UNIVERSITÀ DI PISA

Scuola di Ingegneria
Dipartimento di Ingegneria dell'Informazione

Corso di laurea in
Ingegneria Informatica

TESI DI LAUREA

Sviluppo di un modello di machine learning per l'identificazione di ticker nel testo

Candidato:
Federico Montini

Relatore:
Prof. Marco Avvenuti

Correlatore:
Dott. Leonardo Nizzoli

Anno Accademico: 2021-2022

Indice

Introduzione 3

1 **Capitolo 1 – Feature Engineering & Model Tuning** 6

1.1 Dataset 6

1.2 Features Engineering 7

1.2.1 Problematiche nel dataset 9

1.3 Addestramento dei modelli 9

1.3.1 Effetto del One-Hot Encoding sui dati 11

1.3.2 Gestione dei valori nulli e mancanti 12

1.3.3 Split del dataset e Grid-Search Cross-Validation 13

1.3.4 Addestramento 13

1.3.5 Parametri della Grid-Search CV 16

1.3.6 Best Model 16

2 **Capitolo 2 - Risultati** 18

2.1 Modelli di condronto: Dummy 18

2.2 Presentazione dei risultati 18

2.3 Feature importance 19

3 **Capitolo 3 - Conclusione** 22

4 **Appendice** 23

5 **Bibliografia** 28

6 **Sitografia** 29

Introduzione

Dal 1602, quando è stata aperta la prima borsa valori ufficiale ad Amsterdam, il mercato azionario è diventato fondamentale per tutte quelle aziende che cercano fondi privati per potersi fare strada all'interno di una realtà complessa e, spesso, molto avversa.

L'avvento della rivoluzione digitale ha sicuramente influenzato anche questo settore, rendendo le transazioni finanziarie sempre più veloci e complesse, ma anche direttamente accessibili alla grande massa dei piccoli risparmiatori. Questi, tramite piattaforme come https://robinhood.com/us/en/, possono acquistare e vendere azioni direttamente dal proprio smartphone.

Un'ulteriore svolta è avvenuta con l'avvento dei Social Network. Questi strumenti digitali hanno influenzato ogni aspetto della quotidianità, compreso il processo di investimento: il privato può adesso confrontare le proprie opinioni con altre persone che condividono lo stesso interesse e ha accesso a un'immensa mole di informazioni.

Un social su cui questo fenomeno ha preso piede è Reddit. Gli utenti di Reddit, conosciuti come "Redditors", possono pubblicare post testuali, con allegati, link e contenuti multimediali. Questo social è organizzato in subreddit che, come le classiche pagine del più famoso Facebook, sono divise in base agli argomenti discussi al loro interno: infatti, in alcune di loro, è stato pubblicato un regolamento delle tematiche trattabili. Inoltre, al fine di incoraggiare la pubblicazione di buoni contenuti, Reddit ha implementato un meccanismo denominato "karma" tale per cui l'utente guadagna un punto per ogni "like" (che su reddit è indicato con una freccia rivolta verso l'alto) e perde un punto per ogni "dislike" (freccia verso il basso) (Anderson K. E., 2015).

La mia tesi prende spunto proprio da ciò che è avvenuto all'interno di questi gruppi, in particolar modo in un subreddit chiamato r/WallStreetBets, nel quale piccoli investitori

privati hanno deciso di contrastare quei grandi investitori (o Hedge Fund[1]) che, tramite lo "short selling"[2], stavano investendo sul ribasso del valore delle azioni di GameStop. Grazie al subreddit sono riusciti ad accordarsi per comprare simultaneamente una grande quantità di azioni che ha portato il valore dell'azione ad impennarsi, raggiungendo anche dieci volte il valore di partenza e causando gravi perdite agli hedge fund che stavano speculando sul ribasso dell'azione (Di Muzio T., 2021).
Dopo GameStop, ci sono stati tentativi analoghi su altri titoli come AMD e Nokia. Tuttavia, in questi casi, i redditors non hanno avuto altrettanto successo come nella prima campagna di GameStop.

Data la novità e le conseguenze di questa campagna online, che è stata denominata "Gamestonk", diventa fondamentale studiarne la nascita e lo sviluppo, anche al fine di capire se si è trattato di un fenomeno genuino e spontaneo oppure se nasconde tentativi di manipolazione e speculazione.
Data l'enorme mole di messaggi ottenibili da questi Social Network, è necessario che le analisi siano automatiche. Il primo passaggio per ogni applicazione di Social Media Analysis è filtrare i contenuti pertinenti, su cui poi concentrare le analisi successive. Un modo per individuare i messaggi pertinenti, rispetto alla promozione degli acquisti coordinati di Gamestop e altri titoli, è individuare all'interno del testo le volte in cui viene menzionato il ticker corrispondente.
Un ticker è un'abbreviazione utilizzata per identificare univocamente i titoli di una particolare azienda in uno specifico mercato. Sui social viene utilizzata comunemente una convenzione, nata su Twitter, per riferirsi ai titoli. Questa convenzione è rappresentata dall'aggiunta del simbolo "$" prima del ticker che, proprio come il simbolo "#" per l'hashtag, è finalizzato a consentire agli utenti di individuare facilmente i messaggi che li contengono e organizzare le conversazioni in base ai topic.
Nonostante queste convenzioni, non è tuttavia semplice individuare i ticker all'interno dei messaggi, sia perché spesso gli utenti non sfruttano le convenzioni, sia perché i ticker coincidono con parole di senso compiuto.

[1] Gli 'Hedge Fund' sono dei grandi fondi speculativi gestiti da assicurazioni o privati con grandi disponibilità monetarie che hanno lo scopo di ottenere il maggior guadagno possibile dai loro investimenti.
[2] Lo 'short selling' è una strategia di investimento finalizzata nel trarre profitto dal calo del valore di uno specifico titolo azionario sul quale effettuiamo l'operazione.

Ecco, quindi, che si rende necessario un modello di machine learning che automatizzi questa classificazione dei messaggi.

Per fare ciò mi sono servito di due modelli di Machine Learning, addestrati su un dataset costituito da 5000 messaggi estratti da questi subreddit.

Con **algoritmi machine learning** si intende una forma di intelligenza artificiale capace di apprendere dai dati iterativamente tramite l'utilizzo di algoritmi finalizzati a prevedere e descrivere risultati.

