Politecnico di Torino

Facoltà di Ingegneria
Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea:

Analisi di situazioni emergenziali tramite la classificazione automatica di tweet

Relatore:
Prof. Paolo Garza

Candidato:
Francesco Vergona
s232976

Anno Accademico 2017/2018

Indice

Elenco delle figure

Capitolo 1

Introduzione

Considerati i più recenti aggiornamenti a cui è andato incontro il popolare servizio di micro-blogging **Twitter**[1], appare lampante come questo non sia più da considerare un semplice social network, ma più come un servizio informativo.
Certo, Twitter assume tale funzione perchè le notizie circolano attraverso le connessioni sociali degli utenti, ed è proprio questo il segreto che permette al servizio di rilasciare, riassumere e sintetizzare notizie in note da soltanto 280 caratteri.
Studi recenti [2] relativi all'utilizzo di Twitter in concomitanza di disastri naturali illustrano accuratamente il *paradosso del conversational micro-blogging.* [3]

La maggioranza dei retweet (termine per indicare un re-post o messaggio inoltrato in Twitter) immessi sulla piattaforma durante l'esondazione del Red River, North Dakota nel 2009 [4] erano vere e proprie news, intese come informazioni che non esistevano precedentemente su Twitter, o addirittura sul web.
Ma l'importanza di tale aspetto, a sostegno dell'immensa utilità del servizio, assume un rilievo ancora maggiore se si prendono in considerazione i tweet derivativi e sintetici generati a seguito di tali tweet "originali".

Molti di questi tweet furono redatti dai media locali e nazionali, ma una sorprendente porzione di essi invece vennero fuori da account creati specificamente per l'alluvione, con lo scopo di informare ed aggiornare gli interessati, fornendo loro preziose indicazioni.

Testate giornalistiche locali e nazionali in particolare, hanno preso parte alla diffusione di tweet derivativi e sintetici, mentre l'80% delle informazioni originali sono state generate da tweet postati da cittadini che si trovavano ad affrontare il disastro in quel preciso istante.

Sempre la stessa ricerca continua paragonando i retweet ad un sistema di raccomandazioni formale. Un retweet ha solitamente il potere di "sindacare" ed esplicitare l'importanza di un'informazione.

Non è però tutto oro quel che luccica. D'altro canto, con retweet, sintesi ed eventuali aggiunte di conoscenza locale, anche se acquisita "guardando fuori dalla finestra di casa", si genera un'enorme quantità di rumore, che va a sporcare lo stream di tweet relativi alla specifica catastrofe, rendendo molto più complicato trarne vantaggi tangibili.

Date queste premesse, non dovrebbe suonare particolarmente esotico il fatto che negli ultimi tempi Twitter risulta essere sempre più utilizzato per divulgare informazioni preziose sia per chi fornisce supporto alle vittime, sia per chi di tale supporto deve sfortunatamente usufruire.

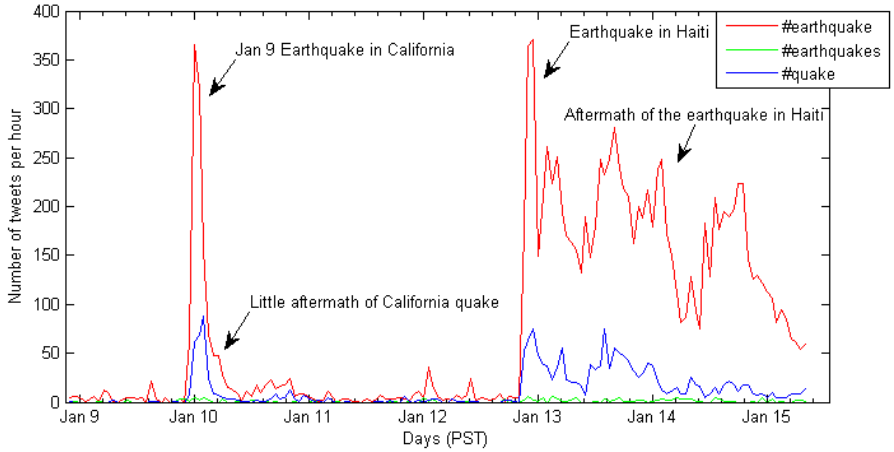

Figura 1.1: Flusso di tweet generato durante alcune catastrofi naturali

Con questo elaborato, pertanto, vogliamo rendere le cose più facili, andando a sfruttare non solo l'inconfutabile potenza di Twitter, ma anche e soprattutto gli enormi passi in avanti che son stati fatti negli ultimi decenni in termini di capacità di elaborazione dei più moderni calcolatori e le possibilità che ne conseguono, tra cui l'attuabilità delle più avanzate tecniche di Machine Learning, previsione e classificazione, di cui oggi tanto si sente parlare.

In particolare, ci siamo sobbarcati l'onere di creare un sistema che sia in grado di filtrare, in totale autonomia, l'enorme flusso di informazioni immesse sulla piattaforma e di etichettare ogni singolo post come **rilevante** o **non rilevante** dal punto di vista di chi deve organizzare o usufruire di soccorsi in situazioni di difficoltà.

Nelle prossime sezioni andremo a prendere in esame le diverse valutazioni effettuate in fase di progettazione ed implementazione dei diversi classificatori che abbiamo realizzato per lo scopo appena introdotto, soffermandoci sulle motivazioni che hanno portato a prendere determinate scelte, passando per i pro e contro di ciascuna soluzione adottata, fino ad arrivare alla comparazione finale delle diverse alternative.

1.1 Lavori correlati

Considerato quanto esposto nella precedente sezione, non è difficile immaginare che un sistema simile a quello che abbiamo realizzato esista già, e non è nostra intenzione elevarci a grandi innovatori o salvatori del pianeta.
In questa sezione accenneremo brevemente ai principali lavori, svolti da terzi, ai quali ci siamo maggiormente ispirati e dai cui risultati siamo ripartiti per realizzare la nostra soluzione, la quale si differenzia per alcune particolari scelte di progettazione.

L'articolo al quale questo nostro elaborato deve di più è sicuramente *"A Language-agnostic Approach to Exact Informative Tweets during Emergency Situations" - Jacopo Longhini, Claudio Rossi, Claudio Casetti, Federico Angaramo.* In tale documento gli autori raccontano di come hanno realizzato un classificatore in grado di inferire automaticamente se un determinato tweet sia correlato o meno ad un particolare evento

catastrofico preso in esame, il tutto rimanendo completamente indipendenti dal testo contenuto nei post e dal linguaggio ivi utilizzato. L'approccio scelto dai sopracitati autori si differenzia dal nostro, come vedremo successivamente, in quanto:

- Alcune delle nostre soluzioni analizzano e sfruttano il contenuto del tweet, perdendo la caratteristica dell'indipendenza dal linguaggio utilizzato, ma sfruttando tecniche di *Natural Language Processing* per ottenere migliori performance.
- Nella quasi totalità dei casi, le nostre soluzioni non richiedono alcun intervento umano per funzionare, a differenza del loro, che prevede una fase di "tagging" manuale dei primi tweet estratti, senza la quale il classificatore non riuscirebbe ad operare.

Di fondamentale aiuto è stato inoltre il sito `http://www.crisislex.org`, il quale mette a disposizione tantissime collezioni di dati che sono state acquisite da Twitter (e non solo) durante più o meno recenti eventi catastrofici.
Grazie al lavoro svolto dai gestori della piattaforma è stato infatti possibile avere accesso a tweet catturati in tempo reale durante una catastrofe, senza i quali non saremmo stati in grado di creare alcun classificatore, in quanto post relativi ad eventi così specifici non sono ovviamente recuperabili in qualsiasi momento, essendo generati prevalentemente nel periodo immediatamente successivo all'avvenimento.

1.2 Strumenti e librerie

Per la realizzazione dei diversi classificatori abbiamo utilizzato vari strumenti e librerie, in particolare:

- **Python 3.6.5** [5]: per l'estrazione, pre-processing, ed elaborazione dei dati per renderli utilizzabili nella classificazione.
- **Rapidminer Studio 8** [6]: per l'importazione dei dati, progettazione, addestramento e applicazione del classificatore.
- **Stanford NLP NER Tagger** [7]: per l'applicazione delle tecniche di Natural Language Processing, in particolare la **feature extraction**

- **Tecnica Doc2Vec** [8]: per il mapping del testo in vettori numerici, codificati in modo da essere particolarmente adatti al confronto e alla classificazione del testo.

Vale la pena menzionare le principali librerie Python (sia di prime che di terze parti) utilizzate per il nostro scopo:

- **Pandas** [9]: per l'importazione e manipolazione dei dataset
- **Twython** [10]: per l'estrazione di informazioni rilevanti dai tweet, come *User ID* dell'autore di un post o *Numero di retweet* di un tweet. In particolare queste librerie forniscono un'interfaccia in Python delle *API Twitter*, le quali permettono di ottenere tali informazioni tramite l'utilizzo delle *Rest API*.
- **Natural Language Toolkit (NLTK)** [11]: per l'implementazione di tecniche di NLP e ottenere un'interfaccia Python dello *Stanford NLP NER Tagger* [7], nativamente implementato in *Java*.
- **Gensim** [12]: per l'implementazione della tecnica *Doc2Vec*.

Capitolo 2

Fondamenti di Machine Learning: La classificazione

Il **Machine Learning** è una branca dell'Intelligenza Artificiale che fa uso di tecniche statistiche per permettere ai computer (più precisamente a moduli software) di ottenere la capacità di apprendere ed imparare dai dati, senza essere esplicitamente programmati [13].

L'obiettivo principale del *Machine Learning* è quello di fare in modo che le macchine manifestino un comportamento che, se esibito dall'uomo, implicherebbe l'uso dell'intelligenza. Si occupa in particolare della progettazione e dello studio di sistemi che possano apprendere conoscenza attraverso l'analisi dei dati in modo da poter **riconoscere automaticamente modelli complessi** ed essere in grado di prendere decisioni, o, nel nostro caso, categorizzare dei dati. In questa sezione introdurremo una panoramica sulle tecniche cardine del Machine Learning e delle principali problematiche che si presentano quando si approccia a contesti che ne richiedono l'utilizzo, in modo tale da avere una visione generale che faciliterà la comprensione dei successivi capitoli, nei quali inevitabilmente saranno presenti riferimenti ad argomenti teorici e tecnicismi.

Ad oggi esistono tantissimi campi nei quali il Machine Learning è sfruttato ed applicato. Moltissime tra le più importanti aziende al mondo fanno affidamento su tali tecniche ed

algoritmi per meglio comprendere i loro clienti e per individuare eventuali opportunità di guadagno [14]. In particolare, tra le più comuni applicazioni vi sono:

- **Customer Lifetime Value Modeling**: insieme di operazioni effettuate per identificare, comprendere e mantenere i clienti più preziosi per una certa azienda. Tali modelli prevedono il futuro guadagno generato da ogni singolo cliente, in un periodo prefissato. Con queste informazioni, è possibile di conseguenza pianificare o adattare strategie di marketing, cercando di incoraggiare tali clienti ad interagire maggiormente con il brand o addirittura attrarre nuovi clienti, simili a quelli identificati.

- **Customer Churn Modeling**: insieme di operazioni che permettono di prevedere quali sono i clienti che più verosimilmente cesseranno la relazione (che sia un abbonamento, una sottoscrizione, o un semplice acquisto ricorrente) che hanno con l'azienda, e il perchè lo faranno. Tali previsioni, non diversamente dal caso del *Customer Lifetime Value Modeling*, rappresentano una componente essenziale per la pianificazione di una strategia di ritenzione della clientela, supportando l'ottimizzazione delle offerte, campagne di e-mail o altre iniziative di marketing.

- **Pricing dinamico**: la pratica di assegnare prezzi variabili agli oggetti in vendita su un sito di e-commerce, basando tale variabilità su fattori come il livello di interesse dello specifico cliente, la richiesta al momento dell'acquisto o se il cliente è o meno soggetto ad una determinata campagna di marketing. Sarà capitato a tutti di individuare un oggetto di proprio interesse su Amazon e vedere in pochissimo tempo (anche più volte in una sola giornata) tale oggetto salire di prezzo, anche con significative variazioni.

- **Segmentazione dell'utenza**: piuttosto che affidarsi all'intuizione di un esperto di marketing, e assegnare lui il compito di segmentare l'utenza per realizzare campagne di marketing ad hoc, i data scientist possono utilizzare tecniche come il **clustering** e la **classificazione** per creare gruppi di clienti e differenziare, per ognuno di essi, la strategia di marketing.

- **Riconoscimento di immagini**: sistemi che permettono di assegnare un certo valore, scelto tra un set prefissato di etichette, ad una qualsiasi immagine presentata in input. Per questi scopi sono spesso utilizzate metodologie di **Deep Learning**, in quanto risultano essere più adatte ad identificare feature rilevanti all'interno dell'immagine, nonostante l'eventuale presenza di complicazioni quali variazioni del punto di vista, illuminazione, o scala degli oggetti ivi rappresentati.

- **Sistemi di raccomandazione**: rappresentano un altro metodo per incrementare il valore d'impresa. Colossi come Amazon e Netflix fanno di tali sistemi il segreto del loro successo. Tramite questi sistemi infatti, è possibile predire quanto verosimilmente un singolo cliente comprerà un certo oggetto o gradirà un certo contenuto, in modo tale da poter personalizzare la sua esperienza d'uso e di conseguenza aumentare le possibilità che tale cliente tornerà a fare uso della piattaforma.

Figura 2.1: Esempio di Recommendation System di Amazon

2.1 Apprendimento supervisionato e non supervisionato

I data scientist di tutto il mondo fanno uso di svariati algoritmi di *Machine Learning* per individuare pattern all'interno dei loro dati che forniscano loro interessanti spunti per supportare decisioni strategiche di vario tipo.
Assumendo un elevato livello di astrazione, tali algoritmi possono essere classificati in due gruppi, basati nel loro modo di **apprendere dai dati** e di conseguenza **effettuare predizioni**.

L'**apprendimento supervisionato** è sicuramente l'approccio più conosciuto ed utilizzato. Esso include algoritmi come la **logistic regression**, **classificazione** e le cosiddette **Support Vector Machine**. Il nome di questa classe di algoritmi deriva dal fatto che il data scientist farà da *"guida"*, o meglio da *"supervisore"* per insegnare all'algoritmo come prendere determinate decisioni. Tali tecniche richiedono tassativamente che le classi assegnabili agli input siano conosciute a priori e che i dati di input che vengono utilizzati per l'addestramento siano già provvisti dell'etichetta corretta. Ad esempio, se vogliamo realizzare un classificatore che sia in grado di identificare se in una certa immagine c'è un fiore o meno, è necessario essere provvisti di una serie di immagini le quali saranno ognuna associata ad un'etichetta, ad indicare o meno la presenza del fiore al suo interno.

L'approccio duale, l'**apprendimento non supervisionato**, è più vicino a quella che oggi è considerata vera intelligenza artificiale, ovvero l'idea che un computer sia in grado di imparare ad identificare processi e pattern di diversi livelli di complessità **senza alcun supporto da parte di esseri umani**, come era invece necessario nel caso precedente. Nonostante l'apprendimento non supervisionato sia notoriamente più complesso anche per la più semplice delle applicazioni, apre la strada alla soluzione di problemi che gli esseri umani solitamente non affrontano. Alcuni algoritmi che rientrano in questa categoria sono il **K-Means Clustering**, la **Principal Component Analysis (PCA)** e le cosiddette **regole associative**.

La scelta tra i due tipi di apprendimento dipende solitamente da fattori relativi alla

struttura ed al volume dei dati, oltre che ovviamente dal caso d'uso che si sta prendendo in considerazione. Non è inoltre esclusivo l'uso di uno dei due approcci: diversi programmi avanzati utilizzano entrambe le tipologie di algoritmo.

Tornando invece alla soluzione del problema introdotto nel capitolo precedente, il nostro sistema rientra nella categoria di **Machine Learning supervisionato**, in quanto richiede per l'addestramento del classificatore l'utilizzo di dati, nello specifico tweet, già etichettati come rilevanti o non rilevanti. Analizzeremo questi aspetti nelle sezioni successive.

2.2 Output discreti, binari o continui

La suddivisione in categorie presentata nella precedente sezione non è però l'unica possibile. Per differenziare algoritmi supervisionati e non supervisionati abbiamo posto sotto la lente i dati di input, in particolare se questi necessitano di un'etichetta per poter addestrare il modello oppure no. Poniamo adesso l'attenzione sul formato dell'output, per poter distinguere **algoritmi di classificazione** da **algoritmi di regressione**.

Gli **algoritmi di classificazione** sono utilizzati quando l'output desiderato è un'**etichetta discreta**, ovvero può assumere uno tra un insieme di valori predeterminati (Es.: Colore = {Bianco, Nero, Rosso, Blu}).

Tante casistiche come quella di determinare se un'email è spam o no, o anche il problema di classificazione che abbiamo introdotto e che ci apprestiamo a risolvere in questo elaborato, hanno soltanto due possibili risultati. Questi problemi sono chiamati problemi di **classificazione binaria**.

La **classificazione multi-etichetta** invece comprende tutti gli altri casi di output discreti, ed è particolarmente utile per segmentazione dell'utenza, categorizzazione di audio e immagini e per l'analisi testuale per fini di *sentiment analysis*.

Tra i principali algoritmi di classificazione figurano **Naive-Bayes**, **alberi di decisione** e **Random Forest**.

Gli **algoritmi di regressione** sono invece utilizzati per **predire output continui**. Ciò significa che la risposta alla domanda posta in origine è rappresentata da una quantità che non rientra in un'insieme di possibilità, ma può essere determinata con maggiore flessibilità. La **regressione lineare** è di gran lunga l'algoritmo di regressione più utilizzato, nonostante sia spesso sottovalutato per via della sua semplicità. Nonostante ciò, si è rivelato più volte essere particolarmente versatile e utilizzabile per diversi scopi, come la previsione del valore di immobili, probabilità di abbandono di un servizio da parte di un cliente o la previsione del guadagno generato da un utente.

2.3 Problemi e tecniche di classificazione

Appurato che il nostro problema di riconoscimento di tweet è un problema di classificazione, dovendo il sistema riconoscere automaticamente se un tweet è rilevante o meno per l'organizzazione o la fruizione di soccorsi in caso di eventi catastrofici, ci soffermiamo adesso sui principali aspetti teorici che caratterizzano gli algoritmi rientranti in tale categoria.

La classificazione è il processo di apprendimento di una determinata **funzione target** f, la quale effettua un mapping tra un **vettore di input** x in una delle **etichette predefinite** y. La funzione target è anche indicata, in modo informale, come **modello di classificazione** [32].

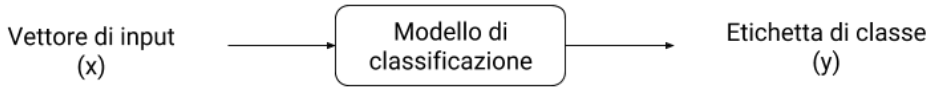

Figura 2.2: Classificazione: Processo base

In via del tutto generale, i modelli di classificazione possono distinguersi in:

- **Descriptive Modeling**: Il modello di classificazione può fare da strumento esplicativo per oggetti di diverse classi. Ad esempio, potrebbe essere utilizzato un modello descrittivo per riassumere un dataset di animali e spiegare quali sono le feature che definiscono se tale animale è un mammifero, rettile, uccello o anfibio (classe).

- **Predictive Modeling**: Il modello di classificazione può essere usato per predire l'etichetta di classe di record sconosciuti. Seguendo lo stesso esempio di prima, è possibile tramite la classificazione prevedere se una nuova specie animale, finora sconosciuta, rientra tra i mammiferi o no.

Le tecniche di classificazione sono particolarmente adatte per la predizione o la descrizione di dataset con categorie binarie o nominali. Risultano invece meno efficienti per categorie ordinali (Es.: Classificare una persona come molto, mediamente, o poco abbiente) in quanto non prendono in considerazione l'ordine implicito tra le categorie. Altre forme di relazione, come la relazione gerarchica tra le categorie (Es.: Gli uomini sono scimmie, e le scimmie a loro volta sono primati, che a loro volta sono una sottoclasse di mammifero), sono generalmente ignorate.

2.3.1 Soluzione di un problema di classificazione

Una tecnica di classificazione è un approccio sistematico per costruire modelli di classificazione dato un dataset di input. Ogni classificatore impiega un **algoritmo di apprendimento** per identificare un modello che sia in grado di adattarsi al meglio alla relazione tra il set di attributi (feature) e le classi associate ai dati in input.

Un modello di classificazione generato tramite un algoritmo di apprendimento deve essere in grado sia di adattarsi correttamente ai dati di input, ma anche e soprattutto essere in grado di predire correttamente le etichette di classe di record che non ha mai visto prima. Ovvero, l'obiettivo chiave dell'algoritmo di apprendimento è quello di **costruire modelli con buone capacità di generalizzazione**.

Vediamo adesso un approccio del tutto generale alla soluzione di un problema di classificazione:

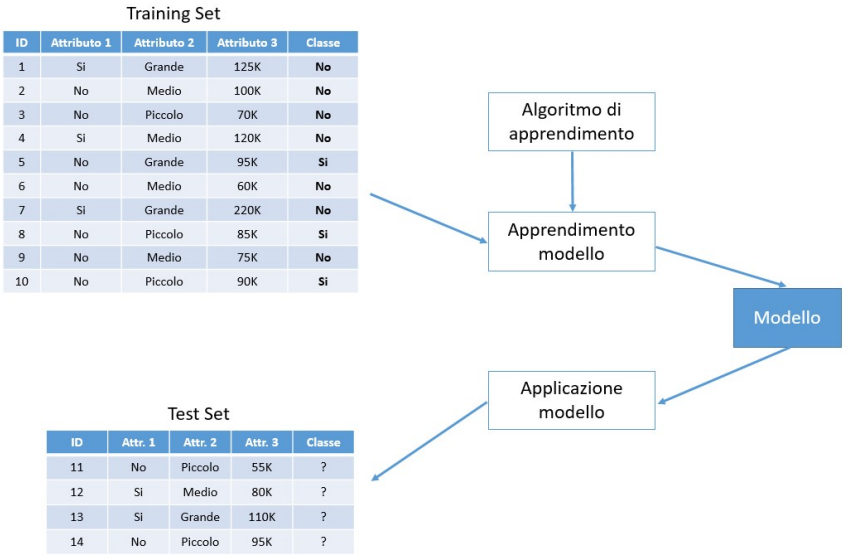

Figura 2.3: Soluzione ad un problema di classificazione

Notiamo in tale schema, la presenza di:

- **Training set**: Insieme di dati le cui classi di appartenenza sono date. Il training set è utilizzato per costruire il **modello di classificazione**, che sarà poi applicato al test set.

- **Test set**: Insieme di dati di input anch'essi dotati di etichetta, che non vengono però utilizzati per l'addestramento bensì per la **valutazione delle performance** del classificatore. In particolare, dopo la fase di training, nella quale vengono presentati in input i dati del training set, si "danno in pasto" al sistema i dati del test set, le cui etichette vengono utilizzate per valutare quanti input vengono correttamente classificati, in modo tale da avere un'indicazione su come il modello si comporterà una volta messo all'opera.

La valutazione delle performance di un classificatore è basata sul conteggio dei record del test set correttamente predetti e di quelli predetti erratamente. Tali conteggi so-

no organizzati in una struttura tabulare che prende il nome di **matrice di confusione**.

