POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale

Text mining e sentiment analysis in ambito sanitario: un approccio community-based

Relatore

Prof. Silvia Anna CHIUSANO

Candidato

DAVIDE MARINO

Anno Accademico 2019/2020

Sommario

Questo elaborato descrive come viene eseguita una sentiment analysis sulle recensioni in ambito medicale, per poi non solo confrontare i risultati ottenuti tramite diverse metodologie di machine learning e modelli di word embeddings, ma anche per comprendere il sentimento espresso e il confronto tra le diverse culture. Inoltre, viene anche presentato un confronto tra i classificatori per la lingua italiana che per quella inglese.

Per far sì che questa classificazione avvenga, è stato necessario reperire il dataset in lingua italiana scrivendo un programma Go e partecipare alla Yelp Challenge 2019. Il dataset ottenuto è stato sufficiente per ottenere un buon classificatore con delle buone metriche prestazionali ed errori contenuti.

Prima della trasformazione del dataset, è stato necessario rimuovere le parole che non avessero a che fare con una valutazione di una prestazione medicale o di un giudizio in generale. Questo processo come si vedrà in seguito ha permesso un miglioramento del modello prodotto.

Una volta estratti i dati, ci si è soffermati quali classi d'appartenenza assegnare ad una recensione in base al punteggio medio che viene associato. Come si è notato per ogni configurazione che è stata scelta di analizzare, il dataset è sempre stato sbilanciato, perciò si è sempre dovuto ricorrere ad un oversampling come tecnica per contenere questo problema data anche la cardinalità del dataset. Si è anche notato con lo sbilanciamento del dataset che è presente un survivorship bias, cioè che le persone tendono a scrivere recensioni molto positive per elogiare un reparto e/o un dottore oppure decisamente negative per evidenziare delle lacune o superficialità di una prestazione sanitaria.

Dopo aver tutto pronto per la creazione del classificatore per la sentiment analysis, si è voluto confrontare non solo i modelli prodotti in lingua italiana e inglese, ma anche i word embeddings ottenuti tramite tecniche di supervised e unsupervised learning, evidenziando le diverse criticità, la complessità algoritmica e la scalabilità come soluzione in base alle metriche ottenute. Nello studio sono state adottate diverse tecniche di apprendimento supervisionato per la costruzione di un classificatore: ognuna di queste sarà presentata con le relative metriche prestazionali.

Applicando diverse tecniche di supervised learning, ci si è soffermati anche sulla scalabilità che hanno per il problema che viene trattato e si conclude che con modelli di word embeddings non supervisionati si potrebbero ottenere metriche prestazionali che sono in grado di generare modelli molto accurati, precisi e generalizzabili a seconda del contesto, dato che le parole che vengono espresse per un sentimento sono pressoché molto simili tra loro, anche in diverse culture.

Come conclusioni vengono presentate delle tecniche alternative a quelle adottate per la costruzione di un classificatore e viene suggerita un'integrazione con altri dataset per aumentare il numero di sentimenti che possono essere compresi. Sempre come miglioria possibile, l'applicazione di un modello semantico basato su un meccanismo di tagging potrebbe migliorare l'apprendimento statistico di una rete neurale in modo da generalizzare il più possibile la casistica e non dipendere dal contesto di provenienza.

Ringraziamenti

Alla mia famiglia

"Meglio aggiungere vita ai giorni che non giorni alla vita"
Rita Levi Montalcini

Tabella dei Contenuti

Elenco delle tabelle

Elenco delle figure

Acronyms

AI

Artificial Intelligence

ADASYN

Adaptive Synthetic Sampling Approach for Imbalanced Learning

AOT

Ahead Of Time

CBOW

Continuoud Bag Of Words

GloVe

Global Vectors for Word Representation

HPC

High Performance Computing

K-NN

K-Nearest Neighbors

NLP

Natural Language Processing

NER

Name Entity Recognition

CNN

Convolutional Neural Network

RNN

Recurrent Neural Network

FFNN

Feed Forward Neural Network

LSTM

Long Short Term Memory

TF-IDF

Term Frequency–Inverse Document Frequency

Introduzione

Oggi viviamo in un'epoca dove internet è entrato nelle nostre case, si è consolidato in questi ultimi 10 anni e l'accesso all'informazione è cresciuto a dismisura, tanto da rendere sempre più complicata la sua elaborazione. Generiamo più dati di quanti ne consumiamo: per esempio secondo Statista, vengono caricati circa 500 ore di video ogni minuto nel 2019, mentre solo 20 nel 2009[1]. I social media che stanno spopolando in questo momento hanno raggiunto l'apice di popolarità, trasmettendo contenuti multimediali, notizie (vere o false), opinioni e molto altro.

In questo contesto, il data mining è una procedura molto comune data la mole di dati presente nel web. In questo studio ci si è voluti focalizzare nell'ambito sanitario e come determinate tecniche possono dare una rappresentazione della qualità ospedaliera e della penetrabilità nel territorio di sua competenza.

L'idea è stata di sviluppare uno strumento in grado di comprendere e analizzare delle recensioni scritte da utenti per la creazione di un modello matematico statistico in grado di riconoscere il sentimento espresso.

Nello svolgimento dello studio sono descritte le fasi per ottenere un modello semantico in grado di comprendere il sentimento espresso in una recensione. Il dataset è stato ottenuto costruendo un programma che scaricasse le recensioni in lingua italiana da QSalute.it, una piattaforma web per recensire gli ospedali di tutta Italia. Un secondo dataset è stato reperito da una challenge del 2019 sponsorizzata da Yelp, una nota piattaforma web per recensire delle attività commerciali, tra cui cliniche ospedaliere.

Lo scopo finale è stato non solo l'analisi e la comprensione di un sentiment espresso in un testo, ma anche il confronto tra le diverse tecniche adottate con le relative metriche prestazionali e tra le diverse lingue a livello culturale (le parole utilizzate, quelle più comuni, e così via).

Come conclusione di questo studio, ci si è focalizzati sull'importanza del progesso tecnologico al servizio delle utenze per migliorare il servizio sanitario, sull'importanza di un dataset corposo e qualitativamente valido e sui possibili sviluppi futuri per migliorare i modelli matematici per sentiment analysis.

Capitolo 1

QSalute e Yelp

1.1 Introduzione

In questo primo capitolo, verranno introdotte le 2 piattaforme web dove è stato possibile reperire i dataset per lo svolgimento di questo studio di tesi e come sono stati elaborati i dati prima dell'analisi del sentimento. I programmi che hanno aiutato a condurre lo studio sono stati scritti tramite i linguaggi Python e Golang, i quali mi hanno permesso di sfruttare le accelerazioni hardware garantendo tempi contenuti di training e pulizia dei dataset. Come approssimazione da qui in avanti, verranno catalogate le recensioni nei seguenti modi:

- positive con punteggi pari superiori al 4 su 5, mentre negative con punteggi pari o inferiori al 3 su 5, scartando le recensioni rimanenti
- positive con punteggi superiori al 3 su 5 e negative viceversa
- positive con punteggi pari o superiorei al 3.6 su 5, neutrali con punteggi tra il 3.6 su 5 e il 2.4 su 5 incluso, negative le recensioni rimanenti

I linguaggi utilizzati non verrano approfonditi, ma verranno indicati i ragionamenti e le assunzioni fatte con allegati i risultati ottenuti.

1.1.1 QSalute

QSalute è nata nel 2008 come piattaforma web per recensire la qualità del servizio sanitario in tutte le zone d'Italia. Sono presenti dalle strutture locali, come ASL, a quelle accreditate, in modo tale da avere il quadro completo non solo per specialistica, ma anche come offerta presente nel territorio.

QSalute è stato creato come strumento per aiutare il cittadino a scegliere e confrontare l'offerta sanitaria nei suoi d'intorni, ciononostante i manutentori della piattaforma specificano che una recensione non sostituisce l'interazione del paziente e dei suoi familiari con il personale medico.

Secondo le loro statistiche pubblicate nel loro sito web, hanno raccolto oltre 100000 contributi spontanei da parte dei pazienti con più di 10 anni di attività online[2]. Di questi contributi spontanei, non tutti sono stati pubblicati nella piattaforma, proprio perché alcuni non sono stati identificati come idonei per diversi motivi: questo è stato proprio riscontrato nella scrittura del programma che ha permesso il reperimento del dataset, il quale ha una cardinalità inferiore di quanto dichiarato nella loro piattaforma. È possibile reperire molte informazioni da quanto scrivono i contribuenti, dalle patologie più comuni a quelle più rare e come sono stati serviti dal personale medico durante il trattamento sanitario.

Molti medici di rilevanza nazionale hanno contribuito alla piattaforma per rendere il contenuto offerto il più oggettivo e affidabile possibile, per evitare che l'informazione sia completamente impulsiva da parte dei recensori e per evitare il plagio, molto comune oggi giorno nelle piattaforme web dove l'informazione non è mediata da arbitri super partes[2]. La struttura di una recensione non si basa solo su un testo scritto, ma anche da un meccanismo di scoring usando 4 criteri e la loro media:

- Competenza percepita
- Servizi presenti in struttura
- Pulizia della struttura
- Assistenza all'interno della struttura

Le recensioni raccolte non sono solamente di strutture ospedaliere, ma anche di centri riabilitativi, cliniche private e altri centri clinici a cui lavora personale di tipo medico. Ancora oggi la piattaforma cerca di mantenere il suo ruolo di arbitro superpartes per garantire l'affidabilità del giudizio di una struttura, dato che il suo impatto potrebbe condizionare in parte il sistema sanitario nazionale. I manutentori di QSalute sono ancora oggi disponibili per collaborazioni con professionisti medici e non e mettono a disposizione strumenti per la segnalazione di recensioni non veritiere o offensive[2].

Per semplicità in questo studio, è stato utilizzato solamente il punteggio medio, in modo tale da avere un miglior grado di confronto tra i 2 dataset e utilizzare una metrica comune per la sentiment analysis. Come lavoro futuro potrebbe essere proposto l'utilizzo delle altre metriche e/o di una loro correlazione.

1.1.2 Yelp

Yelp è una piattaforma web nata a San Francisco nel 2004 che si occupa di recensire ogni tipologia di esercizio commerciale tramite geolocalizzazione e un'applicazione mobile [3]. Nel 2019 Yelp ha lanciato una challenge per condurre ricerche o analisi sui loro dati ed è possibile condividere i risultati scientifici con la compagnia [4].

La challenge si basa su più aspetti di ricerca in ambito d'intelligenza artificiale:

- Computer vision per classificazione di fotografie
- Natural language e sentiment analysis su recensioni
- Graph mining per analisi di pattern

Per motivi di tempistiche non è stato possibile pubblicare dei risultati tramite la piattaforma Yelp Challenge e nonostante ciò lo studio è stato ugualmente condotto in ambito sentiment analysis.

La compagnia presenta il suo fact sheet sempre aggiornato tramite una loro pagina web e secondo le utile statistiche aggiornate al quarto quadrimestre del 2019, i servizi sanitari recensiti costituiscono circa l'8% del totale delle recensioni [3].

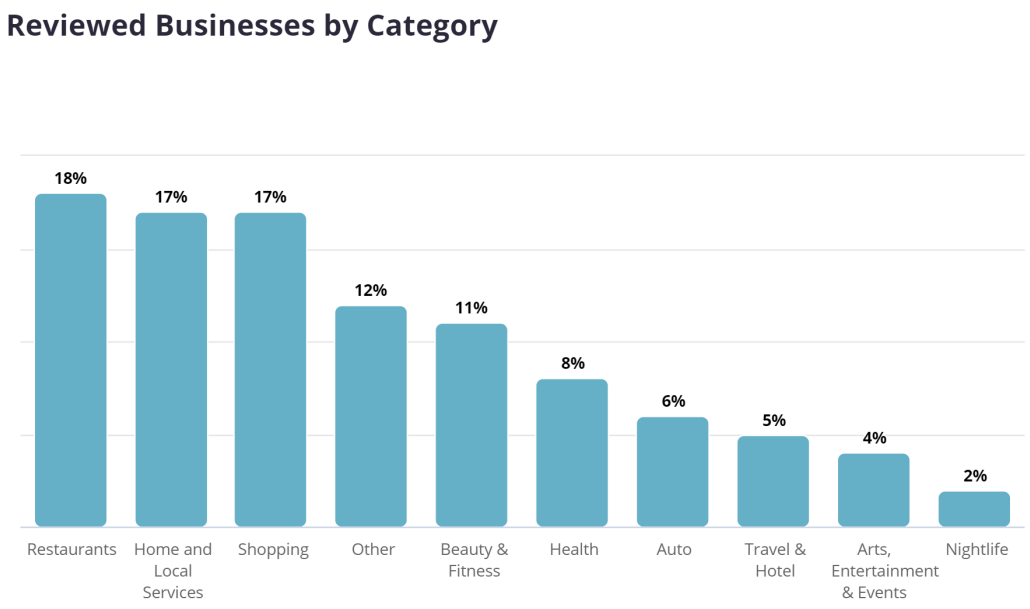

Figura 1.1: Divisione per categoria delle attività recensite su Yelp nel 2019

Nonostante sia una percentuale inferiore alle categorie predominanti, sono state

estratte più recensioni rispetto a quelle presenti su QSalute, ma con lo stesso ordine di grandezza; questo ha permesso di poter esporre e studiare un confronto tra l'espressione di sentimenti in recensioni tra diversi ambienti culturali.

1.2 Struttura del dataset

Per condurre uno studio di sentiment analysis, è necessario strutturare l'informazione in base a:

- Un meccanismo di scoring
- La recensione da analizzare

Per semplicità e come citato in precedenza, il sentimento è definito come solamente positivo o negativo, in modo da garantire la costruzione di un classificatore semplice e robuto per poi in lavori futuri migliorarlo ed espanderlo.

Partendo dal caso di QSalute, il reperimento del dataset non è stato intrapreso tramite un portale dove è possibile scaricarlo interamente, ma è stato necessario scrivere un programma Go per aprire ogni pagina web.

A partire dall'URL base, le strutture sanitarie sono catalogate nel loro sistema per regione amministrativa, quindi per ognuna delle 20 regioni italiane, è stata scaricata la pagina web e registrato l'hyperlink per ogni struttura sanitaria. Ognuna è descritta come:

- Recensioni della struttura
- Votazioni della struttura
- Reparti della struttura (se disponibili)

Per ogni struttura vengono salvate tutte le informazioni utili per l'analisi di sentimento e, se presenti, vengono scaricate le pagine web dei reparti di ogni struttura. Per ogni reparto, sono presenti:

- Recensioni del reparto di una struttura
- Votazioni del reparto di una struttura

Dato che in questo scenario siamo interessati al raiting e al testo della recensione, viene considerato non rilevante l'appartenenza di un reparto ad una struttura sanitaria, perciò il programma memorizzerà solamente il testo e il punteggio medio della recensione. Non sono stati memorizzati i punteggi relativi alla competenza, pulizia, assistenza e servizi, in quanto è stato utilizzato solamente il punteggio medio come anche per le recensioni all'interno del dataset di Yelp.

Nel portale della Yelp challenge, è disponibile scaricare il dataset per intero in formato JSON, il quale al suo interno possiede: [5]

- business.json
- review.json
- user.json
- checkin.json
- tip.json
- photo.json

Per questo studio di tesi, è sono stati tenuti solamente in considerazione business.json e review.json per filtrare le attività commerciali di ambito sanitario per poi reperire le loro recensioni. Il dataset possiede gli esercizi commerciali solamente nel territorio statunitense, perciò sono pur sempre inclusi gli ospedali accreditati e le cliniche di riabilitazione, come per il caso del dataset italiano.

Prima del filtraggio del dataset, sono state ottenute le categorie che sono presenti all'interno dell'intero JSON e infine sono state lasciate per lo studio solamente le categorie che contenessero le parole:

- "Health"
- "Hospital"
- "Medical"
- "Medicine"
- "Clinic"
- "Emergency"

1.3 Estrazione dei dati

Durante l'estrazione e pulizia dei 2 dataset, sono stati contati i record ottenuti e sono state generate le label positive e negative per ogni recensione. La convenzione adottata è stata di inserire 0 per una recensione negativa, 1 viceversa, così da trattare il problema in modo binario.

Le recensioni ottenute dal parser Go e dal filtraggio delle categorie di Yelp sono:

- 70324 recensioni in lingua italiana da QSalute
- 85285 recensioni in lingua inglese da Yelp

Una volta ottenuti i 2 dataset in modo comparabile, sono state filtrate le recensioni che non appartenevano né alla categoria delle recensioni positive, né a quella delle negative, in modo da rendere più binario il problema da trattare. Le cardinalità ottenute sono:

- 66407 recensioni in lingua italiana da QSalute
- 82007 recensioni in lingua inglese da Yelp

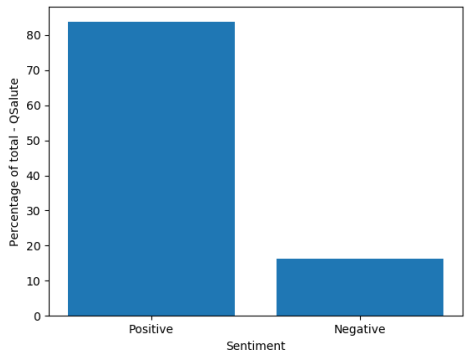

Figura 1.2: QSalute: 83.75% recensioni positive, 16,25% negative

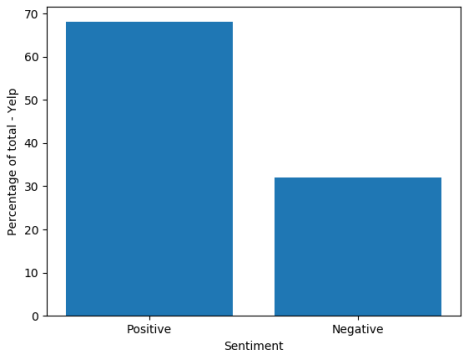

Figura 1.3: Yelp: 68,06% recensioni positive, 31,94% negative

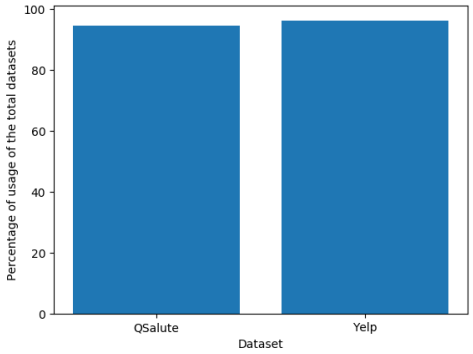

Figura 1.4: Confronto porzione dataset considerata per il training: 94.43% del dataset QSalute, 96.17% di quello di Yelp

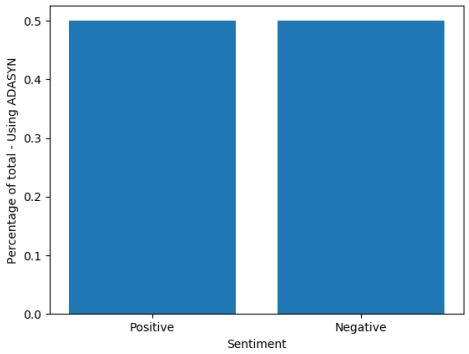

Figura 1.5: L'utilizzo di ADASYN permette il bilanciamento completo delle classi di appartenenza di un determinato dato

Da come si evince dai risultati ottenuti, si nota che c'è un problema di bilanciamento del dataset per entrambi i dataset, perciò, date le cardinalità in gioco, non è stato ritenuto di utilizzare tecniche di downsampling, per mantenere un grado di comparazione tra i 2 dataset dello stesso ordine di grandezza. Dato che lo sbilanciamento del dataset è molto marcato, si è deciso di adottare una tecnica di oversampling nota come Adaptive Synthetic Sampling Approach for

Imbalanced Learning, in breve ADASYN, che verrà affrontata nel dettaglio nel capitolo successivo.

Per quanto riguarda la porzione del dataset considerata, la netta maggioranza delle recensioni presenti vengono considerate, perciò si può affermare che i risultati non sono stati ottenuti con un subset minoritario e il modello creato non sia generalizzabile.

Una seconda metodologia che è stata adottata è di includere l'intero dataset è di definire le recensioni positive con un punteggio pari o superiore a 3 su 5 e le negative le rimanenti. Con questa metodologia, le percentuali di utilizzo del dataset sono pari al 100%, ma le percentuali di bilanciamento tra le recensioni positive e negative rimangono pressochè simili. Dato appunto questa distribuzione a livello di classe di appartenenza, sarà sempre necessario utilizzare una tecnica di oversampling per cercare di ovviare a questo problema.

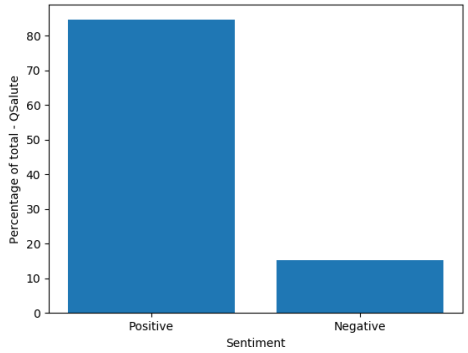

Figura 1.6: QSalute: 84,65% recensioni positive, 15,35% negative

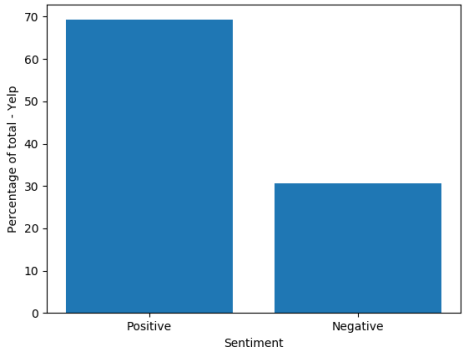

Figura 1.7: Yelp: 69.29% recensioni positive, 30,71% negative

La terza e ultima metodologia per utilizzare l'intero dataset è di definire 3 classi di appartenenza per la singola recensione basandosi sul voto ricevuto:

- Positiva
- Neutrale
- Negativa

Come citato nell'introduzione del capitolo, la suddivisione nelle classi di appartenenza è stata pressochè bilanciata a livello di rating, ma come per i casi precedenti, il dataset rimane sempre sbilanciato, specialmente per le recensioni dentro alla categoria "neutrale". Come per tutti i casi che verranno affrontati, è sempre necessario utilizzare la tecnica di oversampling di ADASYN per ovviare a questo problema di bilanciamento.

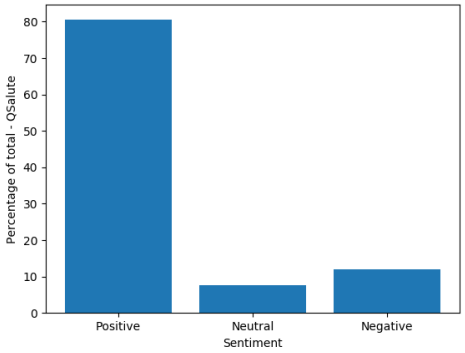

Figura 1.8: QSalute: 80,55 % recensioni positive, 7,54% neutrali e 11,91% negative

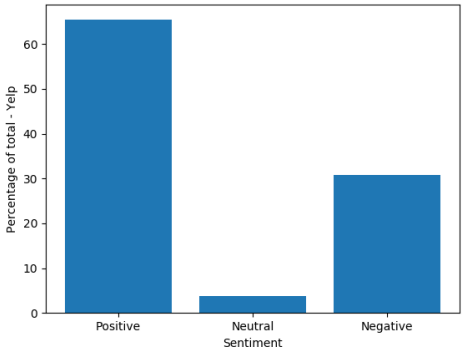

Figura 1.9: Yelp: 65,44% recensioni positive, 3,84% neutrali e 30,72% negative

1.3.1 Golang, Goroutines e Goquery

Come citato in precedenza, è stato utilizzato il linguaggio Go, detto anche Golang, per reperire il dataset in lingua italiana dalla piattaforma QSalute. Molti altri linguaggi potevano essere sfruttati per questo scopo, ma è stato deciso di non solo sfruttare le proprietà del linguaggio per il data retrieval, ma anche per il text mining e per text cleaning.

