quantity, trainSet$y_hat)
    RSE[i] <- rrse(trainSet$Net.Order...Quantity, trainSet$y_hat) ^ 2
    R_2[i] <- 1 - RSE[i]
}
```

L'addestramento viene effettuato su ogni combianzione presente nella gri-

glia, quindi si calcola l'errore medio degli scarti prodotto e i risultati vengono salvati in un file. Successivamente verrà scelta la combinazione che ha riportato il minor RMSE, e si eseguirà il test.

```
# Import CSV into R
data_file <- paste("models/BF/", sales_organization,
    "_gbm_train.csv", sep = "")
trainGBM <- read.csv(file = data_file, header=TRUE, sep=",")
trainGBM <- trainGBM[trainGBM$RMSE == min(trainGBM$RMSE), ]

# Test
dmyTest <- dummyVars(" ~ .", data = testSet)
trsfTest <- data.frame(predict(dmy, newdata = testSet))
testSet$y_hat <- predict(gbm1, trsfTest)
RMSE <- RMSE(testSet$Net.Order...Quantity, testSet$y_hat)
RSE <- rrse(testSet$Net.Order...Quantity, testSet$y_hat) ^ 2
R_2 <- 1 - RSE
```

Extreme Gradient Boosting

Vedremo ora lo svolgimento del processo applicando il modello Extreme Gradient Boosting[22], dopo aver effettuato il one hot encoding.

```
# Set params
df_par <- expand.grid(
    eta = seq(0.01, 0.1, by = 0.01),
    max_depth = seq(2, 10, by = 1),
    nrounds = seq (50, 170, by = 10)
    )
RMSE <- c()
R_2 <- c()
RSE <- c()
```

[22] *La libreria utilizzata è xgboost, spiegata nel Capitolo 2.4*

```
for (i in 1:nrow(df_par)) {
    params <- list(
            eta = df_par$eta[i],
            max_depth = df_par$max_depth[i],
            subsample = 0.5,
            colsample_bytree = 0.5,
            eval_metric = "rmse"
        )

    bst <- xgb.train(
            params = params,
            data = dtrain,
            nrounds = df_par$nrounds[i]
        )

    # RMSE train
    trainSet$y_hat <- predict(bst, sparse_matrix)
    RMSE[i] <- RMSE(trainSet$Net.Order...Quantity, trainSet$y_hat)
    RSE[i] <- rrse(trainSet$Net.Order...Quantity, trainSet$y_hat) ^ 2
    R_2[i] <- 1 - RSE[i]
}
```

Nel listato viene mostrato il one hot encoding necessario per l'utilizzo di xgboost. Successivamente sono stati impostati i valori dei parametri e si è creata la matrice con tutte le possibili combinazioni. Restano valide le criticità sottolineate precedentemente, infatti le prestazioni del modello potrebbero essere limitate impostando valori finiti.
Viene poi eseguito il training su tutte le possibili combinazioni e, dopo aver individuato quella che restituisce un errore minore, verrà effettuato il test.

```
testMatr <-
  sparse.model.matrix(Net.Order...Quantity ~ ., data = testSet)[,-1]
```

```
testSet$y_hat <- predict(bst, testMatr)
RMSE <- RMSE(testSet$Net.Order...Quantity, testSet$y_hat)
RSE <- rrse(testSet$Net.Order...Quantity, testSet$y_hat) ^ 2
R_2 <- 1 - RSE
```

Gradient Boosting Linear

L'ultimo modello applicato è una particolare implementazione di quello appena visto. Vengono utilizzati gli stessi parametri del precedente algoritmo, in aggiunta viene impostato un boosting lineare.

```
for (i in 1:nrow(df_par)) {
    params <- list(
            booster = "gblinear",
            eta = df_par$eta[i],
            max_depth = df_par$max_depth[i],
            subsample = 0.5,
            colsample_bytree = 0.5,
            cv.folds = 5
        )
    bst <- xgboost(
            data = sparse_matrix,
            label = output_vector,
            params = params,
            nrounds = df_par$nrounds[i]
        )