	True 1	**True 0**
Pred 1	TP	FP
Pred 0	FN	TN

Nella matrice di confusione di un problema binario figureranno quattro valori numerici:

- **TP - True Positive**: Numero di elementi del test set correttamente predetti come positivi (Appartenenti alla classe 1)
- **TN - True Negative**: Numero di elementi del test set correttamente predetti come negativi (Appartenenti alla classe 0)
- **FN - False Negative**: Numero di elementi del test set erroneamente predetti come negativi (Predizione: Classe 0 | Classe effettiva: Classe 1)
- **FP - False Positive**: Numero di elementi del test set erroneamente predetti come positivi (Predizione: Classe 1 | Classe effettiva: Classe 0)

Da questa matrice è possibile derivare altre importanti informazioni, più o meno complesse. Ad esempio:

$$Numero\ di\ predizioni\ corrette = TP + TN$$

$$Numero\ di\ predizioni\ errate = FP + FN$$

Nonostante la matrice di confusione fornisca valide indicazioni su quanto bene un modello è in grado di classificare, sintetizzare questa informazione in un singolo indice renderebbe più semplice la comparazione tra due diversi modelli.

Ciò può essere fatto tramite una metrica detta **accuratezza** (Accuracy, in inglese), definita come segue:

$$Accuratezza = \frac{Numero\ di\ predizioni\ corrette}{Numero\ totale\ di\ predizioni} = \frac{TP + TN}{TP + TN + FP + FN}$$

Allo stesso modo è possibile esprimere le performance di un modello in termini di **error rate**, dato dalla seguente equazione:

$$Error\ rate = \frac{Numero\ di\ predizioni\ errate}{Numero\ totale\ di\ predizioni} = \frac{FP + FN}{TP + TN + FP + FN}$$

Vi sono alcuni indici che possono essere calcolati dalla matrice di confusione e che risultano particolarmente utili per verificare le prestazioni di un classificatore: **precisione e richiamo**. In breve, in un processo di classificazione, un valore di precisione di 1.0 per la classe C significa che ogni oggetto che è stato etichettato come appartenente alla classe C vi appartiene davvero (ma non dice niente sul numero di elementi della classe C che non sono stati etichettati correttamente) mentre un valore di recupero pari ad 1.0 significa che ogni oggetto della classe C è stato etichettato come appartenente ad essa. [30]

$$Precisione = \frac{TP}{TP + FP}$$
$$Richiamo = \frac{TP}{TP + FN}$$

2.4 Algoritmi di classificazione

Andiamo adesso a fare una breve panoramica su quelli che sono i principali algoritmi di classificazione. Non ci addentreremo nei dettagli di ogni algoritmo presentato, in quanto non tutti sono necessari alla comprensione del progetto a corredo di tale elaborato e soprattutto non tutti sono stati utilizzati per i nostri scopi.

In particolare, i sistemi di classificazione che abbiamo utilizzato per il progetto, e sui quali porremo maggior attenzione, sono i **Decision Tree** (o Alberi di Decisione), che sono alla base dei nostri classificatori e le **Reti Neurali**, che sono utilizzate internamente dalle tecniche di *Word2Vec* e *Doc2Vec*, e che quindi non abbiamo implementato direttamente con mano nostra.

2.4.1 Albero di decisione

Un **albero di decisione** è una tecnica di classificazione semplice, ma allo stesso tempo molto utilizzata.

Per illustrare il funzionamento di tale tecnica, introduciamo un dataset d'esempio:

Tabella 1: Specie animali

Nome	**Temperatura**	**Pelle**	**Parto**	**Acquatico**	**Volatile**	**Gambe**	**Letargo**	**Mammifero**
uomo	alta	peli	si	no	no	si	no	si
pitone	bassa	scaglie	no	no	no	no	si	no
salmone	bassa	scaglie	no	si	no	no	no	no
balena	alta	peli	si	si	no	no	no	si
rana	bassa	no	no	semi	no	si	si	no
drago								
gatto	alta	pelo	si	no	no	si	no	si
colombo	alta	piume	no	no	si	si	no	no
...								

Supponiamo a questo punto che gli scienziati scoprono una nuova specie di animale. Come possiamo inferire se è un mammifero o no?

Un primo approccio è quello di porre una serie di domande riguardo le caratteristiche di ogni specie. La prima domanda che potremmo fare è se la nuova specie è dal sangue caldo ("$Temperatura$" = "$Alta$") o dal sangue freddo ("$Temperatura$" = "$Bassa$"). Se il suo sangue è freddo, non è sicuramente un mammifero. Se invece è caldo, non possiamo ancora asserire nulla, in quanto ci sono specie dal sangue caldo che non sono mammiferi (Ad esempio: i colombi). In questo caso, dovremo necessariamente porre un'altra domanda: le femmine di questa specie, partoriscono direttamente ("$Parto$" = "si") oppure no ("$Parto$" = "no")? Se la risposta è si, allora la nuova specie apparterrà sicuramente ai mammiferi, altrimenti no.

Questo esempio mostra come possiamo risolvere un problema di classificazione ponendo una serie di domande fatte "ad arte", riguardanti gli attributi dei record del test set. Ogni volta che riceviamo una risposta, poniamo domande successive finchè non riusciamo ad ottenere una conclusione sulla classe di tale record.

La serie di domande e le relative possibili risposte possono essere organizzate in un albero di decisione, che altro non è che una struttura gerarchica composta da **nodi** e **archi direzionali**.

I nodi di un albero di decisione possono essere:

- **Nodo radice**: Non ha archi entranti e ha zero più archi uscenti.
- **Nodo interno**: Ha un solo arco entrante ed almeno due archi uscenti.
- **Nodo foglia**: Ha un solo arco entrante e nessun arco uscente. Ogni nodo foglia è associato ad un'**etichetta di classe**.

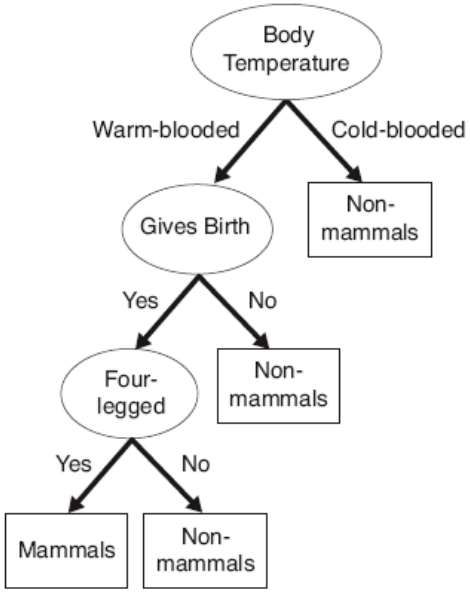

Figura 2.4: Esempio di albero di decisione

I nodi non terminali, quindi nodo radice e nodi interni, contengono al loro interno le **condizioni di test** che permettono di separare i record dei dataset aventi diverse caratteristiche. Ad esempio, il nodo radice mostrato in Figura 2.5 utilizza l'attributo *Body Temperature* (*"Temperatura"* nella Tabella 1), per separare le specie dal sangue caldo da quelle dal sangue freddo. Dato che tutti gli esseri dal sangue freddo non sono mammiferi, viene creato un nodo foglia *"Non mammals"* come figlio di destra del nodo radice.

Successivamente, se la specie è dal sangue caldo, si verifica anche l'attributo *"Gives Birth"* (*"Parto"* nella Tabella 1), tramite il quale si possono separare i mammiferi dalle altre creature dal sangue caldo, che sono tipicamente uccelli.

Classificare un record di test è immediato una volta che l'albero di decisione è stato costruito. Partendo dal nodo radice, applichiamo la condizione di test al record e seguiamo il ramo corretto basandoci sui risultati dei test. Ciò porterà o ad un altro nodo interno, al quale verrà posta un'altra condizione di test, o ad un nodo foglia. L'etichetta di classe associata al nodo foglia è quella assegnata al record.

2.4.1.1 Costruzione di un albero di decisione

In principio ci sono diversi alberi di decisione che possono essere generati da un dato set di attributi. Mentre alcuni alberi sono più precisi di altri nella loro classificazione, trovare l'albero ottimo è computazionalmente impossibile, data la dimensione esponenziale dello spazio di ricerca.

Esistono però degli efficienti algoritmi che sono stati introdotti per generare alberi subottimi in un discreto intervallo di tempo. Questi algoritmi solitamente impiegano una **strategia greedy**, che sviluppa un albero di decisione facendo una serie di scelte locali ottime relative a quale attributo utilizzare per partizionare i dati. Tra questi algoritmi figurano l'**algoritmo di Hunt**, che è alla base della maggior parte degli algoritmi di

generazione di alberi di decisione, inclusi **ID3, C4.5 e CART**.

Un algoritmo per la generazione di alberi deve affrontare due problemi principali:

- **Come splittare i record di training?** Ogni iterazione dell'algoritmo seleziona una condizione di test su un attributo che divide il dataset in subset dalla minore cardinalità. Per implementare questo passo, l'algoritmo deve fornire un metodo per **specificare la condizione di test per diversi tipi di attributi**, oltre che una misura per valutare la bontà di ogni condizione di test.
- **Qual è il criterio di stop per la procedura di splitting?** Una condizione di stop è necessaria per terminare il processo di crescita dell'albero. Una strategia possibile è quella di continuare ad espandere un nodo finchè tutti i record appartengono alla stessa classe o tutti i record hanno gli stessi valori attributo. Nonostante entrambe le soluzioni garantiscono all'algoritmo di terminare, altri criteri possono essere selezionati per permettere alla procedura di terminare anticipatamente.

2.4.1.2 Le condizioni di test

Ogni algoritmo di generazione di alberi di decisione deve fornire un metodo per esprimere una condizione di test su un attributo ed i suoi corrispettivi risultati per i diversi tipi di attributo.

La condizione di test per un **attributo binario** genera solo due potenziali risultati.

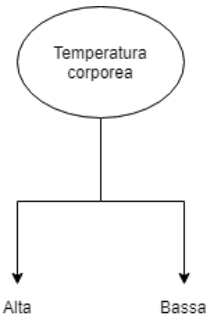

Figura 2.5: Condizione di test per attributi binari

Per quanto riguarda gli **attributi nominali**, dato che possono avere diversi valori, la condizione di test può essere espressa in due modi.

Nel primo caso, detto **multi-way split**, il numero di risultati dipende dal numero di valori distinti per l'attributo corrispondente, come nella Figura 2.7.

Figura 2.6: Esempio di multi-way split

Alcuni algoritmi però, come CART, non prevedono la presenza di multi-way split, e sono in grado di produrre solo split binari. Pertanto, per un attributo con k valori, verranno prodotti $2^{k-1} - 1$ partizioni binarie, come visibile dalla Figura 2.8, che mostra tre diverse possibilità di raggruppamento.

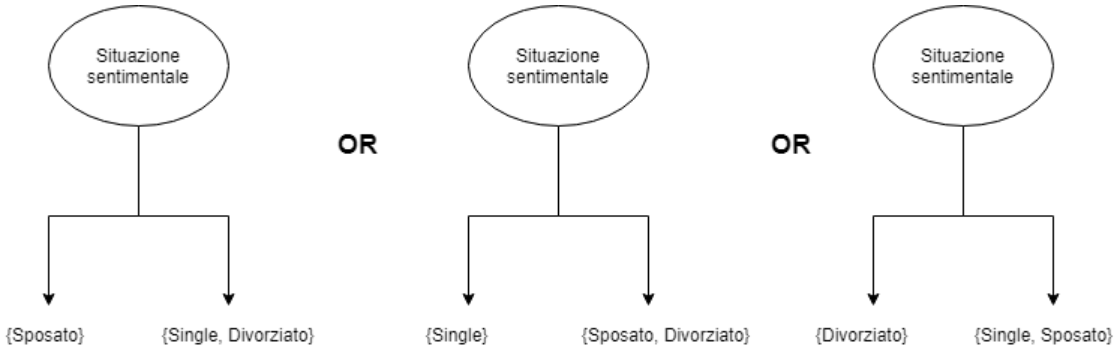

Figura 2.7: Esempio di binary split per attributi nominali

Gli **attributi ordinali** possono produrre anch'essi split binari o multi-way split. I valori degli attributi ordinali possono essere raggruppati finchè il raggruppamento non viola la relazione di ordinamento dei valori di attributi.

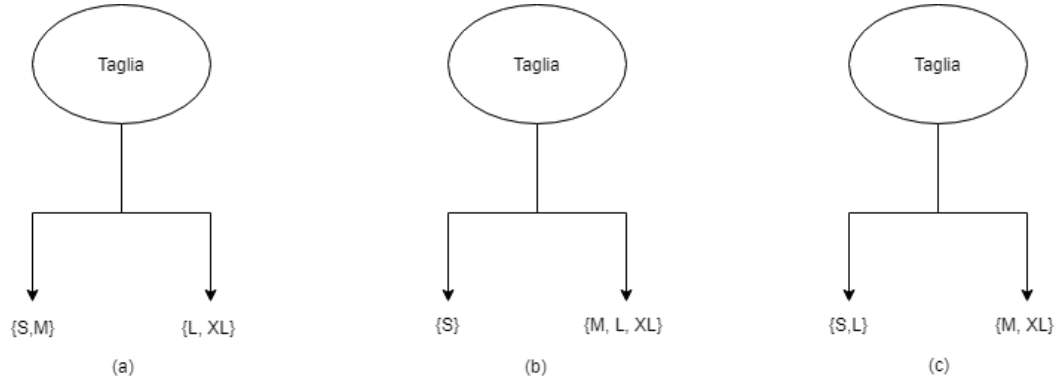

Figura 2.8: Esempio di binary split per attributi ordinali

Nella Figura 2.9 (a) e (b) l'ordine tra i valori degli attributi è preservato, mentre il raggruppamento nella Figura 2.9 (c) viola l'ordinamento in quanto combina i valori S *e* L nella stessa partizione, mentre M *e* XL fanno parte di un'altra partizione.

Per quanto riguarda gli **attributi continui** la condizione di test può essere espressa come un test comparativo $(A < v)$ *or* $(A \geq v)$ con risultati binari, oppure tramite l'introduzione di diversi range con risultati nella forma $v_i \leq A < v_{i+1}$.

Per la soluzione binaria, l'algoritmo deve considerare tutti i possibili valori di v e **selezionare quello che produce il miglior partizionamento**.

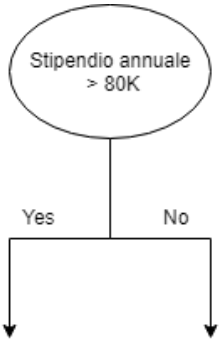

Figura 2.9: Esempio di binary split per attributi continui

Per la soluzione che implementa un multi-way split, l'algoritmo deve **considerare tutti i possibili range di valori continui**. Questo approccio richiede prima l'applicazione della discretizzazione dei valori continui, e poi l'assegnazione di un valore ordinale ad ogni intervallo discretizzato.

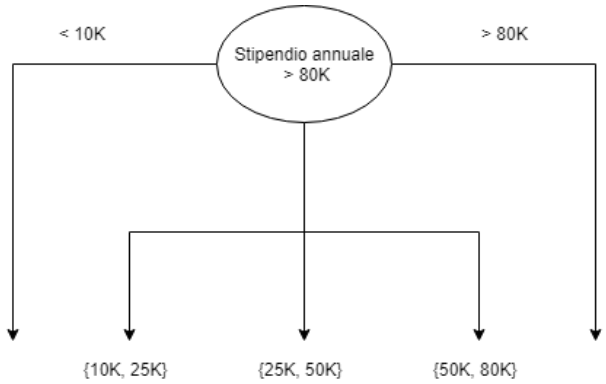

Figura 2.10: Esempio di multi-way split per attributi continui

2.4.1.3 La scelta del miglior split

Esistono diverse misure che permettono di determinare il miglior modo per "splittare" i record. Queste misure sono definite in termini di **distribuzione delle classi** tra i record prima e dopo lo split.

Consideriamo $p(i|t)$ la frazione di record appartenenti alla classe i ad un certo nodo dell'albero t (Spesso indicato semplicemente come p_i). In un problema binario, la distribuzione di classe ad un certo nodo può essere scritta come (p_0, p_1), con $p_1 = 1 - p_0$.

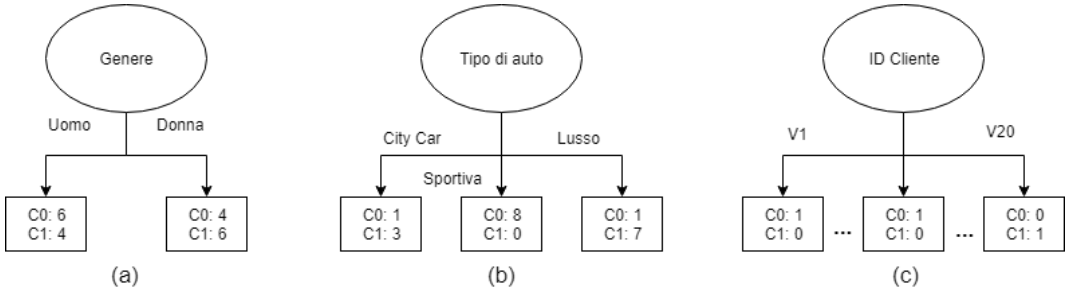

Figura 2.11: Multi-way e binary split a confronto

Consideriamo adesso le condizioni di split della Figura 2.12. La distribuzione delle classi prima dello split è $(0.5, 0.5)$ in quanto vi sono un numero equivalente di record per ogni classe. Splittando i dati utilizzando l'attributo *"Genere"*, la distribuzione delle classi dei nodi diventa $(0.6, 0.4)$ e $(0.4, 0.6)$, rispettivamente. Nonostante le classi non siano più parimente distribuite, i nodi figli possono ancora contenere entrambe le classi. Effettuando lo split su *"Tipo di auto"*, si ottengono partizioni più pure.

Le misure per la selezione del miglior split sono solitamente basate sul **grado di impurità dei nodi figli**. Minore è il grado di impurità, più asimmetrica sarà la distribuzione delle classi. Ad esempio, un nodo con distribuzione $(0, 1)$ ha impurità nulla, mentre un nodo dalla distribuzione uniforme $(0.5, 0.5)$ ha impurità massima.

Alcuni indici di impurità sono:

$$Entropy(t) = -\sum_{i=0}^{c-1} p(i|t) \log_2 p(i|t)$$

$$Gini(t) = 1 - \sum_{i=0}^{c-1} [p(i|t)]^2$$

$$Classification\ error(t) = 1 - \max_i [p(i|t)]$$

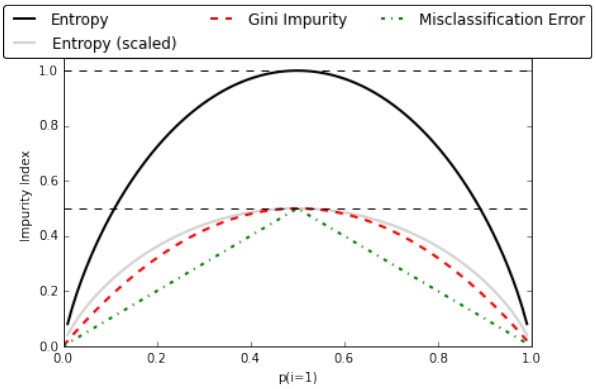

Figura 2.12: Confronto tra indici di impurità

La Figura 4.13 mette a confronto i valori delle misure di impurità per problemi di **classificazione binaria**. $p(i = 1)$ fa riferimento alla frazione di record che appartengono ad una delle due classi. Si osservi che tutte e tre le misure raggiungono il loro massimo valore quando la distribuzione delle classi è uniforme, ovvero quando $p = 0.5$, mentre i valori minimi si ottengono quando tutti i record appartengono alla stessa classe.

Per valutare la bontà di una condizione di test, date queste misure, è necessario confrontare l'indice di impurità (calcolato con una delle tre formule indicate precedentemente) del nodo padre (prima dello split) con quello dei nodi figli (dopo lo split). Maggiore è la differenza, migliore sarà la condizione di test.

Il guadagno Δ è un r che indica la bontà dello split:

$$\Delta = I(Padre) - \sum_{j=1}^{k} \frac{N(v_j)}{N} I(v_j)$$

Dove $I(x)$ è l'**indice di impurità** del nodo x, N è il numero totale di record nel nodo padre, k è il numero di valori attributo e $N(v_j)$ è il numero di record associati al nodo figlio v_j.

Quando per il calcolo di Δ si utilizza l'entropia, la differenza prende il nome di **Information gain** Δ_{info}.

2.4.1.4 Caratteristiche di un albero di decisione

Terminiamo questa sezione con una panoramica sulle caratteristiche degli algoritmi di generazione di alberi di decisione.

- La generazione di alberi di decisione è un approccio non parametrico per la costruzione di modelli di classificazione. In altre parole, non richiede alcuna assunzione a priori riguardo il tipo di distribuzione di probabilità soddisfatta dalle classi e da altri attributi.

- Trovare un albero di decisione ottimo è un problema **NP-Complete**. Molti algoritmi impiegano un approccio euristico che guida la ricerca nello spazio delle ipotesi.

- Tecniche sviluppate per la costruzione di alberi di decisione sono computazionalmente poco onerose, rendendo possibile una veloce costruzione di modelli anche a fronte di training set di dimensioni importanti. Inoltre, una volta costruito l'albero, la classificazione di un record di test è estremamente veloce, con una complessità nel caso peggiore pari a $O(w)$, con w profondità massima dell'albero.

- Gli alberi di decisione, in particolare quelli di piccole dimensioni, sono relativamente facili da interpretare. L'accuratezza di questi alberi è spesso comparabile a quella di altre tecniche di classificazione per dataset non troppo complessi.

- Gli alberi di decisione sono resistenti al rumore, specialmente se associati a metodi per evitare l'overfitting.

- La presenza di attributi ridondanti non influenza negativamente l'accuratezza degli alberi di decisione. Un attributo si dice ridondante quando è fortemente correlato ad un altro attributo nel dataset. Uno dei due (o più) attributi ridondanti semplicemente non verà utilizzato nello split, una volta che l'altro sarà stato scelto. D'altro canto, se il dataset contiene molti attributi non rilevanti (Non utili per la classificazione), è possibile che uno di questi venga scelto accidentalmente durante l'algoritmo di generazione dell'albero, il che porterà alla creazione di un albero di decisione più grande del necessario. Per sopperire ad un problema di

questo tipo vengono a supporto le tecniche di **feature selection**, che permettono di eliminare gli attributi meno rilevanti durante la fase di pre-processing.

- Dato che molti algoritmi per la generazione di alberi di decisione implementano un approccio top-down ricorsivo per il partizionamento, il numero di record diventa sempre minore man mano che si scende nell'albero. Ai nodi foglia, il numero di record può essere troppo piccolo per prendere decisioni statisticamente significative riguardo la rappresentazione delle classi nei singoli nodi. Questo problema è noto come **frammentazione dei dati**. Una possibile soluzione è di impedire ulteriori split quando il numero di record è al di sotto di una determinata soglia.

- Un sottoalbero può essere replicato più volte in un albero di decisione. Questo rende l'albero di decisione più complicato del necessario e pertanto più difficile da interpretare. Dato che la maggior parte degli algoritmi utilizza un approccio ***"Divide and conquer"***, la stessa condizione di test può essere applicata a parti diverse dello spazio degli attributi, portando al problema della replicazione dei sotto-alberi.

- Le condizioni di test descritte fino a questo punto coinvolgono solo un attributo alla volta. Come conseguenza, la procedura di generazione dell'albero può essere vista come un **partizionamento dello spazio degli attributi** in regioni disgiunte, finchè ogni regione non contiene solo record appartenenti alla stessa classe. Il bordo tra due regioni vicine prende il nome di **decision boundary**. Dato che le condizioni di test prendono in considerazione un solo attributo alla volta, i decision boundaries saranno rettilinei. Questo limita l'espressività dell'albero di decisione per il modeling di relazioni complesse tra attributi continui. Nella Figura 2.14 è rappresentato un dataset che non può essere classificato in modo efficace da un algoritmo che utilizza condizioni di test basate su un solo attributo per volta. Gli **alberi di decisione obliqui** vengono incontro a questo problema in quanto permettono di avere condizioni di test basate su più di un attributo contemporaneamente.

