Politecnico di Torino

Facoltà di Ingegneria
Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea:

Analisi e classificazione di tweet inerenti a eventi emergenziali

Relatore:
Prof. Paolo Garza

Candidato:
Flavio Candela
s232758

Anno Accademico 2017/2018

Indice

Elenco delle figure

Capitolo 1

Introduzione

Negli ultimi tempi il processamento di una grande mole di dati sta assumendo un ruolo sempre più fondamentale in numerosi ambiti tecnologici, industriali e non solo. In particolare, uno degli obiettivi che la società moderna si sta ponendo, è quello di riuscire ad estrapolare conoscenze (apparentemente nascoste) da dati già in possesso, con lo scopo di ottenere informazioni utili o addirittura con il fine di riuscire a predire "situazioni" e/o "comportamenti".

Tra i tanti contesti applicabili, uno di enorme interesse, specialmente da un punto di vista etico (e che quindi si discosta da interessi economici e materiali) è quello che riguarda la salute, la salvaguardia e la sicurezza della popolazione.

Ed è proprio in questo ambito che è quindi stata svolta l'attività di tesi, frutto di un lavoro congiunto che ha visto la partecipazione del mio collega di studi Francesco Vergona.

In questi ultimi mesi ci si è posti come obiettivo l'ideazione e progettazione di un **classificatore** che riuscisse autonomamente, mediante un **apprendimento supervisionato**, a riconoscere (nel gergo, classificare) dati, o più propriamente messaggi, contenenti informazioni utili durante cataclismi o situazioni di emergenza, con lo scopo, ad esempio, di velocizzare eventuali soccorsi da parte delle autorità.

Nello specifico, l'idea è stata quella di applicare la classificazione a messaggi, detti propriamente *tweet*, scritti e condivisi da utenti sulla piattaforma *Twitter* (definito come micro-blogging service in quanto i messaggi raggiungono una lunghezza massima

di 140 caratteri). I social network infatti, possono essere facilmente visti come una ricca fonte di informazioni , in quanto gli utenti comunicano col "mondo" in tempo reale, a volte fornendo "dati" utili. Riuscendo ad identificare (in maniera del tutto "automatica") quali sono i tweet *rilevanti* in certe situazioni catastrofiche, l'intervento delle autorità potrà essere più tempestivo ed efficace, riuscendo a limitare i danni oltre a salvare più vite umane. Col termine *rilevante* o *informativo* si intende tutto ciò che può essere considerato come utile per i cittadini e per le autorità durante situazioni di emergenza.
Infatti, secondo quanto riportato dall' *International Telecommunication Union* [1] , ci sono più di 6.8 miliardi di dispositivi mobili attivi nel mondo, numero che sta crescendo globalmente ogni anno, mentre solo Twitter conta più di 300 milioni di utenti attivi al mese. Questi numeri riescono a dare solo un'idea intuitiva di quanto traffico d'informazione viene oramai generato, e che potrebbe essere quindi opportunamente sfruttato in vari ambiti. Un esempio degno di nota è il fatto che, durante recenti crisi (come il bombardamento a New York del Settembre 2016), alcune notizie fossero disponibili su Twitter prima di ogni altro canale convenzionale di informazione. Ovviamente però, una grande mole di dati ha lo svantaggio di poter essere complicata da analizzare e gestire, rischiando di condurre ad una *paralisi* causata dall'enorme traffico di informazioni.
L'attività di laboratorio ha quindi previsto:

- L'acquisizione dei dati (tweet provenienti da Twitter), riguardanti cataclismi o emergenze realmente avvenuti negli anni passati. Per i nostri scopi, è stato anche necessario ottenere dei dati già "etichettati" (usati come **training set**) per poter generare il modello successivamente utilizzabile per la classificazione.

- Studio dei dati ed estrazione di eventuali feature che potessero essere rilevanti per il corretto processo di classificazione. In particolare si è fatto uso delle **API di Twitter** per l'acquisizione di specifici **metadati**, e del **Natural Language Processing (NLP)** per identificare informazioni utili sul significato (semantica) del testo dei tweet.

- Generazione del modello di classificazione con annessa valutazione delle performances, e successiva applicazione su un insieme di dati non etichettati.
- Implementazione di un approccio differente, che si basi su una codifica vettoriale dei messaggi (tramite il modello **Doc2Vec**), così da poter effettuare una classificazione basata solo su testo, che è stato pertanto opportunamente trasformato.