In particolare, utilizzeremo dei modelli di machine learning di classificazione binaria, in grado di stabilire se una nuova istanza (una porzione di testo) sia un ticker oppure no. Per farlo sono stati utilizzati algoritmi di tipo **supervisionato**, ossia modelli addestrati su dataset annotati (ad ogni istanza è assegnata la label da predire). Tra i vari possibili algoritmi sono stati scelti i modelli di tipo **ensemble** basati su alberi di decisione: si tratta di particolari algoritmi costituiti da una serie di modelli più piccoli, ognuno dei quali restituirà la propria soluzione. Il risultato finale sarà quello restituito dalla maggioranza dei modelli.

Il primo modello scelto è **Random Forest**: questo è formato da una "foresta" di alberi decisionali, ognuno dei quali darà la propria risposta al quesito "È un ticker?" e il risultato restituito dall'algoritmo sarà quello che riceverà la maggioranza di voti.

Successivamente procederemo con un secondo algoritmo chiamato **Gradient Boosting Decision Tree**: questo è formato da una serie di modelli più piccoli e soggetti all'errore utilizzati in sequenza in modo tale che ogni albero riconosca e corregga gli errori commessi dai precedenti.

Per concludere, si estrapolano e confrontano le performance di questi due modelli.

La tesi è divisa in tre capitoli:

- Un primo capitolo in cui osserveremo tutto il processo di preparazione del data-set e del modello.
- Un secondo in cui verranno trattati i risultati ottenuti analizzando vari parametri per misurare l'accuratezza dei modelli testati.
- Un terzo capitolo, dedicato alle conclusioni.

I tre capitoli saranno seguiti da un'appendice contenente il codice.

Capitolo 1 – Modelli di classificazione dei ticker

In questo capitolo verrà descritto il dataset, oltre che il processo di annotazione manuale effettuato e il modo in cui è stato spezzato il dataset, in modo da formare training e test set per concludere col processo di tuning dei modelli.

1.1 Dataset

Il data-set fornito è costituito da un file ".csv", all'interno del quale sono presenti 5000 record strutturati su sei colonne separate da virgole:

id,"body","ticker","start","stop","is_ticker"

1. ID: È un identificatore unico per ogni record
2. Body: È il contenuto del commento estratto
3. Ticker: È il ticker individuato all'interno del body
4. Start: È il numero del carattere dopo cui è presente il presunto ticker
5. Stop: È il numero del carattere dopo il quale il presunto ticker è terminato
6. Is_Ticker: Questa è la colonna di annotazione, in cui è stato appuntato singolarmente ogni record, tentando di capire se la sottostringa contenuta tra "Start" e "Stop" fosse o no un ticker e ricercando eventuali altri significati che la sottostringa potesse assumere.

Un esempio di record è il seguente:

gavc3qt,"People vote for incumbents based on how comfortable their own lives are. Material conditions are the ultimate metric for human satisfaction.","BASED",27,32,""

È importante far notare che, specialmente in alcuni casi, la mancanza di contesto è vincolante: come nel caso del ticker "AMC". Questo specifico ticker corrisponde

esattamente al nome dell'azienda, ragion per cui risulta particolarmente complicato, in alcuni casi, capire se si stia parlando dell'azienda o dell'indice azionario.
Nel complesso è un aspetto importante, in quanto andrà ad influenzare le statistiche finali abbassando precision e recall dell'algoritmo.

1.2 Feature Engineering

Durante il processo di annotazione, è stata prestata attenzione a delle caratteristiche comuni tra i record che si sono dimostrati essere, e non essere, ticker.
Le caratteristiche ricorrenti più importanti individuate sono:

- Il ticker viene spesso scritto completamente in maiuscolo.
- La regola precedente ha un significato variabile in base a che rapporto c'è tra maiuscole e minuscole nel body. Un commento scritto completamente in maiuscolo farà perdere rilevanza alla feature scritta precedentemente.
  Per bilanciare ciò sono state inserite alcune feature che verificano che il "body" sia completamente scritto in maiuscolo e il rapporto tra le maiuscole e minuscole nel body e nel ticker.
- I ticker vengono individuati unicamente tramite il simbolo "**$**" andando a formare il "**Cashtag**", largamente utilizzato su Twitter.
- Come indicato precedentemente viene sempre, o quasi, indicato il tipo di opzione di cui si parla per quella determinata azione. Viene esplicitamente scritto "**calls**", "**puts**", "**dip**" e "**short**".
- Se la frase contiene solo la parola interpretata come possibile ticker è molto probabile che rappresenti un ticker.

Oltre a questi aspetti è necessario osservare anche il ruolo grammaticale che i ticker occupano all'interno della frase.
Ha senso, quindi, pensare che il ticker sia utilizzato come nome e che l'abbreviazione rappresenti un'organizzazione.

Per implementare questi due tipi di analisi è possibile sfruttare alcune librerie che hanno le funzionalità:

1. **Part-of-Speech tagging**: È una tecnica di preprocessing attuata tramite un modello di machine learning addestrato a riconoscere e taggare le parole che costituiscono la frase secondo il ruolo grammaticale che ricoprono. Questo modello fornisce anche un valore numerico chiamato 'confidence' che esprime la precisione con cui fornisce ogni previsione.
2. **Name-Entity Recognition**: È una tecnica di preprocessing attuata tramite un modello di machine learning addestrato a riconoscere in una frase delle 'named entities' che è capace di inserire in specifiche categorie.

Per sfruttare i pattern individuati durante il processo di annotazione sono state definite specifiche features:

Nome Feature	Tipo	Descrizione
Ticker_Uppercase	Numerica	Booleano che indica se il ticker è scritto in maiuscolo oppure no
Uppercase_Body	Numerica	Booleano che indica se il corpo del messaggio è scritto tutto in maiuscolo oppure no
First_is_Uppercase	Numerica	Booleano che indica se la prima lettera del ticker è scritta in maiuscolo oppure no
Upper_ratio_total	Numerica	Rapporto tra maiuscole e minuscole nell'intero corpo del messaggio
Upper_ratio_ticker	Numerica	Rapporto tra maiuscole e minuscole nel ticker
Name_Entity_Recognition	Categorica	Interpretazione del significato della stringa ticker secondo il modello di Machine Learning NER
Confidence_NER	Numerica	Sicurezza dell'algoritmo nella predizione del modello NER
Char_prec_$	Numerica	Booleano che indica se il ticker è preceduto dal carattere "$" oppure no
Ticker_only	Numerica	Booleano che indica se il messaggio è costituito solo dal ticker
Keywords_found	Numerica	Booleano che indica se nel messaggio è stata individuata una o più keywords: "dip",

		“calls”, “puts” e “short”
Part_of_speech	Categorica	Ruolo grammaticale che il ticker assume nella frase
Confidence_pos	Numerica	Sicurezza dell’algoritmo nella predizione del modello POS

Tabella 1: Descrizione delle features

Queste features sono state implementate in Python tramite l’utilizzo delle librerie:

- **Pandas**: una libreria open-source di data-analysis che ha permesso di importare un file csv sotto forma di data-frame e di salvare il frame modificato come file csv.
- **Flair**: una libreria che fornisce modelli addestrati ad effettuare il PoS tagging e la NER. In particolare, sono state utilizzate **flair.Sentence** e **flair.SentenceTagger**.