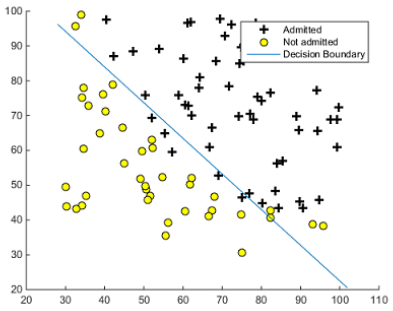

Figura 2.13: Dataset non classificabile con un decision tree standard

2.4.2 Il problema dell'overfitting

Gli errori commessi da un modello di classificazione sono generalmente divisi in due categorie: **errori di addestramento** e **errori di generalizzazione**. Gli **errori di addestramento** rappresentano il numero di errori di classificazione commessi sui **record di training**, mentre gli **errori di generalizzazione** rappresentano l'errore atteso del modello su record mai visti in precedenza.

Un buon classificatore non deve essere in grado soltanto di apprendere correttamente dal training set, ma deve essere anche in grado di classificare accuratamente record che non ha mai visto in precedenza. In altre parole, un buon modello deve avere un **basso errore di addestramento**, ma anche un **basso errore di generalizzazione**. Ciò è fondamentale in quanto un modello troppo adattato ai dati di training avrà una **scarsa capacità di generalizzazione** che porterà ad un errore di generalizzazione più alto. Una situazione del genere è nota come **overfitting**.

2.4.2.1 La soluzione per gli alberi di decisione

Presenteremo adesso due soluzioni possibili per aggirare il problema dell'overfitting nel caso degli alberi di decisione.

Il **pre-pruning** è un approccio che prevede che l'algoritmo di generazione dell'albero di decisione venga fermato prima che si è ottenuto l'albero completamente espanso, che si

adatterebbe perfettamente ai dati di training. Per fare ciò, deve essere selezionata una condizione di stop più restrittiva, come "Smettere di espandere un nodo foglia quando il guadagno della misura di impurità è sotto una certa soglia".

Il **post-pruning** è invece un approccio che prevede la generazione completa dell'albero di decisione. Successivamente si passa ad una fase di **pruning** dell'albero, che comporterà la potatura seguendo un approccio **bottom-up**. La potatura può essere realizzata in due modi:

- Sostituzione di un sottoalbero con un nodo foglia la cui classe è determinata dalla classe di maggioranza dei record associati al sottoalbero.
- Sostituzione di un sottoalbero con il ramo maggiormente usato dello stesso.

La fase di potatura dell'albero si ferma non appena smette di introdurre dei miglioramenti osservabili. Il post-pruning tende a fornire risultati migliori del pre-pruning in quanto prende decisioni basate sull'intero albero, mentre il pre-pruning prende decisioni che possono soffrire di terminazione prematura del processo di generazione dell'albero. D'altro canto però, il post-pruning richiede un maggior onere computazionale in quanto vi saranno parti dell'albero che saranno generate e poi non utilizzate.

2.4.3 Altri algoritmi di classificazione

2.4.3.1 Support Vector Machines

Vale la pena a questo punto fare un accenno ad un altro algoritmo di classificazione che abbiamo utilizzato durante la progettazione del nostro modello, ma che poi abbiamo scelto di scartare in quanto produceva risultati meno convincenti della controparte basata su albero di decisione.

La tecnica delle **Support Vector Machines** pone le sue radici nella teoria dello **statistical learning** ed ha mostrato promettenti risultati in molte applicazioni pratiche, dal riconoscimento di numeri scritti a mano fino alla categorizzazione del testo. Le SVM funzionano bene anche con dati dall'elevata dimensionalità e permettono di aggirare il

problema della maledizione della dimensionalità.

Prima di parlare dell'idea alla base delle SVM, introduciamo il concetto di **iperpiano dal margine massimale** e spieghiamo cosa porta a scegliere tale iperpiano.

Consideriamo la Figura 2.15, la quale mostra uno spazio in cui i dati appartengono a due classi, rappresentate da quadrati e cerchi:

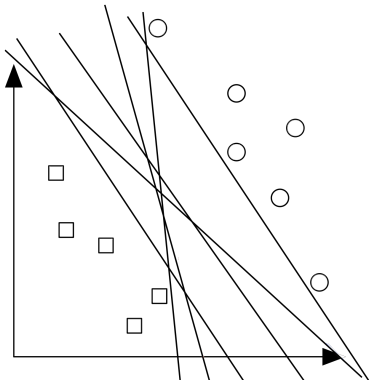

Figura 2.14: Possibili decision boundaries per un dataset linearmente separabile

Il dataset in figura è anche linearmente separabile (È possibile trovare un iperpiano tale da separare tutti i quadrati dai cerchi). Come possiamo notare, ci sono un infinito numero di iperpiani in grado di separare tale dataset. Nonostante il loro errore di training sarà nullo, non c'è garanzia che gli iperpiani avranno performance equivalenti quando si prendono in considerazione nuovi record. Il classificatore dovrà pertanto scegliere uno di questi iperpiani per rappresentare il proprio decision boundary, basando la decisione su quanto bene è previsto funzionare sugli esempi di test.

Per meglio comprendere come le differenti scelte dell'iperpiano di separazione influenzano l'errore di generalizzazione, consideriamo due decision boundaries differenti, ai quali ci riferiremo rispettivamente come B_1 e B_2. Entrambi i decision boundaries possono separare gli esempi di training nelle loro rispettive classi senza commettere alcun errore di classificazione.

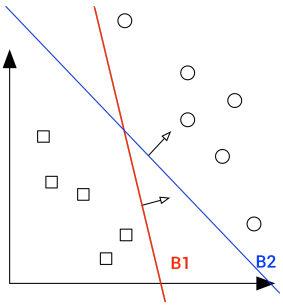

Figura 2.15: Dati separati da due diversi decision boundaries

Ogni decision boundary B_i è associato a due iperpiani, che denotiamo come b_{i1} e b_{i2}, rispettivamente. b_{i1} è ottenuto muovendo un iperpiano parallelo finchè non arriva a toccare l'elemento della Classe 1 (Quadrato) più vicino, mentre b_{i2} è ottenuto facendo lo stesso con l'elemento della Classe 2 (Cerchio) più vicino. La distanza tra questi due iperpiani è detta **margine del classificatore**. Dalla Figura 2.17 si nota come il margine per B_2 è significativamente maggiore del margine per B_1. In questo specifico esempio, B_2 risulta essere l'**iperpiano dal margine massimale** sulle istanze di training.

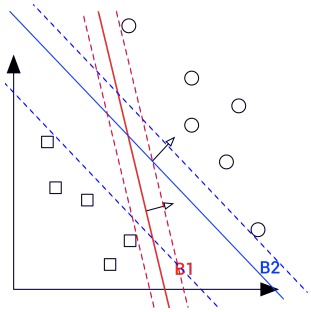

Figura 2.16: Rappresentazione grafica del margine

I decision boundaries con ampio margine tendono ad avere un miglior errore di generalizzazione rispetto a quelli con piccolo margine. Intuitivamente, se il margine è piccolo, sensibili perturbazioni del decision boundary possono comportare un impatto significativo sulla sua classificazione. I classificatori che producono decision boundaries con piccolo margine sono pertanto più succettibili all'overfitting, tendendo a generalizzare poco su record nuovi.

Scopo primario di chiunque vuole realizzare un classificatore lineare è quello di massimizzare i margini dei propri decision boundaries in modo da assicurarsi che il loro errore di generalizzazione, nel caso peggiore, sia minimizzato. Un classificatore di questo tipo è detto **Support Vector Machine Lineare**.

Un SVM lineare è un classificatore che cerca un iperpiano con massimo margine, ed è pertanto conosciuto anche come **classificatore dal margine massimale.** Non ci addentreremo maggiormente nei dettagli per quanto riguarda il modo in cui un SVM apprende il suo decision boundary, in quanto non è obiettivo di questo elaborato entrare nei dettagli delle alternative che sono state scartate in fase di progettazione.

2.4.3.2 Reti neurali

In questa sezione faremo una breve panoramica sui fondamenti delle **Reti neurali artificiali**, senza entrare troppo nei dettagli, ma esponendo sinteticamente quelli che sono i concetti e le idee principali alla base di tale approccio di classificazione, per permettere la comprensione delle sezioni successive, in particolare quando faremo riferimento alla classificazione **Doc2Vec e Word2Vec**.

Lo studio delle reti neurali prende ispirazione dai tentativi di simulare il comportamento dei sistemi neurali biologici. Il cervello umano è composto prevalentemente da **neuroni**, elementi connessi l'uno con l'altro tramite filamenti di fibre chiamati **assoni**. Tali filamenti sono utilizzati per trasmettere impulsi nervosi da un neurone all'altro ogni volta che sono stimolati. Un neurone è connesso agli assoni degli altri neuroni tramite i **dendriti**, che sono estensioni del corpo di un neurone. Il punto di contatto tra un dendrite ed un assone prende il nome di **sinapsi**. I neurologi hanno scoperto che il cervello umano apprende cambiando la forza della connessione sinaptica tra i neuroni a seguito di stimolazioni ripetute dello stesso impulso.

Analogamente alla struttura del cervello umano, una Rete Neurale Artificiale è composta da un insieme di nodi interconnessi e da collegamenti direzionali.

Il modello più semplice di rete neurale prende il nome di **percettrone**. Considerato il diagramma in Figura 2.18, si nota come la tabella sulla sinistra mostri tre variabili booleane (x_1, x_2, x_3) ed una variabile di output y, che assume valore -1 se almeno due dei tre input hanno valore zero, mentre assume valore $+1$ se almeno due input sono maggiori di zero.

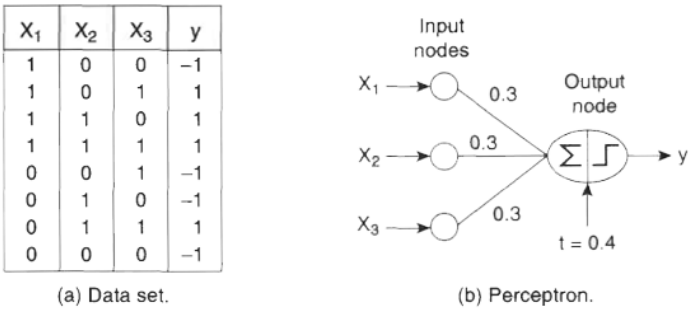

Figura 2.17: Esempio di percettrone

La Figura 2.18 (b) mostra una semplice architettura di rete neurale. Il percettrone è composto da due tipi di nodi:

- **Nodi di input**: utilizzati per rappresentare gli attributi in input.
- **Nodo di output**: utilizzato per rappresentare il modello di output.

I nodi in un'architettura di rete neurale sono indicati comunemente come **neuroni** o unità. In un percettrone, ogni nodo di input è collegato tramite un arco pesato all'unico nodo di output. L'arco pesato è utilizzato per emulare la forza di una connessione sinaptica tra i neuroni di un sistema neurale biologico. Analogamente infatti, l'addestramento di un percettrone si riduce a modificare i pesi dei collegamenti finchè non si adattano alle relazioni input-output dei dati sottostanti.

Il percettrone calcola il suo valore di output $\hat{y}$ facendo la somma pesata dei suoi input, sottraendo tale valore al fattore soglia t e infine verificando se il valore risultante è positivo o meno.

Si noti la differenza sostanziale tra nodi di input e nodo di output. I nodi di input semplicemente trasmettono il valore ricevuto al link in uscita senza effettuare alcuna trasformazione. Il nodo di output invece effettua delle computazioni, calcolando di fatto la somma pesata dei suoi input, sottraendo il valore soglia e producendo un output che dipende dal segno del valore calcolato. Formalmente, l'output di un percettrone può essere espresso matematicamente come segue:

$$\hat{y} = sign(w_1 x_1 + w_2 x_2 + ... + w_d x_d - t)$$

Con w_i pesi dell'arco di input e x_i che rappresentano i valori degli attributi di input.

La funzione $sign$, che agisce come **funzione di attivazione** per il neurone di output, restituisce valore $+1$ se il suo argomento è positivo, -1 se è negativo. In forma vettoriale, il modello del percettrone può essere scritto come:

$$sign(w \cdot x)$$

Durante la fase di addestramento di un modello basato su percettrone, i pesi w sono aggiustati finchè gli output del percettrone diventano consistenti con gli output reali degli esempi di training. La computazione base per questo algoritmo è la **formula di aggiornamento dei pesi:**

$$w_j^{(k+1)} = w_j^{(k)} + \lambda(y_i - \hat{y}_i^{(k)})x_{ij}$$

Con $w^{(k)}$ il peso associato all'i-esimo collegamento di input dopo la k-esima iterazione, λ parametro noto come **learning rate**, e x_{ij} valore del j-esimo attributo del record di training x_i.

La comprensione di tale formula è abbastanza semplice, in quanto l'equazione mostra che il nuovo peso $w^{(k+1)}$ è una combinazione del vecchio peso $w^{(k)}$ ed un termine proporzionale all'errore di predizione $(y - \hat{y})$. Se la predizione è corretta, allora il peso rimane uguale, altrimenti viene modificato. Nella formula di aggiornamento dei pesi, i collegamenti che contribuiscono maggiormente all'errore sono quelli che richiedono un maggior aggiustamento. D'altro canto però, i pesi non dovrebbero essere cambiati troppo drasticamente in quanto l'errore è calcolato solo per il corrente esempio di training,

ed i cambiamenti fatti nelle iterazioni precedenti sarebbero eliminati. Il learning rate λ, un parametro il cui valore è compreso tra 0 e 1, è usato per controllare gli aggiustamenti fatti ad ogni iterazione. Per valori piccoli di λ, il nuovo peso sarà maggiormente influenzato dal valore del vecchio peso. Per valori di λ prossimi ad 1, il nuovo peso sarà maggiormente sensibile all'ammontare dell'aggiustamento effettuato all'iterazione corrente. In alcuni casi si utilizza un learning rate adattivo, che decresce gradualmente all'avanzare delle iterazioni.

Il modello del percettrone appena introdotto è lineare. Per questo motivo, il decision boundary del percettrone, che è ottenuto forzando il valore $\hat{y} = 0$, è un iperpiano lineare che separa i dati in due classi $+1$ e -1.

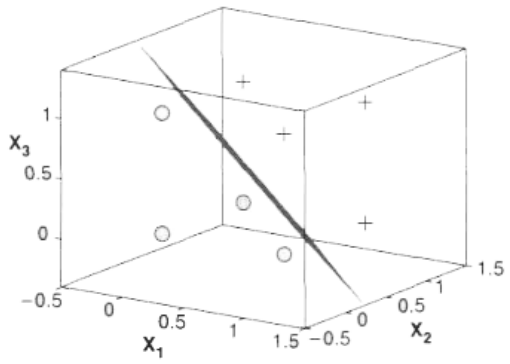

Figura 2.18: Iperpiano di separazione di un percettrone

Una rete neurale artificiale ha una struttura molto più complessa di quella di un percettrone. Tale complessità può essere aggiunta in diversi modi:

- La rete può contenere diversi livelli intermedi tra il livello di input e quello di output. Tali livelli prendono il nome di **hidden layer** (livelli nascosti) e i nodi che si trovano al loro interno prendono il nome di **hidden nodes** (nodi nascosti). In una rete neurale **feed-forward**, i nodi di un livello sono connessi solo ai nodi del livello successivo. Il percettrone è una rete neurale feed-forward con un solo livello, in quanto ha un solo livello di nodi, il livello di output, che effettua complesse operazioni matematiche. In una **recurrent neural network**, i collegamenti

possono connettere nodi anche facenti parte dello stesso livello o nodi da un livello precedente a quello successivo (saltando un livello).

- La rete può usare diversi tipi di funzioni di attivazione oltre che la funzione *sign*. Altre funzioni di attivazione includono funzioni lineari, sigmoidi e tangenti iperboliche. Tali funzioni permettono ai nodi di input e di hidden di produrre valori di output che non sono lineari nei loro parametri di input.

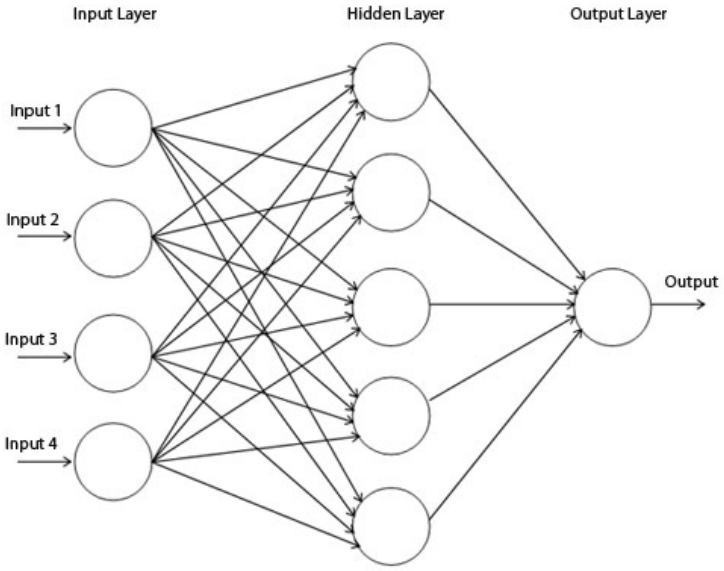

Figura 2.19: Esempio di percettrone multi-livello

2.5 Valutazione di un classificatore

Come abbiamo già accennato più volte nel corso dell'elaborato, risulta particolarmente utile misurare le performance del modello non solo sul training set, ma anche sul test set, in quanto in questo modo è possibile ottenere una stima più precisa dell'errore di generalizzazione.

In questa sezione andremo ad analizzare diversi metodi utilizzati comunemente per valutare le performance di un classificatore.

2.5.1 Metodo holdout

Nel metodo holdout, i dati originali con gli esempi etichettati sono partizionati in due set distinti, rispettivamente **training** e **test set**. Un modello di classificazione è poi creato utilizzando i dati di training, mentre le sue performance sono valutate utilizzando il test set. La proporzione di dati riservati per il training e per il testing è a discrezione dell'analista. L'accuratezza del classificatore viene poi valutata in base all'accuratezza raggiunta dal classificatore stesso sui dati di test.

Il metodo holdout ha diverse, ben note, limitazioni. In primo luogo, si hanno meno record per l'apprendimento in quanto una parte del dataset viene tenuta per la fase di testing. Di conseguenza, il modello generato potrebbe non essere così buono come quello generato utilizzando l'intero dataset.

Inoltre, il modello potrebbe essere molto dipendente dalla composizione del training e del test set. Minore è la dimensione del training set, maggiore sarà la varianza del modello. D'altro canto, se il training set è troppo grande, allora l'accuratezza stimata dal test set, che sarà di minori dimensioni, sarà meno affidabile.

Ultimo ma non meno importante, il training e test set non saranno più indipendenti l'uno dall'altro. Questo perchè entrambi saranno delle partizioni dei dati originali, ed una classe che potrebbe essere preponderante in uno dei due risulterà essere meno presente nell'altro, e viceversa.

2.5.2 Random subsampling

Il metodo holdout può essere ripetuto più volte per migliorare la stima delle performance di un classificatore. Questo approccio è noto come **random subsampling**.

Sia acc_i l'accuratezza del modello all'i-esima iterazione. L'accuratezza totale sarà data da:

$$acc_{sub} = \sum_{i=1}^{k} \frac{acc_i}{k}$$

Il random subsampling presenta ancora alcuni dei problemi introdotti con il metodo holdout in quanto non utilizza tutti i dati possibili per il training. Inoltre non ha alcun controllo sul numero di volte in cui un record è utilizzato per il training ed il testing. Di conseguenza, alcuni record potrebbero essere utilizzati per il training più volte di altri.

2.5.3 Cross Validation

Arriviamo adesso al metodo che maggiormente abbiamo utilizzato per validare i nostri classificatori.

Con la **Cross-Validation**, ogni record è utilizzato esattamente lo stesso numero di volte sia per il training che per il testing del classificatore. Per illustrare questo metodo, supponiamo di partizionare i dati in due partizioni di uguali dimensioni. Inizialmente, selezioniamo una delle due partizioni per il training e l'altra per il testing. Successivamente invertiamo i ruoli in modo tale che il precedente training set diventi il test set e viceversa.

Questo approccio è detto **Two-Folds Cross Validation**. L'errore totale è ottenuto sommando gli errori di entrambe le esecuzioni. In questo esempio, ogni record è utilizzato esattamente una volta per il training e una volta per il testing.

Il metodo **K-Folds Cross Validation** generalizza questo approccio segmentando i dati in k partizioni di egual misura. Durante ogni iterazione, una delle partizioni viene scelta per il testing, mentre tutte le altre sono utilizzate per il training. Questa procedura è ripetuta k volte così che ogni partizione è utilizzata per il testing esattamente una volta. Di nuovo, l'errore totale si calcola sommando gli errori per ognuna delle k iterazioni.

Un caso speciale della K-Folds Cross Validation prevede di impostare $k = N$, con N dimensione del dataset (Numero di record disponibili). Questo approccio prende il nome di **leave-one-out**, e comporta l'utilizzo di **un solo record per il test set**. Questo approccio ha il vantaggio di utilizzare il maggior numero di dati disponibili per

il training. Inoltre, i test set sono mutualmente esclusivi e coprono l'intero dataset. Lo svantaggio di questa soluzione è che è computazionalmente oneroso ripetere la procedura N volte. Inoltre, dato che ogni test set contiene un solo record, la varianza della stima della metrica delle performance tende ad essere elevata.

2.6 Algoritmi di regressione

Chiudiamo questo capitolo facendo una panoramica sugli algoritmi di regressione, in particolare sulla **regressione lineare**, che ricordiamo essere una tecnica di apprendimento supervisionato ad output continui. La regressione lineare è un algoritmo che abbiamo utilizzato in una delle nostre versioni del classificatore di tweet, pertanto per poterlo comprendere a pieno è necessario introdurre alcuni concetti teorici che ne sono alla base.

La regressione è una tecnica di **predictive modeling** in cui la variabile target da stimare è continua. Esempi di applicazione della regressione includono la predizione di indici di mercato azionario o di altri indicatori economici, la previsione delle precipitazioni in una certa zona basandosi sui dati relativi al vento e alle correnti aeree, la proiezione delle vendite totali di una compagnia basate sulla quantità di investimenti fatti per l'advertising o addirittura la stima dell'età di un fossile in accordo con la presenza di componenti organici al suo interno.

Entriamo adesso più nel dettaglio, introducendo qualche formalismo. Consideriamo un dataset D contenente N osservazioni (record):

$$D = \{(x_i, y_i)|i = 1, 2, ..., N\}$$

Con x_i che corrisponde al set di attributi dell'i-esima osservazione (anche detti **variabili esplicative**, che possono essere sia continue che discrete) ed y_i che rappresenta la **variabile target**.

Per definizione, la regressione è il **processo di apprendimento di una funzione target f che effettua il mapping di ogni attributo x in un output continuo**

$\boldsymbol{y}$.