Nel corso della prima parte della trattazione (*Capitolo 2*) verranno brevemente introdotti gli studi e i lavori, precedentemente effettuati in passato, che sono serviti da linee guida per il corrente progetto.
Nella seconda parte (*Capitolo 3*), verrà spiegato con maggior dettaglio il funzionamento e il ruolo della classificazione con riferimento ai più famosi algoritmi implementati per lo scopo. Saranno inoltre introdotti alcuni concetti teorici sulle reti neurali e il Natural Language Processing per poter avere gli strumenti necessari per la comprensione degli argomenti successivi.
Nel *Capitolo 4* invece, verranno elencati gli strumenti, i softwares e le librerie usate nel corso della progettazione.
Successivamente (*Capitolo 5*) verrà spiegata l'attività di acquisizione e pre-processamento dei tweet ponendo l'attenzione sul processo di estrazione di caratteristiche utili.
Nel *Capitolo 6* si porrà il focus su quali caratteristiche possano essere ricavabili dal testo, e di conseguenza, quali tecniche possano essere usate.
Nel *Capitolo 7* finalmente verrà spiegato il funzionamento del classificatore e come esso sia stato implementato sfruttando i metadati e i flag ottenuti dal Natural Language Processing.
Infine, nell'ultima parte (*Capitolo 8*) verrà invece introdotto un approccio completamente diverso per poter effettuare la classificazione su testo opportunamente trasformato in forma vettoriale.

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.

Capitolo 2

Articoli correlati

Nel corso degli anni sono stati avviati numerosi studi che si sono incentrati sull'utilizzo di classificatori per estrarre informazioni utili in caso di disastri. Come già precedentemente affermato, è Twitter ad essere sfruttato come fonte per la raccolta dei dati, riuscendo a prestarsi bene agli scopi prefissati, grazie alle numerose API open source che permettono l'acquisizione di un gran numero di informazioni sui tweet stessi. Tra i lavori più interessanti, che hanno fornito lo spunto per la nostra attività di tesi, vi sono:

- *"AIDR: Artificial Intelligence for Disaster Response"* [2] in cui viene proposta la piattaforma AIDR designata per effettuare la classificazione di brevi messaggi pubblicati su social network (Twitter), relativi a crisi.
- *"A Language-agnostic Approach to Exact Informative tweet during Emergency Situations"* [3] nato dalla collaborazione tra il Politecnico di Torino e l'Istituto Superiore Mario Boella, in cui viene proposto un approccio basato sul machine learning per riuscire a classificare automaticamente i contenuti informativi e non informativi pubblicati su Twitter durante disastri causati da cataclismi naturali.

Nel caso dell'AIDR, l'obiettivo proposto, quindi, è stato quello di riuscire a classificare messaggi che le persone "postano" (scrivono) durante i disastri, in un ben definito set di categorie di informazione (ad esempio, "needs", "damage" etc.). Per effettuare ciò, il sistema si occupa continuamente di "ingerire" dati da Twitter processandoli facendo

uso di tecniche di classificazione e sfruttando la partecipazione ed interazione umana attraverso del crowdsourcing in tempo reale.

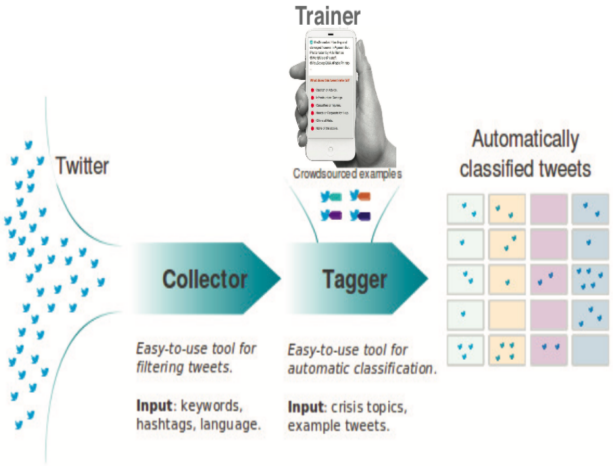

Figura 2.1: AIDR: processo complessivo

Nell'approccio basato su machine learning invece, viene proposta una soluzione del tutto indipendente dal linguaggio e dal contenuto dei tweet (per questo definita come "Language-agnostic"). In particolare, viene fatto uso di dataset di messaggi già campionati ed etichettati, postati su Twitter durante o in seguito a disastri naturali. A partire da questi, è stato definito un set di "feature" di base considerando solo i metadati specifici di ogni tweet (ottenibili mediante le API di Twitter). Per poter inoltre migliorare l'accuratezza del classificatore è stata introdotta una "feature" addizionale, detta "Sorgente" (Source) che viene calcolata considerando il dispositivo o la piattaforma usati per postare il tweet.