In particolare, queste due feature sono state così implementate:

```
#Update feature: Name Entity Recognition e confidence di ciò che è stato individuato tramite flair
#*********************************************************************************************
sentence = Sentence(body)
tagger.predict(sentence)
#Inizializzo il valore a none a priori, nel caso sovrascrivo
file.loc[i,"name_entity_recognition"]="None"
file.loc[i,"confidence_NER"]="None"
#Indice j necessario a scorrere i labels individuati nella sentence
j=0
#Scorro tutti gli span individuati nella sentence
for span in sentence.get_spans('ner'):
    #Se trovo uno span che ha come parte del testo individuata il ticker scrivo nella feature
    #come viene individuato e con quale confidenza
    if span.text==extractedTicker:
        file.loc[i,"name_entity_recognition"]=span.labels[j].value
        file.loc[i,"confidence_NER"]=span.labels[j].score
    j+1
#Update feature: Part-of-Speech tagging
#*********************************************************************************************
sentence = Sentence(body)
posTagger.predict(sentence)
file.loc[i,"part_of_speech"]="None"
file.loc[i,"confidence_POS"]="None"
#Indice j necessario a scorrere gli spans individuati nella sentence
j=0
#Scorro tutti gli span individuati nella sentence
for span in sentence.get_spans('pos'):
    if span.text==extractedTicker:
        file.loc[i,"part_of_speech"]=span.labels[j].value
        file.loc[i,"confidence_POS"]=span.labels[j].score
    j+1
```

1.2.1 Problematiche nel dataset

Una caratteristica del data-set che ha causato problemi durante la parte di feature engineering è rappresentata dai caratteri che codificano le emoticon.

La codifica delle emoticon è costituita da una combinazione di caratteri, non un singolo carattere, e ciò ha causato uno shift verso destra negli effettivi valori “start” e “stop”.

È stato quindi necessario ricalcolare i valori di questi parametri per tutti i record, mettendo tutto il body in minuscolo, tramite la funzione lower(), e cercando la sottostringa all'interno del body, tramite la find(). Entrambe le funzioni sono presenti nelle librerie standard Python.

Questa correzione è stata così implementata:

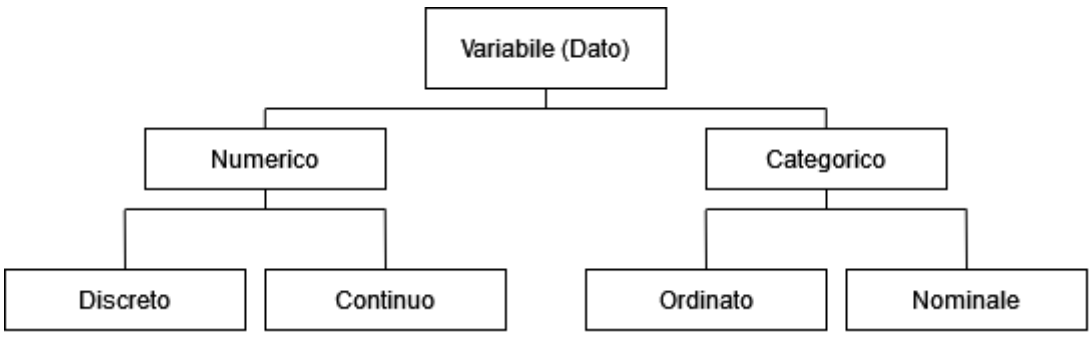
```
#Prelevo i dati necessari alle analisi contenuti nella tupla
#*********************************************************************************************
body = file.loc[i][1]
start=file.loc[i][1].lower().find(file.loc[i][2].lower())
stop = start + len(file.loc[i][2])
file.loc[i,"start_up"]=start
file.loc[i,"stop_up"]=stop
extractedTicker = body[start:stop]
```

Dove con loc[i][1] e loc[i][2] si preleva il campo "body" e "ticker" dell'i-esimo record.

Adesso è stato ottenuto un insieme di features calcolate su tutti i cinquemila record e salvate in un secondo file csv, sul quale dovremo effettuare il preprocessing delle features, al fine di gestire tutte le features di tipo categorico non interpretabili dagli algoritmi di machine learning.

1.3 Addestramento dei modelli

Il primo passaggio da effettuare prima di procedere al vero e proprio addestramento dei modelli è il preprocessing. Con questo temine intendiamo quel passaggio, nella preparazione dei dati, in cui le informazioni vengono trasformate (o codificate) per portarle in uno stato in cui un calcolatore possa interpretarle facilmente.



Le features che hanno bisogno di preprocessing sono tutte quelle categoriche, ovvero quelle che non possono essere interpretate dal modello tramite un coefficiente numerico, essendo rappresentate da stringhe o altri tipi di dati più complessi.

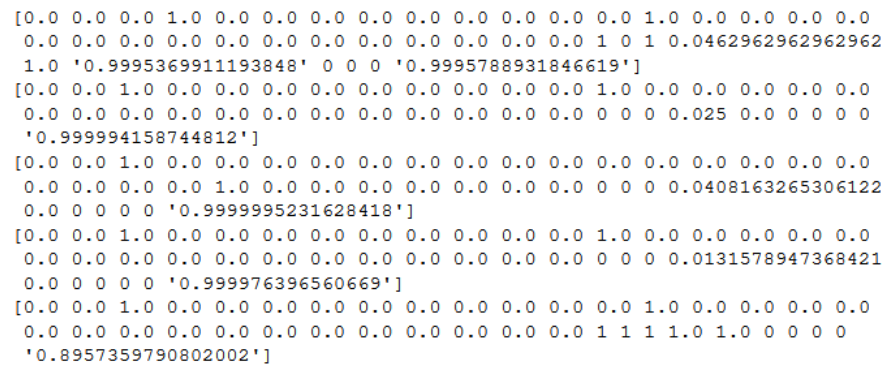
Nel nostro caso le features che necessitano di preprocessing sono quelle derivate dalle predizioni di Name-Entity Recognition e Part-of-Speech tagging, i cui valori sono rappresentati da stringhe. La tecnica scelta per codificare questo tipo di variabili è ”**One-hot encoding**”, dato che assumeranno una quantità di valori limitata. Con questa tecnica di codifica otterremo una serie di features separate, generate da uno stesso campo iniziale, nelle quali solo una di queste conterrà “1.0” mentre le restanti “0.0”.
Infine, sarà necessario gestire i valori “None” o assenti che possono assumere le confidence della Part-of-speech tagging e Name-Entity Recognition.