Go nasce nel 2009 come linguaggio compilato Ahead Of Time (AOT), fortemente tipizzato, orientato alla programmazione ad oggetti e al multithreading. In Go tutti i thread condividono una coda globale dove tutte le astrazione dei thread gestiti dal linguaggio, dette goroutine, sono immagazzinate per poi essere eseguite in como concorrente e asincrono. Golang non nasce come un linguaggio orientato nel modo dell'High Performance Computing (HPC), ma è in grado di nascondere la gestione della concorrenza dando degli strumenti al programmatore per scrivere codice sicuro e veloce[6].

Lo scehduler responsabile del sequenziamento è anche chiamata la goruntime, cioè uno strato di overhead presente all'avvio di ogni programma Go che gestisce quale gruppo di goroutine affidare a quale thread: si fa riferimento a gruppi di goroutine e di dati per ridurre il numero di context-switch. Go non è orientato all'utilizzo di risorse e alla gestione come C e C++, ma alla concorrenza; il linguaggio possiede un garbage collector che distrugge le strutture e le variabili istanziate proprio come Java o C operano a livello funzionale. Vengono chiamate strutture e non classi, proprio perché il concetto di classe come in altri linguaggi orientati agli oggetti non esiste[6]. Golang mette a disposizione delle interfacce che possono essere implementate come in Java ma generalizzabili come in C++.

Un altro aspetto del linguaggio è la facilità di importare codice già scritto e

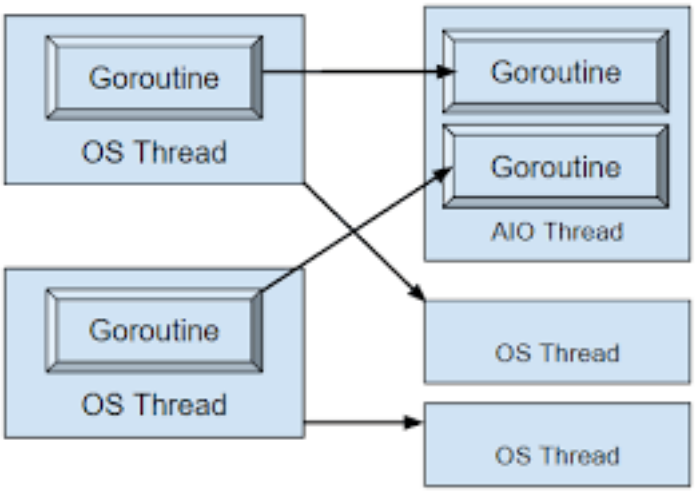

Figura 1.10: Confronto tra goroutine e OS thread[7]

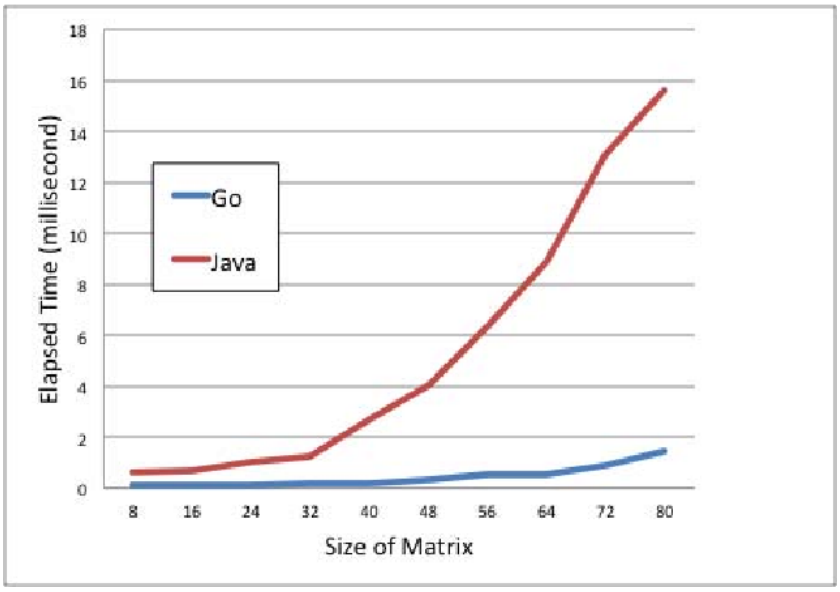

Figura 1.11: Confronto performance Java vs Go nel calcolo matriciale [8]

di esportare il proprio come fosse un package Java. Per questo studio di tesi, si è ricorso ad una dipendenza chiamata goquery. goquery porta una sintassi e un insieme di funzionalità simili a jQuery basandosi sul pacchetto net/html della standard library di Go e sulla libreria CSS Selector Cascadia. Poiché il parser net/html restituisce nodi e non un albero DOM completo, le funzioni di manipolazione stateful di jQuery sono state lasciate fuori[9].

1.4 Analisi qualitativa del dataset

Da quanto si può notare dai grafici precedentemente descritti, la cardinalità di entrambi i dataset ci permette di poter costruire un classificatore di recensioni e di confrontare le occorrenze delle parole tra una lingua e l'altra. Come scelta implementativa, è stato deciso di rimuovere le recensioni che non hanno un punteggio medio superiore a 4 su 5 e inferiore a 3 su 5, in modo tale da permettere l'apprendimento su delle classi ben definite e testare la soluzione ottenuta con l'interno dataset.

Figura 1.12: WordCloud parole positive QSalute

Figura 1.13: WordCloud parole negative QSalute

Figura 1.14: WordCloud parole positive Yelp

Figura 1.15: WordCloud parole negative Yelp

Da quanto si può vedere dalle figure sopra indicate, le parole hanno un significato simile sia nel caso che esprimano sentimenti positivi che negativi. La loro occorrenza e il peso che hanno per determinare quale sentimento è espresso in una recensione verrà affrontato in seguito.

Nella visione del dataset, si è riscontrato che molti testi hanno una lunghezza di gran lunga superiore alle 200 parole e che, nel caso italiano, spesso sono descritti i modus operandi dei dottori e della loro esperienza vissuta all'interno della struttura sanitaria. Per avere un confronto di questo lavoro con altri svolti per lo più utilizzando l'API di Twitter, è stato deciso di limitare la lunghezza delle recensioni, sia italiane che inglesi, a 140 parole, in modo tale da avvicinarsi ad un risultato che possa essere comparabile sia in termini di metriche che di procedimento d'analisi.

Review	Sentiment
struttura ottima sotto ogni aspetto ed immersa nel verde medici e personale tutto semplicemente eccellenti	positive
totalmente impreparati pessime le infermiere pediatriche ostetriche fredde e acide poca attenzione al post parto	negative
What an amazing doctor Dr. XXX is! He is so fast and does everything with swiftness. Would recommend him to all my friends and family	positive
Worst experience ever. Dr. XXX is the worst I've ever encountered, and their new uptown allergy shot clinic is a scourge	negative

Tabella 1.1: Esempi di recensioni considerate nel training

La tendenza di scrittura riscontrata nel dataset proveniente da QSalute ha permesso di verificare che quanto esplicitamente dichiarato dalla piattaforma web sia messo in pratica, proprio per evitare il plagio e garantire un'informazione qualitativamente valida.

Per la pulizia delle parole dai 2 dataset, si è deciso di rimuovere le parole non pertinenti all'espressione di un sentimento per agevolare l'apprendimento di un classificatore. Sono state rimosse:

- I numeri e le date
- I nomi di città, province, stati (Yelp) e regioni (QSalute)

- I digrammi e trigrammi che citano il personale (es. dott. Mario Rossi, prof. Bianca Verdi, etc...)
- Le patologie utilizzando un dizionario reperito dai siti internet dei centri di controllo e prevenzione[10][11]

La pulizia del dataset ha permesso di migliorare le metriche prestazionali del classificatore, ma questa tematica verrà riproposta e citata nel dettaglio nella sezione delle metriche nella sentiment analysis.

Capitolo 2

Classificazione e Normalizzazione

2.1 Introduzione all'analisi del linguaggio

In questo studio di tesi è stato applicata l'analisi del linguaggio per produrre un modello che predica se dato un testo viene espresso un sentimento positivo o negativo. Originariamente si sarebbe potuto produrre un classificatore di sentimenti positivi, neutri o negativi, ma dato lo scarso bilanciamento del dataset, ci si è limitati ad utilizzare solamente 2 classi di appartenenza.

Una delle tante applicazione dei metodi statistici di apprendimento è l'analisi del linguaggio, chiamato anche Natural Language Processing (NLP). L'idea consiste nell'analizzare le parole, la semantica, le occorrenze e molti altri parametri linguistici per ottenere un modello statistico applicabile ad un determinato contesto d'uso pratico.

Il concetto cardine del NLP risiede nella trasformazione del testo in matrici o tensori, perché le operazioni numeriche sono più veloci e scalabili rispetto che alla manipolazione delle stringhe testuali. Questo però introduce una complicazione: la scelta, il criterio e la sua giustificazione di un metodo di trasformazione del testo in formato numerico rispetto ad un altro.

In questo capitolo verranno affrontate delle tecniche di trasformazione e di analisi del testo, ma occorre risolvere il problema del mancato bilanciamento delle classi dei 2 dataset prima di svolgere qualsiasi operazione. Il mancato bilanciamento potrebbe portare alla creazione di un modello non applicabile in un caso generale o ad una performance molto scarsa.

2.2 Bilanciamento del dataset

Durante la scelta di un metodo per bilanciare il dataset prima del training, è stata fatta della ricerca sullo stato dell'arte ed è stato necessario compiere una scelta tra 2 tecniche maggiormente utilizzate:

- Oversampling
- Undersampling

Nella prima la classe con il minor numero di campioni viene ricampionata in modo randomico, mentre nel secondo caso vengono rimossi dei campioni casualmente dalla classe con la cardinalità maggiore. Lo scopo di entrambe le tecniche è quello di ottenere un dataset bilanciato, in modo da limitare (se non eliminare) il problema del bias durante il processo di apprendimento statistico.

Entrambe le tipologie di campionamento implicano che dei campioni presi da una classe vengano considerati maggiormente rispetto ad un'altra. Solitamente i casi che si affrontano sono i seguenti:

- sono presenti molti campioni di una o più classi
- sono presenti pochi campioni di una o più classi

Il bilanciamento di un dataset è una pratica abbastanza comune proprio per via della sparsità dei dati che si possono reperire, specialmente online, e per evitare i casi di overfitting. Questo implica che un modello risultante potrebbe essere molto valido per predirre una determinata classe di appartenenza, mentre non è facilmente generalizzabile a più classi o contesti di applicazione.

Solitamente per combattere il mancato bilanciamento del dataset si potrebbe:

- cercare nuovi dataset (soluzione più semplice ed efficace, ma non sempre possibile)
- utilizzare delle metriche di performance (es. ROC, F1, recall, etc..)
- provare a ricampionare il dataset
- generare dati sintetici
- nuova chiave di lettura del dataset (es. utilizzare un'anomalia a proprio vantaggio, non sempre realizzabile)

Come scelta di approccio per risolvere il mancato bilanciamento del dataset, si è deciso per questo studio di tesi di adottare una tecnica di oversampling che comporti il ricampionare una porzione del dataset in modo randomico per ridurre la probabilità in incorrere nel caso del overfitting.

La scelta di un metodo di oversampling è ricaduta sulla tecnica che prende il nome di Adaptive Synthetic Sampling Approach for Imbalanced Learning, detta anche ADASYN. L'idea dietro all'ADASYN sta nel ricampionare il dataset tramite l'uso di una distribuzione pesata per una moltitudine di classi minoritarie a seconda della loro difficolta di nella fase di apprendimento[12]. Lo scopo d'uso di questa tecnica consiste nel:

- ridurre il bias introdotto dalle classi maggioritarie
- cercare di spostare il confine nel prendere una decisione di appartenenza ad una classe o ad un'altra

Dato un training set con m campioni, vengono definite m_s e m_l come il numero delle classi minoritarie e maggioritarie rispettivamente. Per definizione $m_s < m_l$ e $m_s + m_l = m$.

Per il calcolo del grado di bilanciamento:

$$\frac{m_s}{m_l} \tag{2.1}$$

dove $d \in (0, 1]$. Se $d < d_{th}$, dove d_{th} è il rapporto del massimo grado di bilanciamento allora[12]:

1. calcolare il numero massimo di campioni sintetici da generare per la classe minoritaria
$$G = (m_l - m_s) \times \beta \tag{2.2}$$
dove $\beta \in [0, 1]$ è un parametro utilizzato per specificare il livello di bilanciamento dei dati sintetici. Se $\beta = 1$ allora significa che il dataset è perfettamente bilanciato dopo il processo di generalizzazione.

2. per ogni esempio della classe minoritaria, viene trovato il K-NN basato sulla distanza euclidea e viene calcolato il rapporto r_i definito come
$$r_i = \frac{\triangle_i}{K} \tag{2.3}$$
dove $i = 1, ..., m_s$ e $\triangle_i$ è il numero di campioni vicini all'esempio considerato che appartiene alla classe maggioritaria. Per definizione $r_i \in [0, 1]$

3. normalizzare r_i tramite
$$r'_i = \frac{r_i}{\sum_{i=1}^{m_s} r_i} \tag{2.4}$$
è una densità di distribuzione, cioè la sommatoria di tutte le componenti risulta 1

4. calcolare il numero dei campioni sintetici che hanno bisogno di essere generati per ogni campione di minoranza

$$g_{\mathrm{i}} = r'_{\mathrm{i}} \times G$$

dove G è il totale dei dati sintetici che devono essere generati come definiti in (2.2)

5. per ogni classe minoritaria, viene generato g_{i}

Come gli autori evincono nella loro ricerca, l'algoritmo di ADASYN può migliorare le perfomance del classificatore riducendo il bias introdotto dal dataset original non bilanciato. Viene anche dimostrato dagli autori che in diversi casi c'è una riduzione dell'errore proporzionale al bilanciamento svolto dall'ADASYN.

2.3 Lemming e stemming

Secondo l'Enciclopedia Treccani, la flessione è l'ambito della morfologia che riguarda le diverse forme che una parola può avere a seconda del contesto in cui è usata[13]. La forma flessa di una parola è l'intero testo compreso il significato semantico che possiede e il contesto in cui è utilizzata. A livello morfologico, il tema di una parola è la parte di una parola che resta dopo aver tolto la desinenza[14]. Un esempio per distringuere il lemma dal tema potrebbe essere il seguente:

1. bisc è il tema di biscotto
2. study è il tema di studying
3. pensa è il lemma di pensare
4. good è il lemma di better

L'operazione di stemming è una tecnica che riduce le parole alla sua radice rimuovendo prefissi e suffissi. Gli stemmer non possono descrivere o distinguere una parola a seconda della sua morfologia, ma sono estremamente utili per il text cleaning[15]. Un esempio di criticità nello stemming potrebbe essere:

1. bisc è il tema sia di biscia che di biscotto
2. asto è il tema sia di astomia che di astone (pianta da vivaio)

Morfologicamente le parole presentate negli esempi qui sopra, sono molto distanti, perciò un modello semantico che calcola la distanza euclidea multidimensionale tra loro dovrebbe avere come risultante un valore molto alto. Sono presenti molti stemmer, ma si è deciso di utilizzare quello di Martin Porter[16] per il dataset

in lingua inglese. La scelta è stata influenzata anche dall'utilizzo della librearia Python NLTK, la quale prevede una sua implementazione già presente e funzionante, in modo tale da rendere lo sviluppo più ad alto livello e non badare ai dettagli implementativi. La stessa libreria prevede anche l'uso di un altro stemmer chiamato SnowballStemmer, il quale possiede un'implementazione per la lingua italiana[17].

L'operazione di lemming è una tecnica più complicata dello stemming, perché cerca di andare verso la radice della parola e utilizzare anche il suo significato morfologico. Per il compimento del lemming, è necessario conoscere la semantica e richiede una conoscenza più approfondita del contesto di applicazione.

Per facilità d'uso e per sviluppo, è stato utilizzato solamente uno stemmer e non un lemmer in quanto viene approssimato il processo di data cleaning e in questo studio di tesi non viene approfondita la conoscenza della morfologia di un insieme di parole. Lo scopo è di produrre un modello in grado di predirre il sentimento espresso, il quale è poco condizionato dal contesto in cui una parola viene utilizzata dato che il classificatore utilizza solamente 2 classi (positivo o negativo). Come lavoro futuro, un approfondimento sull'analisi della semantica e della morfologia potrebbe portare ad un'accuratezza migliore dello studio e a miglioramenti delle metriche prestazionali.

2.4 Tecniche di vettorizzazione

Come già citato in precedenza, lavorare con delle stringhe è più dispendioso a livello computazionale, è più difficile scalare con la dimensione del dataset ed è difficile generalizzare l'utilizzo del modello risultate a più aree d'applicazione. L'analisi del testo è un campo di applicazione nota per gli algoritmi di apprendimento statistico, tuttavia, i dataset non possono essere analizzati direttamente agli algoritmi stessi, poiché i documenti di testo possiedono una lunghezza variabile e il grado di qualità dei dati grezzi non può essere quantificato a seconda di una metrica ben definita.

Una tecnica che permette la trasformazione di un testo in una rappresentazione numerica è il processo di vettorizzazione, il quale varia a seconda dell'algoritmo scelto. In questa sezione vengono presentati 2 metodologie che sono state adottate e messe a confronto per la costruzione del classificatore. Gli algoritmi presentati sono:

1. Count Vectorizer
2. TF-IDF Vectorizer

Il primo consiste nella trasformazione di un testo in una matrice di contatori, mentre il secondo di frequenza diretta e inversa di una parola sempre in forma matriciale. La libreria Python che è stata utilizzata si chiama SKLearn, la quale

ha permesso lo sviluppo ad alto livello e più rapido, in modo da focalizzarsi sull'algoritmo generale per la creazione del classificatore statistico di sentimento[18].

2.4.1 Count Vectorizer

L'implementazione del CountVecotrizer in SKLearn produce una rappresentazione tramite una matrice sparsa. Il numero di righe della matrice corrisponde al numero di recensioni presenti nel corpo del testo e le colonne sono tante quanto le parole uniche presenti all'interno del dataset. Per ogni parola trovata all'interno di una recensione, viene incrementato il suo valore alla riga e colonna corrispondente, fino ad ottenere la matrice sparsa completa che rappresenta il testo considerato[19].

Il vantaggio di utilizzare la rappresentazione a matrice sparsa sta nelle operazioni di somma e prodotto che risultano più semplici e la divisione in righe. Le operazioni da svolgere variano a seconda dell'algoritmo utilizzato per generare il classificatore, ma in generale le operazioni più comuni da svolgere sono ottimizzate tramite questa scelta di rappresentazione matriciale[19].

La funzione di CountVectorizer possiede come parametri la scelta di analisi delle parole singolarmente o raccolte in gruppi chiamati grammi. La scelta in questo contesto è di utilizzare (1,3)-grams, in altre parole:

1. i grammi singolarmente, quindi la singola parola come singola occorrenza
2. i digrammi, quindi coppie di parole ripetute
3. i trigrammi, quindi triplette di parole ripetute nel testo

La scelta è stata giustificata dal fatto che molte espressioni possono essere composte da un gruppo di parole piuttosto che dalla parola singola, in modo anche da compensare alla mancata lemmatizzazione del testo. Non si è voluto utilizzare grammi composti da 4 o più parole, in quanto poteva essere poco scalabile come analisi semantica e le metriche prestazionali non ne avrebbero risentito del cambiamento di strategia di analisi del testo.

2.4.2 TF-IDF Vectorizer

Un altro metodo per trasformare il testo in una rappresentazione matriciale è l'uso del TfidfVectorizer, messo a disposizione sepmre dalla libreria Python SKLearn[18]. Prima del suo uso pratico, verrà illustrato il procedimento dietro all'algoritmo dell'analisi della frequenza diretta e inversa dato un corpo di testo.

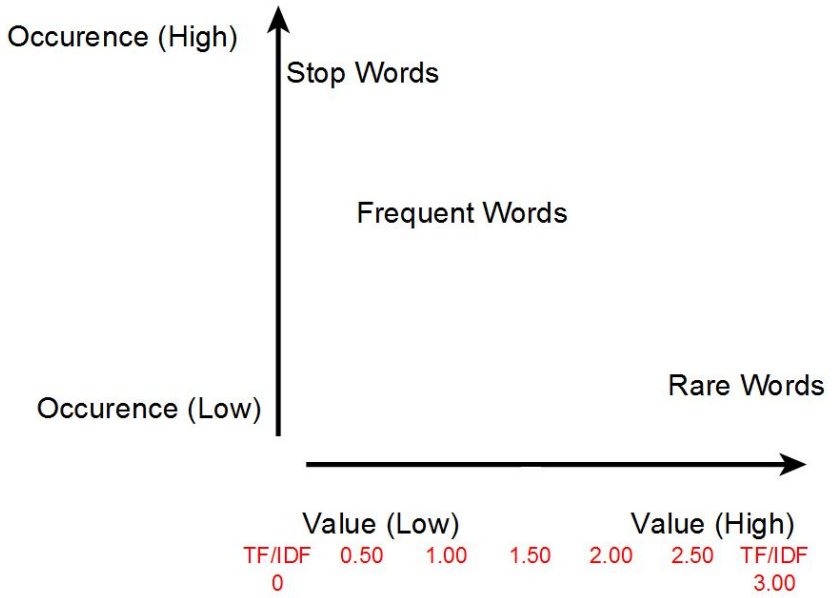

Figura 2.1: Interpretazione dei valori prodotti dal TF-IDF Vectorizer[20]

TF-IDF TF-IDF (Term Frequency–Inverse Document Frequency) è una funzione di descrizione di un peso utilizzata nella raccolta di informazioni in modo tale da misurare l'importanza di una parola o un gruppo di parole rispetto ad un altro facendo riferimento ad una collezione di documenti. In questo contesto, la collezione di documenti sono l'insieme delle nostre recensioni da analizzare, mentre il singolo documento corrisponde alla singola recensione[21].

Utilizzando questo algoritmo di vettorizzazione, una parola assume importanza all'interno del documento in maniera proporzionale al numero di occorrenze presenti e inversamente proporzionale alla sua frequenza all'interno dell'intera collezione di recensioni. Questo implica in altre parole che una parola ha un peso maggiore se utilizzata all'interno di una recensione solamente piuttosto che in tutte quelle considerate[21].

Con l'uso di espressioni matematiche, possiamo descrivere il funzionamento della TF-IDF scomponendo la parte di term frequency con la inverse document frequency per poi combinarle per la risoluzione del problema. Dato un termine i-esimo in un document j-esimo possiamo definire la frequenza di un termine come:

$$tf_{i,j} = \frac{n_{i,j}}{|d_j|} \tag{2.5}$$

dove $n_{i,j}$ è il numero di occorrenze di un termine all'interno di una recensione con d_j parole totali[21]. Dato il numero di documenti in una collezione (in questo

caso il numero di recensioni nell'intero dataset), definiamo l'importanza generale di un termine come:

$$idf_{\mathrm{i}} = \log \frac{|D|}{|\{d : i \in d\}|} \tag{2.6}$$

dove D è il numero di documenti all'interno del dataset, mentre il denominatore è il numero di documenti che contengono il termine i-esimo.[21]. Per definizione quindi il valore TF-IDF per la parola i-esima nel documenti j-esimo risulta:

$$(tfidf)_{\mathrm{i,j}} = tf_{\mathrm{i,j}} \times idf_{\mathrm{i}} \tag{2.7}$$

Sono nate diverse migliorie a partire da questo metodo per produrre una rappresentazione vettoriale in base alla frequenza di una parola all'interno del dataset, ma per l'uso della libreria Python che è stata scelta di utilizzare, è sufficiente per questo studio di tesi.

2.5 Supervised vs Unsupervised

Nelle sezioni precedenti, sono stati presentati i metodi di vettorizzazione di un corpo di testo, ma non sempre queste tecniche sono applicabili. In questo studio di tesi vengono confrontate 5 principali tecniche di apprendimento statistico supervisionato che verranno approfondite nel capitolo successivo:

1. Regressione logistica
2. Support vector machine
3. Albero decisionale
4. Random forest
5. Reti neurali

Le prime 4 tecniche utilizzano i meccanismi di vettorizzazione spiegata precedentemente, mentre le reti neurali potrebbero utilizzare un altro tipo di rappresentazione di un corpo di testo. Come più volte ripetuto, l'apprendimento statistico è basato in primis sulla trasformazione di un corpo di testo in una rappresentazione matriciale, proprio perché è più facile trattare un problema dal punto di vista numerico piuttosto che testuale.