    # RMSE train
    trainSet$y_hat <- predict(bst, sparse_matrix)
    RMSE <- RMSE(trainSet$Net.Order...Quantity, trainSet$y_hat)
    RSE <- rrse(trainSet$Net.Order...Quantity, trainSet$y_hat) ^ 2
    R_2 <- 1 - RSE
}
```

Per brevità non è riportato il codice relativo al test, in quanto simile al precedente modello.

4.2.2 Seconda modalità di ottimizzazione

Il secondo modo in cui è stata effettuata l'ottimizzazione dei parametri è utilizzando la libreria mlr[23].
Anche in questo caso si è resa necessaria una conversione dei dati in modo che potessero essere processati.

La struttura del codice, simile nei tre modelli, è:

1. One hot encoding
2. Creazione del learner
3. Impostazione del valore massimo e minimo assunto da ogni parametro
4. Tuning dei parametri
5. Train della combinazione migliore
6. Test del modello

Generalized Boosted Regression Models

Il codice riportato mostrarà la costruzione del modello Generalized Boosted Regression Models. L'ottimizzazione dei parametri avviene attraverso l'utilizzo della funzione tuneParams.

```
# Create tasks
trainTask <-
    makeClassifTask(data = trainSet, target = "Net.Order...Quantity")

# One Hot Encoding
trainSetDummy <- createDummyFeatures(trainTask)
```

[23] *Si rimanda alla lettura del Capitolo 2.5*

```
# Create learner
gbm_learner <- makeLearner("classif.gbm", predict.type = "response")
gbm_learner$par.vals<- list(n.trees = 200)

# Set params
params <- makeParamSet(
  makeIntegerParam("interaction.depth", lower = 2, upper = 15),
  makeNumericParam("shrinkage", lower = 0.01, upper = 0.08)
)
ctrl <- makeTuneControlMBO()
rdesc <- makeResampleDesc("CV", iters = 5)

myTune <- tuneParams(
  gbm_learner,
  task = trainSetDummy,
  resampling = rdesc,
  measures = acc,
  par.set = params,
  control = ctrl,
  show.info = TRUE)
```

Dopo aver eseguito l'ottimizzazione, verrà costruito un nuovo modello che utilizza gli iperparametri migliori. Con questo modello verrà eseguito il train e successivamente il test.

```
# Create new model using hyperparameters
gbm_tuned_learner <-
    setHyperPars(learner = gbm_learner, par.vals = myTune$x)

# Train model using tuned hyperparametres
gbm_model <- train(gbm_tuned_learner, trainSetDummy)
```

La combinazione di iperparametri viene salvato in un file esterno che verrà poi recuperato dalla funzione che esegue il test. Prima di effettuare il test, è stata eseguita una cross-validation, al fine di individuare il numero di alberi a cui corrisponde la migliore iterazione, con il minimo errore di validazione. Questo metodo permette di evitare che il modello si adatti eccessivamente al set di allenamento.

```
ntrees_opt <- gbm.perf(gbm_model, method="cv")
```

Extreme Gradient Boosting

Il seguente script mostra la costruzione del modello Extreme Gradient Boosting. Per l'ottimizzazione dei parametri viene utilizzata la funzione tune-Params. Gli iperparametri migliori saranno poi impiegati per la costruzione del nuovo modello predittivo e per il successivo train dello stesso.

```
# Create tasks
trainTask <-
  makeClassifTask(data = trainSet, target = "Net.Order...Quantity")

# One Hot Encoding
trainSetDummy <- createDummyFeatures(trainTask)

# Create learner
xgb_learner <- makeLearner("classif.xgboost", predict.type = "response")
xgb_learner$par.vals <- list(nrounds = 200)

# Create model
xgb_model <- xgb.train(xgb_learner, nrounds = 200, data = dtrain)
result <- predict(xgb_model, trainMatr)

# Set param
params <- makeParamSet(
```

```
  makeDiscreteParam("booster", values = "gbtree"),
  makeIntegerParam("max_depth", lower = 2, upper = 15),
  makeNumericParam("eta", lower = 0.01, upper = 0.08)
)
rdesc <- makeResampleDesc("CV", iters = 5)
cv_inst <- makeResampleInstance(rdesc, task = trainSetDummy)
control <- makeTuneControlRandom(maxit = 5)

myTune <- tuneParams(
  learner = xgb_learner,
  task = trainSetDummy,
  resampling = rdesc,
  measures = acc,
  par.set = params,
  control = control,
  show.info = FALSE)

# Create new model using hyperparameters
xgb_tuned_learner <-
  setHyperPars(learner = xgb_learner, par.vals = myTune$x)

# Train model using tuned hyperparametres
xgb_model <- xgb.train(xgb_tuned_learner, nrounds = 20, data = dtrain)
```

Gli iperparametri con cui è stato ottenuto il risultato migliore vengono salvati in un file esterno, per essere recuperati dalla funzione che esegue il test.

Prima di applicare il modello al set destinato al test, viene eseguita una cross-validation che consente di individure il numero di iterazioni con cui si è ottenuto il minor errore di validazione. Successivamente, verrà effettuato un nuovo train e poi il test.