L'obiettivo principale della regressione è quindi quello di trovare la funzione target che sia in grado di adattarsi all'input con il minimo errore. La **funzione di errore** per un task di regressione può essere espressa in termini di:

$$Absolute\ Error = \sum_i |y_i - f(x_i)|$$

$$Squared\ Error = \sum_i (y_i - f(x_i))^2$$

2.6.1 Regressione Lineare Semplice

Per comprendere il funzionamento della **regressione lineare**, consideriamo un dataset fisiologico così composto:

Heat Flux	Skin Temperature
10.858	31.002
10.617	31.021
10.183	31.058
9.7003	31.095
9.652	31.133
10.086	31.188
9.459	31.226
8.3972	31.263
7.6251	31.319
7.1907	31.356
7.046	31.412
6.9494	31.468
6.7081	31.524

Heat Flux	Skin Temperature
6.3221	31.581
6.0325	31.618
5.7429	31.674
5.5016	31.712
5.2603	31.768
5.1638	31.825
5.0673	31.862
4.9708	31.919
4.8743	31.975
4.7777	32.013
4.7295	32.07
4.633	32.126
4.4882	32.164

Heat Flux	Skin Temperature
4.3917	32.221
4.2951	32.259
4.2469	32.296
4.0056	32.334
3.716	32.391
3.523	32.448
3.4265	32.505
3.3782	32.543
3.4265	32.6
3.3782	32.657
3.3299	32.696
3.3299	32.753
3.4265	32.791

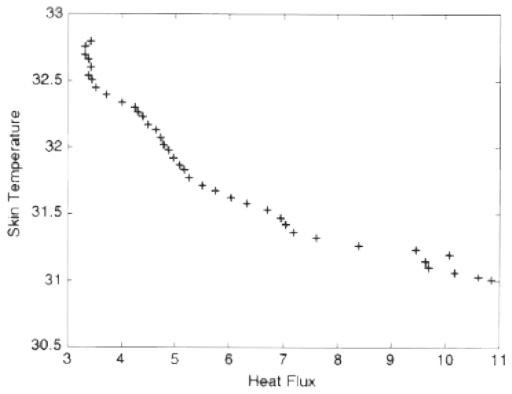

Figura 2.20: Dataset d'esempio per regressione lineare

I dati ivi presentati rappresentano delle misurazioni del flusso di calore e della temperatura corporea di una persona durante il sonno. Supponiamo di esser interessati nel predire la temperatura corporea di una persona basandoci sulle misurazioni del flusso

di calore catturate da un sensore di calore. Lo scatter-plot bidimensionale mostra che c'è una forte relazione lineare tra le due variabili.

Supponiamo di voler adattare il seguente modello lineare ai dati osservati:

$$f(x) = \omega_1 x + \omega_0$$

Con ω_0 e ω_1 parametri del modello, chiamati **coefficienti di regressione**. Un semplice approccio per fare ciò è quello di applicare il metodo di minimizzazione dell'errore quadratico, ovvero cercare i parametri (ω_0, ω_1) che minimizzano la somma:

$$SSE = \sum_{i=1}^{N} [y_i - f(x_i)]^2 = \sum_{i=1}^{N} [y_i - \omega_1 x - \omega_0]^2$$

Tale problema di ottimizzazione è risolvibile prendendo le **derivate parziali di E rispetto a ω_0 e ω_1**, e risolvere il corrispondente sistema di equazioni lineari:

$$\frac{\delta E}{\delta \omega_0} = -2 \sum_{i=1}^{N} [y_i - \omega_1 x_i - \omega_0]$$

$$\frac{\delta E}{\delta \omega_1} = -2 \sum_{i=1}^{N} [y_i - \omega_1 x_i - \omega_0] x_i$$

Senza entrare troppo nei dettagli, questo è un approccio sistematico per addestrare un modello minimizzando l'errore tra il reale valore di y e la sua stima. Nonostante questo sia un modello relativamente semplice, ha dimostrato di fornire una approssimazione ragionevolmente accurata.

Capitolo 3

Il classificatore di tweet: Progettazione

Entriamo finalmente nel cuore pulsante dell'intero progetto.

Come abbiamo già accennato più volte nei precedenti capitoli, l'incarico di nostra competenza prevedeva la **realizzazione di un classificatore** in grado di analizzare i tweet generati durante una qualsiasi catastrofe e di etichettarli come **rilevanti** o **non rilevanti** dal punto di vista di chi deve organizzare o usufruire dei soccorsi.

Per permettere al classificatore di apprendere e di essere in grado di categorizzare in tal modo i tweet, si necessita di una grande quantità di dati prelevati durante catastrofi avvenute nel passato. A tal scopo ci viene incontro la piattaforma `www.crisislex.org` [15], i cui ideatori e manutenitori si sono incaricati di monitorare e memorizzare il flusso di tweet generato durante tantissime catastrofi come terremoti, alluvioni o incendi. Tali dati non erano ovviamente adatti, nella forma in cui vengono distribuiti, ad addestrare il classificatore e renderlo adatto al nostro scopo, pertanto si sono rese necessarie tutte una serie di elaborazioni, dall'estrazione di metadati associati ad ogni tweet fino ad arrivare alla codifica *Doc2Vec* del contenuto di ogni tweet, che andremo ad esporre nel corso di questo capitolo e di quelli a venire.

A seguito di tutta una serie di analisi e prove fatte nei mesi precedenti, dal nostro lavoro sono venuti fuori **due classificatori differenti**, ognuno utilizzante diverse tecniche ed avente pro e contro che analizzeremo nelle prossime sezioni. Distinguiamo i nostri modelli in questo modo:

- **Classificatore 1 (C1) - Metadati e Flag NLP:** La nostra prima versione del classificatore, che prevede la realizzazione di un albero di decisione addestrato su un dataset composto da **metadati associati ai tweet** (tra cui numero di retweet, numero di followers dell'autore ecc.) e dalla presenza di alcuni **flag ad indicare la presenza o meno di alcune caratteristiche all'interno del testo del tweet** (tra cui la presenza di una nazione, di una città ecc.), la cui estrazione è stata realizzata tramite l'utilizzo dello **Stanford NLP NER Tagger**, un *"Name Entity Recognition Tagger"*, il cui funzionamento sarà esposto in una sezione appositamente dedicata.

- **Classificatore 2 (C2) - Doc2Vec:** Una versione alternativa e avanzata del classificatore, realizzata sfruttando le tecniche di codifica *Doc2Vec*, che come vedremo nella sezione apposita, trasformano il testo di ogni tweet in un vettore numerico avente peculiari caratteristiche che risultano essere adatte all'addestramento di un modello di regressione lineare, che abbiamo utilizzato per la classificazione dei tweet nelle due classi rilevante e non rilevante.

In questo capitolo andremo ad analizzare, in ordine, prima gli aspetti che accomunano i due modelli, per poi passare alle tecniche specifiche che ognuno di essi utilizza per il riconoscimento dei tweet, ad esclusione dei modelli di classificazione su cui si basano, ovvero alberi di decisione, reti neurali e regressione lineare, che sono già stati trattati nei capitoli precedenti.

3.1 Il dataset originale

Come abbiamo già spiegato precedentemente, per realizzare i nostri classificatori abbiamo avuto necessità di affidarci ad una piattaforma esterna, `www.crisislex.org`, che ha raccolto il flusso di informazioni condiviso su Twitter durante il verificarsi di

situazioni di pericolo, quali catastrofi naturali e causate dall'uomo.

I flussi di tweet monitorati e registrati dall'organizzazione che abbiamo utilizzato per i nostri scopi, sono relativi in particolari agli eventi di:

- **Incendi boschivi in Colorado, USA - 2012** [16]
- **Alluvioni in Alberta, Canada - 2013** [17]
- **Incendi in New South Wales, Australia - 2013** [18]
- **Attentato alla maratona di Boston, USA - 2013** [19]
- **Alluvioni in Colorado, USA - 2013** [20]
- **Elicottero precipitato a Glasgow, Scozia - 2013** [21]
- **Attentato all'aeroporto di Los Angeles, California - 2013** [22]
- **Deragliamento di un treno a New York City, USA - 2013** [23]
- **Alluvioni a Queensland, Australia - 2013** [24]
- **Esplosione in West Texas, USA - 2013** [25]

Tutti i dataset qui sopra elencati hanno la stessa struttura, la quale, come abbiamo già detto, non rispecchia quella di cui necessitiamo noi per l'addestramento di nessuno dei due classificatori che abbiamo introdotto, e richiedono pertanto operazioni aggiuntive, che esporremo nelle sezioni successive.

I dataset sono distribuiti in un formato `*.csv`, e comprendono i seguenti attributi:

- **Tweet ID**: Identificativo univoco del tweet, tramite il quale si può risalire ai metadati mantenuti da Twitter e dalle informazioni sull'autore del tweet stesso.
- **Tweet Text**: Contenuto del tweet, ovvero il testo postato dall'utente.

- **Information Source**: Fonte dell'informazione, ovvero a che categoria appartiene colui che ha condiviso l'informazione. Alcune delle categorie identificate sono enti governativi (*"Government"*), media (*"Media"*), *Outsiders*, intesi come persone non correlate direttamente all'avvenimento ecc.

- **Information Type**: Tipo dell'informazione condivisa, come supporto morale e compartecipazione (*"Support and sympathy"*), donazioni e volontariato (*"Donation and volunteering"*) ecc.

- **Informativeness (Etichetta di classe)**: Indicazione sulla presenza di una relazione tra il tweet e l'evento che si sta considerando.

L'ultimo attributo merita una menzione ed un approfondimento particolare. Esso infatti, è un attributo nominale che può assumere i seguenti valori:

- **"Related and informative"** (Correlato ed informativo): indica che il tweet non solo è relativo all'evento considerato, ma porta con sè informazioni importanti, quali possono essere aggiornamenti sullo stato dell'evento (Es.: *"Alle ore 12.13 l'incendio si è esteso per 10000 mq"*), comunicazioni cruciali per le operazioni di supporto ecc.

- **"Related but not informative"** (Correlato ma non informativo): indica che il tweet è relativo all'evento considerato, ma non aggiunge informazioni utili (Es.: *"Le mie condoglianze alle famiglie colpite dall'attentato"*).

- **"Not related"** (Non correlato): indica che il tweet non è correlato in nessun modo all'evento considerato (Es.:*"Forza Italia!"*).

Tutti i tweet sono stati etichettati "manualmente" da persone che si sono incaricate di leggerli, comprenderli, ed assegnarli ad una delle diverse categorie. Molti classificatori già esistenti prevedono infatti un processo di tagging manuale, effettuato durante le prime fasi dell'evento catastrofico, in maniera tale da permettere l'apprendimento e la "messa in moto" del sistema immediatamente dopo. Risulta chiaro come però un sistema del genere sia poco efficace, in quanto è necessario prima di tutto avere delle persone a disposizione, che volontariamente accettano di incaricarsi di tale operazione,

e inoltre comporta un inevitabile ritardo, con i tweet prodotti immediatamente dopo l'inizio della catastrofe che devono essere classificati da umani. Vedremo successivamente come abbiamo cercato di aggirare il problema nella realizzazione della nostra versione del classificatore.

Tornando al dataset, notiamo come vi siano delle informazioni irrilevanti dal nostro punto di vista, ovvero:

- **Information Source e Information Type**: Introducono una maggiore granularità della classificazione, alla quale non siamo interessati. Per rendere il sistema più veloce e libero dalla fase di tagging decidiamo di ignorare tali informazioni.
- **Informativeness**: Come abbiamo già detto, il nostro classificatore ha il compito di etichettare i tweet assegnandogli una delle due classi "Rilevante" o "Non rilevante". I dataset che abbiamo ottenuto prevedono la presenza di 3 etichette, delle quali però abbiamo considerato soltanto ***"Relevant but not informative"*** che abbiamo indicato come ***"Non rilevante"***, e ***"Relevant and informative"*** che abbiamo associato all'etichetta ***"Rilevante"***. Abbiamo deciso di ignorare i tweet con etichetta ***"Not relevant"*** in quanto avrebbero potuto portare ad una classificazione fuorviante.

Queste sono però soltanto operazioni preliminari, alle quali seguono poi elaborazioni che variano a seconda della versione del classificatore (C1 e C2), e per le quali dedicheremo delle sezioni apposite nelle prossime righe.

3.2 Estrazione dei metadati

Abbiamo visto quindi qual è la struttura del dataset originale, e quali sono le piccole modifiche che si sono rese necessarie per renderlo compatibile con i nostri scopi. Allo stato attuale, il dataset non contiene feature particolarmente adatte per l'addestramento del *Classificatore C1*, che ricordiamo ancora essere basato su un albero di decisione.

Dunque, la prima operazione che si è resa necessaria è stato il recupero da Twitter delle **meta-informazioni** o **meta-dati**, correlati ai singoli tweet o all'account di chi li ha generati. L'accesso a queste informazioni è permesso dall'utilizzo delle **API di Twitter**, che permettono il prelievo di informazioni correlate ad ogni singolo tweet, tramite l'utilizzo di API Rest e richieste HTTP.

Vale la pena fare un accenno all'oggetto principe delle API di Twitter, ovvero l'**oggetto `tweet`**. Tale oggetto è un riferimento ad uno specifico tweet in **formato JSON**, e contenente al suo interno, tra le altre cose:

- **`created_at:`** Stringa contenente l'orario in cui il tweet è stato generato, in formato UTC (Tempo Coordinato Universale)
- **`id:`** Intero rappresentante l'ID del tweet, ovvero l'identificativo univoco tramite il quale ogni tweet può essere riferito ed acceduto.
- **`text:`** Testo del tweet
- **`source:`** Utility utilizzata per postare il tweet. Identifica l'applicazione ed il sistema operativo tramite il quale il tweet è stato generato e pubblicato (Es.: Twitter App for iOS).
- **`user:`** Oggetto composto che contiene al suo interno le informazioni relative all'utente che ha pubblicato il tweet. Le più importanti sono:
  - `id`: Identificativo univoco dell'utente (Non è l'username, ma un identificativo numerico assegnato automaticamente da Twitter all'atto della registrazione).
  - `name`: Username dell'account che ha generato il tweet.
  - `location`: Posizione geografica dell'utente, come da lui indicato.
  - `description`: Una breve descrizione dell'utente, scritta da lui.
  - `verified`: Flag che indica se l'account è verificato o meno. Gli account verificati sono quelli che si identificano con un badge blu di fianco al nome utente. Il badge blu di account verificato su Twitter permette alle persone

di sapere che un account di interesse pubblico è autentico, ed è assegnato dopo una serie di verifiche svolte direttamente da Twitter.

- `followers_count`: Numero di followers (Persone che seguono gli aggiornamenti) dell'utente che ha generato il tweet.
- `friends_count`: Numero di account seguiti dall'utente che ha generato il tweet.
- `favourites_count`: Numero di tweet piaciuti all'account che ha generato il tweet.
- `statuses_count`: Numero totale di tweet pubblicati dall'utente.
- `geo_enabled`: Flag che indica se l'utente ha dato la possibilità di geo-taggare (Assegnare una posizione ottenuta tramite il modulo GPS del suo dispositivo, o tramite indirizzo IP) i propri tweet.

Molte di queste informazioni sono state utilizzate per l'addestramento del nostro classificatore, mentre altri attributi sono stati generati facendo delle elaborazioni su esse.

Dopo vari tentativi e una accurata operazione di **feature selection**, l'insieme di metadati utilizzato per l'apprendimento è stato così composto:

Tabella 2: Metadati utilizzati per il Classificatore C1

Attributo	Descrizione
TweetID	ID univoco del tweet
CreationTime	Data e ora di pubblicazione
Followers	Numero di followers dell'utente
Followed	Numero di account seguiti dall'utente
GeoTagged	Flag che indica se c'è geo-tag associato
TotalTweets	Numero totale di tweet pubblicati dall'utente
TwitterAge	Giorni da quando l'utente ha creato l'account
nHashtags	Numero di hashtag presenti nel tweet
nMentions	Numero di mentions associate al tweet
nUrls	Numero di URLs presenti nel tweet
Verified	Flag che indica se l'account è verificato
nRetweets	Numero di retweet
DeltaSeconds	Tempo trascorso dal primo tweet della crisi

3.3 Elaborazione del testo: Entity Extraction

Arrivati a questo punto sarebbe possibile già realizzare una prima versione del classificatore, con l'apprendimento basato sui metadati estratti appena elencati. Nonostante essi permettano di realizzare un modello dalle discrete performance, c'è un aspetto che vale la pena sottolineare e tenere in considerazione. Noterete infatti, come tra i metadati non vi sia alcun riferimento al testo del tweet, che ovviamente può portare con sè informazioni molto preziose. Molti studiosi che si son posti il problema simile al nostro infatti, hanno stabilito di non includere alcuna informazione sul testo per una serie di motivazioni che possiamo sintetizzare in questo modo:

- Rendere il classificatore indipendente dal linguaggio utilizzato: includendo informazioni sul testo, come la frequenza di una certa parola all'interno del dataset (E quindi dei diversi testi dei tweet), si vincola inevitabilmente il modello al linguaggio utilizzato nel contenuto del tweet. L'albero di decisione infatti, sarà caratterizzato da nodi con condizioni di split basate sulla frequenza di una parola

(O indici da essa derivati, come il ben noto TF-IDF), la quale sarà però espressa in uno specifico linguaggio. Utilizzando il classificatore per etichettare tweet scritti in altri linguaggi, il modello non funzionerà correttamente.

- Permettere al classificatore di essere utilizzabile in tempi più brevi: le tecniche di Natural Language Processing sono computazionalmente onerose e possono rallentare il riconoscimento dei tweet.

Inoltre, l'utilizzo della frequenza delle parole (o degli indici derivati da essa) per la classificazione comporta l'introduzione di alcune problematiche, tra cui:

- Maggiori probabilità di overfitting: Dato che il classificatore viene addestrato sulla base di dati relativi ad eventi passati, avvenuti in uno specifico luogo, è molto probabile che parole vincolate al singolo evento vengano selezionate come miglior split e inserite nell'albero fin dai livelli più alti. Ad esempio, se decidiamo di addestrare il classificatore utilizzando tweet relativi ad un evento avvenuto in Pakistan, è molto probabile che nell'albero figureranno nodi che baseranno il loro split sulla frequenza di parole come *"Pakistan"* o *"Asia"*, che difficilmente figurano nei tweet relativi a catastrofi avvenute in altri luoghi.

- La lingua inglese, così come tutte le altre lingue, non è "universale", nel senso che vi sono espressioni e modi di dire che variano da luogo a luogo, pertanto l'inglese parlato in Pakistan risulta sicuramente diverso da quello parlato in UK. Di conseguenza bisogna ponderare bene i dati da utilizzare in fase di training, per evitare che il classificatore prenda delle decisioni basate su parole che sono utilizzate soltanto in alcuni paesi, portando a classificazioni sbagliate in fase di utilizzo con dati provenienti da altri luoghi.

- Tali tecniche richiedono di addestrare nuovamente il classificatore ogni volta che lo si utilizza con un nuovo dataset. Questo perchè, i valori degli indici basati sulla frequenza, come il TF-IDF, dipendono dalle parole utilizzate e dai documenti sui quali si verifica la presenza di ogni parola. Per questo motivo, dopo che il classificatore è addestrato, una volta estratto un altro insieme di dati relativo ad

una nuova catastrofe (Il quale introduce ulteriore ritardo, in quanto bisogna avere tutto il dataset disponibile contemporaneamente, e non si possono classificare tweet singolarmente, appena vengono generati) sarà necessario addestrare da capo l'intero classificatore per considerare nell'albero di decisione eventuali nuove parole, non presenti nei precedenti dataset, e ricalcolare il valore dell'indice di tutti quelli precedenti, in quanto avendo introdotto altri documenti (tweet) la frequenza varierà.

É innegabile però, che il testo porta con sè informazioni molto rilevanti per la comprensione e l'etichettatura dei tweet, e trovando il modo di aggirare alcuni dei problemi appena introdotti si possono ottenere prestazioni notevolmente migliori.

Per questo motivo, in fase di progettazione del *Classificatore C1*, siamo venuti fuori con un'idea che riesce ad evitare alcuni dei problemi che la classificazione basata sul testo e sulla frequenza delle parole porta con sè. In particolare, abbiamo scelto di utilizzare tecniche di **Natural Language Processing**, come la **Name Entity Recognition** per estrarre informazioni dal testo e di utilizzarle però come **attributi binomiali** in fase di classificazione, ad indicare la presenza o l'assenza di una determinata entità all'interno del tweet.

In questo modo, si ottengono informazioni rilevanti dal testo del tweet, senza però essere vincolati nè al linguaggio ed alle sue varianti nè alle parole ed alla loro frequenza nel dataset.

3.3.1 Natural Language Processing - POS Tagging

Abbiamo più volte nominato nel corso dell'elaborato le tecniche di Natural Language Processing, spesso indicato semplicemente come NLP. Ma a cosa facevamo riferimento? In questa sezione esporremo i principi fondamentali di questa disciplina, prima di addentrarci nelle tecniche specifiche che abbiamo utilizzato per analizzare ed estrarre informazioni dal testo dei tweet.

Il **Natural Language Processing** è un'area di ricerca ed applicazione che esplora i metodi attraverso i quali i computer possono essere utilizzati per **comprendere e manipolare testi o discorsi in linguaggio naturale** per effettuare poi operazioni utili. Le fondamenta del NLP giacciono in diverse discipline che spaziano dall'informatica alla linguistica, dalla matematica all'intelligenza artificiale e alla robotica, fino a sfociare anche nella psicologia. [26]

Gli uomini comunicano in molti modi: attraverso il parlato e l'ascolto, facendo gesti (ad esempio durante la guida o per fornire indicazioni) e diverse forme di testo. Per testo si intende parole che sono scritte o stampate su carta, o mostrate su uno schermo di un dispositivo elettronico per essere lette dal destinatario prestabilito.

Una delle operazioni basilari che possono esser applicate al testo è la **tokenizzazione**: suddivisione di uno stream di caratteri in parole, simboli di punteggiatura, simboli ed altri elementi discreti. Ad esempio, la frase:

"Dr. Watson, Mr. Sherlock Holmes", said Stamford, introducing us.

Può essere suddivisa come segue:

'"', 'Dr.', 'Watson', ',' , 'Mr.', 'Sherlock', 'Holmes', '"', ',' , 'said', 'Stamford', ',' , 'introducing', 'us', '.'

A questo livello, le parole non sono state ancora classificate in categorie grammaticali e si hanno ancora pochissime indicazioni sulla struttura sintattica. Nonostante ciò, un discreto quantitativo di informazioni può esser ottenuto da un'analisi relativamente superficiale del testo tokenizzato. Ad esempio, supponiamo di voler sviluppare una procedura per trovare tutti i nomi di persona in un certo testo. Quello che sappiamo è che i nomi di persona iniziano sempre con la lettera maiuscola, ma ciò non è abbastanza per distinguerlo dai nomi di città, di paesi, di compagnie, di animali ecc., o dalla lettera maiuscola posta alla prima parola di una frase. Alcuni modi aggiuntivi per identificare i nomi di persona possono essere:

- Presenza di un titolo: *"Dr."*, *"Mr."* ,*"Mrs."*, ecc.

- Una o più parole capitalizzate seguite da una virgola ed un numero, solitamente sotto il 100, rappresenta un modo di riferirsi alle persone negli articoli di giornale, dove il numero rappresenta la loro età. Es.: "Francesco Totti, 42".
- Una parola capitalizzata seguita da un verbo che solitamente si applica agli umani, come *"said"*, *"reported"*, *"claimed"* ecc.