1.3.1 Effetto del One-Hot Encoding sui dati

Al caricamento del file csv avremo complessivamente dodici features e le due features da codificare si trovano rispettivamente in quinta e decima posizione.
Per effettuare questa operazione è opportuno servirsi di due librerie fornite da scikit learn:

- **ColumnTransformer**: Per effettuare la scomposizione della codifica delle features su più colonne.
- **OneHotEncoder**: Che si occupa di effettuare l’encoding delle features categoriche.

Dopo questa operazione le dodici features di partenza risulteranno essere quaranta: cinque dovute alla codifica della NER e venticinque dovute alla codifica del POS, seguite dalle dieci features rimaste immutate.



In questa immagine è possibile osservare le features memorizzate sotto forma di array della libreria "**numpy**". I primi cinque valori rappresentano la codifica della feature NER, mentre i venticinque successivi rappresentano la codifica del valore del POS.

1.3.2 Gestione dei valori nulli e mancanti

Durante il calcolo delle features è stato necessario gestire i casi in cui NER e POS non sono riuscite ad attribuire una predizione al ticker.

Per far ciò è stato attribuito come valore di default: "**None**", che verrà sovrascritto se, durante lo scorrere dei vari label individuati, ne verrà trovato uno che nell'attributo "text" contenga la stringa del ticker.

Come evidenziato nel paragrafo 1.5, i modelli di machine learning non sono in grado di trarre informazioni da parametri che non sono numerici, rendendo necessario modificare tali valori.

I vari "None" contenuti nelle colonne di **confidence** sono stati sostituiti dal valore "0", questo perché concettualmente il parametro di confidence è utilizzato per stimare la correttezza della predizione, e, nel caso in cui questa non sia stata effettuata, non avrebbe senso che tale valore fosse diverso dal valore nullo.

I valori nulli o mancanti contenuti nelle predizioni dei due modelli vengono modificati direttamente dal One-Hot Encoder, che procederà ad attribuire una codifica a "None".

1.3.3 Split del dataset e Grid-Search Cross-Validation

Prima di eseguire il tuning degli iperparametri del modello, il data set viene spezzato in due parti: il "training set" composto da quattromila record, che verrà utilizzato per addestrare e convalidare il modello e il "test set", con i record rimanenti, utilizzato per valutarne le performance.

Questo split viene effettuato mediante la funzione "**train_test_split**", importata da una libreria di scikit-learn. Una caratteristica importante della divisione dei record è che questa è stata eseguita come "stratified": i record vengono quindi riarrangiati nei due subset affichè mantengano la stessa densità di risposte 'positive' e 'negative' posseduta nel dataset iniziale.

```
1.0    2482
0.0    1518
Name: is_ticker, dtype: int64
```

Figura 1: Count eseguito sul train set

```
1.0    3102
0.0    1898
Name: is_ticker, dtype: int64
```

Figura 2: Count eseguito sul totale dei dati.

```
1.0    620
0.0    380
Name: is_ticker, dtype: int64
```

Figura 3: Count eseguito sul test set

Come si può osservare dalle Figure 1, 2 e 3 avremo la divisione dei dati in due sottogruppi, ognuno dei quali con un rapporto positive/totale pari a circa 62%, così come l'insieme iniziale.

La stratificazione risulta molto importante: insieme allo shuffle[3] dei record impostato di default, permette di evitare dei bias dovuti ad un'eccessiva concentrazione di risposte positive (o negative) nel set di train (Hong G., 2010).

1.3.4 Addestramento

I due modelli scelti sono Random Forest e Gradient-Boosting Decision Tree. Entrambi sono basati su:

- Alberi decisionali: ossia modelli predittivi in cui ogni nodo rappresenta una variabile e ogni arco tra due nodi il valore che questa può assumere;

[3] Instanziato con una random_state prefissata al fine di garantire la riproducibilità dell'esperimento.

- Di tipo ensemble: modelli che compiono un'elevata quantità di simulazioni effettuate a partire da condizioni iniziali leggermente diverse tra loro, e stimano un risultato.

Random Forest è un particolare algoritmo che svolge le predizioni parallelamente. Una volta attribuite alcune features ad ogni sottoalbero, si preoccupa di fornirgli il record con le features e di raccogliere le risposte di ogni albero decisionale. La risposta che riceverà la maggioranza di voti sarà quella che l'algoritmo sceglierà come stima.

Gradient-Boosting DT invece utilizza una logica differente in quanto, a differenza di Random Forest che agisce in parallelo, attua sui propri alberi decisionali una predizione di tipo sequenziale finalizzata a correggere gli errori dei modelli precedenti. La risposta finale sarà il risultato dell'ultimo albero decisionale.

Sono stati scelti questi due modelli per la presenza di features categoriche e dal momento che si adattano maggiormente al dataset disponibile, che è piuttosto piccolo.

Per ognuno dei modelli di ML analizzati sono proposti una serie di **iperparametri**[4] su cui effettuare la ricerca:

1. **Random Forest**
    a. **n_estimators**: Rappresenta il numero di alberi che costituiranno la foresta;
    b. **max_features**: Rappresenta il numero di feature assegnate ad ogni albero;
    c. **max_depth**: Rappresenta l'altezza massima fino a cui un albero decisionale si può sviluppare;
    d. **criterion**: Rappresenta la funzione che misura la qualità di uno split.

2. **Gradient-Boosting Decision Tree**
    a. **loss**: Rappresenta la funzione di loss scelta per valutare la modellazione del dataset eseguito dall'algoritmo;
    b. **learning_rate**: Tasso di riduzione del contributo fornito da ogni albero decisionale;

[4] Gli iperparametri sono particolari parametri da specificare prima che il processo di addestramento del modello inizi. Sono infatti utilizzati per controllare il processo di apprendimento stesso dell'algoritmo e di conseguenza non possono essere dedotti durante l'addestramento.

c. **n_estimators**: Rappresenta il numero di volte che la soluzione potrà essere corretta da un altro albero decisionale;
d. **criterion**: Rappresenta la funzione che misura la qualità di uno split.