Le tecniche di vettorizzazione che abbiamo affrontato in precedenza fanno parte di una classe di rappresentazione di un dataset testuale, chiamata Word Embedding. Essenzialmente si tratta di una costruzione di un dizionario che converte un testo in vettori con una certa dimensionalità. Una delle applicazioni possibile dell'analisi del

testo è proprio la sentiment analysis proprio come la classificazione di documenti, la ricerca semantica, il clustering e così via. Le rappresentazioni di Word Embedding possiamo riassumerle in 2 macro categorie[22]:

- In base alla frequenza
- In base alla predizione

La prima categoria l'abbiamo già vista nella sezione precedente introducendo il Count Vectorizer per N-Gram e il TfidfVectorizer per TF-IDF. Un Word Embedding basato sulla predizione significa che data una parola viene restituito un subset in uno spazio vettoriale a N dimensioni dove sono presenti delle parole che hanno una similarità o hanno una probabilità alta di essere simile a quella data o di vivere in un contesto simile alla parola.

Per la generazione di un dizionario in base alla predizione, verranno illustrate in breve 2 tecniche più utilizzate non supervisionate per la generazione di un modello di Word Embedding in base alla predizione: il Word2Vec e GloVe. Quest ultimo sarà quello scelto per questo studio di tesi, in quanto durante il reperimento dei modelli open source, sono stati trovati 2 modelli di lingua italiana e inglese che hanno subito l'apprendimento sulle pagine di Wikipedia delle rispettive lingue, dello stesso anno e con la stessa dimensionalità vettoriale.

2.5.1 Word2Vec

Il Word2Vec è un insieme di 2 algoritmi che generano il modello NLP statistico risultante[22]:

- Continuoud Bag Of Words (CBOW)
- Skip-gram

Entrambe le tecniche apprendono i pesi che poi produrranno il vettore rappresentativo di una parola. CBOW tende a creare un modello in cui si predice la probabilità che una parola appartenga ad un determinato contesto: questa proposizione può essere anche utilizzata per un insieme di parole e non solamente con la singola. Il problema che infatti si presenta con questa tecnica è che il numero di contesti in cui può vivere una parola è molteplice, pertanto occorre utilizzare un meccanismo che dia più priorità ad un contesto piuttosto che ad un altro (es. una media pesata, una semplice media oppure una distribuzione probabilistica). Come ogni modello probabilistico però possiede dalla sua la scalabilità di affrontare più contesti diversi senza aumentare la complessità algoritmica richiedendo un dataset molto grande per avviare un training; per la creazione di modelli così grandi e complessi, vengono utilizzati solitamente dati provenienti da enciclopedie online (es. Wikipedia) o social media [22].

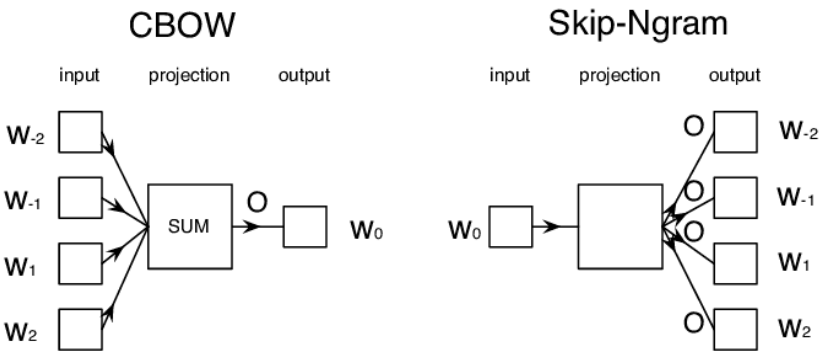

Figura 2.2: CBOW e Skip-NGram messi a confronto dal punto di vista funzionale[23]

Skip-gram agisce contrariamente al CBOW: il suo compito è di predirre un contesto data una parola o un suo insieme. L'output può essere un solo contesto o un insieme dove può vivere una parola: questa tecnica cerca infatti di contenere i problemi del CBOW dando non un solo risultato complessivamente utilizzando Word2Vec, ma una moltitudine che hanno un vicinanza a livello semantico[22].

2.5.2 GloVe

Come il Word2Vec, Global Vectors for Word Representation, o GloVe, è una tecnica di unsupervised learning per ottenere una rappresentazione matriciale in uno spazio vettoriale di un insieme di parole[24].

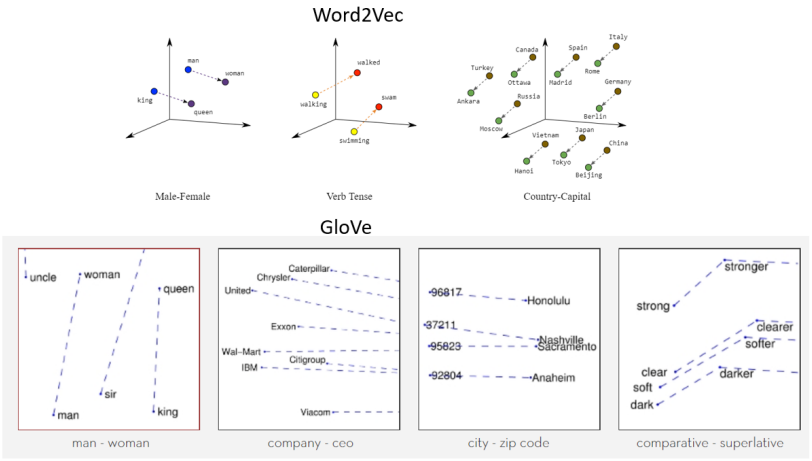

Figura 2.3: Confronto rappresentazione spaziale in Word2Vec e GloVe[25]

Tramite questa rappresentazione di Word Embedding, GloVe cerca di arrivare a 2 soluzioni:

1. creare un vettore per ogni parola che catturi il significato nello spazio
2. utilizzare una statistica globale al posto di utilizzare solamente le informazioni locali del contesto

GloVe avvia il procedimento di learning tramite una matrice di co-occorrenze al posto di imparare tramite uno stream di informazioni come il Word2Vec opera. Questa rappresentazione matriciale definisce una distribuzione di valori che co-occorrono nel testo utilizzato nell'apprendimento: si tengono presenti le volte che una parola compare nel testo e a quale distanza compare nel testo, cioè la stessa compare più volte nell'arco di poche righe o parole di distanza[24].

Il problema che possiede per costruzione il Word2Vec è che tiene conto solamente del contesto locale e non utilizza una statistica globale, la quale in alcuni casi potrebbe approssimare la moltitudine di contesti a cui è applicata la parola in un singolo documento. GloVe in altre parole tiene conto dell'informazione globale e impara le dimensioni di significato mano a mano nel processo di training[24].

Capitolo 3

Tecnologie e tecniche di apprendimento statistico

In questo capitolo affronteremo come viene creato un classificatore per la sua applicazione nella sentiment analysis. Verranno presentate le tecniche più comuni che sono state adottate per il processo di training e messe a confronto in seguito tramite le metriche prestazionali estratte dalla creazione del modello. Sono presentati anche i dettagli matematici, ma non saranno approfonditi nel dettaglio, dato che è necessario solamente dare l'idea di come queste metodologie funzionino e quale campo di applicazione hanno.

3.1 Introduzione al machine learning

Il termine "machine learning" si riferisce al rilevamento automatizzato di significati modelli all'interno dei dataset. Negli ultimi decenni è diventato uno strumento comune in quasi tutte le attività che richiedono l'estrazione di informazioni da grandi quantità di dati. Correntemente utilizziamo tecnologie basate sull'apprendimento automatico: i motori di ricerca imparano come per offrirci i migliori risultati, software anti-spam impara a filtrare i nostri messaggi e-mail e le transazioni con carta di credito sono protette da un software che impara a rilevare le frodi in base alle nostri abitudini e i nostri spostamenti. Le fotocamere digitali negli smartphone di ultima generazione imparano a rilevare i volti e le applicazioni intelligenti di assistenza personale sugli smartphone riconoscono la nostra voce in modo tale da impartire dei comandi vocali che eseguano delle azioni specifiche.[26]

Il machine learning non è solo utilizzato in ambiti commerciali, ma anche in applicazioni scientifiche come bioinformatica, medicina e astronomia. Una caratteristica comune di tutte queste applicazioni è che, al contrario di altre usi tradizionali dei computer, in questi casi, a causa della complessità dei modelli che

devono essere rilevati, un programmatore umano non è in grado di impartire un set o un subset di istruzioni da eseguire per svolgere un determinato compito, perché la trattazione dei dati potrebbe essere ambigua e la programmazione è deterministica e non probabilistica (perciò io dichiaro variabili ed eseguo istruzioni logicamente, non probabilisticamente). Molte delle nostre abilità vengono acquisite o perfezionate attraverso l'apprendimento, la nostra esperienza e spesso non seguiamo delle istruzioni esplicite, perché non possono comprendere il contesto d'azione ed uso corrente e hanno poca tolleranza se non presente alcuna[26].

L'apprendimento statistico automatico non è un processo che può essere trattato a scatola chiusa, bensì richiede conoscenza delle metodologie che si applicano e molta potenza computazionale per la produzione di un modello di classificazione; in sintesi "There ain't no such thing as a free lunch".

3.2 Tecniche di apprendimento

Ci sono numerosi compiti che noi esseri umani eseguiamo regolarmente, ma non siamo in grado di decomporre le azioni e la loro logica in più piccole task per arrivare al risultato voluto e spesso cambiano a seconda dell'ambiente circostante e al contesto d'appartenenza. Dei possibili esempi potrebbero essere la guida, il riconoscimento del parlato e delle immagini. In tutte queste attività, costruire dei modelli di machine learning che imparano dalle nostre esperienze raggiungono risultati abbastanza soddisfacenti, però solamente esposti a sufficienti esempi di formazione, perciò gradi quantità di dataset e di qualità (non sporchi di dati fasulli e/o con errore contenuto in un margine di tolleranza accettabile)[26].

Sempre parlando di grossi dataset, il machine learning porta dei grandi benefici quando occorre analizzare molti dati: dati astronomici, dati medicali, previsioni meteorologiche, analisi di dati genomici, motori di ricerca e sistemi di raccomandazione. Con sempre più dati disponibili in formato digitale, i dataset diventano delle vere e proprie miniere d'oro per l'analisi automatica e l'apprendimento tramite il machine learning: spesso le banche dati sono rimaste dormienti per diversi anni, perché troppo grandi per essere analizzate da un insieme di esseri umani. Oggi con la potenza computazionale a disposizione e la conoscenza più approfondita riguardo a intelligenza artificiale, l'utilizzo di dataset giganteschi di diversi anni fa e i nuovi dati generati a ritmi inimmaginabili rispetto a qualche decennio fa sono un tesoro che portano ad una maggiore conoscenza di un determinato aspetto da analizzare, aprendo nuovi orizzonti, sia in applicazioni scientifiche che non[26].

Ci sono varie distizioni di apprendimento nel machine learning, ma spesso si parla di 2 macro categorie di learning statistico:

1. supervised learning (apprendimento supervisionato): viene in genere eseguito nel contesto della classificazione, quando si desidera mappare l'input alle

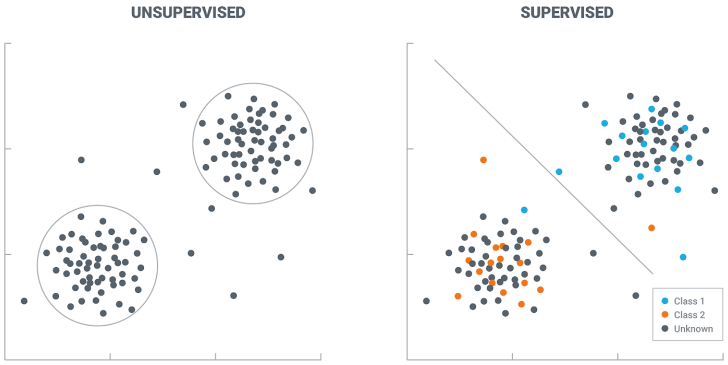

Figura 3.1: Esempio di rappresentazione grafica per distinguere il supervised dall'unsupervised learning[27]

etichette di output. Algoritmi comuni nell'apprendimento supervisionato includono la regressione logistica, le support-vector machines, reti neurali artificiali, gli alberi decisionali e le foreste casuali. Sia nella regressione che nella classificazione, l'obiettivo è trovare relazioni o strutture specifiche nei dati di input che ci consentano di produrre in modo efficace dati di output corretti. Occorre tenere a mente che l'output "corretto" (non è possibile definire a priori sempre cosa sia corretto o meno, specialmente nel caso dell'apprendimento non supervisionato) è determinato interamente dai dati di training, quindi mentre abbiamo una "verità" di base che il nostro modello riterrà vera, non si può dire che le etichette dei dati siano sempre corrette nelle situazioni del mondo reale. Le etichette dei dati rumorose o errate ridurranno chiaramente l'efficacia del modello come già citato in precedenza[26];

2. unsupervised learning (apprendimento non supervisionato): solitamente utilizzato per il raggruppamento, l'apprendimento della rappresentazione grafica/audio e la stima della densità. In tutti questi casi, desideriamo apprendere la struttura intrinseca dei nostri dati senza utilizzare etichette fornite in modo esplicito. Alcuni algoritmi comuni includono il k-means clustering, la PCA e gli autoencoders. Poiché non vengono fornite etichette, non esiste un modo specifico per confrontare le prestazioni del modello nella maggior parte dei metodi di apprendimento senza supervisione. L'apprendimento non supervisionato è molto utile nell'analisi esplorativa, perché può identificare automaticamente la struttura nei dati. Ad esempio, se un analista stesse cercando di segmentare i consumatori, i metodi di clustering senza supervisione sarebbero un ottimo

punto di partenza per la loro analisi. In situazioni in cui è impossibile o impraticabile per un essere umano proporre tendenze nei dati, l'apprendimento senza supervisione può fornire intuizioni iniziali che possono quindi essere utilizzate per verificare le singole ipotesi[26].

In questo studio di tesi verranno affrontate solamente le tecniche di apprendimento supervisionato, perché non vogliamo conoscere nessun pattern tra le recensioni (oppure potrebbe essere un lavoro futuro da proporre come ulteriore sviluppo dalle conclusioni che si traggono da questo), ma vogliamo produrre un classificatore, in modo tale da capire se la recensione in ingresso nel modello statistico sia positiva, negativa o neutrale. Le tecniche principali che verranno affrontate sia dal punto di vista teorico che pratico sono:

1. Logistic Regression
2. Support-Vector Machines (SVM)
3. Decision Tree
4. Random Forest
5. Neural Networks
   (a) Recurrent Neural Network (RNN)
   (b) Feedforward Neural Network (FFNN)
   (c) Convolutional Neural Network (CNN)
   (d) Long Short Term Memory (LSTM)

3.2.1 Regressione Logistica

Nella regressione logistica vogliamo ottenere una famiglia di funzioni h da $\mathcal{R}^d$ nell'intervallo [0, 1], perché lo scopo è quello di trovare una classificazione di recensioni: si può interpretare h(x) come una funzione di probabilità che l'etichetta di x sia 1. L'ipotetica classe associata alla regressione logistica è la composizione di una funziona sigmoidea $\phi_{\text{sig}} : \mathcal{R} - > [0, 1]$ sulla classe di funzioni lineari $\mathrm{L_d}$. In particolar modo, la funzione sigmoid utilizzata nella regressione logistica è la funziona logistica definita in questo modo[26]:

$$\phi_{\text{sig}}(z) = \frac{1}{1 + exp(-z)} \tag{3.1}$$

Il nome "sigmoide" significa "a forma di S", il quale è riferito alla trama di questa funzione, mostrato in figura 3.2[26].

La classe d'appartenenza per ipotesi è quindi (dove per semplicità stiamo usando omogeneo funzioni lineari)[26]:

$$H_{\text{sig}} = \phi_{\text{sig}} \bigcirc L_{\text{d}} = x \rightarrow \phi_{\text{sig}}(w, x) : w \in \mathcal{R}^d \tag{3.2}$$

Notare che quando (h, x) è molto grande, allora $\phi_{\text{sig}}(w, x)$ è vicino a 1, mentre se (h, x) è molto piccolo quindi $\phi_{\text{sig}}(w, x)$ è vicino a 0. Ricordiamo che la previsione

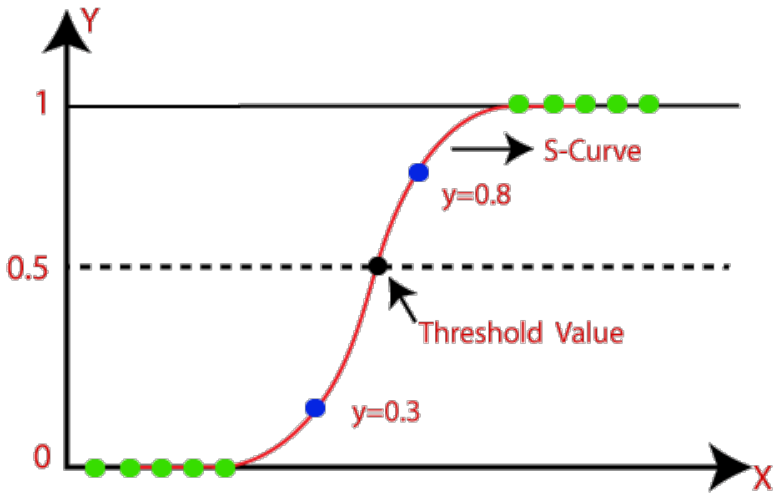

Figura 3.2: Rappresentazione grafica di una regressione logistica[28]

del mezzo spazio corrispondente a un vettore w è $sign(\phi_{\text{sig}}(w, x))$. Pertanto, le previsioni dell'ipotesi dello spazio di mezzo e l'ipotesi logistica sono molto simili ogni volta che |h, x| è grande. Tuttavia, quando |h, x| è vicino a 0 abbiamo che $\phi_{\text{sig}}(w, x) \approx \frac{1}{2}$. Intuitivamente, l'ipotesi logistica non è sicura del valore dell'etichetta, quindi prova ad indovinare l'etichetta di $sign(\phi_{\text{sig}}(w, x))$ con probabilità leggermente superiore al 50%. Al contrario, l'ipotesi di mezzo spazio produce sempre una previsione deterministica di 1 o 1, anche se |h, x| è molto vicino a 0[26].

Successivamente, dobbiamo specificare una funzione di perdita, cioè dovremmo definire quanto male prevedere un insieme di $h_{\text{w}}(x) \in [0, 1]$ dato che l'etichetta vera è $y \in \pm 1$. Chiaramente, vorremmo che $h_{\text{w}}(x)$ fosse grande se y = 1 e che $1 - h_{\text{w}}(x)$ (cioè, la probabilità di prevedere 1) fosse grande se $y = 1$. Nota che[26]:

$$1 - h_{\text{w}}(x) = \frac{1}{1 + exp(w, x)} \tag{3.3}$$

Pertanto, qualsiasi ragionevole funzione di perdita aumenterebbe in modo monotonico con $1 - h_{\text{w}}(x) = \frac{1}{1+exp(y(w,x))}$, o equivalentemente, aumenterebbe sempre in modo monotono con $1 + exp(-y(w, x))$. La funzione di perdita logistica utilizzata nella regressione logistica penalizza h_{w} in base al registro di $1 + exp(-y(w, x))$ (ricorda che il log è una funzione monotona). In sintesi[26]:

$$l(h_{\text{w}}(x, y) = log(1 + exp(-y(w, x)))) \tag{3.4}$$

Pertanto, dato un set di addestramento $S = (x_1, y_1), ..., (x_{\text{m}}, y_{\text{m}})$, il problema dell'Empirical Risk Minimization (detto ERM), cioè della riduzione del range dove

le performance sono accettabili in altre parole, associato alla regressione logistica è[26]:

$$argmin\frac{1}{m}\sum_{\mathrm{i}=1}^{m} log(1+exp(-y_{\mathrm{i}}(w,x_{\mathrm{i}}))) \tag{3.5}$$

Il vantaggio della funzione di perdita logistica è il fatto di essere una funzione convessa con rispetto a w, perciò il problema ERM può essere risolto in modo efficiente usando lo standard metodi[26].

3.2.2 Support-Vector Machines (SVM)

L'alta dimensionalità dello spazio delle funzionalità aumenta sia la complessità del campione che le sfide della complessità computazionale.

Quando il dataset da trattare possiede una dimensionalità elevata a livello di feature, la complessità computazionale aumenta; le support-vector machine (o SVM) cercano di affrontare questa difficoltà utilizzando degli iperpiani come "separatori" tra classi d'apparteneza (in questo caso delle recensioni). Questo iperpiano separa un set durante la fase di training prima con un ampio margine, per poi affinarlo e via via fittarlo al dataset. Limitare l'agoritmo per produrre un separatore di grande margine produce una complessità algoritmica contenuta anche se la dimensionalità delle feature è elevata[26].

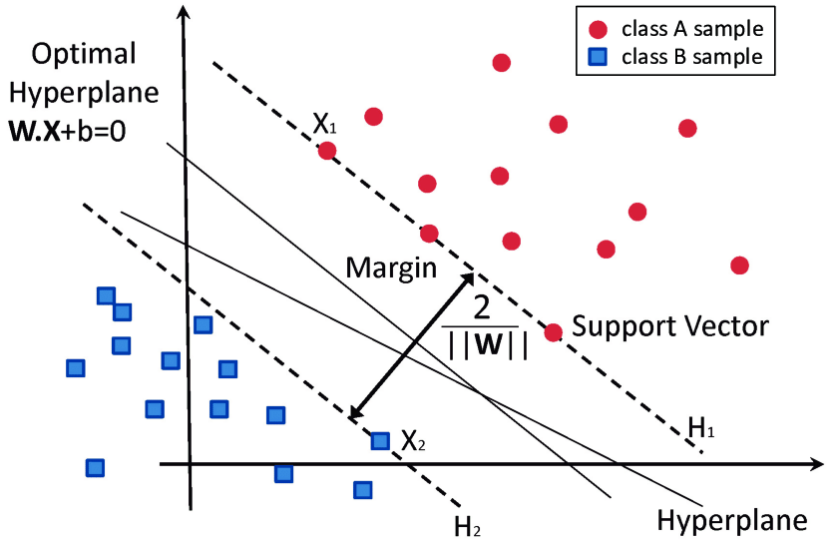

Figura 3.3: Rappresentazione grafica di una support vector machine[29]

Sia $S = (x_1, y_1), ..., (x_{\mathrm{m}}, y_{\mathrm{m}})$ è un insieme di esempi per il training, in cui ogni $x_{\mathrm{i}} \in \mathcal{R}^d$ e $y_{\mathrm{i}} \in \pm 1$. Diciamo che questo set di allenamento è linearmente separabile,

se esiste un halfspace, (w, b), tale che $y_{\mathrm{i}} = sign((w, x_{\mathrm{i}}) + b)$ per ogni i. In alternativa, questo condizione può essere riscritta come:[26]

$$\forall i \in [m], y_{\mathrm{i}}((w, x_{\mathrm{i}}) + b) > 0 \tag{3.6}$$

Tutti gli halfspaces (w, b) che soddisfano questa condizione sono ipotesi ERM (il loro 0-1 errore è zero, che è l'errore minimo possibile). Per qualsiasi train set separabile, ci sono molti halfspaces ERM. Considerando, il set di allenamento descritto nella figura precedente, la nostra intuizione ci porterebbe probabilmente a preferire l'iperpiano che taglia in modo parallelo la classe A da quella B. L'iperpiano che viene preferito infatti è quello descritto dalla linea obliqua incidente con la direttiva parallela della separazione tra le 2 classi trattate[26].