Il passo successivo nell'analisi testuale è quello di **associare ad ogni token una categoria grammaticale**, anche detta **part of speech (POS)**. Esistono diverse classificazioni POS nella linguistica computazionale, e come abbiamo già accennato, la tecnica che abbiamo utilizzato per estrarre dal testo informazioni utili come la città, sono un'evoluzione di una di queste. In generale, le categorie a cui vengono assegnate ai token sono le seguenti:

- **Nomi:** *"fish", "book", "house", "pen", "language"*
- **Nomi propri:** *"John", "France", "Barack", "Python"*
- **Verbi:** *"loves", "hates", "studies", "sleeps", "is", "has"*
- **Aggettivi:** *"grumpy", "sleepy", "happy", "bashful"*
- **Avverbi**: *"slowly", "quickly", "now", "here", "there"*
- **Pronomi:** *"I", "you", "he", "she", "we", "us", "it"*
- **Preposizioni**: *"in", "on", "at", "by", "around", "with", "without"*
- **Congiunzioni:** *"and", "but", "or", "unless"*
- **Determinatori:** *"the", "a", "an", "some", "many", "few", "100"*4

Una distinzione standard fatta in fase di POS Tagging è quella che i nomi si riferiscono generalmente a persone, luoghi, cose o concetti, mentre i verbi descrivono eventi o azioni. Questa caratteristica risulta utile quando si cerca di imparare la terminologia grammaticale ma è molto semplificativa. Infatti, è possibile trovare spesso degli esempi in cui lo stesso concetto è espresso da un nome o da un verbo, da un aggettivo o da un avverbio, senza significative variazioni nella semantica della frase. Dall'altro

lato, esistono molte parole che possono prendere parte a diverse classificazioni POS, a seconda di quello che fanno in una frase. Ad esempio:

1. Rome *fell swiftly*.
2. The *fall* of Rome was *swift*.
3. The enemy *completely destroyed* the city.
4. The enemy's *destruction* of the city was *complete*.
5. John likes to *fish* on the river bank.
6. John caught a *fish*.

Inoltre, alcuni tipi di verbo non corrispondono a nessuna azione particolare ma hanno una funzione puramente grammaticale: tra questi figurano i verbi ausiliari come *"did"*, *"shall"* e così via. Quindi, in sostanza, molto spesso possiamo assegnare una categoria POS ad una parola a seconda della sua funzione nel contesto piuttosto che in funzione a come si relaziona alla realtà o agli eventi nel mondo.
Osservando le frasi presentate prima, si può notare la presenza di pattern ricorrenti, come:

- *Determinatore Nome*
- *Preposizione Determinatore Nome*

Inoltre, è possibile notare come alcuni tipi di frase possono ripresentarsi in contesti simili, ad esempio:

- *John sat on the chair*
- *The cat sat on the bed*

In questo caso, si ha che un nome proprio, o una sequenza *"Determinatore - Nome Proprio"* si presentano prima di un verbo. Alcune possibilità come questa possono essere catturate ed aggregate indicando, ad esempio *(John | The cat) sat*. Inoltre, è possibile aggregare utilizzando tale notazione anche più combinazioni di frasi insieme, come nel seguente caso:

(((the | a) (cat | dog)) (John | Jack | Susan)) (barked | slept)

In questo modo si può effettuare il match con ogni frase che finisce con il verbo *"barked"* o *"slept"*, sia che esso sia preceduto dal determinatore *"a"* o *"the"* seguito dal nome *"cat"* o *"dog"*, sia che esso sia preceduto dal nome proprio *"John"*, *"Jack"* o *"Susan"*.

I pattern che hanno distribuzioni simili, ovvero che possono presentarsi in contesti simili, sono identificati in maniera standard dalle categorie frasali come *Noun Phrase* o *Verb Phrase*.
Un modo comune di rappresentare informazioni riguardo una struttura di questo tipo, che prende il nome di **struttura costituente**, è per mezzo di **regole di produzione** nella forma $X \implies A, B, C$. Utilizzando regole in questa forma, frasi grammaticali possono essere scomposte in frasi costituenti consistenti in diverse combinazioni di POS:

- Sentence $\implies$ Noun Phrase, Verb Phrase
- Noun Phrase $\implies$ Determinatore, Nome (Es.: *The dog*)
- Noun Phrase $\implies$ Nome Proprio (Es.: *Jack*)
- Noun Phrase $\implies$ Noun Phrase, Congiunzione, Noun Phrase (Es.: *Jack and Jill, the owl and the pussycat*)
- Verb Phrase $\implies$ Verbo, Noun Phrase (Ex.: *Saw the rabbit*)
- Verb Phrase $\implies$ Verbo, Preposizione, Noun Phrase (Ex.: *Went up the hill, sat on the mat*)

Finora abbiamo visto pertanto che un **POS Tagger** è in grado di assegnare ad ogni token uno specifico tag, per poi creare delle regole sulla quale effettuare varie analisi. In relazione ai tag che il POS Tagger è in grado di assegnare ad ogni token, è interessante vedere come nel tempo sono stati realizzati diversi insiemi di tag assegnabili, che prendono il nome di **tagset**.

Il tagset probabilmente più popolare è quello del **Penn Treebank Project**. Vale la pena pertanto illustrare i tag inclusi in questo tagset, con la loro descrizione ufficiale [27]

Tabella 3: Penn Treebank Tagset

Tag	**Descrizione**	**Tag**	**Descrizione**
CC	Coordinating conjunction	RB	Adverb
CD	Cardinal number	RBR	Adverb, comparative
DT	Determiner	RBS	Adverb, superlative
EX	Existential there	RP	Particle
FW	Foreign word	SYM	Symbol
IN	Preposition or subordinating conjunction	TO	to
		UH	Interjection
JJ	Adjective	VB	Verb, base form
JJR	Adjective, comparative	VBD	Verb, past tense
JJS	Adjective, superlative	VBG	Verb, gerund or present participle
LS	List item marker		
MD	Modal	VBN	Verb, past participle
NN	Noun, singular or mass	VBP	Verb, non 3rd person singular present
NNS	Noun, plural		
NNP	Proper noun, singular	VBZ	Verb, 3rd person singular present
NNPS	Proper noun, plural		
PDT	Predeterminer	WDT	Wh-determiner
POS	Possessive ending	WP	Wh-pronoun
PRP	Personal pronoun	WP$	Possessive wh-pronoun
PRP$	Possessive pronoun	WRB	Wh-adverb

3.3.2 Estrazione di informazioni sensibili dal testo

Nella sezione precedente abbiamo illustrato i metodi per assegnare ad ogni token, quindi ad ogni parola o simbolo, un tag che ne identifica una categoria, come attributo,

avverbio, simbolo di punteggiatura ecc. Abbiamo anche mostrato in che modo è possibile sfruttare tali token taggati per desumere alcune semplici caratteristiche del testo, ma siamo ancora ben lontani dal poter estrarre da esso informazioni su città, nomi di persone pubbliche o celebrità e così via.

In questa sezione mostreremo brevemente come è possibile sfruttare i POS Tag per estrarre informazioni dal testo, ancora però in via generale, mentre nella prossima sezione mostreremo il funzionamento dello Stanford NLP NER Tagger, che abbiamo utilizzato con i nostri tweet. [28]

Un generico sistema di estrazione di informazioni è così strutturato:

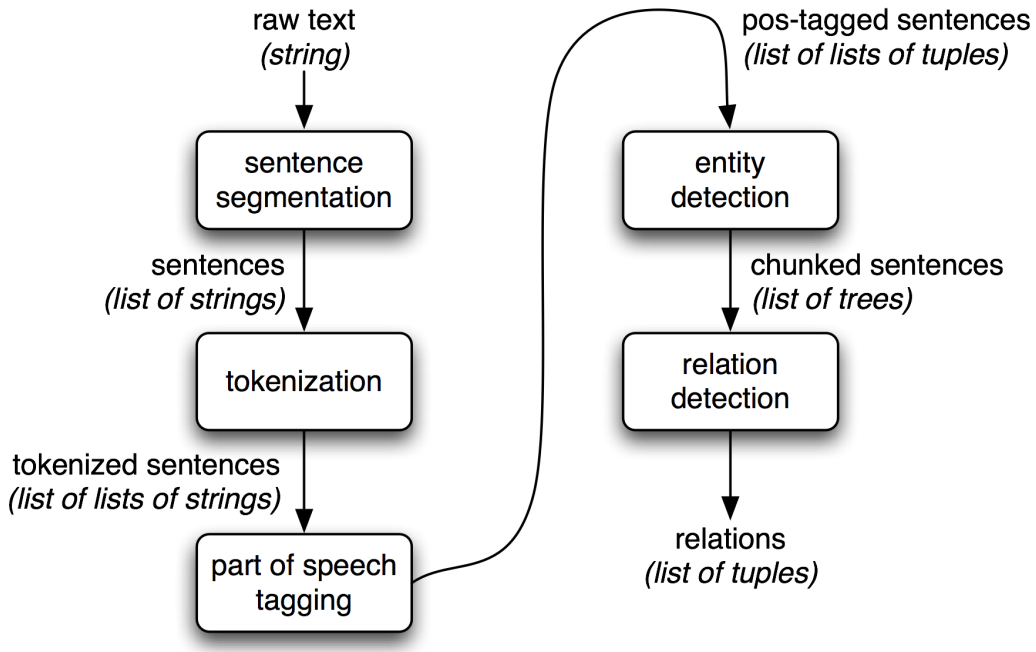

Figura 3.1: Esempio di architettura di estrazione di informazioni

Come è mostrato dal flow chart in Figura 3.1, in primo luogo è eseguita un'operazione di **segmentazione del testo**, che non fa altro che suddividere un documento in frasi. Questa fase è solitamente ignorata nel nostro caso, in quanto essendo i tweet da massimo 280 caratteri vengono considerati come singole frasi. Successivamente ogni

frase è suddivisa in parole tramite un **tokenizer**. Successivamente ad ogni parola è associato un **POS tag**, che risulta fondamentale per la fase successiva, ovvero la **entity detection** (Nella quale rientra lo Stanford NLP NER Tagger che abbiamo più volte nominato). In questo step si cerca per potenziali entità interessanti in ogni frase. Infine, c'è la fase di **relation detection** che cerca possibili relazioni tra le diverse entità nel testo.

Facciamo adesso un accenno ad una delle più utilizzate tecniche di **entity detection**, per comprendere il modo in cui le tecniche di questo tipo operano. La tecnica a cui facciamo riferimento è il **chunking** che effettua segmentazione e etichettatura di sequenze di token, come illustrato nell'immagine qui di seguito:

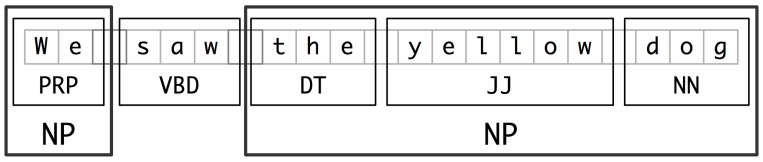

Figura 3.2: Esempio di segmentazione effettuata dal chunking

Nella Figura 3.2 si possono notare sia i quadrati interni, che rappresentano la tokenizzazione e il POS Tag, sia i quadrati esterni, che rappresentano il chunking ad alto livello. Ogni quadrato esterno è un **chunk**. Come la tokenizzazione omette gli spazi, il chunking solitamente seleziona un subset dei token. Inoltre, sempre analogamente alla tokenizzazione, i chunk non si sovrappongono nel testo originale.

Uno dei chunking che è possibile effettuare è quello delle frasi nominative (Noun phrases), in cui si cercano di associare i chunk ad una frase nominativa appunto. Ad esempio, consideriamo la seguente frase, al quale è stato applicato il **Noun Phrase Chunking** a seguito del POS Tagging:

[The/DT market/NN] for/IN [system-management/NN software/NN] for/IN [Digital/NNP] ['s/POS hardware/NN] is/VBZ fragmented/JJ enough/RB that/IN [a/DT giant/NN] such/JJ as/IN [Computer/NNP Associates/NNPS] should/MD do/VB well/RB there/RB ./.5

Nell'esempio appena introdotto, il chunking è evidenziato dalla presenza delle parentesi quadre. Come possiamo vedere, i **Noun-Phrase Chunk** (NP-Chunk), sono spesso parti più piccole di una frase nominativa. Ad esempio, "*The market for system-management software for Digital's hardware*" è una frase nominativa, ma l'NP-Chunking effettua una suddivisione più granulare, selezionando come NP-Chunk la componente *"The market"*. Una delle motivazioni per tale differenza è che gli NP-Chunk sono realizzati in modo tale da non contenere altri NP-Chunk. Conseguentemente, frasi preposizionali o clausole subordinate che modificano un nominale non saranno incluse nel corrispondente NP-Chunk, dato che molto probabilmente conterranno altre frasi nominali.

Per formare un NP-Chunk bisogna definire una **Chunk Grammar**, che contiene un insieme di regole che indicano in che modo le frasi devono essere suddivise in chunk. Ad esempio, una regola può indicare che un chunk può essere realizzato prendendo un determinatore `DT`, seguito da un numero indefinito di aggettivi `JJ` a loro volta seguiti da un nome `NN`. Applicando queste regole a dei dati forniti in input, si ottiene come risultato un **albero**. Un esempio è mostrato qui di seguito:

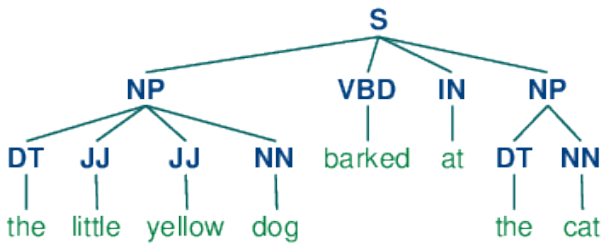

Figura 3.3: Esempio di rappresentazione di chunking

Esistono tante altre tecniche alternative per effettuare il chunking sulle quali però non ci soffermeremo.

3.3.3 Stanford NLP NER Tagger

Lo **Stanford Name Entity Recognition Tagger** è un'implementazione Java di un **Entity Recognizer**. Tale strumento permette di etichettare sequenze di parole, associando loro nomi di persona o di aziende, nomi di geni o proteine ecc. Inoltre, lo strumento è dotato non solo della possibilità di fare feature extraction, ma è realizzato

in modo tale da permettere la definizione di feature extractor arbitrari. [7]

3.3.3.1 Conditional Random Field

Lo Stanford NER è anche noto come **Conditional Random Field** (CRF), un popolare modello probabilistico per la predizione strutturata. Per predizione strutturata si intende l'insieme di metodi che combinano la classificazione e la modellazione grafica, mettendo insieme l'abilità dei modelli grafici di rappresentare in maniera compatta dati multivariabili con e la capacità dei metodi di classificazione di effettuare predizioni utilizzando enormi set di feature di input.

Fondamento di molte applicazioni è l'abilità di predire variabili multiple, le quali dipendono l'una dall'altra. Tali applicazioni possono essere molto diverse tra loro, in quanto spaziano dalla classificazione delle immagini alla previsione del risultato finale di un gioco, dalla segmentazione dei geni in un frammento di DNA fino all'estrazione della sintassi da un testo in linguaggio naturale. In tali applicazioni, si vuole predire un vettore $y = y_0, ..., y_T$ di variabili random dato un vettore di feature x. Un esempio relativamente semplice relativo all'NLP è il POS Tagging, che abbiamo appena visto nella sezione precedente. Infatti, in quel caso ogni variabile y_s è il POS Tag della parola nella posizione s, e l'input x è suddiviso nei vettori di feature $\{x_0, ..., x_t\}$. Ogni componente x_s contiene diverse informazioni riguardanti la parola nella posizione s, come la sua identità, feature ortografiche quali prefissi e suffissi, appartenenza ad un lessico specifico di un qualche particolare dominio e informazioni in database semantici come WordNet.

Un primo approccio a questo problema di predizione, specialmente se il nostro obiettivo è la massimizzazione del numero di etichette y_s correttamente classificate, è quello di addestrare un classificatore indipendente per ogni posizione, che faccia il mapping tra x e y_s per ogni s. La difficoltà principale ad emergere è che le variabili di output hanno dipendenze complesse. Ad esempio, parole vicine in un documento di testo o regioni vicine all'interno di una stessa immagine tendono ad avere etichette simili.

Un modo naturale per rappresentare la relazione di dipendenza tra le variabili di output è fornito dai modelli grafici. I **modelli grafici**, che includono diverse famiglie, dalle reti Bayesiane alle reti neurali, rappresentano una distribuzione complessa su diverse variabili come un prodotto di fattori locali su piccoli subset di variabili. In questo modo risulta possibile descrivere come una certa fattorizzazione della densità di probabilità corrisponde ad un particolare set di relazioni di indipendenza condizionale soddisfatte dalla distribuzione. Questa corrispondenza rende la modellazione molto più semplice, in quanto spesso la nostra conoscenza del dominio presenta delle ragionevoli assunzioni di indipendenza condizionale, che quindi determineranno la nostra scelta dei fattori.

La maggior parte degli sforzi effettuati con i modelli grafici, specialmente nel NLP, si sono concentrati su **modelli generativi** che provano esplicitamente a modellare la distribuzione di probabilità congiunta $p(y, x)$ tra input e output. Nonostante tale approccio presenti numerosi vantaggi, ha anche importanti limitazioni. Non solo la dimensionalità di x può essere molto grande, ma le feature possono avere dipendenze complesse, pertanto costruire una distribuzione di probabilità su esse può risultare complesso. Modellare le dipendenze tra gli input può portare alla creazione di modelli intrattabili, ma di contro ignorare tali relazioni può portare a modelli dalle pessime performance.

Una soluzione a questo problema può essere la modellazione della distribuzione di probabilità condizionata $p(y|x)$, che è tutto ciò che serve per la classificazione. Questo è esattamente quello che prende il nome di **Conditional Random Field**. Un CRF è essenzialmente un modo per combinare i vantaggi della classificazione e della modellazione grafica, mettendo insieme la capacità di modellare in maniera compatta dati multivariabile e l'abilità di sfruttare un elevatissimo numero di feature di input per la predizione. Il vantaggio principale è che le dipendenze che coinvolgono solo le variabili di x non hanno alcun ruolo nel modello condizionale, in maniera tale che tale modello condizionale può avere una struttura molto più semplice rispetto ad un modello congiunto.

Introduciamo adesso alcune strutture di modelli grafici che combinati con la predizione permettono di realizzare il tagging, ovvero il matching tra una parola e la categoria a cui appartiene.

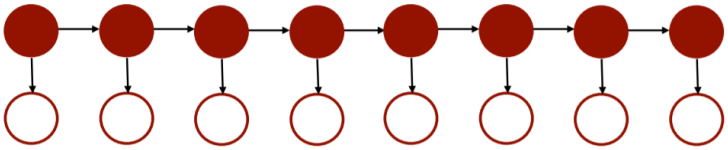

Figura 3.4: Hidden Markov Models (HMM)

Il **Modello HMM (Hidden Markov Model)** è un modello **generativo**, che si pone l'obiettivo di trovare i parametri per **massimizzare $p(X, Y)$**, pertanto si basa sulla **probabilità congiunta**. Esso si basa sull'assunzione che tutte le **feature sono indipendenti**. Inoltre, durante l'etichettatura dell'input x_i, vengono considerate anche le osservazioni future, oltre che quelle passate (**Forward-Backward observation**).

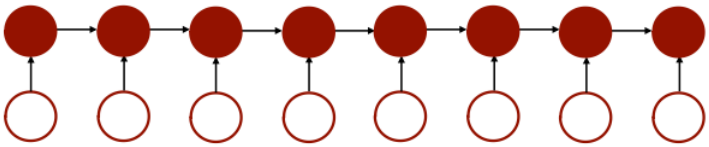

Figura 3.5: MaxEnt Markov Models (MEMMs)

Il **Modello MEMMs** è un modello **discriminativo**, che si pone l'obiettivo di trovare i parametri per **massimizzare $p(Y|X)$**, basandosi quindi sulla **probabilità condizionata**. Esso **elimina l'assunzione dell'indipendenza delle feature**. Inoltre, nel calcolo dell'etichetta per l'input x_i, non tiene in considerazione le osservazioni future, ma solo quelle passate **(No Forward-Backward observation)**.

Il **Modello CRF** (Conditional Random Fields) è un modello **discriminativo**, che prende il meglio dalle due precedenti soluzioni, senza fare l'assunzione dell'indipenden-

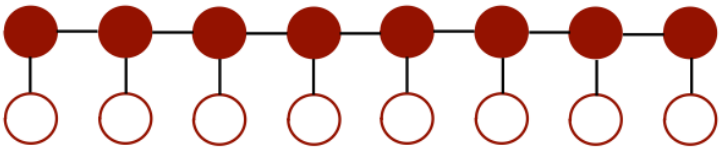

Figura 3.6: Conditional Random Field (CRF)

za condizionale delle feature, implementando un approccio discriminativo e facendo Forward-Backward observation.

Modello	**Velocità**	**Tipo**	**Normalizzazione**
HMM	Alta	Generativo	Locale
MEMM	Media	Discriminativo	Locale
CRF	Bassa	Discriminativo	Globale

Dal punto di vista implementativo, è interessante analizzare in che modo lo Stanford NLP NER Tagger estende il concetto di CRF per l'etichettatura. Come abbiamo accennato prima, esso prende in considerazione anche gli input successivi a quello che si sta considerando, ovvero analizza:

- La parola corrente
- La parola precedente
- La parola successiva o tutte le parole in una determinata finestra

Ogni parola apparirà in uno o più contesti, e il NER calcola una **distribuzione della parola sui diversi contesti**. Successivamente, le parole vengono clusterizzate in base a quanto sono simili le loro distribuzioni (In un certo contesto esistono parole che appaiono spesso insieme) e considera l'identificativo di ogni cluster come feature per la classificazione. Per la realizzazione del Tagger sono stati creati più di 200 cluster con più di 100 milioni di parole, solo per la lingua inglese.

Tornando invece al lato pratico Tagger dell'università di Stanford, è interessante notare come, a differenza del POS Tagger standard, esso non venga distribuito con uno

specifico tagset. Pertanto abbiamo deciso di estrarre personalmente tutti i tag che è in grado di assegnare tale strumento, e li riportiamo qui di seguito:

Tag	**Descrizione**
CRIMINAL_CHARGE	Evento criminale
ORGANIZATION	Organizzazione
SET	Indicazione quantitativa
O	Punteggiatura / Irrilevanza
COUNTRY	Paese
MONEY	Denaro
CAUSE_OF_DEATH	Causa di morte
DURATION	Durata
RELIGION	Religione
ORDINAL	Attributo ordinale
TITLE	Titolo
TIME	Informazione temporale
PERCENT	Percentuale
STATE_OR_PROVINCE	Stato o provincia
LOCATION	Posizione
IDEOLOGY	Ideologia
NATIONALITY	Nazionalità
CITY	Città
PERSON	Persona
NUMBER	Numero
URL	URL
DATE	Data
MISC	Varie

Quello che salta all'occhio, facendo il paragone con il Tagset Penn Treebank introdotto con il POS Tagging, è che le etichette assegnate dal NER Tagger di Stanford sono di livello molto più elevato, permettendo di individuare nel testo la presenza di informa-

zioni potenzialmente interessanti quali eventi criminali, organizzazioni ecc.

Buona parte dei tag appena presentati inglobano un contenuto informativo importante, che se incluso nel modello di classificazione basato su albero di decisione, che noi abbiamo chiamato *Classificatore C1*. In particolare, dopo una serie di tentativi ed analisi sulla loro rilevanza, abbiamo scelto di utilizzare i seguenti tag, indicando in un attributo booleano la presenza o meno di tale informazione nel testo del tweet:

Tag
CRIMINAL_CHARGE
ORGANIZATION
SET
COUNTRY
MONEY
CAUSE_OF_DEATH
DURATION
RELIGION
TIME
STATE_OR_PROVINCE
LOCATION
IDEOLOGY
CITY
PERSON
NUMBER
DATE

Tali informazioni sono state ovviamente aggiunte ai metadati prelevati da Twitter, che abbiamo indicato nelle sezioni precedenti.