Per testare quali siano gli iperparametri più adatti è stata utilizzata la Grid-Search Cross Validation, una funzione, fornita da sci-kit learn, che riserva una sezione del dataset di train per convalidare il modello, al fine di capire le prestazioni del nostro modello, senza però dover riutilizzare come test gli stessi record utilizzati per addestrarlo.
Nello specifico i parametri testati durante l'esecuzione della Grid-Search Cross Validation sono rispettivamente:

```
param_grid = {
    'n_estimators': [130,150,170,180,190,200],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth' : [8,9,10,11],
    'criterion' :['gini', 'entropy']
}
```

Figura 4: Valori iparametri testati per Random Forest

```
param_grid = {
    'loss' : ['deviance','exponential'],
    'learning_rate' : [0.04,0.06,0.08],
    'n_estimators' : [80,90,100],
    'criterion' : ['friedman_mse', 'squared_error'],
    'max_depth' : [4,5,6]
}
```

Figura 5: Valori iperparametri testati per Gradient-Boosting Decision Tree

I valori che gli iperparametri possono assumere sono stati adattati attraverso più esecuzioni in modo da ottenere valori che non cadessero ai margini dell'intervallo preso in esame. Se, ad esempio, per la "max_depth" avessimo ricevuto come valore ottimo "6", avremmo avuto garanzia che questo sarebbe stato il migliore dei tre proposti, ma non il migliore in generale. Se il valore è interno all'intervallo è possibile affermare che ci troviamo in un punto di massimo, almeno relativo.

1.3.5 Parametri della Grid-Search CV

Durante l'allocazione dell'oggetto "GridSearchCV", sempre fornito da librerie di sci-kit learn, ne sono stati specificati alcuni parametri:

1. L'algoritmo da ottimizzare[5];
2. La griglia contenente i parametri con i vari valori che questi possono assumere;
3. Un parametro numerico denominato "**cv**", utilizzato per determinare la strategia di split per la cross validation. Per questo parametro è stato scelto il valore "4": in tal modo ogni quattro record utilizzati per apprendere ne verrà utilizzato uno per validare.
4. Un parametro denominato "**scoring**", utilizzato per specificare quale metrica prestazionale vogliamo ottimizzare. Per questo parametro è stato scelto il valore "f1", che rappresenta la media armonica tra precision[6] e recall[7] nel caso in cui il valore da predire sia di tipo binario. L'F1 è così calcolata:

$$\mathrm{F1} = \frac{2}{\frac{1}{precision} + \frac{1}{recall}}$$

   Ottimizzando questo parametro otterremo un buon bilanciamento tra precision e recall, nonostante ci sia una distribuzione dei risultati non equa (come visto nel paragrafo 1.6: 62% positive e 38% negative).
5. Infine, il parametro denominato "**error_score**" settato al valore "raise", al fine di notificare eventuali errori durante il calcolo dello score del modello.

Non rimane che eseguire il codice e addestrare il modello sul train set; dopodichè potremmo valutare le prestazioni dei nostri modelli.

1.3.6 Best Model

Eseguito il codice e instanziati i modelli si procede all'addestramento e alla validazione del modello tramite la funzione "fit" fornita dalla GridSearchCV.

[5] Instanziato con una random_state prefissata al fine di garantire la riproducibilità dell'esperimento.
[6] La precision rappresenta la relazione tra esempi 'true positive' e tutti gli esempi che il modello classifica come positivi.
[7] La recall indica la relazione tra gli esempi classificati come positivi rispetto al totale di esempi positivi.

Al termine di questa operazione è possibile accedere al parametro denominato “best_params_”, che conterrà i migliori valori per gli iperparametri scelti tra le opzioni fornite alla funzione.

I valori ottenuti sono per Random Forest:

- criterion: Entropy;
- max_depth: 9;
- max_features: auto;
- n_estimators: 170.

Per Gradient-Boosting D.T.:

- criterion: squared_error;
- learning_rate: 0.6;
- loss: exponential;
- max_depth: 5;
- n_estimators: 90.

Il modello ottimo, inizializzato con tali iperparametri, sarà accessibile tramite l’attributo “best_estimator_”: non rimane che fargli eseguire la predict sui mille record riservati alla fase di test.

Capitolo 2 – Risultati

2.1 Modelli di confronto: Dummy

Come metro di paragone sono stati addestrati due semplici modelli predittivi forniti dalla libreria sci-kit learn. Il modello, con il nome "**dummy**", fornisce più strategie di classificazione.

Dummy è stato eseguito con le due strategie:

1. "constant", tramite cui predice sempre il risultato più comune presente nel dataset, che, come abbiamo potuto osservare nel paragrafo 1.6, è "1.0".
2. "stratified", nella quale l'algoritmo genera predizioni rispettando la densità di risposte presente nel dataset iniziale fornitogli.

2.2 Presentazione dei risultati

I risultati ottenuti dai quattro modelli sono:

	Precision	**Recall**	**F1-score**
Random Forest	**89%**	86%	88%
Gradient-Boosting DT	87%	**90%**	**89%**
Dummy("constant")	62%	100%	77%
Dummy("stratified")	63%	64%	64%

Tabella 2: Confronto dei risultati.

Osservando i risultati si può notare che:

1. Il GBDT ha risultati nel complesso leggermente migliori rispetto al RF, superandolo nel parametro ottimizzato durante la Grid-Search dell'1%;
2. Mentre RF ha una precision del 2% più alta, GBDT supera la recall di RF del 4%.

È quindi consigliabile:

- Utilizzare Random Forest nel caso in cui sia preferibile avere maggiore sicurezza che i record classificati come ticker lo siano effettivamente;

- Utilizzare Gradient-Boosting Classifier nel caso in cui sia preferibile avere qualche record falso-positivo rispetto a perdere qualche record positivo tra le risposte segnalate come negative.

È importante notare che l'F1 si discosta tra i due algoritmi solamente dell'1%, rendendo in ogni caso validi entrambi i modelli. Inoltre, Gradient-Boosting è un algoritmo computazionalmente più oneroso e richiede più tempo per essere eseguito.
Prima di scegliere è quindi necessario stimare il carico di lavoro e il tipo di prestazione da preferire.