Una SVM è un algoritmo di training in cui restituiamo un hyperplane ERM che separa il set per il training con il margine più ampio possibile. Per definire una SVM formalmente, per prima cosa esprimiamo la distanza tra un punto x a un iperpiano usando i parametri che definiscono il semispazio. In altre parole quello che si vuole trovare con questo metodo dato in input $S = (x_1, y_1), ..., (x_{\mathrm{m}}, y_{\mathrm{m}})$ è[26]:

$$(w_0, b_0) = argmin||w||^2 \quad \forall i, y_{\mathrm{i}}((w, x_{\mathrm{i}}) + b) >= 1 \tag{3.7}$$

Restituendo in output:

$$w' = \frac{w_0}{||w_0||}, b' = \frac{b_0}{||b_0||} \tag{3.8}$$

3.2.3 Decision Tree

Un albero decisionale è un predittore, h: X → Y, che prevede l'etichetta associata un'istanza x viaggiando da un nodo radice di un albero a una foglia. Per semplicità ci concentriamo sull'impostazione della classificazione binaria, ovvero Y = 0, 1, ma gli alberi decisionali possono essere applicati anche per altri problemi di predizione (come per esempio la classificazione di recensioni positive, negative e neutrali). Per ogni nodo sul percorso radice-foglia, il figlio successore viene scelto sulla base di una scissione del spazio di input. Di solito, la divisione si basa su una delle caratteristiche di x o su un set predefinito di regole di divisione che viene appreso durante il training del modello. Ogni foglia contiene un'etichetta specifica dove si può predirre la classe di appartenenza seguendo un pattern specifico[26].

Sfortunatamente, l'utilizzo degli alberi decisionali comporta nella risoluzione di problema molto difficile dal punto di vista computazionale, di conseguenza gli algoritmi di machine learning si basano sull'euristica come un approccio "best-fit", in cui l'albero viene costruito gradualmente e localmente le decisioni ottimali vengono prese alla costruzione di ciascun nodo. Tali algoritmi non può garantire di restituire

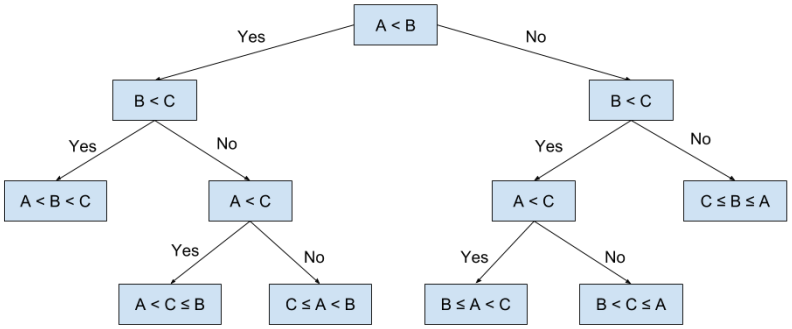

Figura 3.4: Rappresentazione grafica di un albero decisionale[30]

l'albero decisionale globale ottimale, ma tende a funzionare ragionevolmente bene in pratica[26].

Un quadro generale per la crescita di un albero decisionale segue il seguente schema:[26]

1. si inizia con un albero con una singola foglia (la radice) e assegnare a questa foglia un'etichetta secondo il voto di maggioranza sul training set
2. per ogni iterazione che viene compiuta, si esaminano gli effetti dello splitting per ogni singola foglia
3. per ogni iterazione si definisce il guadagno come misura che quantifica il miglioramnento tramite l'aggiunta di una biforcazione (o split dell'albero)
4. su tutti i possibili split, viene scelto quello che massimizza il guadagno e si performa o no lo split sulla singola foglia

Si possono utilizzare diversi algoritmi per scegliere il guadagno migliore e per decidere quando è necessario eseguire o meno lo split durante l'attraversamento dell'albero[26].

Il problema di questo approccio è che l'albero risultante potrebbe diventare molto grande anche perché poco fattibile a livello computazionale la sua ottimizzazione. Una soluzione è il processo di pruning, dove vengono ripercorse le foglie con un approccio bottom-up, cioè dalle foglie senza figli fino alla radice. Ogni nodo o insieme di nodi potrebbe essere rimpiazzato da un albero piccolo basato sul processo di splitting e di guadagno che viene deciso al momento di compiere il training del modello[26].

3.2.4 Random Forest

Gli alberi decisionali possono raggiungere dimensioni notevoli durante il processo di training, perciò sono limitate le sue dimensioni tramite il pruning e la scelta di un criterio di splitting e guadagno. Col il processo di pruning si incorre nel rischio di produrre un albero decisionale overfittato, perciò per ridurrlo, vengono costruiti più alberi[26].

Una foresta casuale è un classificatore costituito da una raccolta di alberi decisionali, dove ognuno di essi è costruito applicando un particolare algoritmo sul dataset per il training. La previsione del classificatore basato sul random forest è ottenuta con il voto di maggioranza o pesato sulle previsioni dei singoli decision trees generati[26].

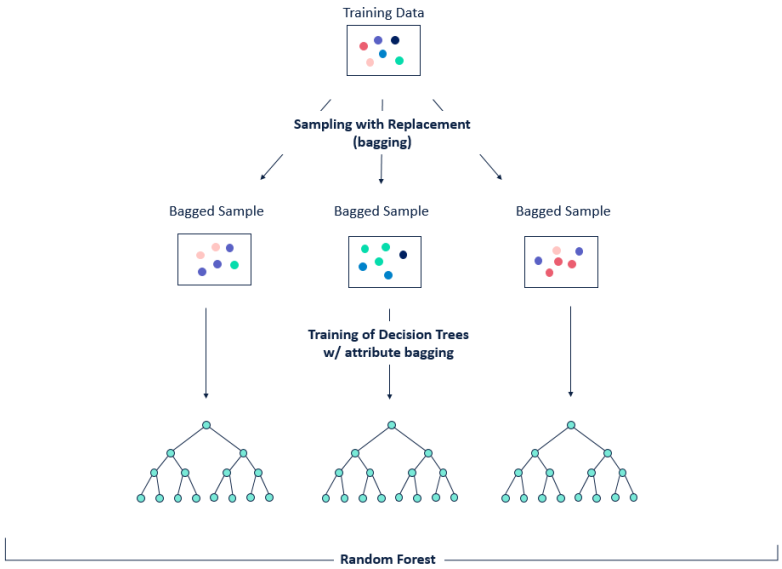

Figura 3.5: Rappresentazione grafica di una foresta casuale[31]

Solitamente con un determinato algoritmo si genera una distribuzione con i passaggi seguenti:[26]

1. si creano dei subset del dataset utilizzato per il training
2. si costruiscono più sequenze di etichette e valori per ogni subset creato
3. viene applicato l'algoritmo per far crescere l'albero decisionale su ogni subset e vengono valutati gli split e i guadagni sulla singola sequenza

I decison trees sono utilizzati perché sono predittori molto intuitivi e facilmente comprensibili i loro processi decisionali dal programmatore, ma la loro complessità algoritmica è molto elevata e poco scalabile con l'hardware a disposizione. Le GPU

oggi tramite anche firmware e software aggiornati sono in grado di parallelizzare determinati workflow e sono utilizzate nell'ambito del deep learning, una branca del machine learning dove la conoscenza del ramo di applicazione ha un ruolo inferiore per generare un classificatore generico[26].

3.3 Introduzione al deep learning

Nonostante le reti neurali facciano parte del supervised learning, sono spesso catalogate all'interno di una sezione a parte: il deep learning è una branca del machine learning dove gli algoritmi vengono creati e funzionano in modo similare a quelli del machine learning, ma esistono numerosi livelli di questi, ognuno dei quali fornisce un'interpretazione diversa dei dati da cui impara. Una rete di algoritmi è anche detta rete neurale artificiale, chiamata così proprio per il suo funzionamento ispiratosi al funzionamento del cervello umano. Ciò che si vuole replicare è come la mente funziona tramite neuroni, dendriti e assoni[32].

Il deep learing adotta un approccio diverso per risolvere questo problema. Il vantaggio principale del deep learning è che non è necessario avere dati strutturati/etichettati delle immagini per classificare i due animali. Le reti neurali artificiali che utilizzano il deep learning inviano l'input (le parole all'interno delle recensioni in questo caso) attraverso diversi strati della rete, con ciascuna rete che definisce gerarchicamente caratteristiche specifiche delle recensioni. Questo è un modo simile a come il nostro cervello umano lavora per risolvere i problemi, passando domande attraverso varie gerarchie di concetti e domande correlate per trovare una risposta[32].

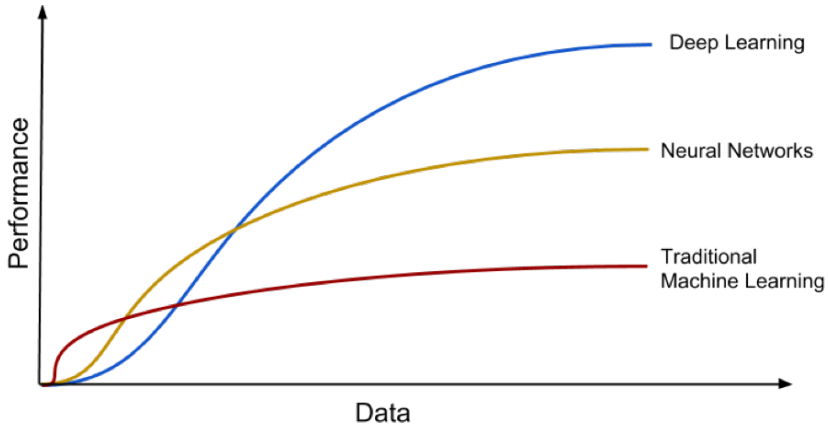

Figura 3.6: Analisi delle performance di un modello di machine learning vs deep learning all'aumentare della cardinalità di un dataset[33]

Un vantaggio che il deep learning e le reti neurali offrono è che si adattano molto bene a dataset di grandi dimensioni, offrendo una migliore scalabilità alla risoluzione di un problema di classificazione all'aumentare della dimensione del dataset. È importante notare che il dominio del deep learning è solo su dataset molto grandi, perché nel regime di piccoli dataset, non è chiaro quale tipo di modello sia migliore e molto probabilmente occorrerà utilizzare una qualche forma di esperienza nel dominio delle caratteristiche per far funzionare bene il modello generato[33].

3.4 Reti neurali

Dato che abbiamo citato spesso le reti neurali, è d'obbligo una loro introduzione propria con una spiegazione di alcune tipologie che sono state adottate.

Una rete neurale è una modalità per creare modelli di calcolo ispirati al cervello.Quello che si cerca di replicare è l'attività celebrale attraverso componenti software che emulino il loro comportamento: le comunicazioni, le interazioni neuronali e la loro stimolazione. Lo scopo di questa simulazione via software è far sì che vengano svolti dei calcoli complessi per ottenere delle probabilità che mettano in relazione un input con la sua classe di appartenenza in output[26].

Una rete neurale può essere descritta come un grafo diretto ciclio o aciclico a seconda della topologia che si vuole adottare; i nodi corrispondono a neuroni e i vertici agli assoni.

Durante la fase di apprendimento, possiamo definire un insieme di classi di ipotesi di appartenenza di un determinato input per costruire un classificatore. In questa fase, tutti i collegamenti con i neuroni devono essere tarati e avere un peso in base al dataset utilizato per questo processo di training e validato tramite il test su una parte dei dati per valutarne le metriche prestazionali. Non solo i collegamenti, il loro peso e la topologia sono fondamentali, ma anche il numero di nodi per ogni strato (una rete neurale può avere più strati, detti layers) determina l'efficacia del classificatore risultante[26].

Molte tipologie sono ancora in fase di studio e alcune vengono teorizzate solamente in quest ultimo decennio, ma per lo scopo di questo studio sono state adottate le strutture più comuni e consolidate sia dal punto di vista della ricerca che dal punto di vista di una produzione di software. Le reti neurali che verranno affrontate sono le seguenti:

1. Convolutional Neural Network (CNN)
2. Recurrent Neural Network (RNN)
3. Feedforward Neural Network (FFNN)
4. Long Short Term Memory (LSTM)

3.4.1 Convolutional Neural Network (CNN)

Le reti neurali convoluzionali, note anche come CNN, sono un tipo specializzato di rete neurale per l'elaborazione dei dati che ha una topologia nota, simile a una griglia o una matrice. I dati possono contenere delle serie temporali, che possono essere pensate come una griglia 1D, che preleva campioni a intervalli di tempo regolari, e dati di immagine, che possono essere considerati una griglia 2D di pixel. Le reti convoluzionali hanno avuto in generale un enorme successo in applicazioni pratiche, specialmente in ambito della computer vision. Il nome di una CNN indica che la rete svolge un'operazione matematica chiamata convoluzione: è un tipo specializzato di operazione lineare utilizzata al posto di una moltiplicazione per matrici[34].

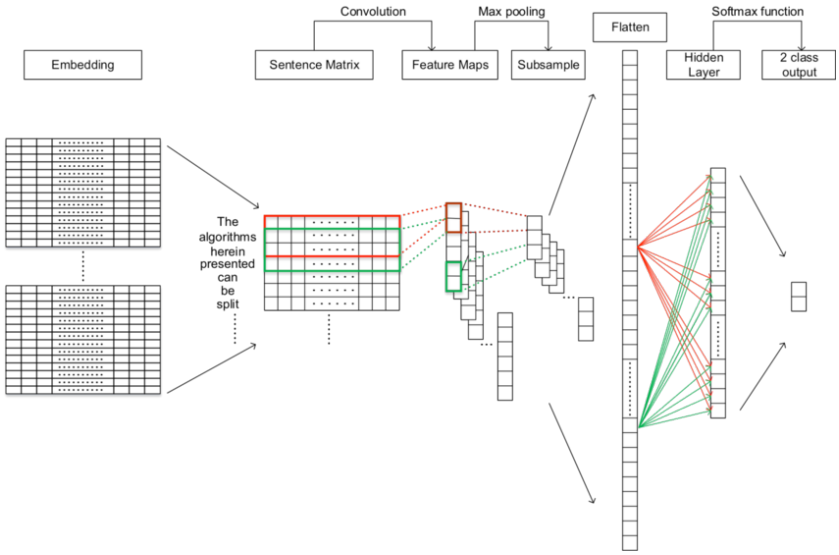

Figura 3.7: Topologia di una CNN[35]

La convoluzione sfrutta tre idee importanti che possono aiutare a migliorare una macchina sistema di apprendimento:[34]

1. interazioni sparse
2. condivisione dei parametri
3. rappresentazioni equivarianti

I livelli di una rete neurale tradizionale usano la moltiplicazione di matrici per un'altra matrice di parametri con un parametro separato che descrive l'interazione tra ciascuno unità di input e ciascuna unità di output: questo implica che ogni unità di uscita interagisce con ogni unità di input. Nel caso di un'interazione sparsa, solamente una parte della matrice interagisce con gli strati più profondi della rete:

per esempio, se avessimo un'immagine divisia in più matrici, l'interazione avviene solamente con la sezione d'immagine che ci interessa per attivare la classificazione. Questo implica meno operazioni da svolgere e meno parametri da memorizzare, migliorando l'efficienza del training della rete neurale[34].

La condivisione dei parametri si riferisce all'utilizzo dello stesso parametro per più di uno funzione in un modello. In una rete neurale tradizionale, ogni elemento della matrice del peso viene usato esattamente una volta quando si calcola l'output di un layer, cioè si moltiplica per uno elemento dell'input e non viene mai rivisitato. La condivisione dei parametri utilizzata dall'operazione di convoluzione significa che invece di apprendere un insieme separato di parametri per ogni posizione, impariamo solamente un insieme. Ciò non influisce sulla runtime della propagazione diretta, ma riduce ulteriormente i requisiti di archiviazione del modello, rendendo il processo più efficiente[34].

In caso di una CNN, la particolare forma di condivisione dei parametri provoca per ogni strato profondo della rete di avere una proprietà chiamata equivalenza alla traduzione: dire che una funzione è equivariante significa che se l'ingresso cambia, l'uscita cambia allo stesso modo[34].

3.4.2 Recurrent Neural Network (RNN)

Le reti neurali ricorrenti o RNN sono una famiglia di reti neurali per l'elaborazione di dati sequenziali. Un po' come una rete convoluzionale è una rete neurale specializzata nell'elaborazione di una griglia di valori X come un'immagine, una rete neurale ricorrente è una rete neurale per cui è specializzata elaborazione di una sequenza di valori $x_1, ..., x_t$. Proprio come le CNN può prontamente ridimensionarsi in immagini con larghezza e altezza elevate e alcune convoluzionali le reti possono elaborare immagini di dimensioni variabili, le reti ricorrenti possono ridimensionarsi molto usando sequenze più lunghe di quanto sarebbe pratico per le reti senza una specializzazione in sequenza. La maggior parte delle reti ricorrenti può anche elaborare sequenze di variabili lunghezza.[34].

Le RNN condividono i parametri in modo diverso rispetto alle altre tipologie di reti neurali: ogni membro dell'output è una funzione dei membri precedenti dell'output e utilizza la stessa regola di aggiornamento del grafo, perciò si comporta come una macchina a stati finiti. Questa formulazione ricorrente si traduce nella condivisione di parametri attraverso un grafico computazionale molto profondo[34].

In pratica, le reti ricorrenti di solito operano su minibatch di queste sequenze, con una lunghezza della stessa diversa τ per ciascun membro del minibatch, senza contare gli indici per semplicità. Inoltre, l'indice del passo temporale non è necessario fare letteralmente riferimento al passare del tempo nel mondo reale, ma solo alla posizione nella sequenza. Le RNN possono anche essere applicate in dati spaziali in 2D come le immagini, e anche se applicati a dati che coinvolgono il

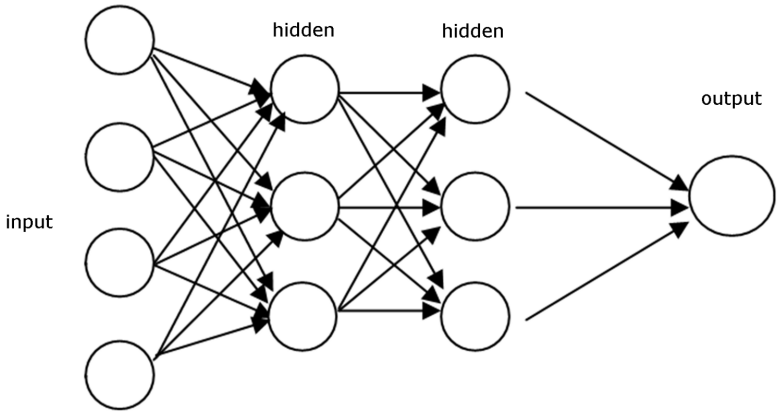

Figura 3.8: Topologia di una RNN[36]

tempo, la rete di una RNN può avere connessioni che vanno indietro nel tempo, a condizione che l'intera la sequenza viene osservata prima di essere fornita alla rete.[34].

3.4.3 Feedforward Neural Network (FFNN)

Le reti di feedforward, spesso chiamate anche FFNN, sono i modelli di apprendimento profondo per antonomasia, il quale obiettivo è approssimare una funzione. Questi modelli sono chiamati feedforward, perché le informazioni fluiscono attraverso una funzione valutata da x, attraverso i calcoli intermedi utilizzati per definire f e infine all'output y. Non ci sono connessioni di feedback in cui le uscite del modello vengono reinserite in se stesse[34].

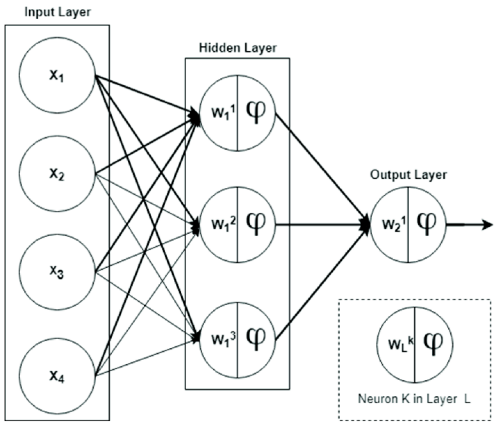

Figura 3.9: Topologia di una FFNN[37]

Le reti feedforward sono di estrema importanza e costituiscono la base di molte importanti applicazioni commerciali, come per ad esempio, le reti convoluzionali utilizzate per il riconoscimento di oggetti dalle foto sono a tipo specializzato di rete feedforward. Le FFNN sono concettuali un trampolino di lancio sul percorso di reti ricorrenti, che alimentano molti naturali applicazioni linguistiche[34].

Questo principio generale di miglioramento dei modelli attraverso l'apprendimento delle funzionalità si estende oltre le reti feedforward ed è un tema ricorrente di apprendimento che si applica a tutti i tipi di reti neurali. Le FFNN sono l'applicazione di questo principio all'apprendimento deterministico mappature da x a y che mancano di connessioni di feedback[34].

3.4.4 Long Short Term Memory (LSTM)

A differenza delle reti neurali di feedforward standard come abbiamo appena visto, le memorie a lunga e breve termine o anche dette LSTM hano connessioni di feedback. Può non solo elaborare singoli punti dati come le immagini, ma anche intere sequenze di dati, come il parlato, il testo o il video[34].

Un'unità di una LSTM è composta da una cella, un gate di ingresso, un gate di uscita e un gate di dimenticanza. La cella ricorda i valori su intervalli di tempo arbitrari e le tre porte regolano il flusso di informazioni all'interno e all'esterno della cella. Ogni sua componente può essere adattata a seconda della funzione che si vuole scegliere e per quanto tempo una determinata informazione deve rimanere all'interno della singola cella[34].

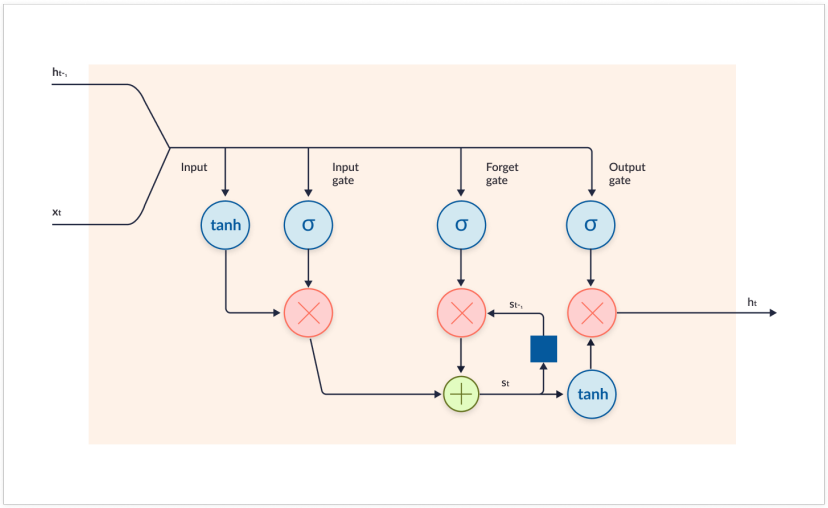

Figura 3.10: Topologia di una LSTM[38]

Molte altre varianti di questo schema possono essere progettate, ad esempio l'output del reset gate o dimenticare gate potrebbe essere condiviso tra più unità nascoste e vale la stessa cosa per l'input. In alternativa, il risultato di un gate globale che copre un intero gruppo di unità, come ad esempio un intero strato, e un gate locale per unità potrebbero essere usati per combinare il controllo globale e controllo locale[34].

Le reti LSTM sono adatte per classificare, elaborare e fare previsioni basate su dati di serie temporali, poiché possono esserci ritardi di durata sconosciuta tra eventi importanti in una serie temporale. Le LSTM sono stati sviluppati per affrontare i problemi di gradiente che esplode computazionalmente durante l'addestramento di RNN tradizionali, infatti spesso le LSTM sono applicabili ad attività come il riconoscimento della scrittura, vocale e il rilevamento di anomalie nel traffico di rete o Intrusion Detection Systems (IDS)[34].

Capitolo 4

Sentiment analysis

4.1 Introduzione

Come già citato nell'introduzione, la sentiment analysis ha suscitato interesse con l'avvento dei social media a partire da circa 20 anni fa. Le attività online sono cresciute esponenzialmente e sono nate nuove professioni orientate al web, come web marketing, web development, blog moderator e così via. Non solo nuove mansioni sono nate, ma le già esistenti si sono espanse su internet aumentando la diffusione di contenuti già presenti, ma poco noti a determinate fasce della popolazione.

Nonostante sia cambiato negli ultimi anni il nostro modo di utilizzare internet, le nostre necessità di reperire informazioni e di divulgarle sono rimaste identiche. In generale reperire contenuti che abbiano un bias ridotto o nullo è arduo, proprio perché ogni persona tende ad esprimere il proprio giudizio, che sia positivo, neutro o negativo.

In questo capitolo affronteremo come è stata svolta la sentiment analysis utilizzando i framework Keras e SKLearn per svolgere delle analisi sulle recensioni. Saranno spiegati i procedimenti adottati, i parametri utilizzati e affinati e le performance ottenute. Sono stati svolte diverse procedure sia per arrivare al modello finale che aggiungere diversi livelli all'interno di una rete neurale. Verranno poi comparati i metodi di machine learning tradizionali, come la regressione lineare, con architetture di reti neurali assieme all'utilizzo di word embeddings trainati con algoritmi di tipo supervised e unsupervised[39].