Con questa sezione termina la disamina delle tecniche, degli algoritmi e delle elaborazioni effettuate per il *Classificatore C1 - Metadati e Flag*. Nelle prossime sezioni intro-

durremo le idee alla base del *Classificatore C2 - Doc2Vec* per poi passare ai dettagli implementativi di entrambi.

3.4 Rappresentazione numerica

La rappresentazione numerica dei documenti di testo è tra le pratiche più complesse e stimolanti del Machine Learning. Una rappresentazione di questo tipo può essere utilizzata per scopi variegati, dal recupero di documenti alla ricerca web, dal filtraggio anti-spam al topic modeling. Non esistono però tante tecniche valide per questo scopo. Molte operazioni fanno uso del ben noto ma semplicistico metodo **Bag of Word**, ma i risultati sono pressocchè mediocri, in quanto il BOW non considera alcune sottigliezze di una possibile buona rappresentazione, come l'ordinamento delle parole.

Un'altra tecnica ben nota è la **LDA - Latent Dirichlet Annotation**, molto utilizzata per il topic modeling ma molto complicata da regolare, ed i risultati sono difficili da valutare.

La tecnica **Doc2Vec**, introdotta nel 2014 e sulla quale ci soffermeremo in questa sezione, è una tecnica semplice da utilizzare, che garantisce buoni risultati e si basa sulla precedente tecnica **Word2Vec**, sulla quale faremo una piccola digressione nelle prossime righe.

3.4.1 Rappresentazione Word2Vec

In questa sezione introduciamo prima il concetto di rappresentazione di vettori di parole (word vector) distribuita, per poi passare all'analisi di una particolare tecnica per ottenere una tale rappresentazione, ovvero la tecnica **Word2Vec**.
Un approccio ben noto è quello mostrato nella seguente figura:

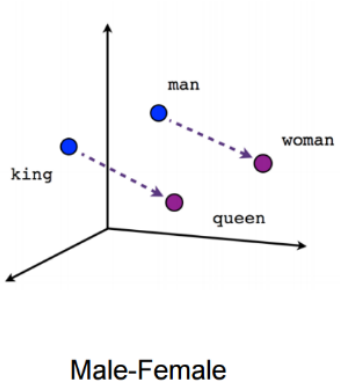

Figura 3.7: Esempio di rappresentazione vettoriale di parole

Ogni parola viene mappata ad un vettore unico, rappresentato da una colonna di una matrice W. La colonna è indicizzata dalla posizione della parola nel vocabolario. La concatenazione o la somma dei vettori è poi utilizzata come feature per la **predizione della prossima parola in una frase**.

Formalmente, data una sequenza di parole di training $w_1, w_2, ..., w_T$, l'obiettivo del modello è quello di massimizzare la probabilità:

$$\frac{1}{T}\sum_{t=k}^{T-k} \log p(w_t|w_{t-k}, ..., w_{t+k})$$

L'operazione di predizione è poi tipicamente svolta da un classificatore multiclasse, ad esempio un **softmax**. Pertanto si ha:

$$p(w_t|w_{t-k}, ..., w_{t+k}) = \frac{e^{y_{w_t}}}{\sum_i e^{y_i}}$$

Ogni y_i rappresenta la probabilità logaritmica, non normalizzata, per ogni parola di output i, calcolata come:

$$y = b + Uh(w_{t-k}, ..., w_{t+k}; W)$$

Dove U, b sono parametri softmax, h è costruita da una concatenazione o media di word vector estratti da W.

I word vector calcolati tramite reti neurali (softmax, ad esempio) sono solitamente addestrati utilizzando la **discesa del gradiente** ottenuto tramite **backpropagation**. Questa tipologia di modelli è conosciuta comunemente come **modelli di linguaggio neurali**.

Dopo che il training converge, le parole con significato simile sono mappate in una posizione vicina nello spazio dei vettori. Ad esempio, le parole *"powerful"* e *"strong"* sono vicine nello spazio, mentre *"powerful"* e *"Paris"* sono più distanti. Inoltre, la differenza tra word vectors ha un forte significato. Ad esempio, i word vectors possono essere utilizzati per rispondere a delle domande di analogia, utilizzando una semplice algebra vettoriale: "$King$" − "man" + "$woman$" = "$Queen$". [**?**]

Passiamo adesso a parlare della tecnica **Word2Vec**, che sfrutta molti dei concetti introdotti finora. Word2Vec è un insieme di modelli basati su **reti neurali a due livelli**, addestrati per ricostruire contesti linguistici delle parole. Infatti, Word2Vec prende in input un testo e produce uno **spazio di vettori**, tipicamente di diverse centinaia di dimensioni, con ogni parola del testo assegnata ad un determinato vettore nello spazio. I word vectors generati da Word2Vec hanno le stesse caratteristiche di quelli esposti precedentemente, ovvero le parole che condividono lo stesso contesto nel testo sono localizzate in zone vicine dello spazio.

Per creare la rappresentazione distribuita delle parole, Word2Vec può utilizzare due algoritmi:

- **Continuous Bag of Words (CBOW)**: Il modello predice la parola corrente da una finestra di parole che la circondano. L'ordine delle parole non influenza la predizione (Questa è un'assunzione tipica delle tecniche BOW).
- **Skip-gram:** Il modello assegna alle parole dal contesto simile pesi maggiori rispetto a quelle meno simili.

In generale, Word2Vec sfrutta una tecnica particolare, usata spesso nel Machine Learning. Esso infatti, addestra una semplice rete neurale con un singolo hidden layer per effettuare un certo task, nonostante essa non verrà utilizzata realmente per quello scopo. Infatti, l'obiettivo è quello di apprendere i pesi dell'hidden layer ed utilizzarli come word-vectors.

Tale processo assegnato alla rete neurale prende il nome di **fake task**. In particolare, data una parola all'interno di una frase (Alla quale ci riferiremo come input word), seleziona casualmente una parola nel suo vicinato. La rete fornirà in input la probabilità, per ogni parola del vocabolario, di essere la parola vicina che scegliamo. Tipicamente si utilizza una finestra di 5 parole, ovvero si tengono in considerazione le 5 parole precedenti e le 5 parole seguenti quella considerata.

Le probabilità in output rappresenteranno la possibilità di trovare ogni parola del vocabolario nel vicinato dell'input word. Ad esempio, fornendo alla rete la parola *"Soviet"*, le probabilità di output saranno molto alte per le parole *"Union"* e *"Russia"*, e molto basse per le parole *"watermelon"* e *"kangaroo"*.

Prima di passare alla codifica dei paragrafi e alla tecnica Doc2Vec, introduciamo un semplice esempio per illustrare il funzionamento della tecnica Word2Vec. Consideriamo una semplice frase con una finestra di 2 parole, quindi tenendo in considerazione le due parole precedenti e quelle successive all'input word. Consideriamo a frase *"The quick brown fox jumps over the lazy dog"*. Nell'immagine seguente, è indicato in blu l'input word.

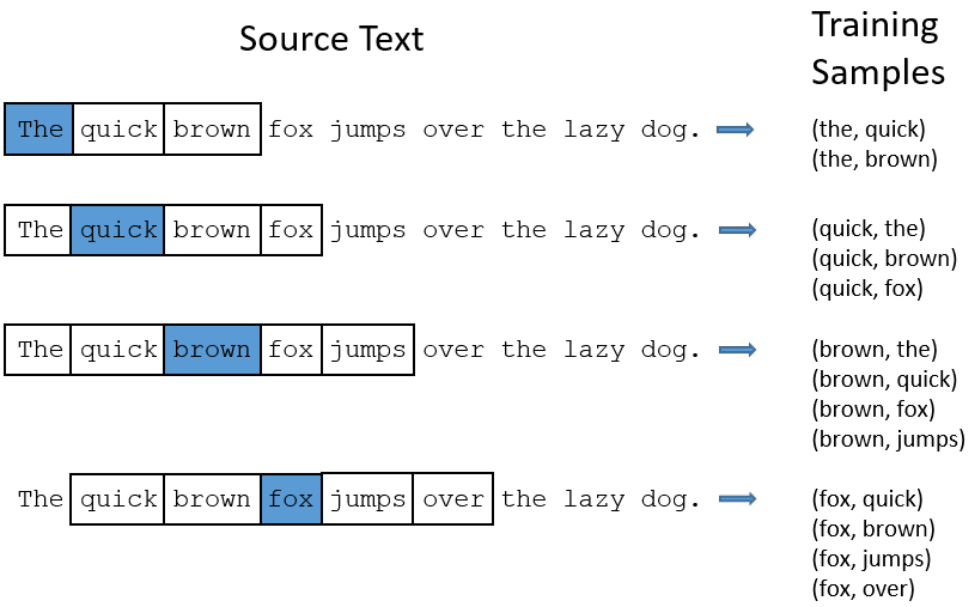

Figura 3.8: Esempio di Word2Vec

In questo esempio, si considerano tutte le coppie di parole nel vicinato, come mostrato nella Figura 3.8. La rete apprenderà le statistiche dal numero di volte che ogni coppia appare nel contesto. Ad esempio, considerando l'esempio di prima sull'Unione Sovietica, la rete probabilmente avrà molte più coppie di training *("Soviet", "Union")* che *("Soviet", "Sasquatch")*. Quando la fase di training è finita, fornendo in input la parola *"Soviet"*, la rete fornirà una probabilità molto più alta per *"Union"* o *"Russia"* che per *"Sasquatch"*.

Dato che non è possibile fornire in input ad una rete neurale una stringa testuale, la prima cosa da fare è costruire un vocabolario di parole prese dai documenti di training. Tali parole saranno rappresentate tramite un vettore one-hot, ovvero, si avranno tanti vettori quante sono le parole, ognuno con **il numero di dimensioni pari alla dimensione del vocabolario** (Se si hanno 10000 parole, si avranno 10000 vettori 10000-dimensionali). Tali vettori sono rappresentati ponendo un solo elemento a 1, e tutti gli altri a 0.

L'output della rete sarà un **vettore da 10000 componenti** contenente, per ogni parola del vocabolario, la probabilità che una parola selezionata randomicamente nel suo vicinato sia esattamente quella parola. L'architettura della rete neurale è così composta:

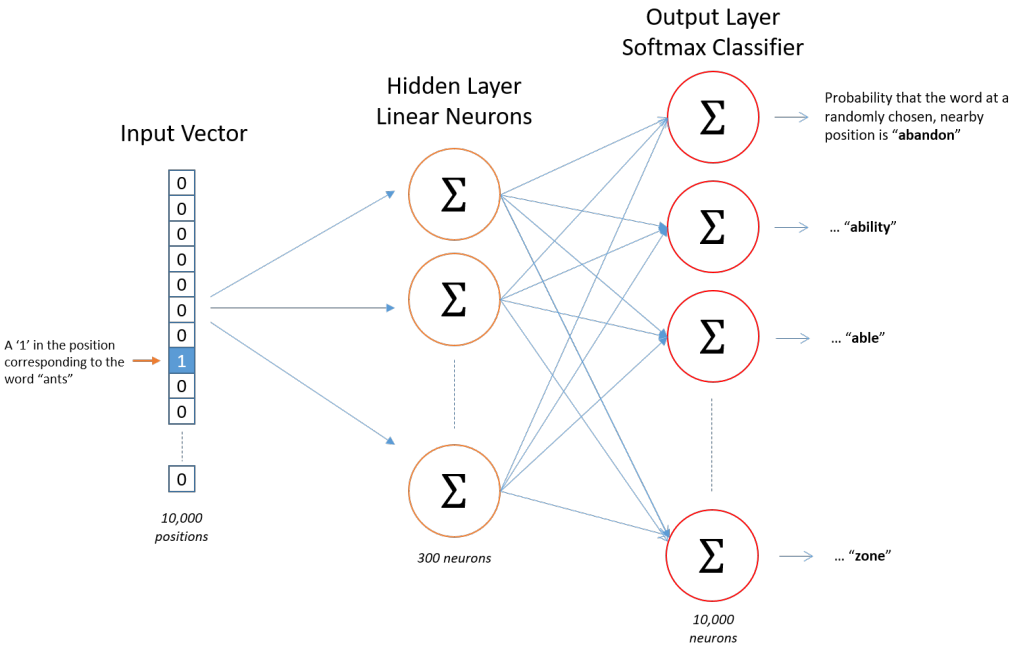

Figura 3.9: Architettura di una rete neurale Word2Vec

Quindi, durante la fase di training, sia i vettori di input che il vettore di output sono un **one-hot vector**. In fase di testing invece, l'output vector sarà la distribuzione di probabilità delle parole.

Consideriamo un livello di hidden da 300 nodi (facendo l'assunzione che le feature peculiari presenti nei vettori, che devono essere individuate, sono 300). L'**hidden layer** sarà pertanto rappresentato da una **matrice dei pesi con 10000 righe** e 300 colonne, una per ogni neurone. Analizzando le righe di questa matrice, si avrà che esse rappresenteranno esattamente i word vectors che vogliamo ottenere, quindi, l'obiettivo finale di questo processo è soltanto quello di apprendere e memorizzare la matrice dei pesi del livello hidden. L'output layer sarà semplicemente ignorato, una volta raggiunta la convergenza.

Essendo che l'input vector è un one-hot vector, quando viene moltiplicato per la matrice di pesi hidden, semplicemente andrà a selezionare la riga della matrice corrispondente alla posizione che nel vettore in input ha valore 1. Ciò significa che l'hidden layer del modello opera semplicemente come una **lookup table**. L'output dell'hidden layer sarà esattamente il word vector corrispondente alla parola immessa in input.

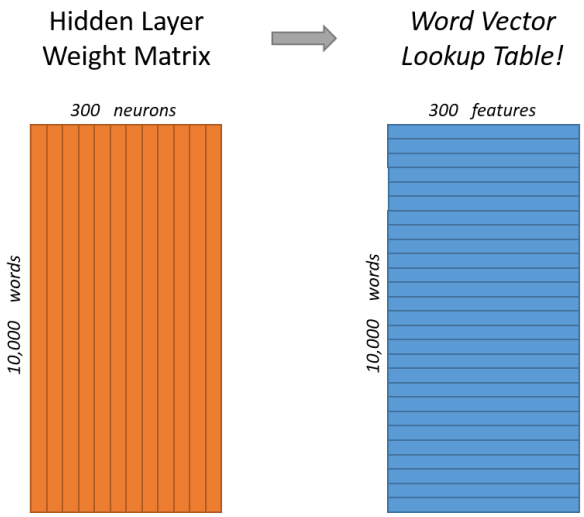

Figura 3.10: Visualizzazione dell'hidden layer come una lookup table

L'output layer infine, riceverà il word vector corrispondente ad una certa parola. Essendo un classificatore softmax, esso risponderà con ogni neurone che fornisce in output un valore tra 0 e 1, la cui somma di tutti i valori farà 1. Ovvero, ogni neurone di output ha un vettore di pesi che moltiplica con il word vector fornito dall'hidden layer, e poi applica al risultato una funzione esponenziale. [29]

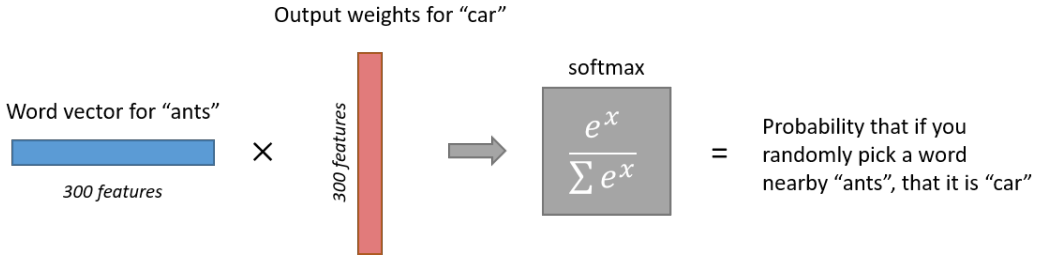

Figura 3.11: Esempio di funzionamento di output layer

3.4.2 Rappresentazione Doc2Vec

Introduciamo in questa sezione un problema simile a quello precedente, che non preveder però la rappresentazione tramite vettori numerici di parole, bensì di interi documenti, paragrafi o, nel nostro caso, tweet.

Nel **Paragraph Vector**, la rappresentazione vettoriale è addestrata in modo da esser utile per la predizione di parole in un paragrafo. Più precisamente, si concatena il paragraph vector (Vettore che identifica il paragrafo) con i diversi word vector (Vettori che identificano le parole) in modo da predire la parola seguente dato un certo contesto. Entrambi i tipi di vettori sono addestrati utilizzando la discesa del gradiente e la back-propagation. Bisogna notare però che mentre i paragraph vector sono unici per ogni paragrafo, i word vector sono condivisi. In fase di predizione, i paragraph vector sono inferiti tenendo fissi i word vectors e addestrando un nuovo paragraph vector fino a raggiungere la convergenza.

L'approccio che introdurremo per l'apprendimento di paragraph vector trae forte ispirazione dai metodi di apprendimento dei word vector. Come abbiamo infatti accennato prima, i word vector forniscono un contributo nel task di predizione della prossima parola nella frase. Perciò, nonostante i vettori siano inizializzati casualmente, possono contenere una semantica come risultato indiretto della predizione. Utilizzeremo questa idea similmente per i paragraph vector, in quanto essi saranno utilizzati per contribuire al task di predizione della prossima parola, considerando però diversi contesti estratti dal paragrafo.

Ogni paragrafo è mappato in un unico vettore, rappresentato da una colonna nella matrice D e ogni parola è mappata anch'essa ad un vettore unico, rappresentato da una colonna della matrice W. Il paragraph ed i word vector sono pesati o concatenati per predire la prossima parola in un certo contesto.
La componente relativa al paragrafo può essere pensata come se fosse un'altra parola. Essa opera come una memoria che ricorda cosa manca dal contesto corrente, o dall'argomento del paragrafo. Per questo motivo, questo modello è spesso noto come

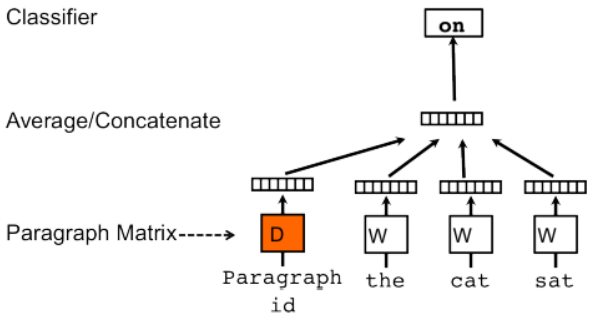

Figura 3.12: Esempio di Doc2Vec

Distributed Memory Model of Paragraph Vectors (PV-DM). I contesti sono di lunghezza fissa e campionati da una finestra scorrevole lungo il paragrafo. Il paragraph vector è condiviso attraverso tutti i contesti generati dallo stesso paragrafo, ma non tra i diversi paragrafi. La matrice di word vector W invece, è condivisa tra i paragrafi.

Sia i paragraph vector che i word vector sono addestrati utilizzando la tecnica della discesa del gradiente, il quale è ottenuto tramite back-propagation. Ad ogni passo, un contesto di lunghezza fissa viene prelevato da un paragrafo casuale, se ne calcola l'errore dalla rete in Figura 3.12 e si usa il gradiente per aggiornare i parametri del modello. In fase di predizione, il calcolo del paragraph vector per un nuovo paragrafo è ottenuto anch'esso tramite metodo di discesa del gradiente. In questo passo il resto dei parametri del modello, i word vectors W e i pesi sono fissati.

Supponiamo che vi siano N paragrafi, M parole nel vocabolario e si vuole apprendere dei paragraph vector affinchè ogni paragrafo sia mappato in p dimensioni e ogni parola sia mappata in q dimensioni, quindi il modello ha un totale di $N \times p + M \times q$ parametri. Nonostante il numero di parametri può essere molto grande in caso di elevati valori di N, gli aggiornamenti durante la fase di addestramento sono tipicamente sparsi e pertanto efficienti.

Dopo l'addestramento, i paragraph vector possono essere utilizzati come feature del paragrafo. Tali feature possono essere utilizzate direttamente con tecniche di Machine

Learning, quali logistic regression (Come abbiamo fatto per la realizzazione del nostro *Classificatore C2*) o K-Means clustering.

Quindi, ricapitolando, l'algoritmo ha due passi fondamentali:

- **Fase di addestramento:** Training delle word vectors W, dei pesi U, b e dei paragraph vector D sui paragrafi già visti.
- **Fase di inferenza**: Per il calcolo dei paragraph vectors D per nuovi paragrafi (mai visti prima) aggiungendo più colonne in D e utilizzando su esso la discesa del gradiente, mantenendo W, U e b fissi. Si utilizza poi D per effettuare la predizione riguardo alcune particolari etichette, utilizzando un classificatore standard.

Un vantaggio fondamentale dei paragraph vector è che sono **appresi da dati non etichettati** e pertanto possono funzionare bene per processi che non hanno abbastanza dati etichettati.

Inoltre, essi risolvono alcune debolezze tipiche dei modelli basati su bag-of-words. In primo luogo, ereditano un'importante caratteristica dei word vectors: la semantica delle parole. In questa rappresentazione, *"powerful"* è più simile a *"strong"* che a *"Paris"*, come è logico che sia. Il secondo vantaggio di questa rappresentazione è che tiene in considerazione l'ordine delle parole, quanto meno per piccoli contesti. Tale modello è inoltre anche migliore di un modello bag-of-n-grams in quanto quest'ultimo creerebbe una rappresentazione dall'elevata dimensionalità con scarse capacità di generalizzazione.

Finora abbiamo considerato la tecnica PV-DM, che prevede la concatenazione del paragraph vector con i word vector per la predizione della prossima parola in una finestra. Un altro modo per effettuare una codifica simile è quello di ignorare le parole di contesto in input, e forzare il modello a **predire parole scelte randomicamente dal paragrafo in output**. In realtà, ciò significa che ad ogni iterazione dell'algoritmo di discesa del gradiente si seleziona una finestra e dal suo interno si sceglie una parola

casualmente. Questa tecnica, mostrata in Figura 3.10, prende il nome di **Distributed Bag of Words (PV-DBOW)**.

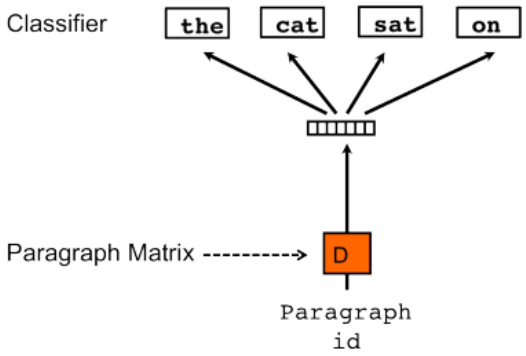

Figura 3.13: Esempio di Doc2Vec tramite PV-DBOW

Oltre ad essere concettualmente semplice, questo modello richiede la memorizzazione di un quantitativo inferiore di dati, in quanto necessita di memorizzare i pesi, mentre la tecnica PV-DM richiedeva anche lo storage dei word vectors. Molto spesso le due tecniche vengono utilizzate insieme, creando dei paragraph vector sfruttando la combinazione di due diversi vettori: uno appreso tramite PV-DM e l'altro appreso tramite PV-DBOW. Questo approccio garantisce solitamente una maggiore consistenza in molti processi ed è fortemente consigliato.

Capitolo 4

Il classificatore di tweet: Implementazione

Questo capitolo è da considerare un'integrazione del capitolo precedente ed è pertanto raccontato mostrando un evidente parallelismo nei suoi confronti, già dal punto di vista della struttura, in modo tale da fornire una visione uniforme su cosa è stato fatto e le tecniche, le librerie, gli algoritmi e gli strumenti utilizzati per implementare il progetto precedentemente esposto.