	0	**1**
0	3.1×10^2	67
1	84	5.4×10^2

Tabella 3: Matrice di confusione per algoritmo Random Forest

	0	**1**
0	3×10^2	82
1	62	5.6×10^2

Tabella 4: Matrice di confusione per algoritmo Gradient Boosting Decision Tree

A conferma delle statistiche viste precedentemente nelle tabelle 3 e 4 possiamo osservare, attraverso delle **matrici di confusione**, il numero di errori fatti dai due algoritmi rispettivamente per "false negative" (1,0) e "false positive" (0,1).

2.3 Feature importance

Una caratteristica da osservare dopo l'addestramento dei modelli è l'importanza che ogni modello ha dato alle features.

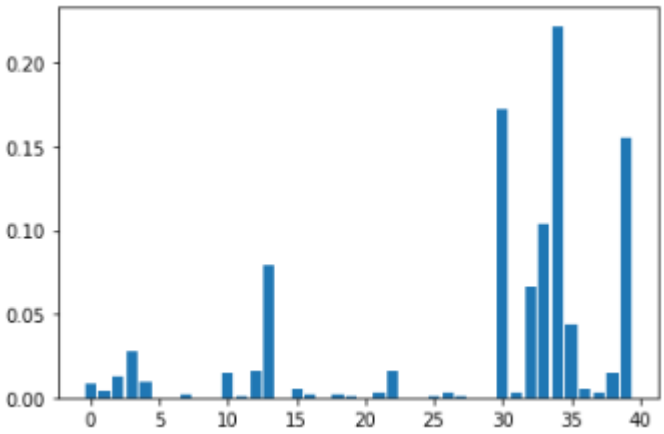


Figura 6: Feature importance per Random Forest

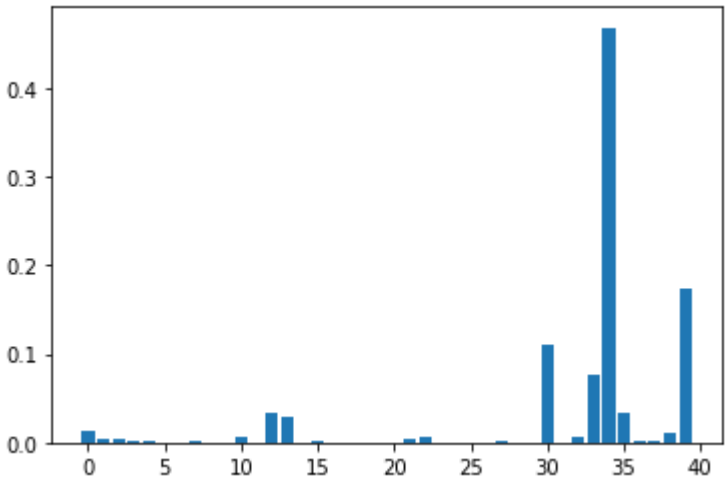


Figura 7: Feature importance GBDT

Come è possibile osservare da Figura 10 e Figura 11 l'importanza che i due algoritmi danno alle features sono sostanzialmente diverse:

- **Random Forest**: Attribuisce alle features un'importanza più bilanciata e considera maggiormente le prediction del POS (feature 13) e della NER (feature 3) rispetto al GBDT. Grande importanza viene data anche alle ultime dieci features (30-39) non codificate tramite l'One-Hot Encoding: su tutte spiccano la feature 39 (15%), che rappresenta la confidence del POS tagging, la feature 34 (22%), che rappresenta il rapporto tra maiuscole e minuscole nella citazione del ticker, e la feature 30 (17%), variabile booleana, che indica se il ticker è stato scritto in maiuscolo.
- **Gradient-Boosting Decision Tree**: Attribuisce alle features un'importanza molto sbilanciata. A differenza del RF non dà importanza alla predizione dell'algoritmo

NER e diminuisce l'importanza anche della predizione dell'algoritmo POS. Le features considerate più importanti sono le stesse viste nel caso precedente ma più accentuate: la feature 34 raggiunge un peso pari al 46% del totale sulla decisione finale.

Capitolo 3 - Conclusioni

Come si è visto all'interno del capitolo 2, sono estratti dei commenti e collezionati all'interno di un set, dove vengono annotati manualmente per creare un insieme di record su cui poter addestrare dei modelli.

In base alle caratteristiche individuate durante il processo di annotazione sono state scelte delle features su cui basare le scelte dei nostri algoritmi per concludere la preparazione dei dati tramite l'encoding delle feature categoriche.

Successivamente è stato effettuato il tuning degli iperparametri e l'addestramento dei modelli per concludere ottenendo i seguenti risultati:

- Una F1 pari all'88% per l'algoritmo Random Forest;
- Una F1 pari all'89% per l'algoritmo Gradient-Boosting Decision Tree.

Questi modelli, data la loro alta precisione, si prestano ad essere utilizzati in vari ambiti che riguardino lo studio di ticker. In particolare, potrebbe aver senso effettuare una ricerca sui ticker più discussi all'interno dei vari subreddit e, conseguentemente, del motivo per cui questi sono acquistati/venduti; o uno studio che osservi come le discussioni all'interno dei vari subreddit, con le relative compravendite di titoli, influenzino l'andamento del titolo stesso.

Appendice

Preparazione del dataset

```
import pandas as pd
from flair.data import Sentence
from flair.models import SequenceTagger
tagger = SequenceTagger.load('ner')
posTagger= SequenceTagger.load("flair/pos-english")
#Lettura del file CSV
file=pd.read_csv('reddit_csv_annotation.csv')
#Creazione colonne in cui inserire le features per ogni tupla
file["start_up"]=""
file["stop_up"]=""
file["Ticker_Uppercase"]=""
file["Uppercase_Body"]=""
file["First_is_Uppercase"]=""
file["upper_ratio_total"]=""
file["upper_ratio_ticker"]=""
file["name_entity_recognition"]=""
file["confidence_NER"]=""
file["char_prec_$"]=""
file["ticker_only"]=""
file["keyword_found"]=""
file["part_of_speech"]=""
file["confidence_POS"]=""
```