4.2 Reperimento e preparazione del dataset

La sentiment analysis è semplicemente quel passaggio per il riconoscimento di un'espressione di un'opinione dato un testo scritto da un utente generico: l'applicazione di questa tecnica è spesso utilizzata per raccomandare un prodotto o servizio di

una certa natura oppure per scopi di marketing del medesimo. Per la creazione di un modello semantico che riconosca un sentimento dato un testo, è necessario avere un dataset con delle recensioni di prodotti o servizi e per ognuna un rating associato(spesso da 1 a 5 come in molti siti di e-commerce oppure un meccanismo booleano come like-dislike).

Per i dataset presi in questione, è stato utilizzato il rating espresso da 1 a 5 e sono stati condotti 3 diverse tipologie di training suddividendo il dataset:

1. Recensioni positive con punteggio pari o superiore a 4 su 5 e negative con punteggio inferiore a 3 su 5
2. Recensioni positive con punteggio pari o superiore a 3 su 5 e viceversa per le negative
3. Recensioni positive con punteggio pari o superiore a 3.6 su 5, recensioni negative con punteggio inferiore a 2.4 su 5, neutrali le restanti

Come già citato in precedenza nel primo capitolo introduttivo, il problema del bilanciamento è sempre presente, pertanto è sempre necessario un ricampionamento sintetico in modo tale da avere 50-50 per i primi 2 casi e 33-33-33 per l'ultimo caso di studio di classificazione. Come scelta implementativa, le etichette utilizzate per determinare a quale classe appartenesse una recensione è stato utilizzato il seguente schema:

- 1 se recensione positiva
- 0 se recensione neutrale
- -1 se recensione negativa

Dopo aver definito il meccanismo di tagging per la singola recensione, è stato deciso di applicare un processo di pulizia di entrambi i dataset per aiutare il classificatore nel processo di apprendimento. I passaggi che il dataset ha subito sono i seguenti:

1. rimozione delle parole non utili al training (nomi, città, etc..)
2. rimozione delle stopwords
3. processo di stemming del testo delle recensioni
4. generazione del tag dato il punteggio e il caso affrontato
5. bilanciamento delle classi generate tramite ADASYN
6. generazione del word embedding

Il primo passaggio illustrato è l'unico assieme al reperimento dei dati scritto in Go, i restanti sono stati svolti tramite l'uso di un Jupyter Notebook con del codice Python. Dopo i passaggi illustrati sono state confrontate diverse tecniche di

training attraverso le metriche che sono risultate; i 2 dataset sono stati suddivisi in 4 parti, dove 1 di queste è stata utilizzata per la valutazione del classificatore risultante, mentre le restanti 3 hanno contribuito al processo di decision making.

I word embeddings sono stati ottenuti tramite 3 differenti metodologie, 2 supervised e 1 unsupervised:

1. CountVectorizer per N-Gram
2. TfidfVectorizer per TF-IDF
3. Tokenizer e matrice Numpy per GloVe

Una volta ottenuti i word embeddings, è possibile cominciare il training del dataset trasformato in matrici numeriche.

4.2.1 Approccio al problem solving

In questo studio di tesi ci si è voluti soffermare sulle tecniche supervisionate di apprendimento mettendo a confronto i 2 dataset di lingua italiana e inglese e le metodologie supervised e unsupervised per la generazione dei word embeddings. Come in precedenza citato, sono state utilizzate diverse tecniche di training e ciascuna deve essere utilizzata tramite un fine tuning dei parametri in ingresso. Nella tabella sottostante vengono descritti i parametri che sono stati "tarati" per ottenere i risultati che verranno esposti in seguito.

Tecnica	C	Solver	Criterion	Splitter	Estimators
LR	0÷1	newton-cg	gini/entropy	best/random	N.D.
SVM	0÷1	newton-cg	gini/entropy	best/random	N.D.
Decision Tree	N.D.	N.D.	gini/entropy	best/random	N.D.
Random Forest	N.D.	N.D.	gini/entropy	N.D.	50÷250

Tabella 4.1: Elenco parametri per fine-tuning per ogni tecnica di supervised learning senza l'uso di reti neurali

Per l'utilizzo delle reti neurali, i parametri per il fine-tuning sono diversi in quanto utilizzano uno schema diverso dalle classiche metodologie di machine learning. Le reti neurali utilizzano i word embeddings sia con tecniche supervised che unsupervised, tuttavia il processo di training è di tipo supervisionato. L'unica tipologia di rete neurale che utilizza un processo di training unsupervised è la Kohenon's Self Organizing Map (KSOM) che è utilizzata per il clustering con un dataset ad alta dimensionalità. Di solito è utilizzata com alternativa al tradizionale algoritmo di clustering K-Mean, ma solitamente le reti neurali non sono utilizzate

per estrarre dei pattern, bensì delle probabilità definite perché scalano bene come architettura nell'affrontare dei problemi dove si conoscono input e output[40].

Un altro vantaggio dell'utilizzo delle reti neurali rispetto alle classiche tecniche di machine learning è utilizzo delle GPU: queste permettono di abbattere le tempistiche di training di diversi ordini di grandezza, rendendo le reti neurali la scelta per un ambiente dinamico, come nella produzione di software e modelli semantici. In seguito sono elencati i parametri di fine-tuning per ogni tipologia di rete neurale che è stata utilizzata per svolgere i training:

Reti Neurali	Activation	Optimizer	Loss
CNN	relu/sigmoid	Adam/SGD	binary/categorical crossentropy
LSTM	relu/sigmoid	Adam/SGD	binary/categorical crossentropy
RNN	relu/sigmoid	Adam/SGD	binary/categorical crossentropy
FFNN	relu/sigmoid	Adam/SGD	binary/categorical crossentropy

Tabella 4.2: Elenco parametri principali per fine-tuning per ogni tecnica di supervised learning usando reti neurali

Per entrambe le topologie di tecniche di apprendimento, i parametri possono essere molteplici, tuttavia si è deciso di soffermarsi su quelli più importanti e di utilizzare delle tecniche ibride piuttosto che soffermarsi sul fine-tuning della singola. Nel caso di combinazione di 2 tecniche, vedremo in seguito una rete neurale RNN con una CNN, la quale darà i risultati migliore utilizzando questo tipologia di architettura. In generale, l'uso delle tecniche tradizonali di machine learning hanno dato risultati migliori con l'utilizzo di word embeddings generati tramite algoritmi di training di tipo supervised; nel caso delle architetture di reti neurali, i migliori word embeddings sono stati i GloVe e non Count Vectorizer o TF-IDF Vectorizer, tuttavia questi ultimi hanno permesso di ottenere delle metriche migliori senza l'adozione del deep learning. Contrariamente da quanto si potrebbe concludere, l'utilizzo dei word embeddings di tipo supervised hanno performato in modo decisamente peggiore con l'utilizzo delle reti neurali.

4.3 Confronto supervised vs. unsupervised learning

Come abbiamo già visto nel capitolo 2, i word embeddings utilizzati in questo studio di tesi sono di 3 tipologie:

1. N-Gram Count Vectorizer
2. TF-IDF Vectorizer
3. GloVe

Sono presenti molti algoritmi per la generazione dei word embeddings di tipo unsupervised, come per esempio Word2Vec, BERT e altri, ma per avere una metrica di giudizio migliore e la possibilità di fare un confronto a parità di tecnica, è stato deciso di adottare il modello GloVe. Il modelli che sono stati utilizzati per il confronto vengono dai seguenti laboratori di ricerca in ambito NLP:

1. Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo" del Consiglio Nazionale delle Ricerche con sede a Pisa[41]
2. Computer Science Department della Stanford University a Stanford in California[42]

Nel caso del word embeddings di lingua inglese, è presente il codice della sua generazione opensource su GitHub[43], in modo da rendere trasparente la metodologia e l'implementazione dell'algoritmo globalvectors. Per quanto riguarda quello in lingua italiana, non ci sono rimandi al codice sorgente per verificare l'effettivo risultato ottenuto, ma sono presenti dei test per virificare che il processo di training abbia avuto successo nel modo sperato [41].

4.3.1 Metriche e risultati

Questa sezione affronterà tutti i casi di suddivisione e trattamento delle recensioni in questo studio di tesi. L'analisi del dataset si svolgerà in 3 blocchi principali utilizzando il dataset nei seguenti modi:

1. suddivisione classi positive con scoring >= 4 su 5 e negative con scoring < 3 su 5 e
   (a) dataset pulito, come citato alla fine del capitolo 1
   (b) dataset senza alcuna pulizia
   (c) dataset pulito dai nomi delle patologie citate
   (d) dataset pulito dai digrammi/trigrammi dei dottori/professori
   (e) dataset pulito dai digrammi/trigrammi dei dottori/professori e dai nomi dalle patologie citate
   (f) dataset pulito dai digrammi/trigrammi dei dottori/professori e dai nomi delle città, province, stati e regioni
2. suddivisione classi positive con scoring >= 3 su 5 e negative con scoring < 3 su 5 e

   (a) dataset pulito, come citato alla fine del capitolo 1

3. suddivisione classi positive con scoring > 3.5 su 5, neutrali con scoring compreso tra 2.5 e 3.5, negative con scoring < 2.5 su 5 e

   (a) dataset pulito, come citato alla fine del capitolo 1

Il motivo per cui sono state svolte diverse analisi del dataset con diversi gradi di pulizia è per determinare se i risultati ottenuti derivassero da una pulizia e per poter capire se i procedimenti di apprendimento svolti potessero produrre un modello semantico overfittato e poco generalizzabile. Ogni singolo punto ha sempre affrontato la procedura di rimozione delle stopwords e lo stemming, in modo da avere dei word embeddings senza vettori con parole ripetute e/o simili (es. pulito e pulita diventano pulit e la matrice delle occorrenze prodotta avrà solamente una entry). Per ogni metodologia e tecnica, i dataset sono stati divisi sempre in 75%-25% in modo tale da verificare il modello ottenuto ad ogni training.

Cominciando dal primo caso affrontato, i risultati prodotti sono stati molto promettenti sia per le tecniche tradizionali di machine learning che per l'utilizzo di architetture di reti neurali.

Lang	Vectorizer	C	Acc.	Prec.	Recall	F1	MAE	MSE
IT	N-Gram	0.05	0.9009	0.9141	0.9748	0.9423	0.0567	0.0567
IT	TF-IDF	0.75	0.9031	0.9129	0.9791	0.9438	0.0587	0.0587
EN	N-Gram	0.75	0.9476	0.9543	0.9576	0.9443	0.0041	0.0041
EN	TF-IDF	0.95	0.9423	0.9499	0.9529	0.9514	0.0020	0.0020

Tabella 4.3: Caso 1A: Metriche tramite l'uso della LogisticRegression per il training con il solver newtoniano

Lang	Vectorizer	C	Acc.	Prec.	Recall	F1	MAE	MSE
IT	N-Gram	0.05	0.9009	0.9141	0.9748	0.9443	0.0567	0.0567
IT	TF-IDF	0.15	0.9031	0.9129	0.9791	0.9438	0.0587	0.0587
EN	N-Gram	0.05	0.9476	0.9543	0.9576	0.9443	0.0041	0.0041
EN	TF-IDF	0.7	0.9423	0.9499	0.9529	0.9514	0.0020	0.0020

Tabella 4.4: Caso 1A: Metriche tramite l'uso della SVM per il training con kernel di tipo lineare

Da quanto si può vedere dalle tabelle in alto, la regressione logistica e la support vector machine sono le tecniche di training che hanno permesso di ottenere

Lang	Vectorizer	Accuracy	Precision	Recall	F1	MAE	MSE
IT	N-Gram	0.8585	0.8832	0.9548	0.9168	0.0516	0.0511
IT	TF-IDF	0.8515	0.8824	0.9454	0.9135	0.0540	0.0519
EN	N-Gram	0.8549	0.8723	0.9576	0.9443	0.0041	0.0041
EN	TF-IDF	0.8492	0.8931	0.9592	0.9514	0.0020	0.0020

Tabella 4.5: Caso 1A: Metriche tramite l'uso del DecisionTree per il training con il criterion gini e best splitter

Lang	Vectorizer	Trees	Acc.	Prec.	Recall	F1	MAE	MSE
IT	N-Gram	100	0.8504	0.8473	0.9542	0.9152	0.0210	0.0310
IT	TF-IDF	150	0.8631	0.8592	0.9791	0.9438	0.0587	0.0468
EN	N-Gram	100	0.9111	0.9241	0.9990	0.9151	0.0079	0.0089
EN	TF-IDF	100	0.9217	0.9169	0.9542	0.9353	0.0079	0.0090

Tabella 4.6: Caso 1A: Metriche tramite l'uso del RandomForest per il training con il criterion gini

le performance migliore e in generale, data anche la cardinalità, il dataset in lingua inglese ha metriche migliori in generale rispetto a quello italiano. Le tabelle mostrano solamente i risultati migliori delle prove sperimentali utilizzando i parametri citati, ma come già in precedenza citato, sono state condotte diverse simulazione con la variazione dei parametri per avere un fine-tuning per ciascuna tecnica di machine learning adottata.

Per quanto riguarda le architetture di reti neurali, le performance non sono così buone come avute con il tradizionale machine learning, ma è stato possibile raggiungere risultati molto vicini in alcuni casi. Nelle tabelle precedentemente pubblicate, si può notare che la soluzione RNN+CNN con il GloVe come word embedding è quella più promettente a livello di metriche di performance.

Language	Accuracy	Precision	Recall	F1	Cohens kappa	ROC AUC
Italian	0.6164	0.5586	0.4768	0.5144	0.2007	0.5930
English	0.5962	0.5473	0.1125	0.1867	0.0538	0.5528

Tabella 4.7: Caso 1A: Metriche tramite l'uso di una rete FFNN con l'uso della TF-IDF al posto del modello GloVe

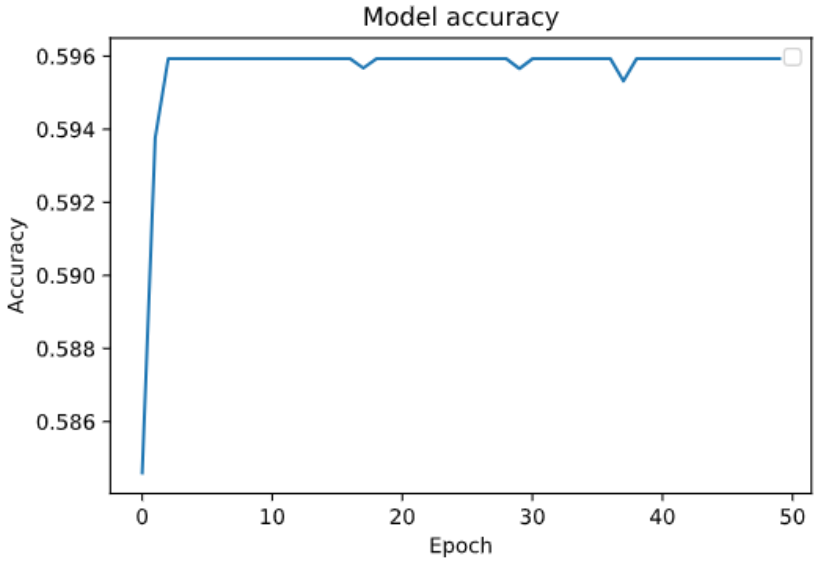

Figura 4.1: Caso 1A: accuratezza di una RNN in base al numero di epoche

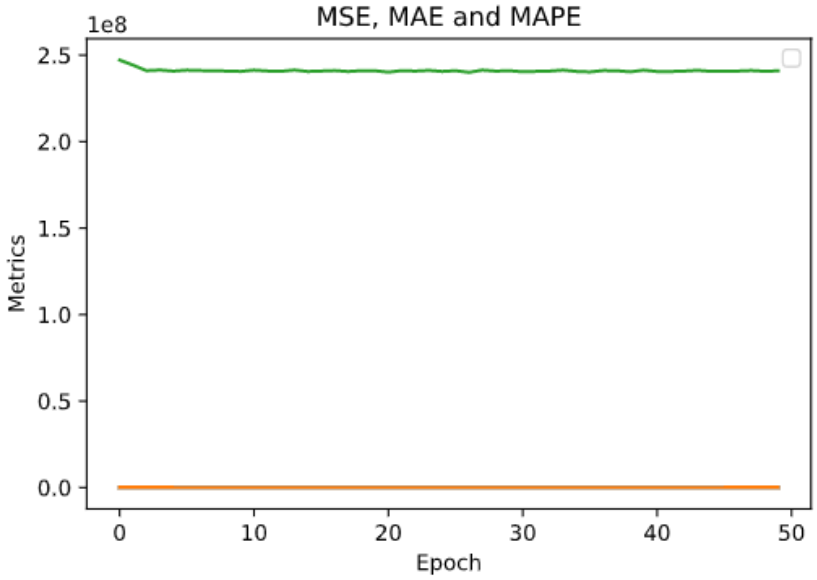

Figura 4.2: Caso 1A: andamento errore in base al numero di epoche per una RNN

Affrontando il caso 1B, si può notare che per ogni tecnica utilizzata le metriche prestazionali sono peggiorate, principalmente dove i risultati erano più promettenti. Le recensioni così dette non pulite hanno solamente subito il processo di rimozione delle stopwords e stemming: questo potrebbe concludere che il preprocessing e il

Language	Accuracy	Precision	Recall	F1	Cohens kappa	ROC AUC
Italian	0.7959	0.7600	0.7619	0.7609	0.5830	0.8444
English	0.9189	0.9174	0.8824	0.8996	0.8316	0.9601

Tabella 4.8: Caso 1A: Metriche tramite l'uso di una rete FFNN

Language	Accuracy	Precision	Recall	F1	Cohens kappa	ROC AUC
Italian	0.3760	0.8367	0.1173	0.2306	0.2661	0.2306
English	0.3669	0.9392	0.0548	0.0653	0.2881	0.2174

Tabella 4.9: Caso 1A: Metriche tramite l'uso di una rete LSTM

Language	Accuracy	Precision	Recall	F1	Cohens kappa	ROC AUC
Italian	0.5897	0.5331	0.3021	0.3856	0.1117	0.5514
English	0.5395	0.4112	0.2734	0.3284	0.2710	0.5011

Tabella 4.10: Caso 1A: Metriche tramite l'uso di una rete RNN

Language	Accuracy	Precision	Recall	F1	Cohens kappa	ROC AUC
Italian	0.6857	0.7097	0.4444	0.5466	0.3250	0.7155
English	0.5131	0.3594	0.2330	0.2827	0.5628	0.4613

Tabella 4.11: Caso 1A: Metriche tramite l'uso di una rete CNN

Language	Accuracy	Precision	Recall	F1	Cohens kappa	ROC AUC
Italian	0.8243	0.7976	0.7877	0.7926	0.6403	0.8672
English	0.9334	0.9276	0.9091	0.9183	0.8621	0.9676

Tabella 4.12: Caso 1A: Metriche tramite l'uso di una rete RNN+CNN

text mining prima di un training è fondamentale per produrre un modello statistico che performi adeguadamente per un ambiente non solo accademico, ma anche di produzione. Tuttavia, le metriche sono sempre rimaste coerenti con i risultati precedentemente ottenuti, rendendo più affidabile il modus operandi per ottenere il modello.

Andando avanti con il caso 1C, i miglioramenti iniziano ad essere senibili in generale: è stato deciso di rimuovere gradualmente una categoria di parole alla

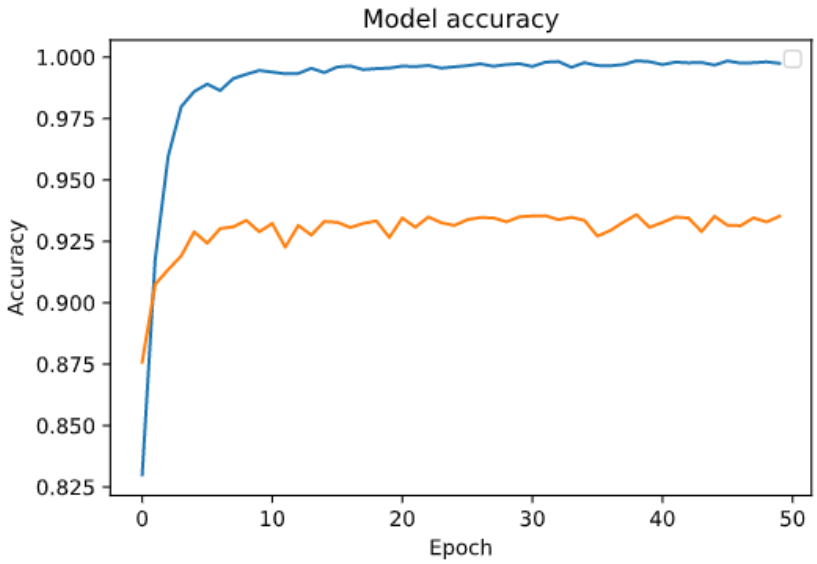

Figura 4.3: Caso 1A: accuratezza di una RNN+CNN per dataset in lingua italiana e inglese in base al numero di epoche

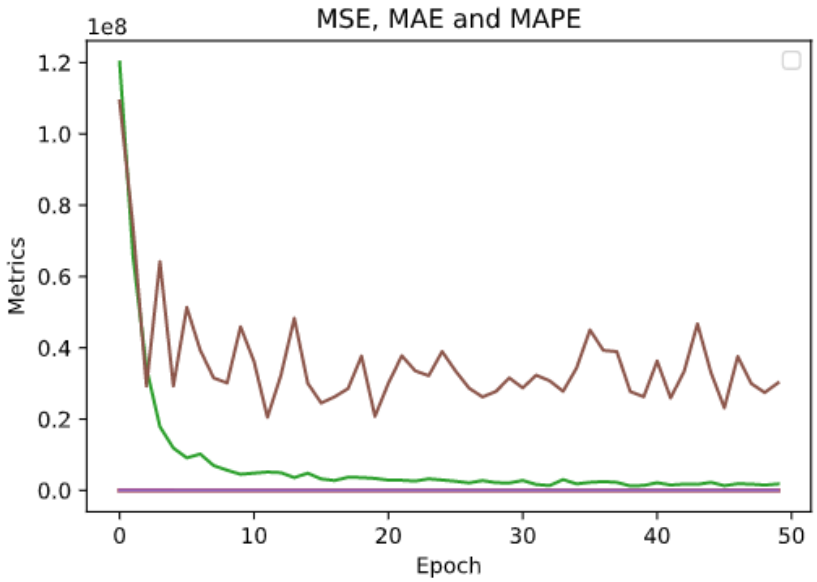

Figura 4.4: Caso 1A: andamento errore in base al numero di epoche per una RNN+CNN

volta proprio per capire quali siano più influenti per il processo di training del modello semantico. La scelta della aggiunta/rimozione delle patologie all'interno

Lang	Vectorizer	C	Acc.	Prec.	Recall	F1	MAE	MSE
IT	N-Gram	0.15	0.7109	0.9422	0.8419	0.8892	0.0511	0.0511
IT	TF-IDF	0.60	0.7031	0.9199	0.8712	0.8949	0.0532	0.0532
EN	N-Gram	0.75	0.7265	0.9013	0.8576	0.8789	0.0041	0.0041
EN	TF-IDF	0.80	0.7472	0.9091	0.8529	0.8801	0.0112	0.0112

Tabella 4.13: Caso 1B: Metriche tramite l'uso della LogisticRegression per il training con il solver newtoniano

Lang	Vectorizer	C	Acc.	Prec.	Recall	F1	MAE	MSE
IT	N-Gram	0.15	0.7011	0.9198	0.8231	0.8687	0.0492	0.0492
IT	TF-IDF	0.60	0.6971	0.9173	0.8432	0.8787	0.0591	0.0591
EN	N-Gram	0.75	0.6959	0.9011	0.8688	0.8847	0.0052	0.0052
EN	TF-IDF	0.80	0.6988	0.8997	0.8691	0.8841	0.0131	0.0131

Tabella 4.14: Caso 1B: Metriche tramite l'uso della SVM per il training con kernel di tipo lineare

Lang	Vectorizer	Accuracy	Precision	Recall	F1	MAE	MSE
IT	N-Gram	0.6451	0.8191	0.8371	0.8280	0.0711	0.0711
IT	TF-IDF	0.6522	0.8243	0.8222	0.8232	0.0698	0.0698
EN	N-Gram	0.6534	0.8456	0.8242	0.8348	0.0099	0.0099
EN	TF-IDF	0.6419	0.8419	0.8364	0.8391	0.0101	0.0101

Tabella 4.15: Caso 1B: Metriche tramite l'uso del DecisionTree per il training con il criterion gini e best splitter

delle recensioni è stata fatta perché nell'esplorazione del dataset si è notato che in generale molti utenti tendono a descrivere un caso clinico o perlomeno a citare la loro esperienza.