```
cv <- xgb.cv(data = dtrain,
```

```
            eta = trainxgb$eta,
            max_depth = trainxgb$max_depth,
            nrounds = 200,
            nthread = 2,
            nfold = 5,
            metrics = "rmse",
            early_stopping_rounds = 10)
best_iteration <- cv$best_ntreelimit

bst <- xgb.train(data = dtrain,
                 eta = trainxgb$eta,
                 max.depth = trainxgb$max_depth,
                 nrounds = best_iteration,
                 nthread = 2,
                 eval.metric = "rmse")

# Predict values
resultest <- predict(bst, testMatr)
RMSE <- RMSE(labelTest, resultest)
RSE <-  rrse(labelTest, resultest) ^ 2
R_2 <- 1 - RSE
```

Gradient Boosting Linear

Per l'implementazione del terzo modello occorre impostare il booster in fase di creazione del modello. Il tuning, il train e il test rimangono uguali al precedente algoritmo, implementato negli script visti.

```
params <- makeParamSet(
    makeDiscreteParam("booster", values = "gblinear"),
    makeIntegerParam("max_depth", lower = 2, upper = 15),
    makeNumericParam("eta", lower = 0.01, upper = 0.08)
)
```

Capitolo 5

Risultati ottenuti

In questo capitolo verranno mostrati i risultati ottenuti con i tre modelli applicati per la previsione dell'ordinato netto.
L'accuratezza degli algoritmi viene confrontata considerando l'errore quadratico medio della radice ($RMSE$) e il coefficiente di determinazione (R^2), precedentemente introdotti.

5.1 I parametri

L'applicazione dei tre modelli all'insieme di dati destinati al test ha permesso di ottenere previsioni differenti.
Nella **Tabella 5.1** e nella **Tabella 5.2** sono riportate le combinazioni di parametri con cui, per ogni modello, si è ottenuta una miglior accuratezza di previsione attraverso l'ottimizzazione manuale. Sono state crete tabelle differenti, in quanto i modelli presentano parametri non comuni.

È possibile notare che per i modelli risultano maggiormente performanti con un tasso di appendimento basso (**eta** e **shrinkage**), combinato con un elevato numero di iterazioni (**nrouds** e **n.trees**) e con una bassa profondità degli alberi costruiti (**max_depth** e **interaction_depth**).

Mercato	Modello	eta	max_depth	nrounds
Italia	xgboost	0.01	2	250
Italia	gblinear	0.01	6	250
Cina	xgboost	0.01	2	220
Cina	gblinear	0.01	6	220

Tabella 5.1: xgboost e gblinear, ottimizzazione manuale