Figura 8: Creazione colonne features

```
for i in range(1000):
    #Prelevo i dati necessari alle analisi contenuti nella tupla
    #*************************************************************************************************
    body = file.loc[i][1]
    start=file.loc[i][1].lower().find(file.loc[i][2].lower())
    stop = start + len(file.loc[i][2])
    file.loc[i,"start_up"]=start
    file.loc[i,"stop_up"]=stop
    extractedTicker = body[start:stop]
    #Update feature: Il potenziale ticker Ã¨ scritto completamente in maiuscolo?
    #*************************************************************************************************
    if extractedTicker.isupper():
        file.loc[i,"Ticker_Uppercase"]="1"
    else:
        file.loc[i,"Ticker_Uppercase"]="0"
    #Update feature: Tutto il body Ã¨ scritto in maiuscolo?
    #*************************************************************************************************
    if body.isupper():
        file.loc[i,"Uppercase_Body"]="1"
    else:
        file.loc[i,"Uppercase_Body"]="0"
    #Update feature: La prima lettera del potenziale Ticker Ã¨ maiuscola?
    #*************************************************************************************************
    if extractedTicker[0].isupper():
        file.loc[i,"First_is_Uppercase"]="1"
    else:
        file.loc[i,"First_is_Uppercase"]="0"
```

Figura 9: Ciclo scorrimento record e calcolo delle prime features

```
#Update feature: Conteggio di lettere maiuscole e minuscole e calcolo del rateo nella frase
#*************************************************************************************************
lowerCases=sum(map(str.islower, body))
upperCases=sum(1 for c in body if c.isupper())
file.loc[i,"upper_ratio_total"]=upperCases/(upperCases+lowerCases)
#Update feature: Conteggio di lettere maiuscole e minuscole e calcolo del rateo nel presunto ticker
#*************************************************************************************************
lowerCases=sum(map(str.islower, extractedTicker))
upperCases=sum(1 for c in extractedTicker if c.isupper())
file.loc[i,"upper_ratio_ticker"]=upperCases/(stop-start)
#Update feature: La frase Ã¨ costituita solo dal ticker?
#*************************************************************************************************
if len(extractedTicker)==len(body):
    file.loc[i,"ticker_only"]="1"
else:
    file.loc[i,"ticker_only"]="0"
#Update feature: Il carattere precedente il ticker Ã¨ un '$'?
#*************************************************************************************************
if body[start-1]=='$':
    file.loc[i,"char_prec_$"]="1"
else:
    file.loc[i,"char_prec_$"]="0"
```

Figura 10

```
    #Update feature: Name Entity Recognition e confidence di ciÃ² che Ã¨ stato individuato tramite flair
    #*************************************************************************************************
    sentence = Sentence(body)
    tagger.predict(sentence)
    #Inizializzo il valore a none a priori, nel caso sovrascrivo
    file.loc[i,"name_entity_recognition"]="None"
    file.loc[i,"confidence_NER"]="None"
    #Indice j necessario a scorrere i labels individuati nella sentence
    j=0
    #Scorro tutti gli span individuati nella sentence
    for span in sentence.get_spans('ner'):
        #Se trovo uno span che ha come parte del testo individuata il ticker scrivo nella feature come viene individuato
        #e con quale confidenza
        if span.text==extractedTicker:
            file.loc[i,"name_entity_recognition"]=span.labels[j].value
            file.loc[i,"confidence_NER"]=span.labels[j].score
        j+1
```

Figura 11: Codice Name-Entity Recognition

```
    #Update feature: Part-of-Speech tagging
    #*************************************************************************************************
    sentence = Sentence(body)
    posTagger.predict(sentence)
    file.loc[i,"part_of_speech"]="None"
    file.loc[i,"confidence_POS"]="None"
    #Indice j necessario a scorrere gli spans individuati nella sentence
    j=0
    #Scorro tutti gli span individuati nella sentence
    for span in sentence.get_spans('pos'):
        if span.text==extractedTicker:
            file.loc[i,"part_of_speech"]=span.labels[j].value
            file.loc[i,"confidence_POS"]=span.labels[j].score
        j+1
    #Update feature: Il body contiene le parole 'dip' 'put' 'call'
    #*************************************************************************************************
    if ((body.lower().find('dip')!=-1) or (body.lower().find('call')!=-1) or (body.lower().find('put')!=-1)):
        file.loc[i,"keyword_found"]="1"
    else:
        file.loc[i,"keyword_found"]="0"
#Salvataggio risultato da dataframe a file csv
file.to_csv("Ann_With_FeatureEngineering.csv")
```

Figura 12: Part-of-Speech tagging e salvataggio

Import e Preprocessing

```
import pandas as pd
import numpy as np
import scikitplot as skplt
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.metrics import classification_report
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
#Preparo il dataframe
from sklearn.metrics import confusion_matrix
dataFrame=pd.read_csv('Ann_With_FeatureEngineering.csv')
#Aggiusto i valori mancanti (None) nella colonna Confidence_NER
dataFrame['confidence_NER'].replace('None', 0, inplace=True)
dataFrame['confidence_POS'].replace('None', 0, inplace=True)
#************************************
#Sezione pre-processing
#************************************
x_set=dataFrame.iloc[:,9:21]
y_set=dataFrame['is_ticker']
#One-hot-encoding
ct = ColumnTransformer(transformers=[('encoders',OneHotEncoder(),[5,10])], remainder='passthrough')
x_set = np.array(ct.fit_transform(x_set))
#Split del data-set nelle sezioni di train e test
x_trainSet,x_testSet,y_trainSet,y_testSet=train_test_split(x_set,y_set,test_size=0.2,stratify=y_set,random_state=17)
```

Figura 13

Grid-Search Cross-Validation

```
#**************************************
#Sezione Grid-search Cross-Validation
#**************************************
rfc=RandomForestClassifier(random_state=17)
# Significato parametri:
# n_estimators: E' il numero di alberi che costituiranno la foresta
# max_features: E' il numero di feature assegnate ad ogni albero
# max_depth: E' l'altezza massima a cui un albero si può sviluppare
# criterion: E' la funzione che misura la qualità di uno split
param_grid = {
    'n_estimators': [130,150,170,180,190,200],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth' : [8,9,10,11],
    'criterion' :['gini', 'entropy']
}
grid_search=GridSearchCV(rfc,param_grid,cv=4,error_score='raise',scoring='f1')
```