Feature	Description
ΔT	Delta time since first *tweet* of the crisis
nFollowers	Number of followers (Fw) of the user
nFollowee	Number of followees (Fe) of the user
ΔregDate	user age in days with respect to its registration date
nTotTw	Total *tweets* posted by the user
vUser	Boolean: TRUE if User is Verified
geoTag	Boolean: TRUE if the *tweet* is geotagged
nHash	Number of # in the *tweet*
nLink	Number of URLs in the *tweet*
nAt	Number of @ sign in the *tweet*

Figura 2.2: Language-agnostic Approach: elenco dei metadati estratti per ciascun tweet

Seguendo le linee guida di questi lavori (specialmente il Language-agnostic approach), abbiamo proposto il nostro classificatore, che andasse ad introdurre altre nuove caratteristiche - tra le quali alcune derivabili dal contenuto del testo - così da cercare di migliorare i risultati.

Capitolo 3

Richiami teorici

Prima di inoltrarci negli aspetti pratici del progetto, in cui sono stati provati ed utilizzati gli algoritmi di **Machine Learning**, è necessario fare un'introduzione teorica per poter dare un quadro generale su cosa sia la classificazione, a quale scopo venga utilizzata, e come possa essere implementata. In seguito verranno invece introdotti i principi delle **Reti Neurali** e **Natural Language Processing**.

3.1 Data Mining

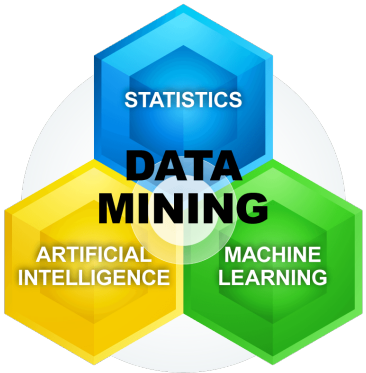

Figura 3.1: Le origini del Data Mining [4]

L'importanza che oggigiorno viene sempre più riconosciuta ai dati, ha portato alla nascita di nuove scienze che possano essere in grado di sfruttare queste numerose in-

formazioni. Tra queste vi è il **Data Mining**, definito come l'insieme di tecniche e metodologie che si occupano dell'estrazione di informazioni implicite, precedentemente sconosciute e potenzialmente utili, a partire dai dati disponibili.
Spesso esso viene usato nell'ambito dei databases come fase di un processo di acquisizione di conoscenze - detto nel gergo **Knowledge Discovery Process** - che prevede la selezione, il processamento e la trasformazione dei dati, per poi poter estrarre conoscenza facendo uso di tecniche di data mining.
Tipicamente si individuano due famiglie di tecniche di analisi:

- *Metodi descrittivi:* in cui vengono estratti modelli interpretabili per descrivere i dati. Tra i più diffusi vi è la *client segmentation* che si occupa di dividere i clienti in gruppi in cui gli elmenti tra loro sono uguali o perlomeno simili.

- *Metodi predittivi:* in cui vengono sfruttati i valori delle variabili conosciute (dette anche *feature*) con l'obiettivo di predirre sconosciuti o futuri valori di altre variabili. L'impiego più noto è per il rilevamento di email di spam. La classificazione appartiene tipicamente a questa categoria di tecniche.

3.2 Classificazione

Nel campo del Data Mining, e nello specifico in quello del Machine Learning e della statistica, la classificazione è vista come il processo di identificazione della categoria adatta (scelta a partire da un insieme ben definito) a cui un certo dato appartiene. Quindi è definita come una metodologia usata per **predire** una etichetta di classe. Per poter effettuare una predizione, il classificatore - ovvero l'algoritmo che implementa la classificazione - si occupa della creazione di un modello. Questo può essere anche usato come strumento per poter interpretare i dati stessi. Per questo motivo la classificazione può essere vista anche come un metodo descrittivo, usato per esplorare i dati.