Mercato	Modello	shrinkage	interaction_depth	n.trees
Italia	gbm	0.02	10	150
Cina	gbm	0.03	6	170

Tabella 5.2: gbm, ottimizzazione manuale

Prendendo ora in considerazione i modelli che hanno ottenuto risultati migliori effettuando il tuning con la libreria mlr, i parametri ottimali saranno i seguenti:

Mercato	Modello	eta	max_depth	nrounds
Italia	xgboost	0.033633..	7	174
Italia	gblinear	0.0404807..	6	200
Cina	xgboost	0.0223284..	5	141
Cina	gblinear	0.0223284..	5	174

Tabella 5.3: xgboost e gblinear, ottimizzazione guidata

Come detto sopra, anche in questo caso si ha un modello migliore con un tasso di apprendimento basso. In particolare, confrontando questo parametro con i precedenti si può osservare che, con l'ottimizzazione automatica, il valore del tasso di apprendimento è molto inferiore rispetto al precedente approccio.

Mercato	Modello	shrinkage	interaction_depth	n.trees
Italia	gbm	0.0108425..	5	250
Cina	gbm	0.0118845..	5	470

Tabella 5.4: gbm, ottimizzazione guidata

5.2 Accuratezza

Nei seguenti grafici sarà possibile valutare l'accuratezza dei modelli utilizzati. In particolare si farà riferimento all'errore quadratico medio della radice e al coefficiente di determinazione.

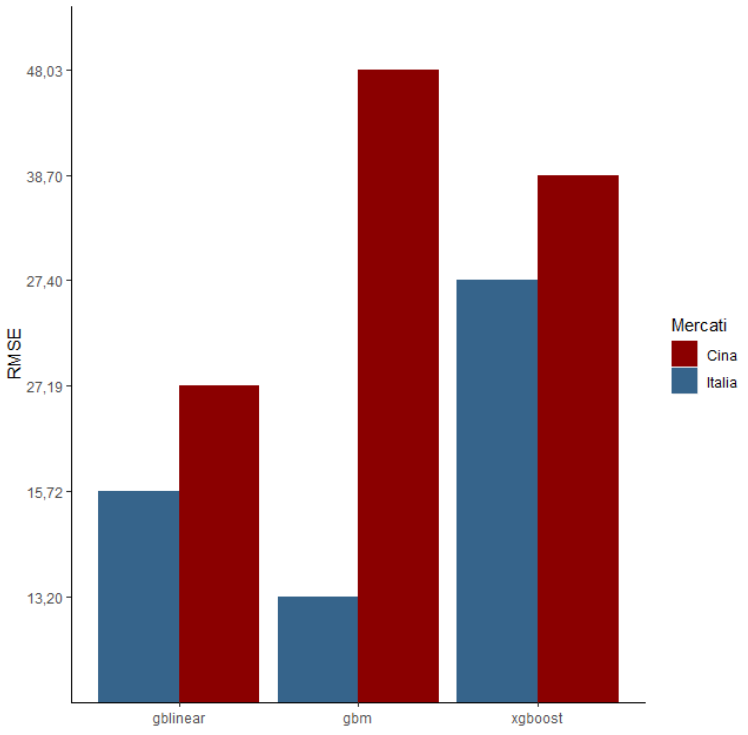

Figura 5.1: RMSE modelli ottimizzazione manuale

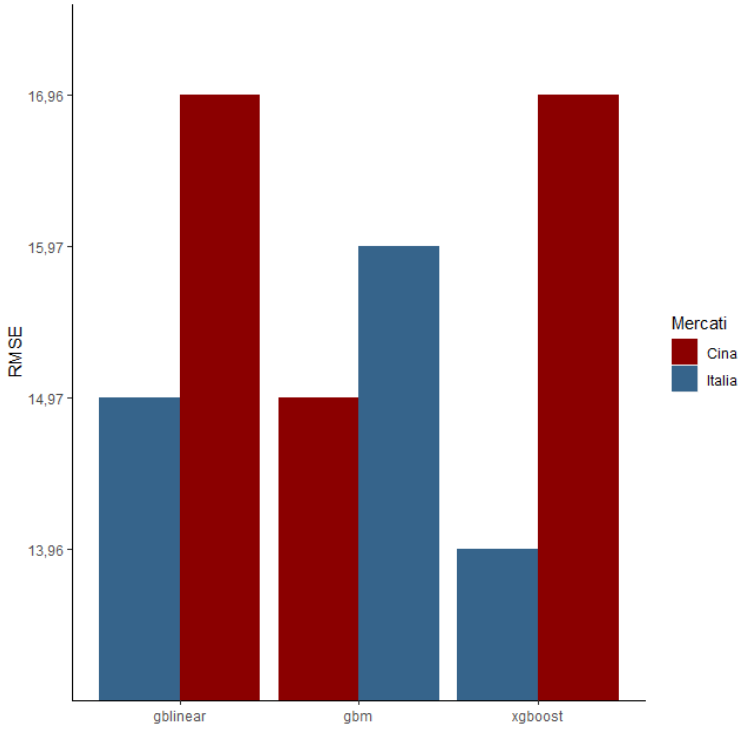

Figura 5.2: RMSE modelli ottimizzazione guidata

Nei precedenti grafici (**Figura 5.1** e **Figura 5.2**) viene riportata l'accuratezza espressa in termini di RMSE e relativa ai modelli applicati. In particolare, per ogni modello vengono confrontati i risultati ottenuti nel mercato italiano e in quello cinese. Le due figure riportano i risultati ottenuti in base al tipo di ottimizzazione applicato.
Nei grafici viene utilizzato un indice di errore quindi prestazioni migliori corrispondono a un indice minore.
Confrontando le figure si può notare che la performance di un modello cambia in base al dataset utilizzato e all'ottimizzazione applicata.

Nei successivi grafici sarà possibile individuare, per ciascun mercato, il modello che produce un minor errore.

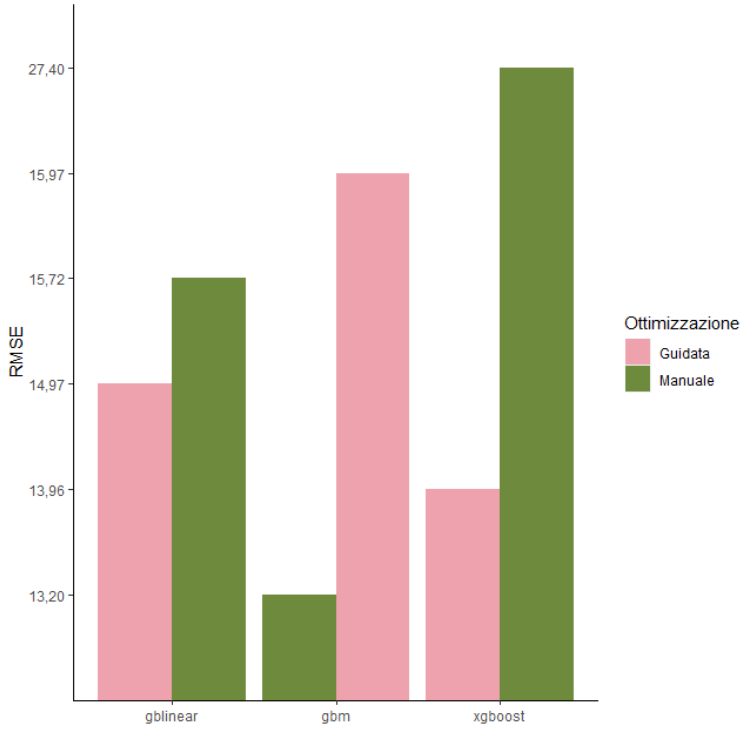

Figura 5.3: RMSE modelli mercato italiano

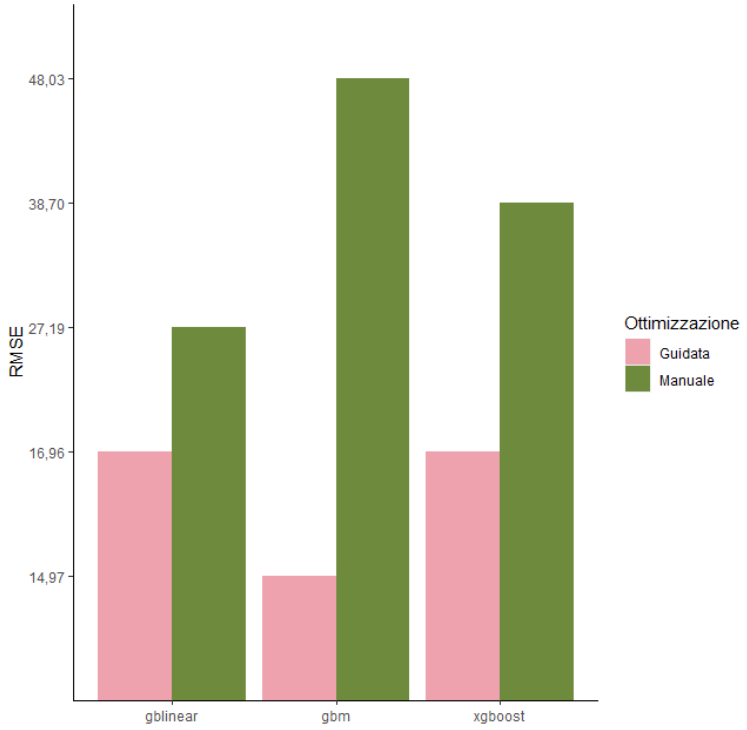

Figura 5.4: RMSE modelli mercato cinese