Figura 14: Grid-Search CV per Random Forest

```
#************************************
#Sezione Grid-search e Cross-Validation
#************************************
gbc = GradientBoostingClassifier(random_state=17)
# Significato parametri:
# loss: Rappresenta la funzione di loss scelta per valutare la modellazione del dataset eseguito dall'algoritmo
# learning_rate: Tasso di riduzione del contributo fornito da ogni albero decisionale
# n_estimators: Rappresenta il numero di volte che la soluzione potrà essere corretta da un altro albero decisional
# criterion: E' la funzione che misura la qualità di uno split di features
# max_depth:
param_grid = {
    'loss' : ['deviance','exponential'],
    'learning_rate' : [0.04,0.06,0.08],
    'n_estimators' : [80,90,100],
    'criterion' : ['friedman_mse', 'squared_error'],
    'max_depth' : [4,5,6]
}
grid_search=GridSearchCV(gbc,param_grid,cv=4,error_score='raise',scoring='f1')
```

Figura 15: Grid-Search CV per GBDT

Addestramento

```
#Addestro il modello sul training-set
grid_search.fit(x_trainSet,y_trainSet)
#Mostro il valore degli iperparametri per cui voglio mostrare le performance  del k-esimo fold
print(grid_search.best_params_)
#Eseguo il test del modello trovato su dei record che non ha ancora visto
predictions = grid_search.best_estimator_.predict(x_testSet)
#Salvo i risultati della prediction
score=classification_report(y_testSet, predictions)
print(score)
conf_mat = confusion_matrix(y_testSet, predictions)
print(conf_mat)
import seaborn
seaborn.heatmap(conf_mat,annot=True)
plt.show()
```

Figura 16: Stampa iperparametri, statistiche e matrice di confusione

Feature importance

```
importance = grid_search.best_estimator_.feature_importances_
for i,v in enumerate(importance):
    print('Feature: %0d, Score: %.5f' % (i,v))
plt.bar([x for x in range(len(importance))], importance)
plt.show()
```

Figura 17: Stampa valori features importance e grafico

Dummy

```
#*************************************
#Allocazione DummyClassifier
#*************************************
dc=DummyClassifier(random_state=17, strategy='stratify')
#Addestro il modello sul training-set
dc.fit(x_trainSet,y_trainSet)
#Eseguo il test del modello trovato su dei record che non ha ancora visto
predictions = dc.predict(x_testSet)
#Mostro i risultati della prediction
score=classification_report(y_testSet, predictions, zero_division=0)
print(score)
```

Figura 18:Dummy con strategia "stratified"

```
#*************************************
#Allocazione DummyClassifier
#*************************************
dc=DummyClassifier(random_state=17, strategy='constant',constant=1.0)
#Addestro il modello sul training-set
dc.fit(x_trainSet,y_trainSet)
#Eseguo il test del modello trovato su dei record che non ha ancora visto
predictions = dc.predict(x_testSet)
#Mostro i risultati della prediction
score=classification_report(y_testSet, predictions, zero_division=0)
print(score)
```

Figura 19: Dummy con strategia "constant"

Bibliografia

1. Dietterich, T. G., & Kong, E. B. (1995). *Machine learning bias, statistical bias, and statistical variance of decision tree algorithms* (pp. 0-13). Technical report, Department of Computer Science, Oregon State University.

2. Feurer, M., Klein, A., Eggensperger, K., Springenberg, J. T., Blum, M., & Hutter, F. (2019). Auto-sklearn: efficient and robust automated machine learning. In *Automated Machine Learning* (pp. 113-134). Springer, Cham.

3. Hong, G. (2010). Marginal mean weighting through stratification: Adjustment for selection bias in multilevel data. *Journal of Educational and Behavioral Statistics*, *35*(5), 499-531.

4. Bardenet, R., Brendel, M., Kégl, B., & Sebag, M. (2013, May). Collaborative hyperparameter tuning. In *International conference on machine learning* (pp. 199-207). PMLR.

5. Lipton, Z. C., Elkan, C., & Naryanaswamy, B. (2014, September). Optimal thresholding of classifiers to maximize F1 measure. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases* (pp. 225-239). Springer, Berlin, Heidelberg.

6. Di Muzio, T. (2021). GameStop Capitalism. Wall Street vs. The Reddit Rally (Part I).

7. Chohan, U. W. (2021). Counter-hegemonic finance: The gamestop short squeeze. *Available at SSRN*

8. Anderson, K. E. (2015). Ask me anything: what is Reddit?. *Library Hi Tech News*.

Sitografia

1. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html, consultato il 15/10/2021
2. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html?highlight=gradientboostingclassifier#sklearn.ensemble.GradientBoostingClassifier, consultato il 19/10/2021
3. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html?highlight=gridsearch#sklearn.model_selection.GridSearchCV, consultato il 15/10/2021
4. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html, consultato il 20/10/2021
5. https://scikit-learn.org/stable/modules/generated/sklearn.dummy.DummyClassifier.html, consultato il 22/10/2021

Ringraziamenti

Ringrazio tutti coloro che mi hanno supportato durante questo percorso, che spesso si è rivelato tortuoso.

Il Professor Avvenuti e il Dottor Leonardo Nizzoli per avermi seguito durante le varie fasi e per la loro disponibilità e gentilezza.

I miei genitori per la vicinanza che mi hanno dimostrato nei periodi più difficili, per non aver mai smesso di credere in me e per tutti i sacrifici fatti per permettermi di portare avanti questa mia passione.

Mio fratello Matteo e mia sorella Elisa che, prendendomi in giro e consolandomi quando era necessario, non hanno mai smesso di incoraggiarmi a modo loro.

Cristina e Andrea, per non aver mai smesso di credere nelle mie potenzialità anche quando io pensavo di non averne.

I miei compagni di corso: Emanuele, Tommaso e Fabrizio per essere riusciti a rendere divertenti anche le ore più noiose, per tutto il supporto morale e per l'amicizia sincera che mi accompagnerà per sempre.

Agli amici, sia quelli di vecchia che di nuova data, per avermi supportato e sopportato, vi ringrazio in nome di tutte le birre, le uscite e le risate fatte insieme.

Aurora, per essermi stata vicino ogni singolo giorno, farmi stare bene e per avermi insegnato che l'impegno viene sempre ripagato, stimolandomi a dare sempre di più. Grazie soprattutto per il supporto durante la stesura di questa tesi, so di essere stato distratto e particolarmente noioso, ma non mi hai mai abbandonato. Se oggi sono qua è in larga parte merito tuo, anche se non lo ammetterai mai. Sei unica, ti amo.

E infine ai miei nonni, sia quelli presenti che quelli che non ci sono più. Vi ho sempre pensato durante questo percorso e la voglia di rendervi orgogliosi, come spero di riuscire a fare, è stata una delle maggiori spinte a impegnarmi. Grazie perché questo traguardo è anche merito vostro.