Analizzandolo ad alto livello, il processo di classificazione:

- Parte da un insieme di dati - records - già classificati, detto training set. Ogni record contiene un set di attributi (detti anche feature), tra i quali vi è la *classe*.

- L'algoritmo di classificazione fa uso del training set per generare un modello per l'attributo di classe. La classe infatti, può essere vista come il risultato di una funzione applicata sui valori degli altri attributi.
- Il modello generato viene usato per assegnare, il più accuratamente possibile, una classe a nuovi records. Viene fatto utilizzo quindi di un test set per determinare l'accuratezza del modello. Tipicamente infatti, il data set originario viene diviso in training set e test set, dove il primo vien usato per costruire il modello e il secondo è usato per validarlo.

Numerosi algoritmi sono stati definiti per poter implementare la classificazione. Tra i più famosi e conosciuti ci sono:

- *Alberi di decisione*
- *Classificatore Bayesiano*
- *Regole di classificazione*
- *Reti neurali*
- *K-nearest Neighours*
- *Support Vector Machine (SVM)*

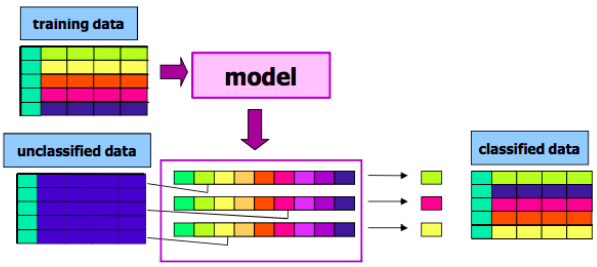

Figura 3.2: Schema dei passi della classificazione [5]

Ciascuno di questi algoritmi viene valutato sulla base di determinati parametri o requisiti. La scelta dell'algoritmo più adeguato è infatti fortemente influenzata da quali requisiti si ritengono necessari. I parametri tipicamente considerati sono:

- *Accuratezza:* quante volte opera correttamente il classificatore
- *Interpretabilità:* quanto un determinato risultato (classificazione) viene compreso
- *Scalabilità:* quanta mole di dati è in grado di gestire
- *Robustezza:* quanto correttamente riesce a gestire rumore, dati mancanti ed eccezioni.
- *Efficienza:* quanto è veloce nel processo di classificazione

A prescindere dall'algoritmo utilizzato, la classificazione fa uso di un insieme di dati disponibili già classificati - detto **training set** - che viene utilizzato per generare un modello che verrà poi sfruttato per la classificazione di un insieme di dati non etichettati, detto **test set**.
La classificazione viene usata in molte applicazioni. In alcune di queste è impiegata come procedura di data mining, in altre come tecnica statistica. Tra i più comuni ambiti vi sono:

- Computer vision
- Rilevamento di frodi
- Classificazione di tipi di patologie differenti
- Analisi delle immagini in ambito medico
- Riconoscimento di caratteri
- Video tracking
- Scoperta e sviluppo di medicine
- Geostatistica
- Riconoscimento del parlato e della scrittura
- Identificazione biometrica
- Classificazione biologica

- Classificazione di documenti
- Motori di ricerca internet
- Pattern recognition

Visto il gran numero di algoritmi di classificazione esistenti, il focus verrà posto solo su quelli più adatti al contesto in cui abbiamo operato e che quindi sono stati provati durante l'attività di tesi.

3.2.1 Alberi di decisione

Gli alberi di decisione sono uno dei più diffusi algoritmi usati per la classificazione. Nello specifico, i modelli di albero in cui la feature da classificare può assumere un set di valori discreti, sono detti **alberi di classificazione**, dove le foglie rappresentano le etichette di classe, e i rami rappresentano l'insieme delle feature che conducono a quelle etichette di classe. Gli alberi di decisione in cui la variabile da etichettare può assumere valori continui (tipicamente numeri reali), sono invece detti **alberi di regressione**. Queste due tipologie di alberi presentano molte similarità, ma anche alcune differenze, come ad esempio la procedura utilizzata per determinare lo *split*.
Come avviene per gli altri noti classificatori, il modello - in questo caso l'albero - viene prima costruito a partire dai dati di training, e poi sarà usato per l'assegnazione di una etichetta di classe.