La **Figura 5.3** e la **Figura 5.4** riportano l'errore quadratico medio della radice ottenuto per ogni modello. In ciascun grafico viene confrontata l'accuratezza di ogni modello ottenuta utilizzando l'ottimizzazione manuale e l'ottimizzazione guidata. Le figure si riferiscono ai due mercati presi in analisi.

Osservando i grafici precedenti si verifica che in generale l'ottimizzazione guidata porta a delle previsioni più accurate. Si può vedere inoltre che le prestazioni dei modelli sul mercato cinese sono più basse rispetto a quello italiano, infatti i valori dell'*RMSE* relativi alla **Figura 5.4** sono più elevati, indicando una maggiore diffusione dei residui.
Questo avviene perchè si tratta di mercati strutturati in modo differente. Il mercato italiano, basandosi principalmente su una vendita al dettaglio, riporta una dispersione dei valori contenuta e questo rende più semplice la previsione. Il mercato cinese invece è basato sulla vendita all'ingrosso, quindi la dispersione dei valori è elevata, complicando così i tentativi di previsione.

Queste differenze rendono difficile la ricerca di un modello che sia in grado di effettuare previsioni accurate su entrambi i dataset.

L'RMSE è un indice di errore, ma non neccessariamente un modello che riporta un RMSE basso è in grado di fornire previsioni più attendibili.
Per una visione più completa e dettagliata è necessario effettuare un confronto sulla base del coefficiente di determinazione R^2.

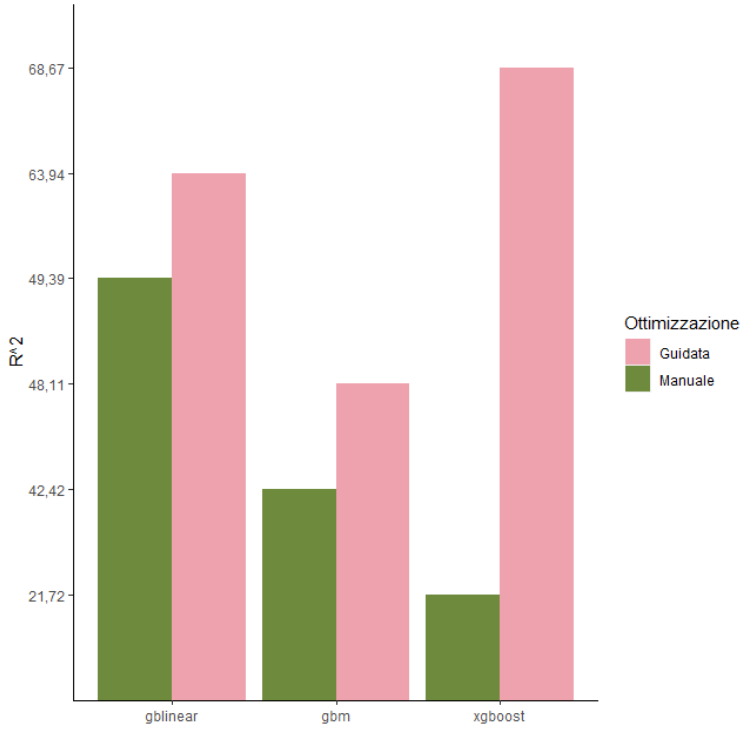

Figura 5.5: R^2 modelli mercato italiano

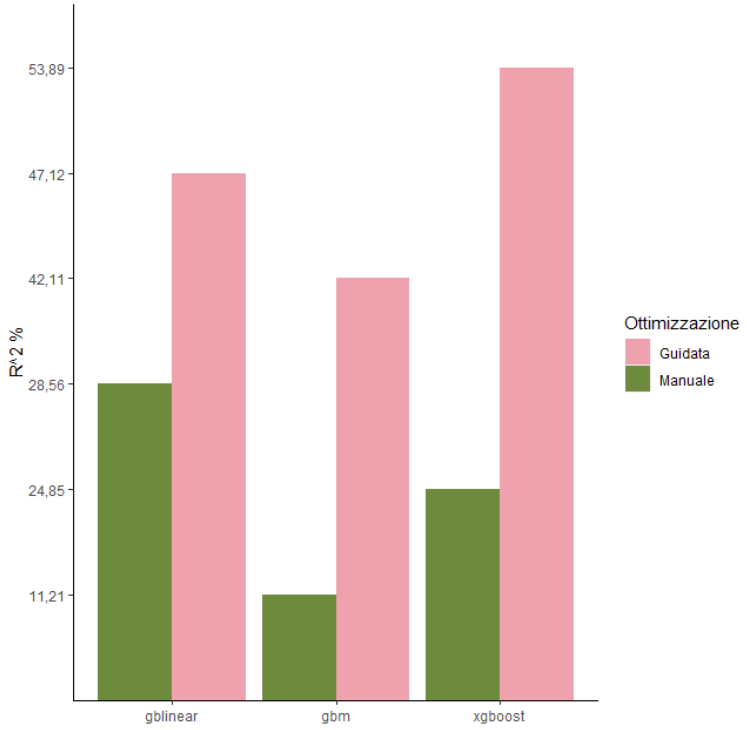

Figura 5.6: RMSR^2E modelli mercato cinese

Infine accostando i valori di RMSE a quelli di R^2 si può dire che modello che ottiene previsioni migliori per il mercato italiano è xgboost con i parametri riportati nella **Tabella 5.3**. Anche nel mercato cinese, il modello con un'accuratezza maggiore risulta essere xgboost con i parametri indicati nella **Tabella 5.3**.

Mercato	Modello	Ottimizzazione	RMSE	R^2%
Italia	xgboost	manuale	27.39673	21.71555
Italia	xgboost	guidata	13.95682	68.66721
Italia	gblinear	manuale	15.72005	49.38543
Italia	gblinear	guidata	14.97457	63.94258
Italia	gbm	manuale	13.20362	42.42347
Italia	gbm	guidata	10,96925	51,11462
Cina	xgboost	manuale	38.69667	24.85033
Cina	xgboost	guidata	16.95829	53.89006
Cina	gblinear	manuale	27.19064	28.55959
Cina	gblinear	guidata	16.95833	47.11542
Cina	gbm	manuale	48.02757	9.20792
Cina	gbm	guidata	14.96925	42.11462