Con il caso 1D, si può notare come l'aggiunta/rimozione dei digrammi/trigrammi dei dottori e/o professori non influisca in modo determinante sul processo di training del modello statistico. Le metriche ottenute infatti sono molto simili al caso 1B precedentemente citato.

Guardando nel dettaglio il caso 1E, possiamo riaffermare quanto già espresso con la sola rimozione della categoria delle patologie, tuttavia è difficile affermare che la rimozione congiunta delle 2 cateogire di parole possa avere degli effetti migliori

Lang	Vectorizer	Trees	Accuracy	Precision	Recall	F1	MAE	MSE
IT	N-Gram	150	0.6837	0.8934	0.8733	0.8832	0.0600	0.0597
IT	TF-IDF	150	0.6912	0.9091	0.8720	0.8902	0.0581	0.0574
EN	N-Gram	200	0.6871	0.8988	0.8964	0.8976	0.0083	0.0085
EN	TF-IDF	200	0.6892	0.8973	0.8842	0.8907	0.0085	0.0085

Tabella 4.16: Caso 1B: Metriche tramite l'uso del RandomForest per il training con il criterion gini

Language	Accuracy	Precision	Recall	F1	Cohens kappa	ROC AUC
Italian	0.5062	0.4250	0.6983	0.5284	0.2454	0.5432
English	0.4911	0.4156	0.2196	0.2874	0.0463	0.4428

Tabella 4.17: Caso 1B: Metriche tramite l'uso di una rete FFNN con l'uso della TF-IDF al posto del modello GloVe

Language	Accuracy	Precision	Recall	F1	Cohens kappa	ROC AUC
Italian	0.6992	0.6713	0.6520	0.6615	0.5882	0.7485
English	0.8187	0.8139	0.7860	0.7997	0.7314	0.8604

Tabella 4.18: Caso 1B: Metriche tramite l'uso di una rete FFNN

Language	Accuracy	Precision	Recall	F1	Cohens kappa	ROC AUC
Italian	0.3340	0.8288	0.1350	0.2321	0.1350	0.1640
English	0.3349	0.8287	0.1285	0.2225	0.2331	0.1915

Tabella 4.19: Caso 1B: Metriche tramite l'uso di una rete LSTM

Language	Accuracy	Precision	Recall	F1	Cohens kappa	ROC AUC
Italian	0.5736	0.4994	0.1604	0.2428	0.0448	0.5279
English	0.5470	0.4024	0.2060	0.2725	0.0087	0.4953

Tabella 4.20: Caso 1B: Metriche tramite l'uso di una rete RNN

sulle metriche prestazionali. I risultati sono confrontabili facilmente con il caso 1C.

Prima di affrontare i casi 2 e 3 precedentemente elencati, ci soffermiamo sul

Language	Accuracy	Precision	Recall	F1	Cohens kappa	ROC AUC
Italian	0.6992	0.7713	0.4520	0.5699	0.3882	0.8485
English	0.6187	0.4139	0.3860	0.3994	0.5314	0.5604

Tabella 4.21: Caso 1B: Metriche tramite l'uso di una rete CNN

Language	Accuracy	Precision	Recall	F1	Cohens kappa	ROC AUC
Italian	0.6432	0.7091	0.5643	0.6284	0.6099	0.7743
English	0.7126	0.8199	0.6431	0.7208	0.8271	0.8694

Tabella 4.22: Caso 1B: Metriche tramite l'uso di una rete RNN+CNN

Lang	Vectorizer	C	Acc.	Prec.	Recall	F1	MAE	MSE
IT	N-Gram	0.20	0.7672	0.9322	0.8642	0.8969	0.0477	0.0477
IT	TF-IDF	0.55	0.7601	0.9292	0.8637	0.8952	0.0493	0.0493
EN	N-Gram	0.75	0.7743	0.9133	0.8781	0.8953	0.0037	0.0037
EN	TF-IDF	0.65	0.7806	0.9124	0.8697	0.8905	0.0089	0.0089

Tabella 4.23: Caso 1C: Metriche tramite l'uso della LogisticRegression per il training con il solver newtoniano

Lang	Vectorizer	C	Acc.	Prec.	Recall	F1	MAE	MSE
IT	N-Gram	0.25	0.7601	0.9220	0.8740	0.8973	0.0412	0.0412
IT	TF-IDF	0.55	0.7713	0.9247	0.8691	0.8960	0.0487	0.0487
EN	N-Gram	0.75	0.7694	0.9103	0.8635	0.8863	0.0047	0.0047
EN	TF-IDF	0.70	0.7726	0.9012	0.8590	0.8796	0.0096	0.0096

Tabella 4.24: Caso 1C: Metriche tramite l'uso della SVM per il training con kernel di tipo lineare

caso 1F, dove è possibile notare come le metriche prestazionali siano migliorate in generale, ma non in modo significativo. Questo infatti potrebbe sottolineare l'importanza di un dataset pulito per un training di un modello di machine learning non solamente dalle parole più generiche, ma avere una scelta ben definita di una loro categoria, quindi la conoscenza del contesto del dataset.

Affrontando il caso 2, si può subito notare la somiglianza delle metriche ottenute con il primo caso affrontato. Come citato nel capitolo 1, la distribuzione delle

Lang	Vectorizer	Accuracy	Precision	Recall	F1	MAE	MSE
IT	N-Gram	0.6841	0.8941	0.8803	0.8871	0.0601	0.0596
IT	TF-IDF	0.6901	0.9012	0.8781	0.8895	0.0583	0.0577
EN	N-Gram	0.6863	0.8978	0.8749	0.8862	0.0079	0.0079
EN	TF-IDF	0.6884	0.8993	0.8790	0.8890	0.0083	0.0087

Tabella 4.25: Caso 1C: Metriche tramite l'uso del DecisionTree per il training con il criterion gini e best splitter

Lang	Vectorizer	Trees	Accuracy	Precision	Recall	F1	MAE	MSE
IT	N-Gram	150	0.7107	0.9392	0.8471	0.8908	0.0506	0.0506
IT	TF-IDF	150	0.7029	0.9202	0.8567	0.8873	0.0511	0.0511
EN	N-Gram	100	0.7241	0.9001	0.8592	0.8792	0.0048	0.0048
EN	TF-IDF	150	0.7327	0.9068	0.8584	0.8819	0.0094	0.0094

Tabella 4.26: Caso 1C: Metriche tramite l'uso del RandomForest per il training con il criterion gini

Language	Accuracy	Precision	Recall	F1	Cohens kappa	ROC AUC
Italian	0.5476	0.4956	0.7160	0.5858	0.2787	0.5432
English	0.5547	0.5011	0.4042	0.4475	0.1533	0.4428

Tabella 4.27: Caso 1C: Metriche tramite l'uso di una rete FFNN con l'uso della TF-IDF al posto del modello GloVe

Language	Accuracy	Precision	Recall	F1	Cohens kappa	ROC AUC
Italian	0.7193	0.6988	0.6829	0.6908	0.6092	0.7623
English	0.8349	0.8287	0.8010	0.8146	0.7590	0.8799

Tabella 4.28: Caso 1C: Metriche tramite l'uso di una rete FFNN

classi di appartenenza delle recensioni è sempre sbilanciata e la sua distribuzione è comparabile tra i 2 casi. Lo spostamento della soglia di appartenenza, perciò, non ha influito sul modello risultante e non è possibile cambiarla in modo da avere un bilanciamento delle classi, perché non potremo giudicare una recensione con un punteggio superiore a 3 su 5 come negativa.

Infine analizzando le metriche prodotte dal caso 3, è stato possibile produrre

Language	Accuracy	Precision	Recall	F1	Cohens kappa	ROC AUC
Italian	0.3549	0.7821	0.1607	0.2666	0.1791	0.1933
English	0.3607	0.7794	0.1866	0.3011	0.2508	0.2264

Tabella 4.29: Caso 1C: Metriche tramite l'uso di una rete LSTM

Language	Accuracy	Precision	Recall	F1	Cohens kappa	ROC AUC
Italian	0.6018	0.5191	0.2038	0.2927	0.0567	0.5489
English	0.5897	0.4895	0.2278	0.3109	0.0173	0.5300

Tabella 4.30: Caso 1C: Metriche tramite l'uso di una rete RNN

Language	Accuracy	Precision	Recall	F1	Cohens kappa	ROC AUC
Italian	0.7119	0.7910	0.4964	0.6099	0.4231	0.8485
English	0.6699	0.5129	0.4708	0.4909	0.5743	0.5604

Tabella 4.31: Caso 1C: Metriche tramite l'uso di una rete CNN

Language	Accuracy	Precision	Recall	F1	Cohens kappa	ROC AUC
Italian	0.6704	0.7315	0.6021	0.6605	0.6370	0.7837
English	0.7481	0.8304	0.6748	0.7446	0.8453	0.8520

Tabella 4.32: Caso 1C: Metriche tramite l'uso di una rete RNN+CNN

Lang	Vectorizer	C	Acc.	Prec.	Recall	F1	MAE	MSE
IT	N-Gram	0.15	0.7131	0.9397	0.8503	0.8928	0.0506	0.0506
IT	TF-IDF	0.65	0.6987	0.9271	0.8688	0.8970	0.0532	0.0532
EN	N-Gram	0.70	0.7293	0.9156	0.8600	0.8869	0.0038	0.0038
EN	TF-IDF	0.80	0.7399	0.9148	0.8537	0.8832	0.0101	0.0101

Tabella 4.33: Caso 1D: Metriche tramite l'uso della LogisticRegression per il training con il solver newtoniano

un modello più generalizzabile rispetto a tutti i precedenti casi. Nonostante le metriche siano peggiori rispetto ai casi 1 e 2, è stato ottenuto un risultato che è più facile da generalizzare rispetto a quelli generati dagli altri modelli semantici. L'uso

Lang	Vectorizer	C	Acc.	Prec.	Recall	F1	MAE	MSE
IT	N-Gram	0.20	0.6998	0.9231	0.8450	0.8823	0.0473	0.0473
IT	TF-IDF	0.60	0.7014	0.9204	0.8498	0.8837	0.0534	0.0534
EN	N-Gram	0.75	0.7006	0.9194	0.8716	0.8847	0.0048	0.0048
EN	TF-IDF	0.75	0.6974	0.9086	0.8711	0.8949	0.0117	0.0117

Tabella 4.34: Caso 1D: Metriche tramite l'uso della SVM per il training con kernel di tipo lineare

Lang	Vectorizer	Accuracy	Precision	Recall	F1	MAE	MSE
IT	N-Gram	0.6534	0.7951	0.8261	0.8103	0.0672	0.0672
IT	TF-IDF	0.6498	0.8027	0.8205	0.8115	0.0654	0.0654
EN	N-Gram	0.6573	0.8432	0.8238	0.8334	0.0124	0.0124
EN	TF-IDF	0.6604	0.8482	0.8210	0.8343	0.0137	0.0137

Tabella 4.35: Caso 1D: Metriche tramite l'uso del DecisionTree per il training con il criterion gini e best splitter

Lang	Vectorizer	Trees	Accuracy	Precision	Recall	F1	MAE	MSE
IT	N-Gram	150	0.6912	0.8857	0.8632	0.8743	0.0597	0.0573
IT	TF-IDF	150	0.7002	0.8801	0.8610	0.8704	0.0601	0.0592
EN	N-Gram	200	0.6924	0.8732	0.8773	0.8752	0.0086	0.0086
EN	TF-IDF	200	0.6857	0.8801	0.8719	0.8760	0.0091	0.0091

Tabella 4.36: Caso 1D: Metriche tramite l'uso del RandomForest per il training con il criterion gini

Language	Accuracy	Precision	Recall	F1	Cohens kappa	ROC AUC
Italian	0.5103	0.4197	0.6912	0.5223	0.2634	0.5478
English	0.5012	0.4235	0.2410	0.3072	0.0678	0.4639

Tabella 4.37: Caso 1D: Metriche tramite l'uso di una rete FFNN con l'uso della TF-IDF al posto del modello GloVe

di un modello non binario ma multiclasse porta come lavoro futuro ad una profonda analisi delle recensioni per saper riconoscere non solo un sentimento generico (come

Language	Accuracy	Precision	Recall	F1	Cohens kappa	ROC AUC
Italian	0.6921	0.6843	0.6626	0.6733	0.5930	0.7400
English	0.8036	0.8106	0.7799	0.7950	0.7304	0.8523

Tabella 4.38: Caso 1D: Metriche tramite l'uso di una rete FFNN

Language	Accuracy	Precision	Recall	F1	Cohens kappa	ROC AUC
Italian	0.3456	0.7923	0.1567	0.2617	0.1540	0.1843
English	0.3509	0.8345	0.1432	0.2445	0.2468	0.2137

Tabella 4.39: Caso 1D: Metriche tramite l'uso di una rete LSTM

Language	Accuracy	Precision	Recall	F1	Cohens kappa	ROC AUC
Italian	0.5712	0.5008	0.1832	0.2683	0.0511	0.5321
English	0.5749	0.4359	0.2114	0.2847	0.0094	0.5020

Tabella 4.40: Caso 1D: Metriche tramite l'uso di una rete RNN

Language	Accuracy	Precision	Recall	F1	Cohens kappa	ROC AUC
Italian	0.6967	0.7823	0.4634	0.5820	0.4029	0.8534
English	0.6341	0.5236	0.4251	0.4692	0.5408	0.6027

Tabella 4.41: Caso 1D: Metriche tramite l'uso di una rete CNN

Language	Accuracy	Precision	Recall	F1	Cohens kappa	ROC AUC
Italian	0.6506	0.7104	0.5601	0.6264	0.6127	0.7645
English	0.7124	0.8222	0.6395	0.7194	0.8234	0.8628

Tabella 4.42: Caso 1D: Metriche tramite l'uso di una rete RNN+CNN

per il positivo e negativo), ma uno nello specifico, come per esempio gioia, sarcasmo, rabbia, e così via.

Lang	Vectorizer	C	Acc.	Prec.	Recall	F1	MAE	MSE
IT	N-Gram	0.15	0.7734	0.9134	0.8537	0.8825	0.0498	0.0498
IT	TF-IDF	0.60	0.7627	0.9221	0.8633	0.8917	0.0507	0.0507
EN	N-Gram	0.60	0.7598	0.9344	0.8670	0.8994	0.0035	0.0035
EN	TF-IDF	0.95	0.7704	0.9304	0.8813	0.9052	0.0081	0.0081

Tabella 4.43: Caso 1E: Metriche tramite l'uso della LogisticRegression per il training con il solver newtoniano

Lang	Vectorizer	C	Acc.	Prec.	Recall	F1	MAE	MSE
IT	N-Gram	0.30	0.7713	0.9021	0.8578	0.8794	0.0450	0.0450
IT	TF-IDF	0.40	0.7692	0.9137	0.8536	0.8826	0.0496	0.0496
EN	N-Gram	0.80	0.7736	0.8992	0.8690	0.8838	0.0051	0.0051
EN	TF-IDF	0.80	0.7798	0.9008	0.8574	0.8786	0.0107	0.0107

Tabella 4.44: Caso 1E: Metriche tramite l'uso della SVM per il training con kernel di tipo lineare

Lang	Vectorizer	Accuracy	Precision	Recall	F1	MAE	MSE
IT	N-Gram	0.6884	0.8874	0.8731	0.8802	0.0597	0.0597
IT	TF-IDF	0.6872	0.8988	0.8769	0.8877	0.0604	0.0604
EN	N-Gram	0.6831	0.8821	0.8770	0.8795	0.0091	0.0091
EN	TF-IDF	0.6902	0.8860	0.8736	0.8798	0.0084	0.0084

Tabella 4.45: Caso 1E: Metriche tramite l'uso del DecisionTree per il training con il criterion gini e best splitter

Lang	Vectorizer	Trees	Accuracy	Precision	Recall	F1	MAE	MSE
IT	N-Gram	150	0.7256	0.9054	0.8558	0.8799	0.0500	0.0500
IT	TF-IDF	150	0.7276	0.8955	0.8427	0.8683	0.0493	0.0493
EN	N-Gram	100	0.7287	0.8947	0.8487	0.8711	0.0044	0.0044
EN	TF-IDF	150	0.7233	0.9145	0.8600	0.8864	0.0101	0.0101

Tabella 4.46: Caso 1E: Metriche tramite l'uso del RandomForest per il training con il criterion gini

Language	Accuracy	Precision	Recall	F1	Cohens kappa	ROC AUC
Italian	0.5675	0.5144	0.6971	0.5920	0.2860	0.5323
English	0.5658	0.5169	0.5383	0.5274	0.2669	0.4787

Tabella 4.47: Caso 1E: Metriche tramite l'uso di una rete FFNN con l'uso della TF-IDF al posto del modello GloVe

Language	Accuracy	Precision	Recall	F1	Cohens kappa	ROC AUC
Italian	0.7035	0.7078	0.7016	0.7047	0.6026	0.7719
English	0.8150	0.8011	0.7840	0.7925	0.7396	0.8544

Tabella 4.48: Caso 1E: Metriche tramite l'uso di una rete FFNN

Language	Accuracy	Precision	Recall	F1	Cohens kappa	ROC AUC
Italian	0.3718	0.7691	0.2338	0.3586	0.2154	0.2300
English	0.3660	0.7520	0.2267	0.3484	0.2689	0.2454

Tabella 4.49: Caso 1E: Metriche tramite l'uso di una rete LSTM

Language	Accuracy	Precision	Recall	F1	Cohens kappa	ROC AUC
Italian	0.6119	0.5079	0.2460	0.3315	0.1730	0.5584
English	0.6266	0.5310	0.2699	0.3579	0.0529	0.5456

Tabella 4.50: Caso 1E: Metriche tramite l'uso di una rete RNN

Language	Accuracy	Precision	Recall	F1	Cohens kappa	ROC AUC
Italian	0.6974	0.7881	0.5167	0.6242	0.4863	0.8520
English	0.7073	0.6015	0.4991	0.5455	0.5885	0.5997

Tabella 4.51: Caso 1E: Metriche tramite l'uso di una rete CNN

Language	Accuracy	Precision	Recall	F1	Cohens kappa	ROC AUC
Italian	0.6871	0.7219	0.6142	0.6637	0.6485	0.7656
English	0.7379	0.8199	0.6831	0.7453	0.8504	0.8157

Tabella 4.52: Caso 1E: Metriche tramite l'uso di una rete RNN+CNN

Lang	Vectorizer	C	Acc.	Prec.	Recall	F1	MAE	MSE
IT	N-Gram	0.15	0.7601	0.9241	0.8720	0.8973	0.0479	0.0479
IT	TF-IDF	0.05	0.7691	0.9247	0.8876	0.9058	0.0472	0.0472
EN	N-Gram	0.65	0.7888	0.9387	0.8815	0.9092	0.0034	0.0034
EN	TF-IDF	0.95	0.7700	0.9194	0.8763	0.8973	0.0091	0.0091

Tabella 4.53: Caso 1F: Metriche tramite l'uso della LogisticRegression per il training con il solver newtoniano

Lang	Vectorizer	C	Acc.	Prec.	Recall	F1	MAE	MSE
IT	N-Gram	0.20	0.7841	0.9071	0.8817	0.8942	0.0469	0.0469
IT	TF-IDF	0.15	0.7869	0.9101	0.8900	0.8999	0.0483	0.0483
EN	N-Gram	0.50	0.7968	0.9188	0.8801	0.8990	0.0047	0.0047
EN	TF-IDF	0.70	0.7888	0.9301	0.8859	0.9075	0.0096	0.0096

Tabella 4.54: Caso 1F: Metriche tramite l'uso della SVM per il training con kernel di tipo lineare

Lang	Vectorizer	Accuracy	Precision	Recall	F1	MAE	MSE
IT	N-Gram	0.7071	0.9011	0.8970	0.8990	0.0584	0.0596
IT	TF-IDF	0.7029	0.8906	0.8948	0.8927	0.0596	0.0577
EN	N-Gram	0.7037	0.8992	0.8811	0.8900	0.0087	0.0083
EN	TF-IDF	0.7089	0.9088	0.8900	0.8993	0.0091	0.0091

Tabella 4.55: Caso 1F: Metriche tramite l'uso del DecisionTree per il training con il criterion gini e best splitter

Lang	Vectorizer	Trees	Accuracy	Precision	Recall	F1	MAE	MSE
IT	N-Gram	150	0.7356	0.9160	0.8632	0.8888	0.0511	0.0511
IT	TF-IDF	150	0.7249	0.9158	0.8644	0.8894	0.0499	0.0499
EN	N-Gram	100	0.7255	0.9400	0.8828	0.9105	0.0047	0.0047
EN	TF-IDF	150	0.7401	0.9376	0.8811	0.9084	0.0091	0.0091

Tabella 4.56: Caso 1F: Metriche tramite l'uso del RandomForest per il training con il criterion gini

Language	Accuracy	Precision	Recall	F1	Cohens kappa	ROC AUC
Italian	0.5771	0.5280	0.6917	0.5989	0.2825	0.5419
English	0.5683	0.5369	0.5426	0.5397	0.2526	0.4679

Tabella 4.57: Caso 1F: Metriche tramite l'uso di una rete FFNN con l'uso della TF-IDF al posto del modello GloVe

Language	Accuracy	Precision	Recall	F1	Cohens kappa	ROC AUC
Italian	0.7249	0.7116	0.6901	0.7007	0.6017	0.7433
English	0.8436	0.8401	0.8166	0.8282	0.7422	0.8564

Tabella 4.58: Caso 1F: Metriche tramite l'uso di una rete FFNN

Language	Accuracy	Precision	Recall	F1	Cohens kappa	ROC AUC
Italian	0.3654	0.7609	0.2329	0.3566	0.21	0.2125
English	0.3889	0.7700	0.2551	0.3832	0.26	0.2367

Tabella 4.59: Caso 1F: Metriche tramite l'uso di una rete LSTM

Language	Accuracy	Precision	Recall	F1	Cohens kappa	ROC AUC
Italian	0.6219	0.5429	0.2617	0.3532	0.0512	0.5204
English	0.6266	0.5564	0.2599	0.3543	0.0197	0.5143

Tabella 4.60: Caso 1F: Metriche tramite l'uso di una rete RNN

Language	Accuracy	Precision	Recall	F1	Cohens kappa	ROC AUC
Italian	0.7218	0.7821	0.5370	0.6368	0.4335	0.8585
English	0.7183	0.6467	0.5453	0.5917	0.5429	0.5240

Tabella 4.61: Caso 1F: Metriche tramite l'uso di una rete CNN

Language	Accuracy	Precision	Recall	F1	Cohens kappa	ROC AUC
Italian	0.6990	0.7515	0.6237	0.6817	0.6485	0.7821
English	0.7401	0.8356	0.6681	0.7425	0.8323	0.8597

Tabella 4.62: Caso 1F: Metriche tramite l'uso di una rete RNN+CNN

Lang	Vectorizer	C	Acc.	Prec.	Recall	F1	MAE	MSE
IT	N-Gram	0.15	0.8585	0.8997	0.8068	0.8507	0.0090	0.0090
IT	TF-IDF	0.75	0.8665	0.9080	0.8156	0.8592	0.0101	0.0101
EN	N-Gram	0.85	0.8694	0.8821	0.8262	0.8532	0.0037	0.0037
EN	TF-IDF	0.95	0.8621	0.8781	0.8128	0.8442	0.0040	0.0040

Tabella 4.63: Caso 2: Metriche tramite l'uso della LogisticRegression per il training con il solver newtoniano