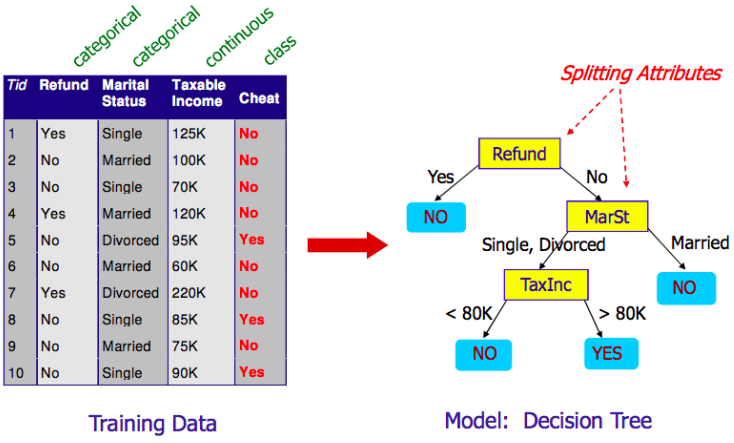

Figura 3.3: Esempio di albero generato a partire da un traning set [5]

Una volta costruito il modello, la classificazione di un dato (record) risulta essere molto semplice: l'albero viene attraversato partendo dalla radice e arrivando fino ad un nodo foglia, sulla base dei valori delle feature del record stesso. Infatti, come già precedentemente affermato, sono gli altri attributi del record a determinare a quale classe appartiene il record stesso.
Per la generazione di un albero di decisione, nel corso degli anni sono stati proposti diversi algoritmi (detti **tree induction algorithms**):

- **Hunt's Algorithm:** uno dei primi algoritmi definiti.
- **CART**
- **ID3**, e il suo successore **C4.5**.
- **SLIQ, SPRINT**

Prendendo in considerazione l'algoritmo di Hunt, uno dei più diffusi in letteratura per spiegare la costruzione del modello, la generazione dell'albero procede nella seguente maniera: a partire dal training set, si sceglie l'attributo *"migliore"* su cui effettuare la divisione (detto "split"), si crea quindi un nodo corrispondente a quell'attributo e si generano tanti archi uscenti in base al valore che l'attributo stesso può assumere (gli attributi non categorici continui o discreti vengono gestiti in maniera differente, come vedremo dopo).
Più formalmente, considerando D_t come insieme di records di training che hanno raggiunto un nodo t, la procedura generale è la seguente:

- Se D_t contiene records che appartengono tutti alla stessa classe y_t, allora t è un nodo foglia che viene etichettato come y_t.
- Se D_t è un insieme vuoto, allora t è un nodo foglia che viene etichettato con un valore di default della classe. Il valore di default può essere definito in varie maniere, ad esempio come etichetta più frequente.
- Se D_t contiene records che appartengono a più classi, viene scelto un attributo per separare i records in sottoinsiemi più piccoli. Vengono generati tanti nodi figli in base ai valori che l'attributo può assumere.

Tipicamente gli algoritmi di generazione degli alberi fanno utilizzo di una strategia **greedy** in cui si sceglie localmente l'attributo migliore, definendo quindi ogni volta l'ottimo locale.
A questo punto le questioni che quindi devono essere prese in considerazione sono:

- Come specificare la condizione di test, e quindi anche il numero di archi uscenti dal nodo. Ciò è fortemente influenzato dal tipo di attributi (se nominali, ordinali o continui) e dal numero di modi in cui può essere effettuato lo split (**2-way/binary split** o **Multi-way split**).
- Come determinare quello che viene definito come *"migliore"* split.
- Determinare quando fermare la suddivisione.

È importante specificare che il binary split divide i records in due sottoinsiemi, e quindi necessita di trovare il partizionamento ottimo, mentre il multi-way split definisce il numero di partizioni pari al numero di valori possibili assumibili da quell'attributo. Ovviamente nel caso di valori continui, per poter usare queste due tecniche di partizione sono necessarie diverse strategie di gestione: la **discretizzazione** (trasformazione in attributi ordinali e categorici) o la **decisione binaria** (in cui si considerato tutti i possibili tagli e si sceglie quello "migliore", es. A < v).
Per quanto riguarda la seconda questione, differenti algoritmi usano differenti metriche per misurare lo split "migliore". Tipicamente viene misurata **l'omogeneità** della distribuzione di classe. Queste metriche sono applicate ad ogni sottoinsieme generato da un ipotetico split, e i risultati sono combinati per fornire una misura della qualità dello split stesso.