Tabella 5.5: Riassunto risultati ottenuti

Conclusioni

Il progetto svolto aveva l'obiettivo valutare le performance degli algoritmi di boosting al fine di effettuare previsioni sull'ordinato netto. Sono stati proposti e valutati tre modelli molto simili fra loro, ma che hanno riportato risultati differenti.

Per migliorare le prestazioni è stata implementata una fase iniziale di ottimizzazione dei modelli, che si è sviluppata in due modi. Il primo approccio seguito è stata l'ottimizzazione manuale, in cui i parametri sono stati inseriti in una matrice. Dopo aver fatto il training su ogni possibile combinazione, sono stati scelti i valori ottimali. Il secondo metodo seguito è stata l'ottimizzazione guidata, in cui è stato definito un valore minimo e uno massimo per i parametri scelti. Il risultato di questa esecuzione è stata la combinazione di valori ottimali.
Infine è avvenuta la fase di training e test dei modelli ottenuti.

In conclusione all'analisi svolta si può dire che Extreme Gradient Boosting è risultato essere il modello migliore, riportando un maggior R^2.
Nonostante questo non è abbastanza performante da effettuare previsioni attendibili. Il modello infatti ha un $R^2\,\%$ inferiore a 80% e di conseguenza le previsioni che fornisce possono essere solo un supporto, non vanno quindi considerate singolarmente ma inserite nel contesto.

In base a questi risultati possono essere fatte diverse considerazioni.

La prima è relativa ai dati a disposizione. Entrambi i dataset su cui si è basata l'analisi sono composti solamente da dati tecnici. Trattandosi di una previsione in ambito fashion, l'ordinato netto non può essere influenzato solamente dal colore, dal materiale o altre caratteristiche specifiche dell'occhiale. Ci sarebbero molte variabili esterne da tenere in considerazione, come l'influenza delle mode, delle pubblicità, il ruolo dei social network o la concorrenza del mercato.
La fase di raccolta dei dati richiederebbe quindi più tempo, ma potrebber rivelarsi fondamentale per ottenere delle previsioni attendibili.

Un'altra considerazione da fare è che gli algoritmi di boosting potrebbero non rivelarsi la miglior soluzione per affrontare questo tipo di problema.