Lang	Vectorizer	C	Acc.	Prec.	Recall	F1	MAE	MSE
IT	N-Gram	0.05	0.8710	0.8998	0.8351	0.8661	0.0098	0.0098
IT	TF-IDF	0.15	0.8614	0.9301	0.7815	0.8493	0.0094	0.0094
EN	N-Gram	0.05	0.8720	0.8876	0.8262	0.8558	0.0040	0.0040
EN	TF-IDF	0.70	0.8627	0.8783	0.8139	0.8449	0.0021	0.0021

Tabella 4.64: Caso 2: Metriche tramite l'uso della SVM per il training con kernel di tipo lineare

Lang	Vectorizer	Accuracy	Precision	Recall	F1	MAE	MSE
IT	N-Gram	0.8158	0.8131	0.9295	0.8674	0.0507	0.0509
IT	TF-IDF	0.8185	0.8099	0.9194	0.8612	0.0537	0.0521
EN	N-Gram	0.8294	0.8295	0.9292	0.9294	0.0036	0.0034
EN	TF-IDF	0.8229	0.8214	0.9143	0.8654	0.0021	0.0022

Tabella 4.65: Caso 2: Metriche tramite l'uso del DecisionTree per il training con il criterion gini e best splitter

Lang	Vectorizer	Trees	Acc.	Prec.	Recall	F1	MAE	MSE
IT	N-Gram	100	0.8111	0.8285	0.9138	0.8691	0.0203	0.0294
IT	TF-IDF	150	0.8217	0.8292	0.9451	0.8834	0.0537	0.0434
EN	N-Gram	100	0.8531	0.8441	0.9359	0.8876	0.0073	0.0089
EN	TF-IDF	200	0.8586	0.8393	0.9224	0.8789	0.0079	0.0091

Tabella 4.66: Caso 2: Metriche tramite l'uso del RandomForest per il training con il criterion gini

Language	Accuracy	Precision	Recall	F1	Cohens kappa	ROC AUC
Italian	0.5689	0.5841	0.4850	0.5300	0.1382	0.5871
English	0.5099	0.5052	0.7554	0.6055	0.0217	0.5199

Tabella 4.67: Caso 2: Metriche tramite l'uso di una rete FFNN con l'uso della TF-IDF al posto del modello GloVe

Language	Accuracy	Precision	Recall	F1	Cohens kappa	ROC AUC
Italian	0.7532	0.7499	0.7616	0.7557	0.5064	0.8263
English	0.7651	0.7743	0.7454	0.7596	0.5300	0.8453

Tabella 4.68: Caso 2: Metriche tramite l'uso di una rete FFNN

Language	Accuracy	Precision	Recall	F1	Cohens kappa	ROC AUC
Italian	0.3676	0.3797	0.3479	0.3631	0.5354	0.2450
English	0.3691	0.3589	0.3861	0.3720	0.5384	0.2488

Tabella 4.69: Caso 2: Metriche tramite l'uso di una rete LSTM

Language	Accuracy	Precision	Recall	F1	Cohens kappa	ROC AUC
Italian	0.6011	0.5011	0.6381	0.5614	0.0000	0.4996
English	0.5979	0.4979	0.6238	0.5538	0.0000	0.4996

Tabella 4.70: Caso 2: Metriche tramite l'uso di una rete RNN

Language	Accuracy	Precision	Recall	F1	Cohens kappa	ROC AUC
Italian	0.4989	0.4969	0.5288	0.5123	0.000	0.4993
English	0.5021	0.6118	0.5787	0.5948	0.000	0.5000

Tabella 4.71: Caso 2: Metriche tramite l'uso di una rete CNN

Language	Accuracy	Precision	Recall	F1	Cohens kappa	ROC AUC
Italian	0.7977	0.8070	0.7837	0.7952	0.5954	0.8770
English	0.8171	0.8197	0.8111	0.8154	0.6342	0.8985

Tabella 4.72: Caso 2: Metriche tramite l'uso di una rete RNN+CNN

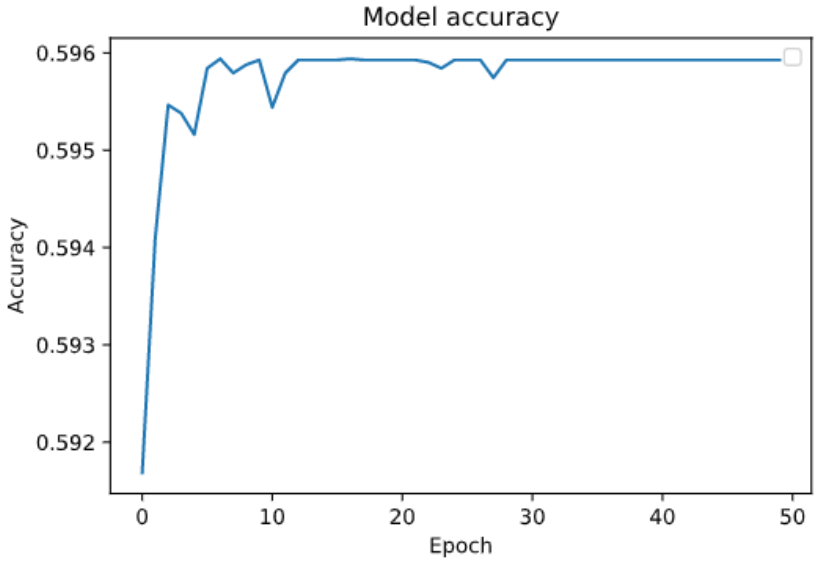

Figura 4.5: Caso 2: accuratezza di una RNN+CNN per dataset in lingua italiana e inglese in base al numero di epoche

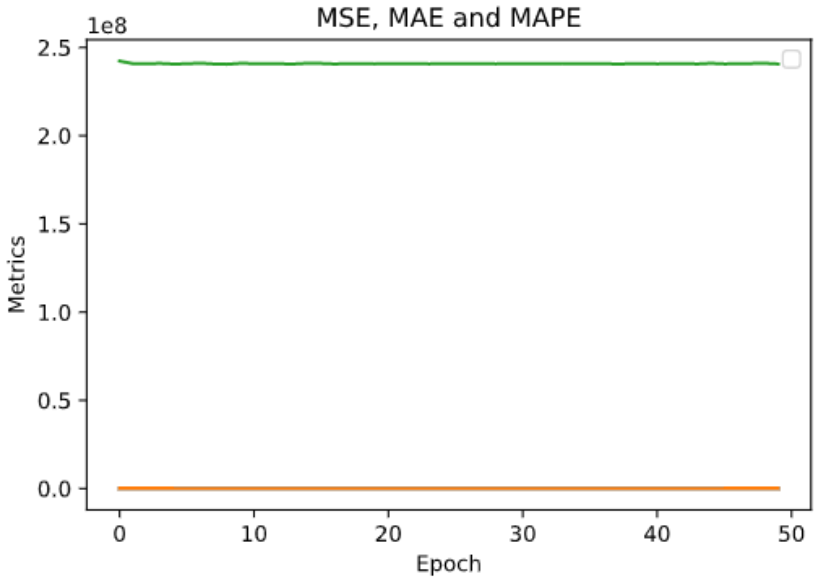

Figura 4.6: Caso 2: andamento errore in base al numero di epoche per una RNN+CNN

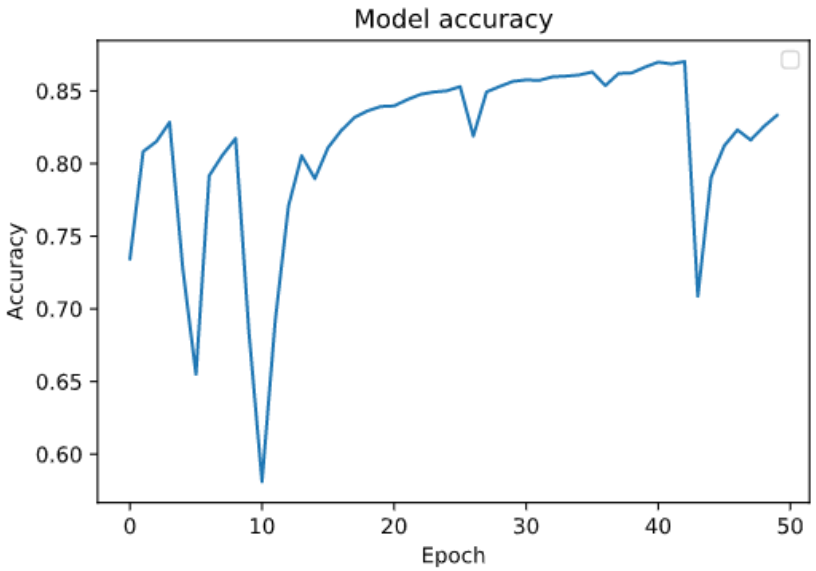

Figura 4.7: Caso 2: accuratezza di una RNN+CNN per dataset in lingua italiana e inglese in base al numero di epoche

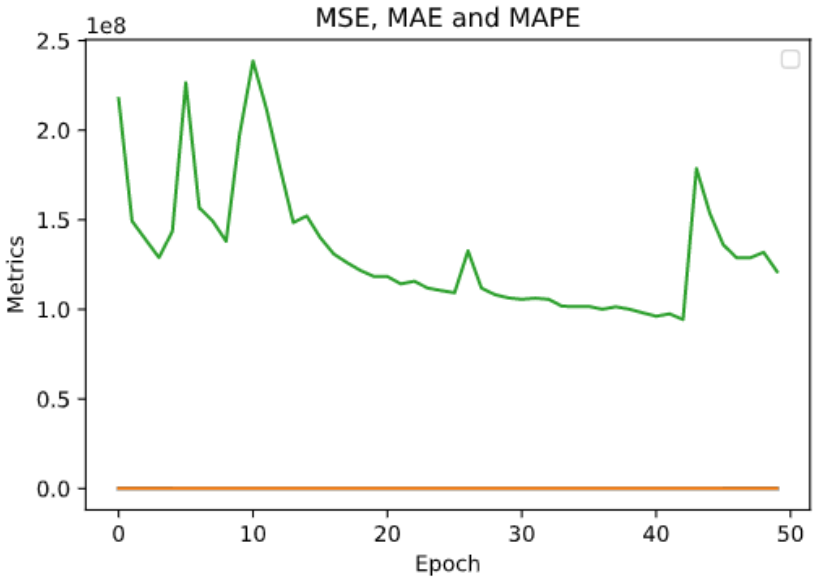

Figura 4.8: Caso 2: andamento errore in base al numero di epoche per una RNN+CNN

Lang	Vectorizer	C	Acc.	Prec.	Recall	F1	MAE	MSE
IT	N-Gram	0.05	0.6016	0.6007	0.6016	0.5979	0.0132	0.0369
IT	TF-IDF	0.75	0.6046	0.6044	0.6046	0.6018	0.0067	0.0198
EN	N-Gram	0.75	0.6473	0.6515	0.6473	0.6484	0.0034	0.0103
EN	TF-IDF	0.95	0.6445	0.6472	0.6445	0.6446	0.0036	0.0032

Tabella 4.73: Caso 3: Metriche tramite l'uso della LogisticRegression per il training con il solver newtoniano

Lang	Vectorizer	C	Acc.	Prec.	Recall	F1	MAE	MSE
IT	N-Gram	0.05	0.6028	0.6098	0.6028	0.6051	0.0060	0.0209
IT	TF-IDF	0.15	0.5899	0.6077	0.5899	0.5949	0.0094	0.0309
EN	N-Gram	0.05	0.6547	0.6652	0.6547	0.6575	0.0065	0.0097
EN	TF-IDF	0.70	0.6513	0.6585	0.6513	0.6532	0.0044	0.0083

Tabella 4.74: Caso 3: Metriche tramite l'uso della SVM per il training con kernel di tipo lineare

Lang	Vectorizer	Accuracy	Precision	Recall	F1	MAE	MSE
IT	N-Gram	0.5237	0.5358	0.5237	0.5285	0.0048	0.0109
IT	TF-IDF	0.5156	0.5175	0.5156	0.5158	0.0009	0.0281
EN	N-Gram	0.5274	0.5296	0.5274	0.5282	0.0094	0.0281
EN	TF-IDF	0.5173	0.5206	0.5173	0.5182	0.0060	0.0175

Tabella 4.75: Caso 3: Metriche tramite l'uso del DecisionTree per il training con il criterion gini e best splitter

Lang	Vectorizer	Trees	Accuracy	Precision	Recall	F1	MAE	MSE
IT	N-Gram	100	0.5909	0.5813	0.5909	0.5847	0.0226	0.0500
IT	TF-IDF	150	0.5983	0.5866	0.5983	0.5882	0.0158	0.0404
EN	N-Gram	100	0.6005	0.6186	0.6005	0.6033	0.0049	0.0108
EN	TF-IDF	150	0.5904	0.5882	0.5904	0.5891	0.0016	0.0036

Tabella 4.76: Caso 3: Metriche tramite l'uso del RandomForest per il training con il criterion gini

Language	Accuracy	Precision	Recall	F1	Cohens kappa
Italian	0.3320	0.1102	0.3320	0.1655	0.0000
English 0.4428	0.3221	0.2086	0.3221	0.1642	-0.0026

Tabella 4.77: Caso 3: Metriche tramite l'uso di una rete FFNN con l'uso della TF-IDF al posto del modello GloVe

Language	Accuracy	Precision	Recall	F1	Cohens kappa
Italian	0.3998	0.3926	0.3998	0.2869	0.1012
English	0.3477	0.3891	0.3477	0.2109	0.0349

Tabella 4.78: Caso 3: Metriche tramite l'uso di una rete FFNN

Language	Accuracy	Precision	Recall	F1	Cohens kappa
Italian	0.3801	0.3970	0.3801	0.2562	0.0719
English	0.3339	0.3642	0.3339	0.1825	0.0148

Tabella 4.79: Caso 3: Metriche tramite l'uso di una rete LSTM

Language	Accuracy	Precision	Recall	F1	Cohens kappa
Italian	0.3320	0.1103	0.3320	0.1655	0.0000
English	0.3237	0.1048	0.3237	0.1583	0.0000

Tabella 4.80: Caso 3: Metriche tramite l'uso di una rete RNN

Language	Accuracy	Precision	Recall	F1	Cohens kappa
Italian	0.3648	0.4125	0.3648	0.2295	0.0489
English	0.3213	0.1667	0.3213	0.1614	-0.0038

Tabella 4.81: Caso 3: Metriche tramite l'uso di una rete CNN

Language	Accuracy	Precision	Recall	F1	Cohens kappa
Italian	0.3990	0.3969	0.3990	0.2864	0.1000
English	0.4558	0.5004	0.4558	0.4771	0.0468

Tabella 4.82: Caso 3: Metriche tramite l'uso di una rete RNN+CNN

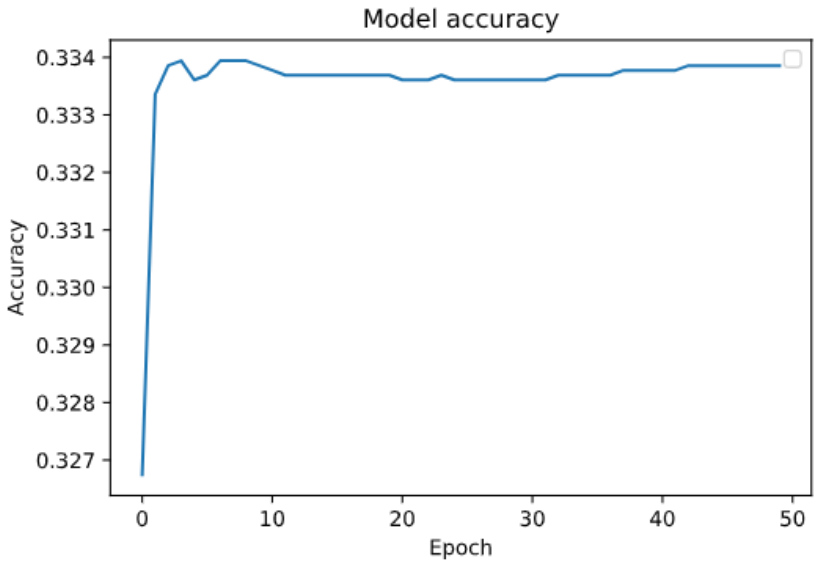

Figura 4.9: Caso 3: accuratezza di una RNN+CNN per dataset in lingua italiana e inglese in base al numero di epoche

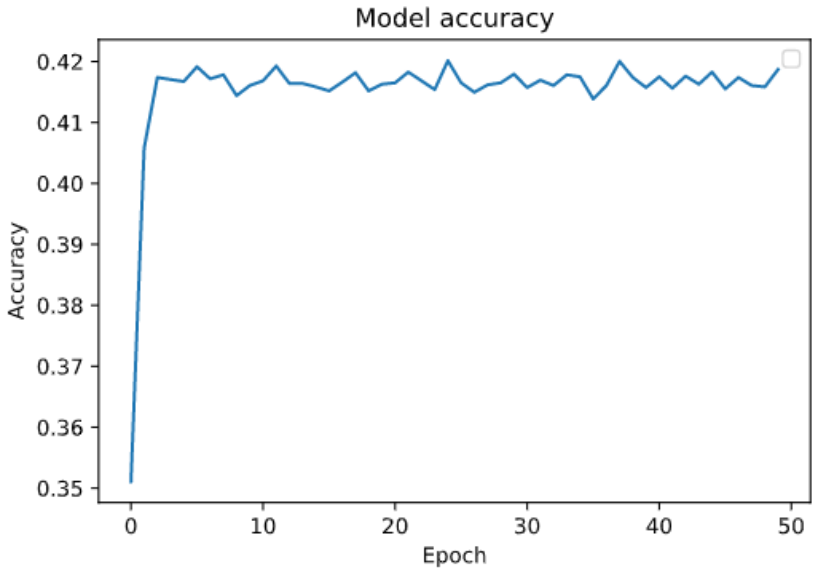

Figura 4.10: Caso 3: accuratezza di una RNN+CNN per dataset in lingua italiana e inglese in base al numero di epoche

Capitolo 5

Conclusioni

Da quanto si evince dai risultati sperimentali, il pre processing delle recensioni ha aiutato nel processo di training del classificatore. Tuttavia, occorre non solo adottare dei criteri di pulizia del dataset coerenti con il contesto in cui è utilizzato il modello, ma anche studiare un metodo per la creazione di un modello multi-classe per non incorrere nell'overfitting.

I metodi trandizionali di machine learning hanno ottenuto degli ottimi risultati in tutti i casi trattati, ma rispetto alle architetture di reti neurali, non sono in grado di comprendere un contesto se non utilizzando dei word embedding con tecniche di unsupervised learning. I modelli GloVe hanno senz'altro ottenuto delle ottime metriche utilizzando la rete neurale più performante e coerente con il metodo di applicazione.

Dato che il contesto di applicazione è influente con il processo di apprendimento del classificatore, un lavoro futuro potrebbe essere l'applicazione di approcci NER per etichettare il dataset prima di procedere con il training del modello. Facendo così, il classificatore potrebbe imparare meglio l'importanza e la rilevanza di una parola all'interno di una recensione, senza dover utilizzare del pre processing, perché era stato utilizzato per rimuovere le parole che non esprimessero dei sentimenti.

Lo studio del caso multiclasse ha portato alla luce dei fattori che potrebbero essere di studio futuro: il riconoscimento di un testo come positivo e negativo sono delle categorie tutto sommato generiche, ma il riconoscimento di un testo volgare o di una persona frustrata o euforica sono problemi di complessità maggiore. Con l'ausilio di un dataset corposo e bilanciato, potrebbe essere possibile approfondire lo studio della classificazione di una recensione per produrre un modello statistico più complesso, ma applicabile come se fosse un detector e un servizio online offerto.

Da come si può vedere anche dalle metriche prestazionali, è possibile concludere che non solo la giusta architettura di reti neurali influisce sul risultato ottenuto, ma anche l'uso di un modello GloVe è determinante, infatti spesso (se non sempre) le performance migliori sono state ottenute in lingua inglese. Nonostante non sia

stato rilasciato il codice sorgente, potrebbe essere una fonte di approfondimento lo studio del modello GloVe italiano proveniente dall'HLT del CNR, in modo tale da valutarne il suo utilizzo in ambiti NER-NLP sia accademico che di produzione[41].

Un predittore di score data una recensione potrebbe essere un lavoro che potrebbe essere svolto in futuro: la sua applicazione potrebbe essere la determinazione di un punteggio dato un testo, il quale dovrà essere sempre limitato in lunghezza per sia il preprocessing che per l'introduzione all'interno di una rete neurale. Il modello prodotto potrebbe avere un'applicazione pratica anche tramite uno studio sul metodo di tagging.

Un altro lavoro futuro potrebbe essere la realizzazione di un'architettura REST che, una volta ottenuto un modello semantico con metriche prestazionali valide, possa fungere come servizio. Una problematica da affrontare non è solamente la realizzazione del classificatore, ma anche come interagire con il modello caricato in RAM (come molti framework di machine learning fanno). Una costruzione di un motore di ricerca vero e proprio potrebbe superare queste difficoltà (come per esempio l'uso di Apache Solr) e introdurrne delle altre per via della memorizzazione del vettore risultante e della ricerca di similarità tra una parola e/o recensione.

I campi di applicazione della sentiment analysis sono molteplici e le sue potenzialità sono state messe in risalto dall'ascesa dei social network. L'utilizzo di framework per deep learning come Keras permettono una maggiore scalabilità nella creazione del classificatore e ancora oggi molti studi sono in fase sperimentale in ambito NLP[39]. Questo studio si è focalizzato principalmente sulle recensioni, ma non vengono escluse le piattaforme web, le quali sono ottime fonti per il reperimento del dataset. Molti studi condotti fino ad ora hanno utilizzato le API di Twitter per creare un classificatore per sentiment analysis, ma non ci si è soffermati sull'applicazione delle reti neurali e dei modelli di rappresentazione vettoriale delle parole[44].

Con questo studio condotto fino ad oggi, si vuole aprire una frontiera di sviluppo di modelli come GloVe, Word2Vec e fastText[45] di Facebook, non solo per dare una rappresentazione delle parole in base al contesto, ma anche per aiutare le reti neurali come classificatori nell'ambito della sentiment analysis.