Come abbiamo già detto nelle sezioni precedenti, abbiamo creato due classificatori molto differenti nell'approccio e nell'implementazione, e pertanto suddivideremo questo capitolo dedicando ad ognuno di essi una sezione apposita.

Entrambi i classificatori sono disponibili sulla repository GitHub `https://github.com/FlavioC182/Crisis-tweets-analysis` sulla quale sono presenti sia gli script Python sia i processi Rapidminer, oltre che ovviamente ai dataset modificati.

4.1 Il classificatore C1 - Metadati e Flag NLP

Iniziamo questa sezione introducendo un breve recap sulla composizione del *Classificatore C1* e delle operazioni che si sono rese necessarie a realizzarlo.

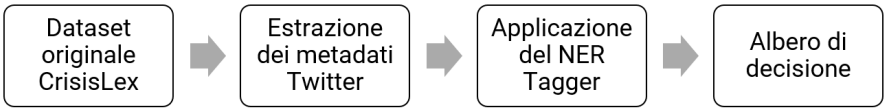

Figura 4.1: Flow Chart del Classificatore C1

Con riferimento alla Figura 4.1, ad eccezione del primo task, che ignoreremo essendo stato realizzato semplicemente effettuando il download dei dataset messi a disposizione sulla piattaforma `crisislex.com`, nelle prossime sezioni analizzeremo i tools e le tecniche utilizzate per realizzare appunto l'estrazione dei metadati, l'applicazione dello Stanford NLP NER Tagger e l'implementazione pratica dell'albero di decisione.

4.1.1 Estrazione dei metadati: Twython

Come già più volte indicato nel corso delle precedenti sezioni, per ottenere le informazioni aggiuntive su ogni tweet, delle quali necessitavamo per l'addestramento del *Classificatore C1* basato su decision tree, abbiamo fatto uso delle API di Twitter, in particolare delle **Twitter Search API**.

Le Twitter's Search API permettono di effettuare semplici interrogazioni utilizzando gli indici di tweet pubblicati dal 2006 in poi. Esse sono delle **RESTful API** che permettono l'interazione con la piattaforma Twitter. In particolare, permettono di creare, recuperare e cancellare *Tweets, Retweets e Likes.*

Essendo il nostro progetto realizzato prevalentemente in Python, abbiamo scelto di utilizzare però una libreria apposita, la quale non fa altro che da **interfaccia alle Twitter APIs**. Ci stiamo riferendo a **Twython** [10], le cui principali caratteristiche sono:

- Possibilità di effettuare query relative a:
  - Informazioni sugli utenti
  - Liste Twitter

  - Timelines
  - Messaggi diretti
  - In generale tutto ciò che può essere trovato sulla documentazione Twitter
- Possibilità di fare upload di immagini:
  - Aggiornamento di stato dell'utente tramite immagini
  - Cambio dell'avatar utente
  - Cambio dell'immagine in background dell'utente
  - Cambio del banner dell'utente
- Supporto ad OAuth2 Application
- Supporto alle Streaming API di Twitter
- Supporto a Python 3

4.1.1.1 I metodi principali

Senza entrare troppo nei dettagli, vogliamo menzionare brevemente le funzioni, ed il loro corrispettivo funzionamento, che abbiamo sfruttato maggiormente per ottenere le informazioni aggiuntive sui tweet presenti nei dataset CrisisLex.

Il costruttore della classe Twython **`Twython`** che permette di effettuare l'autenticazione, prendendo in input i vari token dell'applicazione Twitter tramite la quale inviare richieste HTTP ai loro server.

Il metodo **`Twython.show_status()`**, al quale basta indicare il Tweet ID del tweet da recuperare, e restituisce il **Tweet Object corrispondente**. I campi principali presenti nel Tweet Object sono esposti nella Sezione 3.2.

L'eccezione **`twython.exceptions.TwythonRateLimitError`**, che viene lanciata nel caso in cui si effettuino troppe richieste in una determinata finestra temporale. Nel nostro caso, essendo la nostra applicazione Twitter abilitata per effettuare al massimo

900 richieste ogni 15 minuti, abbiamo sfruttato tale eccezione per implementare un meccanismo di attesa e di conseguente riprova per recuperare i diversi tweet.

L'eccezione **`twython.exceptions.TwythonError`**, versione più generica, che abbiamo utilizzato per modellare il comportamento del programma a seguito della ricezione di risposte HTTP quali `401 - Unauthorized`, `403 - User suspended` e `404 - Not Found` che si verificavano cercando di accedere ad alcuni tweet non più esistenti o privati.

4.1.2 NLP - Named Entity Recognition

Il Named Entity Recognition Tagger, realizzato dall'università di Stanford, che abbiamo utilizzato per estrarre entità rilevanti dal testo dei tweet, è realizzato in Java.

Analogamente a quanto fatto con l'estrazione dei metadati, abbiamo scelto di utilizzare una libreria Python, tramite la quale richiamare il NER Tagger Java, senza però scrivere codice in altri linguaggi. La libreria a cui facciamo riferimento è la ben nota **Natural Language Toolkit (nltk)**. Essa rappresenta lo stato dell'arte del Natural Language Processing in Python, fornendo interfacce verso più di 50 corpora e risorse lessicali, come WordNet, contemporaneamente ad una suite di processing del testo per la classificazione, tokenizzazione, stemming, tagging, parsing ed altro. [11].

I componenti della libreria che abbiamo utilizzato per il tagging e che vale la pena menzionare sono soltanto due.

Il costruttore della classe **`CoreNLPNERTagger`**, tra i quali parametri è necessario indicare la porta sulla quale si trova in ascolto il processo d'esecuzione del NER Tagger. Essendo questo modulo soltanto un wrapper di quello che è il vero e proprio processo Java che analizza ed etichetta il testo, la classe `CoreNLPNERTagger` rappresenta una semplice interfaccia.

Il metodo **`CoreNLPNERTagger.tag()`**, che prende in input la lista di elementi da taggare, e restituisce una coppia *(Parola, Tag)* per ogni parola passata.

Per permettere al sistema di funzionare, è necessario eseguire in locale il processo di tagging, e per eseguirlo è fondamentale lanciare il seguente comando da terminale (Windows), che farà partire il processo principale del NER e lo mette in attesa di eventuali richieste di tagging, effettuabili tramite le funzioni appena introdotte:

```
java -mx4g -cp "*" edu.stanford.nlp.pipeline.StanfordCoreNLP
Server -port 9000 -timeout 15000
```

4.1.3 Rapidminer - L'albero di decisione

Passiamo finalmente adesso in rassegna quello che rappresenta il cuore di questa prima parte del capitolo, ovvero la realizzazione del classificatore basato su albero di decisione. Mostreremo vari dettagli implementativi, introducendo brevemente i moduli RapidMiner che si sono resi necessari per la realizzazione e spiegando le varie scelte di progettazione.

Prima però, introduciamo l'intero processo per poter avere una visione d'insieme e poi successivamente entriamo nei dettagli di ogni sua singola componente.

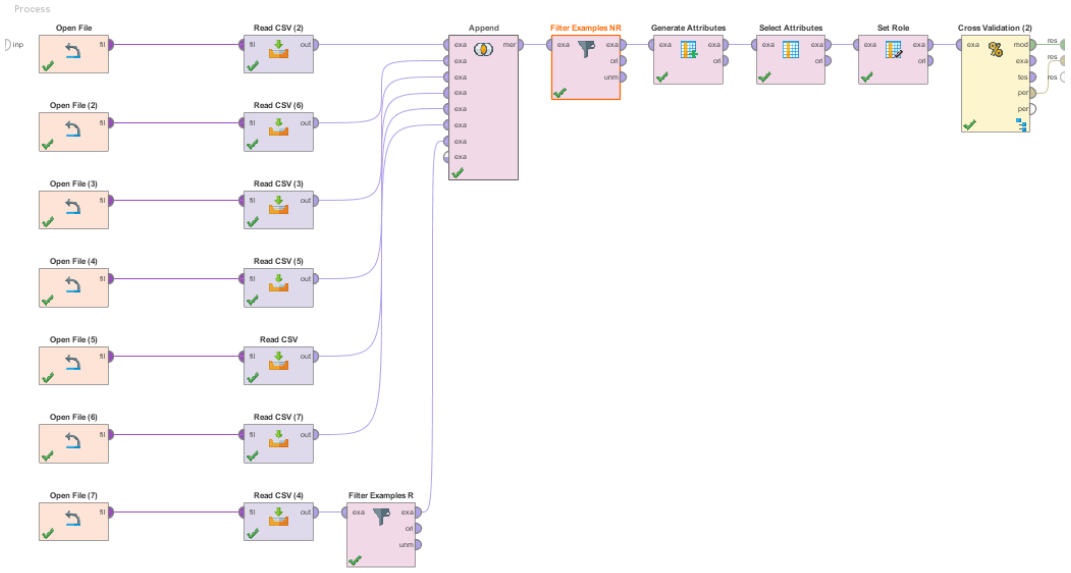

Figura 4.2: Processo RapidMiner del classificatore C1

I nodi **Open File e Read CSV** servono rispettivamente per recuperare da un link remoto il dataset e successivamente importarlo in RapidMiner. Il motivo per il quale vi sono più di uno di questi moduli è che ognuno rappresenta un dataset relativo ad un determinato evento.

Il nodo **Filter Examples** ha il compito di applicare un filtro al dataset ricevuto in input. Con riferimento all'immagine appena introdotta distinguiamo:

- **Filter Examples R**: Elimina i record con etichetta "Related and informative". Questo viene fatto per introdurre un maggiore bilanciamento nella distribuzione delle etichette in input al classificatore. Questo viene applicato soltanto ad un dataset.
- **Filter Examples NR**: Elimina i record con etichetta "Not related", che sono completamente scorrelati dall'evento e quindi non vengono utilizzati in fase di training. (Es.: "Ciao Mamma").

Il nodo **Append** è invece un semplice processo che concatena i dati passatigli in input, creando una tabella unica. Ovviamente, per poter funzionare, è necessario che ogni

dataset datogli in input abbia la stessa struttura (Stesso set di attributi).

Il nodo **Generate Attribute** permette di creare uno o più nuovi attributi, utilizzando una certa logica e sfruttando caratteristiche dei valori degli altri attributi. In questo caso infatti, lo abbiamo utilizzato per diversi scopi:

- Rinominare i valori dell'attributo dell'etichetta di classe, ovvero *"Informativeness"*, sostituendo *"Related and informative"* con *"Relevant"* e *"Related but not informative"* con *"Not Relevant"*.
- Creare un nuovo attributo per ogni tipo di "Source", ovvero per ogni tipo di applicazione dalla quale il tweet può essere generato. Tale attributo sarà un booleano, ad indicare se il tweet è stato generato da quella determinata sorgente o meno. Ad esempio, considerate le sorgenti *"Mobile App"* e *"Client Web"*, vengono generati due nuovi attributi booleani, denominati *"ClientMobile"* e *"ClientSource"*, ad indicare se il tweet è stato generato dalla corrispondente sorgente o no.
- Raggruppamento delle informazioni sulla posizione. Abbiamo introdotto in precedenza i diversi flag basati sui tag generati dal NER Tagger. Tra questi figuravano 4 attributi relativi alla posizione che possono risultare fortemente dipendenti tra loro, e come abbiamo visto nella sezione teorica sugli alberi di decisione, questi possono portare alla generazione di un albero errato. Pertanto abbiamo deciso di collassare queste informazioni in un solo nuovo attributo, denominato *"hasPlace"*, un flag che sarà *True* se anche solo uno dei quattro tag (*Position, Country, StateOrProvince, City*) è presente nel tweet, *False* altrimenti.

Il nodo **Select Attribute** è l'equivalente di una `SELECT` in SQL, permettendo di selezionare un sottoinsieme di attributi del dataset in input, eliminando gli altri. In questo caso lo abbiamo utilizzato per eliminare attributi non più necessari, come quelli sulla posizione (In quanto abbiamo generato il nuovo attributo *hasPlace* che collassa tali informazioni in un singolo attributo). Piuttosto che indicare tutti gli attributi filtrati ed eliminati (feature selection) rimandiamo alle righe successive in cui introdurremo la tabella con tutti gli attributi utilizzati per la classificazione.

Il nodo **Set Role** permette di indicare al sistema qual è l'attributo etichetta di classe, per non considerarlo in fase di addestramento assieme agli altri attributi.

Il nodo **Cross Validation**, è un nodo complesso. Infatti, è possibile definire la sua composizione. Espandendolo si nota come esso presenti due diverse sezioni: **Training** e **Testing**, nelle quali si può indicare separatamente come utilizzare i dati corrispondenti. Si consiglia di riguardare il funzionamento della Cross Validation, esposto nella Sezione 2.5.3.

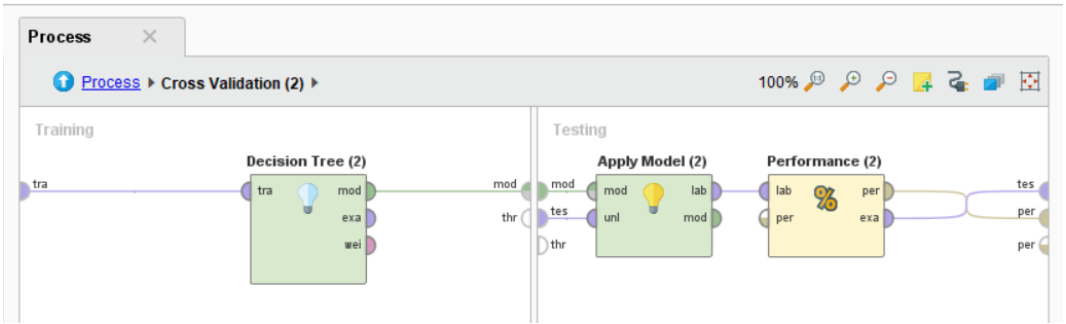

Figura 4.3: Composizione del nodo Cross Validation

Sul lato sinistro, nella sezione **Training**, va ovviamente addestrato il modello, fornendo la porzione di dataset che rappresenta il training set. Sul lato destro invece, nella sezione **Testing**, va applicato il modello per valutarne le performance, utilizzando la porzione di dataset rappresentante il test set.

Menzioniamo, per la Cross Validation, il valore di due parametri fondamentali. In particolare, abbiamo scelto $Number\ of\ folds = 10$ e $Sampling\ type =$ "$Stratified\ Sampling$". Per quest'ultimo è necessario un approfondimento: il Sampling Type indica la modalità con cui la Cross Validation selezionerà i record da splittare nei diversi fold. Con lo **Stratified Sampling**, si effettua un campionamento random ma mantenendo costante, attraverso i folds, la distribuzione delle etichette. In questo modo si riduce l'aleatorietà del classificatore stesso, rispetto al caso in cui il campionamento è puramente randomico.

All'interno del nodo di Cross Validation, nella sezione Training, troviamo finalmente il **Decision Tree**, il nodo che implementa l'algoritmo di generazione di alberi di decisione. Anche del funzionamento degli alberi di decisione abbiamo parlato abbastanza, vale la pena qui indicare soltanto alcune scelte fatte in relazione ai parametri. In particolare, il criterio di split scelto è il **Gini Index**, mentre la profondità massima dell'albero è 9. Si è scelto di non applicare nè post-pruning, nè pre-pruning.

L'ultimo nodo, sempre interno a quello di Cross Validation, stavolta però nella sezione di Testing, è l'**Apply Model**, che non presenta parametri particolari. Esso infatti, non fa nient'altro che prendere i dati in input, applicarvi il modello e misurarne le performance facendo il confronto con le etichette associate.

Vale la pena menzionare inoltre il nodo **Optimize Parameters (Grid)**, che pur se non presente nell'immagine che mostra il processo nella sua interezza, si è dimostrato molto utile per la scelta dei parametri. Esso infatti non è nient'altro che un processo che lancia tante iterazioni del sotto-processo inserito al suo interno quante sono le combinazioni di parametri scelti possibili. In pratica, implementa un approccio **brute force**, lanciando il processo tante volte, ogni volta con una combinazione di parametri diversa, e fornendo in output le performance di ogni istanza.

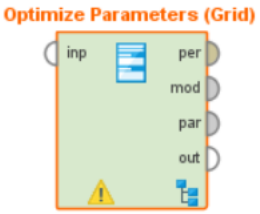

Figura 4.4: Il nodo Optimize Parameters

Prima di passare all'applicazione dell'albero ed alle relative performance, ricapitoliamo quali sono gli attributi utilizzati per la generazione dell'albero di decisione. Dopo atten-

ta feature selection e diversi tentativi effettuati, la combinazione migliore di attributi (esattamente quella che viene fuori dal nodo "Select Attributes") è la seguente:

Attributi scelti		
TweetID	nLikes	TweetBotSource
Followers	DeltaSeconds	TwitterDeckSource
TotalTweets	hasCriminal	TwitterFeedSource
TwitterAge	hasDeath	UnofficialSource
nHashtags	hasPerson	MobWebSource
nMentions	hasPlace	InstagramSource
nUrls	MobileSource	FacebookSource
Verified	ClientSource	DeveloperSource
nRetweets	TabletSource	

Ricapitolando brevemente, i primi 11 attributi, da *TweetID* a *DeltaSeconds* sono quelli estratti da Twitter tramite la libreria `Twython`, mentre quelli che presentano il prefisso *"has-"* sono quelli estratti utilizzando lo *Stanford NER Tagger*. Gli ultimi, quelli che presentano il suffisso *"-Source"*, sono invece i booleani che indicano la fonte che ha generato del tweet.

Illustrare su carta l'albero di decisione che viene generato seguendo le linee guida introdotte nell'elaborato è un po' complicato per motivi di spazio. Ne mostriamo pertanto solo la struttura radiale, scelta per motivi di compattezza. Si consiglia comunque di analizzare l'albero direttamente da RapidMiner, in modo tale da poterlo esplorare più nel dettaglio.

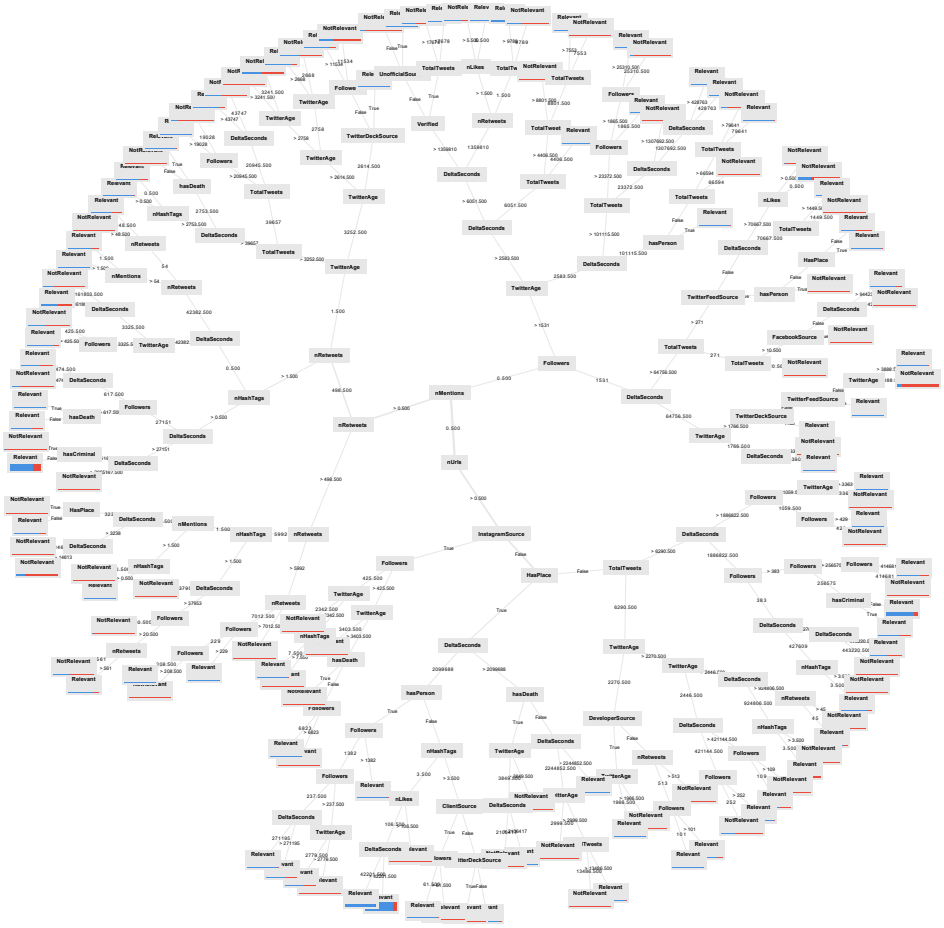

Figura 4.5: Struttura del decision tree

Essendo l'albero molto esteso, è difficile dall'immagine appena introdotta comprenderne la struttura e le decisioni sullo split. Introduciamo pertanto una porzione dell'albero descritta in formato testuale, in quanto risulta interessante comprendere quali sono stati selezionati come attributi più rilevanti per la classificazione.

```
nUrls > 0.500
|   InstagramSource = False
|   |   HasPlace = False
|   |   |   TotalTweets > 6290.500
|   |   |   |   DeltaSeconds > 1886822.500
|   |   |   |   |   Followers > 1059.500
|   |   |   |   |   |   TwitterAge > 3363: Relevant
                            {Relevant=1, NotRelevant=0}
|   |   |   |   |   |   TwitterAge <= 3363: NotRelevant
                            {Relevant=0, NotRelevant=5}
|   |   |   |   |   Followers <= 1059.500
|   |   |   |   |   |   Followers > 429: Relevant
                            {Relevant=2, NotRelevant=0}
|   |   |   |   |   |   Followers <= 429: NotRelevant
                            {Relevant=0, NotRelevant=1}
|   |   |   |   DeltaSeconds <= 1886822.500
|   |   |   |   |   Followers > 383
|   |   |   |   |   |   Followers > 256575
|   |   |   |   |   |   |   Followers > 414681: Relevant
                            {Relevant=10, NotRelevant=4}
|   |   |   |   |   |   |   Followers <= 414681: NotRelevant
                            {Relevant=0, NotRelevant=3}
|   |   |   |   |   |   Followers <= 256575
|   |   |   |   |   |   |   hasCriminal = False: Relevant
                            {Relevant=336, NotRelevant=41}
|   |   |   |   |   |   |   hasCriminal = True: Relevant
                            {Relevant=9, NotRelevant=5}
```

Dal listato appena introdotto, si nota come l'attributo maggiormente significativo per la suddivisione dei dati sia **nUrls**, ovver il numero di URL presenti nel tweet. Seguendo un ramo che porta fino alla prima foglia, notiamo come il classificatore riesce a distinguere correttamente tweet non rilevanti da quelli rilevanti, quando questi possiedono le

seguenti caratteristiche:

- Il tweet **ha almeno un URL**
- La fonte che ha generato il tweet **non è Instagram**
- Nel tweet **non vi sono informazioni sulla posizione**
- Il tweet è stato **generato dopo più di 21 giorni dall'inizio della crisi**
- L'autore del tweet ha **più di 1059 followers**

A questo punto, tweet che hanno queste caratteristiche sono suddivisi in rilevanti e non rilevanti a seconda del loro attributo **TwitterAge**, ovvero da quanto tempo esiste l'account che ha generato il tweet. In particolare:

- Se l'account **esiste da più di 3363 giorni**: **il tweet non è rilevante**
- Se l'account **esiste da massimo 3363 giorni**: **il tweet è rilevante**

Analizzando l'intero albero è possibile fare tantissime assunzioni diverse come quella appena introdotta, ognuna con una corrispettiva probabilità di inferire correttamente la classe di appartenenza del tweet.