Ricordiamo però che la scelta di un algoritmo viene fortemente influenzata dal dataset su cui si lavora. Una prova di questo è data dalle performance ottenute applicando i modelli al dataset relativo al mercato cinese, che risultano inferiori a quelle ottenute sul dataset italiano. Questo viene confermato nei grafici del capitolo precedente, in particolare dalla **Figura 5.5** e **Figura 5.6**.

Risulta necessario quindi prendere in considerazione diversi tipi di algoritmi e, valutando i risultati, scegliere quello più performante.

Bibliografia

[1] Hurwitz J., Kirsch D.
"Machine Learning", IBM Limited Edition

[2] Mitchell T.
"Machine Learning"

[3] "Intelligenza Artificiale"
[http://www.intelligenzaartificiale.it/machine−learning/#Un_po8217_di_storia]

[4] Murphy K.
"Machine Learning a probabilistic perspective"

[5] Knox S.
"Machine Learning a concise introduction"

[6] Breiman L.
"Bagging predictors", in Machine learning, 1996.

[7] Breiman L., Cutler A.
"Random Forests"

[8] Rey T., Wells C., Kordon A.
"Applied Data Mining for Forecasting Using SAS", Chapter 1

[9] Kilimci Z. H., Akyuz O., Uysal M., Akyokus S., Uysal O., Bulbul B. A.,
Ekmis M. A.
Research Article

"An Improved Demand Forecasting Model Using Deep Learning Approach and Proposed Decision Integration Strategy for Supply Chain"

[10] "SkuBrain 101 Documentation"
[https://skubrain.com/docs/]

[11] Kaya G. O., Turkyilmaz A.
"Intermittent Demand Forecasting Using Data Mining Techniques"

[12] The R Project for Statistical Computing - "Getting Started"
[https://www.r-project.org/]

[13] What is R? - "Introduction to R"
[https://www.r-project.org/about.html]

[14] RStudio
[https://rstudio.com/products/rstudio/]

[15] Kuhn M.
"Package 'caret' "
[https://cran.r-project.org/web/packages/caret/caret.pdf]

[16] Greenwell B., Boehmke B., Cunningham J.
"Package 'gbm' "
[https://cran.r-project.org/web/packages/gbm/gbm.pdf]

[17] Chen T., He T., Benesty M.
"Package 'xgboost' "
[https://cran.r-project.org/web/packages/xgboost/xgboost.pdf]

[18] Bischl B., Lang M., Kotthoff L., Schiffner J., Richter J., Jones Z., Casalicchio G., Gallo M., Schratz P.
"Package 'mlr' "
[https://cran.r-project.org/web/packages/mlr/mlr.pdf]

[19] Girone G., Crocetta C., Massari A.
"Statistica"

Ringraziamenti

Ci tengo innanzitutto a ringraziare il professor Marco Di Felice, che mi ha seguita curando con pazienza e dedizione il mio lavoro.

Un ringraziamento va anche al Dott. Michael Romolo, correlatore di questa tesi, per il tempo che mi ha dedicato nel periodo di tirocinio e per la pazienza con cui ha risolto i miei dubbi.