Appendice A

Uso di Go in data retrieval e text mining

```
func getCurrentStructureOrDepartment(selfUrl string, doc *goquery.Document, isStructure bool) (Structure, Department) {
    var tempReview Review
    var currentStructure Structure
    var currentDepartment Department
    if isStructure {
        currentStructure.self = selfUrl
    } else {
        currentDepartment.self = selfUrl
    }
    // name
    doc.Find(".contentheading").Each(func(i int, s *goquery.Selection) {
        if isStructure {
            currentStructure.name = s.Find("span").Text()
        } else {
            currentDepartment.name = s.Find("span").Text()
        }
    })
    // global ratings and reviews raiting
    doc.Find(".jrCol.jrRatingValue").Each(func(i int, s *goquery.Selection) {
        if i < 5 { // global ratings
            temp := s.Text()
            temp = strings.TrimSpace(temp)
            tokens := strings.Split(temp, "(")
```

```
24              currentDepartment, currentStructure =
    updateDepartmentStructureRatings(i, currentDepartment,
    currentStructure, isStructure)
25          } else { // review rating
26              currentDepartment, currentStructure =
    updateDepartmentStructureReviews(i, currentDepartment,
    currentStructure, isStructure)
27          }
28      })
29      // reviews
30      doc.Find(".description.jrReviewComment").Each(func(i int, s *
    goquery.Selection) {
31          tempReviewText := strings.TrimSpace(s.Text())
32          space := regexp.MustCompile(`\s+`)
33          tempReviewText = space.ReplaceAllString(tempReviewText, " ")
34          if isStructure {
35              currentStructure.reviews[i].text = tempReviewText
36          } else {
37              currentDepartment.reviews[i].text = tempReviewText
38          }
39      })
40      return currentStructure, currentDepartment
41  }
```

Funzione per l'uso goquery nel data retrieval di un reparto o di una struttura sanitaria

```
1       var waitGroup sync.WaitGroup
2       waitGroup.Add(cleaningRowsNumber)
3       for index, row := range sourceRows {
4           if index == 0 || index == (departmentRowsNumber+1) {
5               continue // skip header files
6           }
7           review := ""
8           score := 0.0
9           if index <= departmentRowsNumber {
10              review = row[4] // department review-score file
11              score, _ = strconv.ParseFloat(row[5], 64)
12          } else {
13              review = row[3] // structure review-score file
14              score, _ = strconv.ParseFloat(row[4], 64)
15          }
16          go func(index int, row string, score float64) {
17              defer waitGroup.Done()
18              row = removeSpecialChars(row)
19              rowTokens := strings.Split(row, " ")
20              rowTokens = splitAndRemove(rowTokens, toRemove, len(
    toRemove), true)
```

```
21              if removingDoctors {
22                  for _, name := range titleName {
23                      foundToken := false
24                      countNames := 0
25                      for indexToken, _ := range rowTokens {
26                          if rowTokens[indexToken] == "" || rowTokens[
    indexToken] == " " {
27                              continue
28                          }
29                          if foundToken && countNames < 2 && len(
    rowTokens[indexToken]) > 2 && unicode.IsUpper([]rune(rowTokens[
    indexToken])[0]) {
30                              countNames++
31                              rowTokens[indexToken] = ""
32                          }
33                          if rowTokens[indexToken] == name {
34                              foundToken = true
35                              countNames = 0
36                              rowTokens[indexToken] = ""
37                          }
38                      }
39                  }
40              }
41              rowTokens = splitAndRemove(rowTokens, pathologiesArray,
    pathologiesSize, false)
42              rowTokens = splitAndRemove(rowTokens, regionsCitiesArray,
     regionsCitiesSize, false)
43              row = strings.Join(rowTokens, " ")
44              row = strings.ToLower(row)
45              extraSpaces := regexp.MustCompile(`\s+`)
46              row = extraSpaces.ReplaceAllString(row, " ")
47              if index <= departmentRowsNumber {
48                  cleaningResultReviews[index-1] = row
49                  cleaningResultScores[index-1] = score
50              } else {
51                  cleaningResultReviews[index-2] = row
52                  cleaningResultScores[index-2] = score
53              }
54          }(index, review, score)
55      }
56      waitGroup.Wait()
```

`Uso delle goroutines per multithreaded data cleaning`

Appendice B

imblearn, NLTK e analisi del linguaggio

```
1
2 from imblearn.over_sampling import ADASYN
3
4 def adasyn_bilancing(xtrain, ytrain):
5     # no need for splitting, already done
6     ad = ADASYN(random_state=777, ratio = 1.0)
7     ad_xtrain, ad_ytrain = ad.fit_sample(xtrain, ytrain)
8     return ad_xtrain, ad_ytrain
```

Uso di imblearn per ADASYN

```
1
2 from nltk.corpus import stopwords
3
4 def remove_stop_words(corpus, lang):
5     removed_stop_words = []
6     for review in corpus:
7         if lang == "IT":
8             removed_stop_words.append(
9                 ' '.join([word for word in review.split() if word not
      in stopwords.words('italian')])
10        else:
11            removed_stop_words.append(
12                ' '.join([word for word in review.split() if word not
      in stopwords.words('english')])
13    return removed_stop_words
```

Rimozione stopwords da un corpo di testo tramite NLTK

```
1
2 from nltk.stem.snowball import ItalianStemmer, PorterStemmer
3
4 def get_stemmed_text(corpus, lang):
5     if lang == "IT":
6         stemmer = ItalianStemmer()
7     else:
8         stemmer = PorterStemmer()
9     return [' '.join([stemmer.stem(word) for word in review.split()])
      for review in corpus]
```

Operazione di stemming di un corpo di testo tramite NLTK

```
1
2 def vectorization_count(range_bottom, range_top, reviews_train,
    reviews_test):
3     cv = CountVectorizer(binary=True, ngram_range=(
4         range_bottom, range_top))
5     cv.fit(reviews_train)
6     X = cv.transform(reviews_train)
7     X_test = cv.transform(reviews_test)
8     return cv, X, X_test
```

Vettorizzazione tramite CountVectorizer in SKLearn

```
1
2 def vectorization_tf_idf(reviews_train, reviews_test):
3     tfidf = TfidfVectorizer()
4     tfidf.fit(reviews_train)
5     X = tfidf.transform(reviews_train)
6     X_test = tfidf.transform(reviews_test)
7     return tfidf, X, X_test
```

Vettorizzazione tramite TfidfVectorizer in SKLearn

Appendice C

SKLearn e Machine Learning

```
1
2  def build_classifier_logistic(range_bottom, range_top, X, target,
       given_C=None):
3      best_C = None
4      best_accuracy = None
5      X_train, X_val, Y_train, Y_val = train_test_split(X, target,
       train_size=0.75)
6      if given_C is not None:
7          best_C = given_C
8          lr = LogisticRegression(C=given_C, solver='newton-cg')
9          lr.fit(X_train, Y_train)
10         best_accuracy = float(accuracy_score(Y_val, lr.predict(X_val)
       ))
11     else:
12         for c in C_ARRAY:
13             lr = LogisticRegression(C=c, solver='newton-cg')
14             lr.fit(X_train, Y_train)
15             accuracy = float(accuracy_score(Y_val, lr.predict(X_val))
       )
16             if (best_C is None and best_accuracy is None) or (
       best_accuracy != None and best_accuracy < accuracy):
17                 best_C = c
18                 best_accuracy = accuracy
19     best_C = round(best_C, 2)
20     best_accuracy = round(best_accuracy, 2)
21     if range_bottom != -1 and range_top != -1:
22         print('Best accuracy {}-{}-gram for C={}: {}'.format(str(
       range_bottom), str(range_top), str(best_C),
```

```
23                                                                    str(
    best_accuracy)))
24      else:
25          print('Best accuracy TF-IDF for C={}: {}'.format(str(best_C),
     str(best_accuracy)))
26      return X_train, X_val, Y_train, Y_val, best_C
```

Logistic Regression usando SKLearn

```
1
2  def build_classifier_linear_svm(X, target, given_C=None):
3      best_C = None
4      best_accuracy = None
5      X_train, X_val, Y_train, Y_val = train_test_split(X, target,
    train_size=0.75)
6      if given_C is not None:
7          best_C = given_C
8          svm = SVC(C=given_C, kernel='linear', probability=True)
9          svm.fit(X_train, Y_train)
10         best_accuracy = float(accuracy_score(Y_val, svm.predict(X_val
    )))
11     else:
12         for c in C_ARRAY:
13             svm = SVC(C=c, kernel='linear', probability=True)
14             svm.fit(X_train, Y_train)
15             accuracy = float(accuracy_score(Y_val, svm.predict(X_val)
    ))
16             if (best_C is None and best_accuracy is None) or (
    best_accuracy != None and best_accuracy < accuracy):
17                 best_C = c
18                 best_accuracy = accuracy
19     best_C = round(best_C, 2)
20     best_accuracy = round(best_accuracy, 2)
21     print("Best accuracy SVM for C={}: {}".format(str(best_C), str(
    best_accuracy)))
22     return X_train, X_val, Y_train, Y_val, best_C
```

Support Vector Machine usando SKLearn

```
1
2  def build_classifier_decision_tree(X, target):
3      for criteria in criterias:
4          for splitter in splitters:
5              X_train, X_val, Y_train, Y_val = train_test_split(X,
    target, train_size=0.75)
6              tree = DecisionTreeClassifier(criterion=criteria,
    splitter=splitter)
```

```
7               tree.fit(X_train, Y_train)
8               accuracy = float(accuracy_score(Y_val, tree.predict(X_val
    )))
9               print("Accuracy Decision tree using gini criteria and
    best splitter: {}".format(accuracy))
10              return X_train, X_val, Y_train, Y_val, tree
```

Decision Tree usando SKLearn

```
1
2 def build_classifier_random_forest(X, target):
3     for criteria in criterias:
4         for trees in range(50, 250, 50):
5             X_train, X_val, Y_train, Y_val = train_test_split(X,
    target, train_size=0.75)
6             forest = RandomForestClassifier(n_estimators=trees,
    criterion=criteria, n_jobs=-1)
7             forest.fit(X_train, Y_train)
8             accuracy = float(accuracy_score(Y_val, forest.predict(
    X_val)))
9             print("Accuracy Random Forest using gini criteria and 100
     trees: {}".format(accuracy))
10            return X_train, X_val, Y_train, Y_val, forest
```

Random Forest usando SKLearn

Appendice D

Keras, TensorFlow e Deep Learning

```
1
2  def MyFFNNWithOutEmbeddings(X_train, Y_train, X_test, Y_test, lang):
3      model = Sequential()
4      model.add(Dense(140, activation='relu'))
5      model.add(Dense(2, activation='sigmoid'))
6      model.compile(optimizer='adam', loss='binary_crossentropy',
7                    metrics=['acc', 'mse', 'mae', 'mape'])
8      history = model.fit(X_train, Y_train, epochs=EPOCHS,
9                          batch_size=BATCH, verbose=1, validation_split
       =SPLIT)
10     score = model.evaluate(X_test, Y_test, verbose=0)
11     print("Metrics: {}".format(model.metrics_names))
12     print("Test Score: {}".format(str(score[0])))
13     print("Test Accuracy: {}".format(str(score[1])))
14     list(map(print, score))
15     name = "ffnn-no-embeddings-{}".format(lang)
16     print_graphs(history, name, False)
17     plot_file_model = "model-ffnn-no-embeddings-{}.png".format(lang)
18     plot_model(model, to_file=plot_file_model)
19     return model
```

FFNN senza GloVe usando Keras

```
1
2  def MyFFNN(X_train, Y_train, X_test, Y_test, vocab_size,
       embedding_matrix, lang):
3      model = Sequential()
```

```
    embedding_layer = Embedding(vocab_size, MAX_LEN, weights=[
                                embedding_matrix], input_length=
    MAX_LEN, trainable=False)
    model.add(embedding_layer)
    model.add(Flatten())
    model.add(Dense(140, activation='relu'))
    model.add(Dense(2, activation='sigmoid'))
    model.compile(optimizer='adam', loss='binary_crossentropy',
                  metrics=['acc', 'mse', 'mae', 'mape'])
    print(model.summary())
    history = model.fit(X_train, Y_train, batch_size=BATCH,
                        epochs=EPOCHS, verbose=1, validation_split=
    SPLIT)
    score = model.evaluate(X_test, Y_test, verbose=1)
    print("Metrics: {}".format(model.metrics_names))
    print("Test Score: {}".format(str(score[0])))
    print("Test Accuracy: {}".format(str(score[1])))
    list(map(print, score))
    name = "ffnn-{}".format(lang)
    print_graphs(history, name, False)
    plot_file_model = "model-ffnn-{}.png".format(lang)
    plot_model(model, to_file=plot_file_model)
    return model
```

FFNN usando Keras

```

def MyCNN(X_train, Y_train, X_test, Y_test, vocab_size,
    embedding_matrix, lang):

    model = Sequential()
    embedding_layer = Embedding(vocab_size, MAX_LEN, weights=[
                                embedding_matrix], input_length=
    MAX_LEN, trainable=False)
    model.add(embedding_layer)
    model.add(Conv1D(FILTERS, KERNELS, activation='relu'))
    model.add(MaxPooling1D(pool_size=2))
    model.add(Flatten())
    model.add(Dense(140, activation='relu'))
    model.add(Dense(2, activation='sigmoid'))
    model.compile(optimizer='adam', loss='binary_crossentropy',
                  metrics=['acc', 'mse', 'mae', 'mape'])
    print(model.summary())
    history = model.fit(X_train, Y_train, batch_size=BATCH,
                        epochs=EPOCHS, verbose=1, validation_split=
    SPLIT)
    score = model.evaluate(X_test, Y_test, verbose=1)
    print("Metrics: {}".format(model.metrics_names))
```

```
    print("Test Score: {}".format(str(score[0])))
    print("Test Accuracy: {}".format(str(score[1])))
    print("Try to print what's inside score:")
    list(map(print, score))
    name = "cnn-multi-{}".format(lang)
    print_graphs(history, name, False)
    plot_file_model = "model-cnn-multi-{}.png".format(lang)
    plot_model(model, to_file=plot_file_model)
    return model
```

CNN usando Keras

```

def MyRNN(X_train, Y_train, X_test, Y_test, vocab_size,
    embedding_matrix, lang):
    model = Sequential()
    embedding_layer = Embedding(vocab_size, MAX_LEN, weights=[
                                    embedding_matrix], input_length=
    MAX_LEN, trainable=False)
    model.add(embedding_layer)
    model.add(SimpleRNN(140))
    model.add(Dense(2, activation='sigmoid'))
    model.compile(optimizer='adam', loss='binary_crossentropy',
                  metrics=['accuracy', 'mse', 'mae', 'mape'])
    print(model.summary())
    history = model.fit(X_train, Y_train, batch_size=BATCH,
                        epochs=EPOCHS, verbose=1)
    score = model.evaluate(X_test, Y_test, verbose=1)
    print("Metrics: {}".format(model.metrics_names))
    print("Test Score: {}".format(str(score[0])))
    print("Test Accuracy: {}".format(str(score[1])))
    print("Try to print what's inside score:")
    list(map(print, score))
    name = "rnn-{}".format(lang)
    print_graphs(history, name, True)
    plot_file_model = "model-rnn-{}.png".format(lang)
    plot_model(model, to_file=plot_file_model)
    return model
```

RNN usando Keras

```

def MyLSTM(X_train, Y_train, X_test, Y_test, vocab_size,
    embedding_matrix, lang):
    model = Sequential()
    embedding_layer = Embedding(vocab_size, MAX_LEN, weights=[
```

```
                                    embedding_matrix], input_length=
    MAX_LEN, trainable=False)
    model.add(embedding_layer)
    model.add(Bidirectional(LSTM(FILTERS)))
    model.add(Dropout(0.50))
    model.add(Dense(2, activation='sigmoid'))
    model.compile(optimizer='adam', loss='binary_crossentropy',
                  metrics=['acc', 'mse', 'mae', 'mape'])
    print(model.summary())
    history = model.fit(X_train, Y_train, batch_size=BATCH,
                        epochs=EPOCHS, verbose=1, validation_split=
    SPLIT)
    score = model.evaluate(X_test, Y_test, verbose=1)
    print("Metrics: {}".format(model.metrics_names))
    print("Test Score: {}".format(str(score[0])))
    print("Test Accuracy: {}".format(str(score[1])))
    print("Try to print what's inside score:")
    list(map(print, score))
    name = "lstm-{}".format(lang)
    print_graphs(history, name, False)
    plot_file_model = "model-lstm-{}.png".format(lang)
    plot_model(model, to_file=plot_file_model)
    return model
```

LSTM usando Keras

```

def MyRNNwithCNN(X_train, Y_train, X_test, Y_test, vocab_size,
    embedding_matrix, lang):
    model = Sequential()
    model.add(Embedding(vocab_size, MAX_LEN, weights=[
              embedding_matrix], input_length=MAX_LEN, trainable=
    False))
    model.add(Conv1D(FILTERS, KERNELS, activation='relu'))
    model.add(MaxPooling1D(pool_size=2))
    model.add(SimpleRNN(140))
    model.add(Dense(2, activation='sigmoid'))
    model.compile(optimizer='adam', loss='binary_crossentropy',
                  metrics=['accuracy', 'mse', 'mae', 'mape'])
    history = model.fit(X_train, Y_train, batch_size=BATCH,
                        epochs=EPOCHS, verbose=1)
    score = model.evaluate(X_test, Y_test, verbose=0)
    print("Metrics: {}".format(model.metrics_names))
    print("Test Score: {}".format(str(score[0])))
    print("Test Accuracy: {}".format(str(score[1])))
    print("Try to print what's inside score:")
    list(map(print, score))
    name = "rnn-cnn-{}".format(lang)
```

```
21      print_graphs(history, name, True)
22      plot_file_model = "model-rnn-cnn-{}.png".format(lang)
23      plot_model(model, to_file=plot_file_model)
24      return model
```

`RNN and CNN usando Keras`

Bibliografia

[1] *Volume of data/information created worldwide from 2010 to 2025*. URL: https://www.statista.com/statistics/871513/worldwide-data-created/ (cit. a p. xvii).

[2] *QSalute - ti aiutiamo a trovare la cura migliore*. URL: https://www.qsalute.it/chi-siamo-presentazione/ (cit. a p. 2).

[3] *Yelp - Fast Facts*. URL: https://www.yelp-press.com/company/fast-facts/default.aspx (cit. a p. 3).

[4] *Yelp Challenge - Introduction*. URL: https://www.yelp.com/dataset/challenge (cit. a p. 3).

[5] *Yelp Challenge - Dataset*. URL: https://www.yelp.com/dataset (cit. a p. 4).

[6] *A Review of Lightweight Thread Approaches for High Performance Computing*. URL: https://upcommons.upc.edu/bitstream/handle/2117/100370/A%20Review%20of%20Lightweight%20Thread%20Approaches.pdf (cit. a p. 8).

[7] *An Attempt at Reducing Costs of Disk I/O in Go*. URL: https://encrypted-tbn0.gstatic.com/images?q=tbn%3AANd9GcRzaGG_as9phxDFffbcyd8IYBJk7L5kjdQeYmh1_8UnQ5tnybFn (cit. a p. 9).

[8] N. Togashi e V. Klyuev. «Concurrency in Go and Java: Performance analysis». In: *2014 4th IEEE International Conference on Information Science and Technology*. Apr. 2014, pp. 213–216. DOI: 10.1109/ICIST.2014.6920368 (cit. a p. 9).

[9] *goquery - A little like that j-thing, only in Go*. URL: https://github.com/PuerkitoBio/goquery (cit. a p. 9).

[10] *GVM Care Research*. URL: https://www.gvmnet.it/patologie.aspx (cit. a p. 12).

[11] *Centers for Desease Control and Prevention*. URL: https://www.cdc.gov/diseasesconditions/az/a.html (cit. a p. 12).

[12] Haibo He, Yang Bai, Edwardo A. Garcia e Shutao Li. «ADASYN: Adaptive Synthetic Sampling Approach for Imbalanced Learning». In: (2008), pp. 1–7. URL: https://sci2s.ugr.es/keel/pdf/algorithm/congreso/2008-He-ieee.pdf (cit. a p. 15).

[13] *Treccani - Definizione di Flessione*. URL: http://www.treccani.it/enciclopedia/flessione_(Enciclopedia-dell%5C%27Italiano)/ (cit. a p. 16).

[14] *Treccani - Definizione di Tema*. URL: http://www.treccani.it/enciclopedia/tema (cit. a p. 16).

[15] Wahiba Abdessalem. «A New Stemmer to Improve Information Retrieval». In: *International Journal of Network Security Its Applications* 5 (lug. 2013), pp. 143–154. DOI: 10.5121/ijnsa.2013.5411 (cit. a p. 16).

[16] Martin Porter. «An algorithm for suffix stripping». In: (2006). URL: https://pdfs.semanticscholar.org/a651/bb7cc7fc68ece0cc66ab921486d163373385.pdf (cit. a p. 16).

[17] Edward Loper Bird Steven e Ewan Klein. *Natural Language Processing with Python*. O'Reilly Media Inc., 2009 (cit. a p. 17).

[18] F. Pedregosa et al. «Scikit-learn: Machine Learning in Python». In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. a p. 18).

[19] Pauli Virtanen et al. «SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python». In: *Nature Methods* (2020). DOI: https://doi.org/10.1038/s41592-019-0686-2. URL: https://rdcu.be/b08Wh (cit. a p. 18).

[20] URL: https://www.evemilano.com/wp-content/uploads/2014/10/TFIDF-SEO.jpg (cit. a p. 19).

[21] *Term frequency–inverse document frequency*. URL: https://it.wikipedia.org/wiki/Tf-idf (cit. alle pp. 19, 20).

[22] *An Intuitive Understanding of Word Embeddings: From Count Vectors to Word2Vec*. URL: https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/ (cit. alle pp. 21, 22).

[23] *Illustration of the Skip-Ngram and Continuous Bag of Word models*. URL: https://www.researchgate.net/profile/Wang_Ling/publication/281812760/figure/fig1/AS:613966665486361@1523392468791/Illustration-of-the-Skip-gram-and-Continuous-Bag-of-Word-CBOW-models.png (cit. a p. 22).

[24] *GloVe NLP Stanford Project*. URL: https://nlp.stanford.edu/projects/glove/ (cit. alle pp. 22, 23).

[25] *Word Embeddings for NLP - Understanding word embeddings and their usage in Deep NLP*. URL: https://miro.medium.com/max/2456/1*gcC7b_v7OKWutYN1NAHyMQ.png (cit. a p. 22).

[26] Shai Shalev-Shwartz e Shai Ben-David. *Understanding Machine Learning - From Theory to Algorithms*. 32 Avenue of the Americas, New York, NY 10013-2473, USA: Cambridge University Press, 2014 (cit. alle pp. 24–34).

[27] *Supervised vs Unsupervised Learning*. URL: https://lawtomated.com/wp-content/uploads/2019/04/supVsUnsup.png (cit. a p. 26).

[28] *Logistic Regression in Machine Learning*. URL: https://www.javatpoint.com/logistic-regression-in-machine-learning (cit. a p. 28).

[29] *Hard-Rock Stability Analysis for Span Design in Entry-Type Excavations with Learning Classifiers*. URL: https://www.researchgate.net/publication/304611323_Hard-Rock_Stability_Analysis_for_Span_Design_in_Entry-Type_Excavations_with_Learning_Classifiers/figures?lo=1&utm_source=google&utm_medium=organic (cit. a p. 29).

[30] *Classifying data with decision trees*. URL: https://elf11.github.io/2018/07/01/python-decision-trees-acm.html (cit. a p. 31).

[31] *Seeing the Forest for the Trees: An Introduction to Random Forest*. URL: https://community.alteryx.com/t5/Alteryx-Designer-Knowledge-Base/Seeing-the-Forest-for-the-Trees-An-Introduction-to-Random-Forest/ta-p/158062 (cit. a p. 32).

[32] *A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning*. URL: https://ronan.collobert.com/pub/matos/2008_nlp_icml.pdf (cit. a p. 33).

[33] *Artificial Intelligence vs. Machine Learning vs. Deep Learning: What's the Difference?* URL: https://www.sumologic.com/blog/machine-learning-deep-learning/ (cit. alle pp. 33, 34).

[34] Ian Goodfellow, Yoshua Bengio e Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016 (cit. alle pp. 35–39).

[35] *The NTNU-YZU System in the AESW Shared Task: Automated Evaluation of Scientific Writing Using a Convolutional Neural Network*. URL: https://www.researchgate.net/publication/304124119_The_NTNU-YZU_System_in_the_AESW_Shared_Task_Automated_Evaluation_of_Scientific_Writing_Using_a_Convolutional_Neural_Network (cit. a p. 35).

[36] *RNN or Recurrent Neural Network for Noobs*. URL: https://hackernoon.com/rnn-or-recurrent-neural-network-for-noobs-a9afbb00e860 (cit. a p. 37).

[37] *Dynamic Neural Network Modelling of Soil Moisture Content for Predictive Irrigation Scheduling*. URL: https://www.researchgate.net/publication/328228017_Dynamic_Neural_Network_Modelling_of_Soil_Moisture_Content_for_Predictive_Irrigation_Scheduling (cit. a p. 37).

[38] *Deep Learning Long Short-Term Memory (LSTM) Networks: What You Should Remember*. URL: https://missinglink.ai/guides/neural-network-concepts/deep-learning-long-short-term-memory-lstm-networks-remember/ (cit. a p. 38).

[39] François Chollet et al. *Keras*. https://keras.io. 2015 (cit. alle pp. 40, 69).

[40] *Kohonen self-organising map (KSOM) extracted features for enhancing MLP-ANN prediction models of BOD5*. URL: https://iahs.info/uploads/dms/14037.25-181-187-314-06-Rustumetal.pdf (cit. a p. 43).

[41] *Italian Word Embeddings using GloVe*. URL: http://hlt.isti.cnr.it/wordembeddings/ (cit. alle pp. 44, 69).

[42] Jeffrey Pennington, Richard Socher e Christopher D. Manning. «GloVe: Global Vectors for Word Representation». In: *Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1532–1543. URL: http://www.aclweb.org/anthology/D14-1162 (cit. a p. 44).

[43] *GloVe NLP Stanfors Project - GitHub project*. URL: https://github.com/stanfordnlp/GloVe (cit. a p. 44).

[44] *Sentiment analysis using ensemble methods: an application to Twitter*. URL: http://webthesis.biblio.polito.it/id/eprint/10363 (cit. a p. 69).

[45] *fastText is a library for efficient learning of word representations and sentence classification*. URL: https://github.com/facebookresearch/fastText/ (cit. a p. 69).