4.1.4 Prestazioni

Arriviamo finalmente all'ultimo argomento riguardante il *Classificatore C1*, le prestazioni. Prima di arrivare ai numeri, anticipiamo che per questo approccio abbiamo effettuato diverse suddivisioni dei dataset per garantire che gli errori di classificazione siano il più possibile comparabili con gli errori reali in fase di applicazione del classificatore su dati mai visti prima.

Partiamo dal primo set di prestazioni, che abbiamo ottenuto utilizzando la **Cross Validation** su tutti i dataset indicati all'inizio del Capitolo 3, analogamente a quanto mostrato nel Processo RapidMiner introdotto nella Sezione 4.1.3. La matrice di confusione risultante è la seguente:

	True R	**True NR**	**Class precision**
Pred R	2607	782	76.93%
Pred NR	415	676	61.96%
Class recall	86.27%	46.36%	

Questa soluzione fornisce una **accuratezza del 73.28%**.

Successivamente, abbiamo provato il classificatore utilizzando il **metodo holdout**, utilizzando diversi dataset come test set. Il primo per il quale mostreremo le performance lo abbiamo realizzato addestrando il classificatore con tutti i dataset ad eccezione di quelli relativi all'**incidente di New York, 2013**, che abbiamo utilizzato per il **testing**. I risultati sono i seguenti:

	True R	**True NR**	**Class precision**
Pred R	561	32	94.60%
Pred NR	63	41	39.42%
Class recall	89.90%	56.16%	

In questo caso si raggiunge un'**accuratezza dell' 86.37%**.

Ancora, abbiamo provato il classificatore sempre con il **metodo holdout**, utilizzando come **dati di testing** quelli **relativi all'esplosione in West Texas, 2013**. I risultanti sono i seguenti:

	True R	**True NR**	**Class precision**
Pred R	255	133	65.72%
Pred NR	77	198	72.00%
Class recall	76.81%	59.82%	

Con **accuratezza pari al 68.33%**.

Infine, come ultimo tentativo, sempre con **metodo holdout**, abbiamo utilizzato il solo **dataset delle alluvioni in Queensland per il testing**:

	True R	**True NR**	**Class precision**
Pred R	484	114	80.94%
Pred NR	92	108	54.00%
Class recall	84.03%	48.65%	

Con **accuratezza del 74.19%**.

Nel complesso quindi, volendo indicare un unico valore di accuratezza, facendo una media tra i tentativi fatti potremmo asserire che il **Classificatore C1 ha un'accuratezza pari al 75.54%**.

Può essere interessante confrontare le prestazioni di un classificatore che combina l'utilizzo di metadati e flag, come quello appena introdotto, con quelle di un classificatore che utilizza singolarmente i due approcci.

Ad esempio, considerando un classificatore addestrato tramite **Cross Validation**, realizzato utilizzando gli stessi dati elaborati dai classificatori prima utilizzati, tenendo in considerazione **solo i metadati**, si ottiene la seguente matrice di confusione:

	True R	**True NR**	**Class precision**
Pred R	2565	763	77.07%
Pred NR	457	695	60.03%
Class recall	84.88%	47.57%	

Questo approccio permette di ottenere un'**accuratezza del 72.77%**.

Allo stesso modo, realizzando il classificatore tenendo in considerazione **solo i flag estratti tramite tecniche NLP**, le performance ottenute sono le seguenti:

	True R	True NR	Class precision
Pred R	2956	1412	67.67%
Pred NR	66	46	41.07%
Class recall	97.82%	3.16%	

Il quale approccio permette di ottenere un'**accuratezza del 67.01%**.

Analizzando questi ultimi due più semplici classificatori, si può notare come i flag NLP da soli non siano in grado di ottenere delle performance valide, ma permettono, se combinati con i metadati, di migliorare la situazione e rendere il tutto più stabile.

4.2 Il classificatore C2 - Doc2Vec

Il secondo classificatore che abbiamo realizzato, come abbiamo più volte spiegato, sfrutta un approccio molto differente, facendo uso della codifica Doc2Vec e della classificazione tramite un modello di regressione lineare. A differenza del Classificatore C1 inoltre, non è realizzato in Rapidminer, ma direttamente in Python, sfruttando alcune librerie apposite che introdurremo nelle prossime righe.

In primo luogo, come fatto anche per il *Classificatore C1*, introduciamo una visione ad alto livello del flow-chart del *Classificatore C2*.

Figura 4.6: Flow chart del Classificatore C2

Anche qui, i due blocchi ad alto livello fondamentali, ovvero "Rappresentazione Doc2Vec" e "Regressione Lineare" meritano una maggiore attenzione e saranno esposti qui di seguito.

4.2.1 Rappresentazione Doc2Vec

Per implementare la rappresentazione Doc2Vec, Python mette a disposizione una libreria molto utilizzata per il topic modeling, che prende il nome di **`gensim`** [12]. Anche in questo caso non riteniamo valido entrare eccessivamente nei dettagli della rappresentazione, in quanto sono stati già abbondantemente esposti nella sezione apposita, ma ci soffermeremo sui moduli della libreria che sono risultati essere fondamentali per l'implementazione.

Per la rappresentazione vettoriale abbiamo utilizzato ovviamente un modello di rete neurale già addestrato, data l'enorme capacità computazionale richiesta per addestrare un modello con dizionari ampi come quelli della lingua inglese.

In particolare, abbiamo utilizzato in primo luogo il **costruttore `Doc2Vec`**, il quale prende diversi parametri, tra i quali spiccano sicuramente quello della **dimensione della finestra**, che abbiamo posto uguale a 10 parole, ed alle **dimensioni dell'hidden layer**, che abbiamo posto pari a 300 neuroni.

La funzione **`Doc2Vec.infer_vector()`** prende in input la lista di parole presenti in un tweet e ne calcola il vettore numerico, applicando il modello della rete neurale. Questa l'abbiamo utilizzata per codificare sia gli elementi del training che del test set, per poi successivamente applicarvi una regressione lineare.

Per l'applicazione il modello di regressione lineare sui vettori numerici, abbiamo utilizzato la libreria **`scikit-learn`** ed in particolare la classe **`LogisticRegression`** tramite la quale creare il modello, ed i suoi metodi **`fit()`** per l'addestramento e **`score()`** per la validazione delle prestazioni.

4.2.2 Prestazioni

Non avendo utilizzato RapidMiner per questo classificatore, nè tantomeno alcun albero di decisione da mostrare, passiamo direttamente ai risultati. Anche qui abbiamo fatto diverse prove, distribuendo diversamente i 10 dataset a disposizione tra training e test

set. In particolare, i risultati ottenuti sono i seguenti:

Dataset training	Dataset testing	Accuratezza
7	3	74.41%
9	1	74.79%
3	7	74.12%

Tirando le somme quindi, notiamo come questo classificatore abbia delle performance che, anche se non sempre superiori a quelle del *Classificatore C1*, risultano generalmente più stabili, come dimostrato dalla piccola varianza relativa all'*accuracy* dei diversi tentativi effettuati per il testing. Ciò è dovuto quasi sicuramente alla maggior correlazione che la rappresentazione Doc2Vec riesce ad introdurre nei confronti dell'etichetta di classe, rispetto a quella che si riesce ad ottenere con gli attributi utilizzati nel *Classificatore C1*.

Può essere interessante confrontare questo classificatore con un classico classificatore realizzato utilizzando un indice di frequenza delle parole nei diversi documenti, nel nostro caso tweet, come il TF-IDF. Infatti, un classificatore basato su albero di decisione addestrato tramite **Cross Validation**, e avente come attributi il valore **TF-IDF** delle parole presenti nei diversi tweet, realizzato utilizzando sempre gli stessi dataset, restituisce prestazioni di questo tipo:

	True R	**True NR**	**Class precision**
Pred R	3959	1122	77.92%
Pred NR	143	928	86.65%
Class recall	96.51%	45.27%	

Con un'**accuratezza del 79.44%**, che per quanto possa sembrare promettente, abbiamo deciso di scartare rispetto agli altri approcci in quanto, come indicato già nelle sezioni precedenti, un classificatore basato su un indice come il TF-IDF porta con sè alcuni problemi che non lo rendono applicabile al nostro caso. Infatti simili tecniche

richiedono di addestrare nuovamente il classificatore ogni volta che lo si utilizza con un nuovo dataset. Questo perchè, i valori degli indici basati sulla frequenza, come il TF-IDF, dipendono dalle parole utilizzate e dai documenti sui quali si verifica la presenza di ogni parola. Per questo motivo, dopo che il classificatore è addestrato, una volta estratto un altro insieme di dati relativo ad una nuova catastrofe (Il quale introduce ulteriore ritardo, in quanto bisogna avere tutto il dataset disponibile contemporaneamente, e non si possono classificare tweet singolarmente, appena generati) sarà necessario addestrare da capo l'intero classificatore per considerare nell'albero di decisione eventuali nuove parole, non presenti nei precedenti dataset, e ricalcolare il valore dell'indice di tutte quelle precedenti, in quanto avendo introdotto altri documenti (tweet) la frequenza varierà.

Inoltre, un approccio del genere richiede necessariamente che sia presente la fase di tagging manuale durante il verificarsi dell'evento emergenziale. Difatti, essendo necessaria una nuova fase di addestramento del classificatore, i nuovi dati, anche se ottenuti da pochi secondi, devono essere dotati necessariamente di etichetta. Questa caratteristica rende di conseguenza non utilizzabile questa implementazione per una situazione di questo tipo, ma ritenevamo interessante confrontarne le performance.

Capitolo 5

Conclusioni

Nell'elaborato che si appresta a concludersi abbiamo esposto le operazioni di ricerca, progettazione ed implementazione resesi necessarie per la realizzazione di un sistema che fosse in grado di analizzare tweet prelevati dal social network Twitter, e di inferire automaticamente se questi potessero essere utili per l'organizzazione delle operazioni di soccorso, permettendo ad enti come la Croce Rossa o la Protezione Civile di poter sfruttare a proprio vantaggio l'enorme flusso di informazioni che viene generato durante il verificarsi di eventi che mettono in pericolo vite umane.

Sono stati passati in rassegna prima gli argomenti teorici alla base del progetto realizzato, partendo dai fondamenti del Machine Learning quali classificazione, regressione, alberi di decisione e altri algoritmi, fino ad arrivare ad argomenti più avanzati quali Natural Language Processing e rappresentazione numerica, con attenzione particolare alle tecniche di Named Entity Recognition e Rappresentazione Doc2Vec, sulla quale si basano i due classificatori che abbiamo realizzato, le cui scelte di progettazione e di implementazione sono state esposte in sezioni apposite.

In particolare, sono stati realizzati ed analizzati:

- **Classificatore 1 (C1) - Metadati e Flag NLP:** prima versione del classificatore, che prevede la realizzazione di un **albero di decisione** addestrato su un dataset composto da **metadati associati ai tweet** e dalla presenza di alcuni

flag ad indicare la presenza o meno di alcune caratteristiche all'interno del testo del tweet la cui estrazione è stata realizzata tramite l'utilizzo dello **Standord NLP NER Tagger**.

- **Classificatore 2 (C2) - Doc2Vec:** Una versione alternativa del classificatore, realizzata sfruttando le tecniche di codifica *Doc2Vec*, che trasformano il testo di ogni tweet in un vettore numerico avente caratteristiche adatte all'addestramento di un **modello di regressione lineare**, utilizzato per la classificazione dei tweet nelle due classi rilevante e non rilevante.

Dalla valutazione delle prestazioni dei due classificatori emerge come entrambi abbiano un buon errore di classificazione, seppur migliorabile. Inoltre, si denota una maggiore stabilità del *Classificatore C2*, per via delle caratteristiche insite della rappresentazione utilizzata.

Future implementazioni del classificatore potrebbero infatti essere basate sul miglioramento di questa seconda implementazione, magari andando ad addestrare la rete neurale rendendola adatta a classificare e rappresentare tutte le parole che figurano nei tweet presi in considerazione. Tale compito non è stato svolto in quanto richiede enorme potenza computazionale data l'importante mole di dati e la complessità della rete, potenza che non avevamo a disposizione.

Il progetto realizzato è stato portato avanti con passione e dedizione, nella speranza che possa essere d'aiuto a chi malauguratamente si trova ad affrontare le tragiche situazioni per le quali è stato pensato, uomini, donne e bambini che sono stati sempre nei nostri pensieri negli ultimi mesi e che ci hanno aiutato a trovare la motivazione necessaria a superare gli ostacoli che si sono palesati sul nostro cammino.

Bibliografia

[1] Aggiornamenti sulle notizie in Twitter:

https://www.smartworld.it/internet/twitter-novita-notizie-timeline-notifiche-esplora.html

[2] Chatter on the red: what hazards threat reveals about the social life of microblogged information

http://portal.acm.org/citation.cfm?id=1718965

[3] Beyond Microblogging: Conversation and Collaboration via Twitter

https://www.computer.org/csdl/proceedings/hicss/2009/3450/00/03-05-05-abs.html

[4] 2009 Red River, North Dakota flood:

https://www.wikiwand.com/en/2009_Red_River_flood

[5] Python Programming Language

https://www.python.org/

[6] Rapidminer Studio 8

https://rapidminer.com/

[7] Stanford Named Entity Recognition Tagger

https://nlp.stanford.edu/software/CRF-NER.html

[8] Doc2Vec: Deep Learning paragraph2vec

https://radimrehurek.com/gensim/models/doc2vec.html

[9] Pandas Data Analysis Library

https://pandas.pydata.org/

[10] Twython: Twitter API for Python

https://twython.readthedocs.io/en/latest/#

[11] Natural Language Processing Toolkit in Python

https://www.nltk.org/

[12] Gensim: Topic modeling for humans

https://radimrehurek.com/gensim/

[13] Some Studies in Machine Learning Using the Game of Checkers

https://ieeexplore.ieee.org/document/5392560/

[14] 6 Common Machine Learning Business Applications

https://www.datascience.com/blog/common-machine-learning-business-applications

[15] CrisisLex

http://www.crisislex.org/

[16] Colorado Wildfires, 2012

https://www.wikiwand.com/en/2012_Colorado_wildfires

[17] Alberta Floods, 2013

https://www.wikiwand.com/en/2013_Alberta_floods

[18] Australia Bushfire, 2013

https://www.wikiwand.com/en/2013_New_South_Wales_bushfires

[19] Boston marathon bombings, 2013

https://www.wikiwand.com/en/Boston_Marathon_bombing

[20] Colorado floods, 2013

https://www.wikiwand.com/en/2013_Colorado_floods

[21] Glasgow Helicopter Crash, 2013

https://www.wikiwand.com/en/2013_Glasgow_helicopter_crash

[22] Los Angeles Airport Attack, 2013

https://www.wikiwand.com/en/2013_Los_Angeles_International_Airport_shooting

[23] New York train crash, 2013

https://www.wikiwand.com/en/December_2013_Spuyten_Duyvil_derailment

[24] Queenslands floods, 2013

https://www.bbc.com/news/world-asia-21226178

[25] West Texas Explosions, 2013

https://www.wikiwand.com/en/West_Fertilizer_Company_explosion

[26] Introduction to Natural Language Processing - R. Kibble

https://london.ac.uk/sites/default/files/study-guides/introduction-to-natural-language-processing.pdf

[27] Penn Treebank POS Tagset

https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

[28] Information extraction from text

http://www.nltk.org/book/ch07.html#ex-ie4

[29] Word2Vec Tutorial

http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

[30] Precisione e richiamo

https://www.wikiwand.com/it/Precisione_e_recupero

[31] Distributed Representations of Sentences and Documents - Quoc Le, Tomas Mikolov

https://cs.stanford.edu/~quocle/paragraph_vector.pdf

[32] Introduction to Data Mining - Tan, Steinbach, Kumar

Ringraziamenti

Si conclude oggi un percorso lungo due anni, due anni di studio, risate, disperazione, divertimento ed interminabili viaggi in treno, per trasferirmi da una famiglia all'altra, da quella naturale a Napoli, a quella che si è creata qui a Torino, e viceversa. Ed è grazie a loro che sono riuscito ad arrivare al termine di questo percorso, perchè senza il supporto morale, il divertimento, la disperazione comune, l'amore e l'affetto ricevuti da ognuno di loro, non ce l'avrei mai fatta.

A mio padre Amedeo, la cui assenza è stata una presenza costante e fondamentale in questo percorso, al suo amore che va oltre corpo, mente e tempo. Nello scrivere questa tesi e nel registrarmi alla sessione di laurea ho fatto fatica a mettere il mio nome e non il suo, sotto la voce "Candidato".

A mia madre Luisa, colei che più di tutti ha creduto in me senza mai battere ciglio, dandomi sempre tutto il supporto di cui avevo bisogno, non facendomi mai mancare compagnia durante i più intensi periodi di studio e preoccupandosi ogni volta che la mia voce pareva "scocciata", tirandomi su a seguito di ogni fallimento e gioiendo a fronte di ogni mio piccolo successo.

A mio fratello Giorgio, alle risate che ci siamo fatti ogni sera giocando assieme al computer, ed a quelle fatte ad ogni nostro incontro. Grazie a lui, che in due anni mai mi ha fatto sentire come se avessi davvero lasciato casa, perchè di fatto è stato sempre al mio fianco.

A mia sorella Federica, alle cene domenicali a casa con Pierluigi, ai pranzi fatti alla ricerca della pizzeria "meno peggio" di Torino, ed alla sua incredibile dinamicità e proattività che mi ha trasmesso in ogni occasione.

A mia nonna Margherita, che nonostante gli 800 e passa kilometri di distanza mi ha fatto sentire il suo amore come se vivessi, pranzassi e cenassi ancora con lei, ed ai chili in eccesso che ho portato con me ogni volta che sono tornato da Napoli, per i quali le attribuisco tutto il merito.

A mio zio Mario, ed ai suoi figli Alessandro e Angelica, ai suoi consigli e le nostre giornate a Milano che mi facevano sentire sempre in famiglia, al nostro incredibile viaggio in Argentina, suo vero ed unico posto nel mondo, ed alle emozioni che ho provato nel visitarla assieme a loro, provando sulla mia pelle tutto quanto ci ha sempre raccontato da quando è tornato in Italia.

A mio zio Agostino, ai suoi messaggi di stima e affetto nei miei confronti, ed al mese in cui siamo stati insieme a Torino, ricco di risate, film e chiacchierate in spensieratezza.

Ai miei zii Giorgio e Brigida, Ruggero e Sabrina, che mi hanno accolto a braccia aperte ad ogni mio ritorno, sempre pronti a fare le corse pur di salutarmi, alle nostre cene di famiglia che non mi perderei per niente al mondo e all'affetto smisurato che nutro nei loro confronti, come se fossero i miei secondi genitori.

Alle mie cugine Carolina, Gisella e Clara, al senso di amore ed ammirazione che mi fanno provare ogni volta, del tutto reciproci e per le quali provo un senso di protezione smisurato.

A mio cugino Michele e i nostri sporadici incontri, ed a quella volta che di ritorno da Torino è venuto fino in stazione a Firenze solo per prendersi un caffè con me.

Ai miei amici di sempre: Luca, Lorenzo, Piero, Gianmarco, Fabio e Fernando, con i

quali sono sempre stato felicissimo di rivedermi ad ogni mio rientro, alle loro storie, imbarazzanti media trasmessi su Whatsapp ed alle nostre lunghe telefonate per scambiarci gli ultimi aggiornamenti.

A Flavio, che è stato con me dal primo all'ultimo giorno della mia esperienza a Torino, a partire dal primo, ansiogeno semestre al Politecnico, fino alla consegna del progetto finale, che sono stato onorato di realizzare con lui, passando per ogni singolo esame della magistrale, nei quali il costante confronto con lui è stato di fondamentale aiuto, trasmettendomi il suo senso di dedizione ed attenzione al dettaglio. Ad ogni partita della Roma vista insieme, all'emozionantissima prima volta in trasferta, in occasione dell'indimenticabile Torino-Roma che spero di ripetere da quando siamo usciti dallo stadio, fino a quel 10 Aprile 2018, che rimarrà indelebile nella mia memoria. Ai nostri discorsi sul calciomercato, alle nostre discussioni tecnico-tattiche dalla dubbia veridicità, alle esilaranti dirette di Asso con tanto di dedica del Nostro, che *all'effettivo* ha alleviato la pesantezza dell'estenuante sessione estiva 2017. Alle nostre serate combo "Cinema + KFC", all'amore smisurato per il colonnello, ai suoi nomignoli e fotomontaggi che hanno decorato il mio alloggio torinese fino ad oggi, e potrei continuare per ore, perchè racchiudere due anni di vita in un testo è impossibile.

A Federico, il nostro cucciolo, ed alla nostra fraterna amicizia, alla nostra prima uscita insieme condita con 45 minuti di ritardo dovuti alla sua prontezza nello scegliere il tram sbagliato, ai nostri brevi incontri a Napoli in attesa del treno per Torino, ai tre giorni di visita a Settembre 2017, al weekend tra Gaeta e Scauri con Mattia e Vincenzo, alla sua presenza costante, come se fossimo amici da tutta la vita. Alle nostre esilaranti serate votate al conoscere gente nuova, alle sue minacce di impiccagione o volontario incidente d'auto, ai nostri ben poco fruttuosi mesi di palestra, ed alle nostre continue telefonate, anche quando distanti migliaia di chilometri.

A Uccio, il mio sole di notte, la mia guida nelle giornate più buie, luce dei miei occhi e aria per i miei polmoni. Al suo acquario, al suo pesce suicida, al granchio assassino, alle nostre indimenticabili spese al Simply, alla nostra visita presso Acquarissima 2000

che porterò sempre nel cuore, al nostro weekend a Scauri e Gaeta e alle sue mirabolanti storie che hanno allietato anche le più tristi giornate.

A Giusteppe, l'esemplare perfetto di giustezza e meritocrazia. Alla sua scissione tra corpo e mente, ai suoi preziosi consigli, alla sua infamità in ambito calcistico, che accettiamo come un cazzotto in pancia pur di averlo tra noi.

A Mattia ed alle sue serate giochi nei periodi di studio più intenso, alla sua infinita tranquillità anche nelle situazioni più critiche, ai nostri due giorni a Nizza culminati con sauna e bagno in piscina, ai suoi "Ci sta tempo...", al suo essere pioniere di Tinder con conseguenti imbarazzanti racconti fino ad arrivare al weekend tra Scauri e Gaeta, ricco di storie e eventi da raccontare.

A Nico, alle sue immense mani che producono codice incomprensibile e dalla capacità di carico pari a quella di un carro trasporto, alla sua morte e conseguente rinascita dalle sue ceneri, alle sue malattie evolute e trasportate in ogni ambito ed al suo costante essere nomade, che ha suscitato continue e numerose risate, oltre che lecite domande, ancora irrisolte, nel corso di questi due anni.

A Piero ed al suo catapultarsi nella mia vita nell'ultimo anno, alle serate passate insieme a fare "i matti", al suo rapporto di odio e amore con Federico, alla sua storia da mille e una notte ed alla sua sempre presente voglia di fare festa che più volte ha messo a rischio la mia disciplina e dedizione.

A Toppariello, al suo accettare ogni nostra richiesta anche se priva di apparente logica, al suo imbuto, al suo straziante ed inutile "Farewell Party", alla sua storia d'amore dall'epilogo ancora incerto, ai suoi nickname social che a breve saranno portati all'anagrafe ed ai suoi festeggiamenti per lo scudetto del 2018, che accettiamo non senza remore pur di averlo tra noi.

A Paolo Garza, colui che potrei definire la scelta migliore degli ultimi 6 mesi, relatore

sempre pronto e disponibile a fornire il suo prezioso supporto.

E a te, che se stai leggendo queste righe, avrai avuto sicuramente una parte importante in tutto questo, che la mia testa costantemente tra le nuvole mi ha fatto ingiustificatamente omettere.