Vorrei ringraziare i miei genitori, per aver permesso tutto questo. Grazie per aver sempre creduto in me e per avermi insegnato l'importanza dell'essere ambiziosi e puntare in alto.

Grazie alla mia numerosa famiglia, perchè avete gioito con me per ogni traguardo, non facendomi mai sentire sola.

Un grazie di cuore alle mie coinquiline, Agnese, Linda ed Eleonora, per avermi accompagnata in questi anni. Grazie per avermi accolto con le mie fissazioni e manie, ma grazie soprattutto perchè per me siete casa.

Grazie a Ruben, che da sempre è il mio compagno di vita e mi motiva ogni giorno a dare il meglio di me.

Un grazie speciale a Eleonora, che mi ha sempre capita al volo, rispettando i miei spazi, ascoltandomi, accogliendomi e coccolandomi, grazie per

essere sempre stata quello di cui avevo bisogno.

Grazie ad Alice, che mi ha insegnato che la distanza non separa le persone. Grazie per le infinite risate, per le lacrime, per tutte le volte che mi sono lamentata e per il supporto che mi ha saputo dare in ogni occasione.

Grazie ad Alice, perchè non mi ha mai lasciata sola a partire dai mille viaggi verso Bologna, per essere sempre un'amica sincera che sa come farmi tornare con i piedi a terra.

Grazie a Benedetta, per essere quell'amica della vita che spesso mi conosce meglio di quanto non mi conosca io.

Grazie a Kyra, per la forza e per l'energia con cui mi travolge, dandomi la spinta per ripartire quando mi sento a terra.

Grazie a Sara, per il dolce sorriso che non mi nega mai, per esserci sempre.

Infine vorrei ringraziare di cuore i miei compagni di corso, Virginia, Lucrezia, Beatrice, Alberto, Giacomo e Gabriele per aver condiviso con me questi anni di università, per aver sopportato tutte le mie acidate, e per aver riempito di risate le giornate.
Un ringraziamento particolare a Virginia e Lucrezia, perchè conoscervi è stata una sorpresa, grazie per avermi ascoltata sempre, per aver accettato ogni parte di me e per avermi motivato più volte. Vi ringrazio soprattutto per aver custodito la mia password universitaria e averla usata per consegnare i progetti che mi dimenticavo. Siete state fondamentali.

Grazie a tutte quelle persone che almeno una volta hanno sentito le mie lamentele, paranoie, ansie, le acidate e il nervoso, e mi hanno sopportato pazientemente.