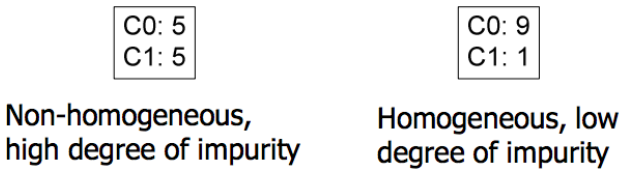

Figura 3.4: Due esempi di partizionamento [5]: l'attributo scelto nel primo caso porta ad uno split non omogeneo.

Le più comuni misure di impurezza di un nodo sono:

- **Gini Index**
- **Entropia**
- **Misclassification error**

In conclusione, la classificazione basata sugli alberi di decisione comporta diversi vantaggi:

- È poco costoso da costruire.
- È estremamente veloce a classificare dati sconosciuti.
- È facile da interpretare nel caso di alberi di dimensione ridotta.
- L'accuratezza è comparabile ad altre tecniche di classificazione nel caso di semplici data sets.

D'altro canto però questo tipo di classificazione è soggetta a:

- Problemi relativi ai valori mancanti, che riducono l'accuratezza.
- Sotto adattamento (quando il modello è troppo semplice e gli errori del training e test set sono grandi) o eccessivo adattamento (quando il modello generato è troppo specifico e vincolato al training set, finendo per non classificare correttamente il test set).

In particolare, **l'overfitting** (eccessivo adattamento) può essere conseguenza della presenza di rumore o di insufficienti records di training. Per gestire e ridurre questo problema vengono utilizzate due tecniche specifiche:

- **Pre-Pruning:** in cui l'algoritmo viene terminato prima che generi un albero completo. Ci possono essere diverse condizioni di terminazione (se il numero di istanze è inferiore ad una certa soglia, se l'espansione del nodo corrente non comporta alcun miglioramento nella misura d'impurità e così via).

- **Post-Pruning:** si sviluppa l'albero completo e poi vengono effettuati i tagli sui nodi meno accurati, in cui viene quindi inserito un nodo voglia con valore di classe uguale alla classe più frequente del sottoalbero che deve essere tagliato.

In conclusione, è possibile inoltre far utilizzo di classificatori che generano più alberi di decisione per effettuare la classificazione. Tra questi il **random forest** è uno dei più famosi ed usati.

3.2.2 Macchine a vettori di supporto (SVM)

Le macchine a vettori di supporto (più comunemente note come **Support Vector Machines**) sono dei modelli di apprendimento supervisionato usati per la classificazione binaria e per regressione. Nonostante tipicamente appartengano alla famiglia dei classificatori lineari, le SVMs possono anche efficientemente effetturare la classificazione non lineare facendo un mapping dei loro input in spazi dimensionali più grandi (con più feature).
A partire da un insieme di addestramento, in cui ogni elemento è classificato, un algoritmo di training per SVM costruisce un modello che si occuperà di assegnare i nuovi dati ad una classe o ad un'altra. Un modello SVM consiste in una rappresentazione degli elementi del set di addestramento (records o dati) come punti nello spazio, tracciati così che gli elementi di categorie separate siano chiaramente divisi tra loro da un *margine* il più largo possibile (come visibile dalla Figura 3.5). I nuovi elementi (la cui classe non è nota) vengono quindi tracciati nello stesso spazio generato precedentemente, e vengono assegnati alla classe relativa al lato del margine in cui essi cadono.
Come si può facilmente dedurre quindi, un elemento rappresentato come un punto è visto come un vettore di dimensione n e l'obiettivo è quello di determinare se sia possibile separare questi punti mediante un iperpiano di dimensione $n - 1$. La scelta più ottimale è quella di selezionare come iperpiano quello che permette la maggiore separazione possibile tra le due classi. Per far ciò viene quindi scelto l'iperpiano in maniera tale che la distanza con il punto più vicino di ciascuna delle due classi sia massimizzata (Figura 3.6).

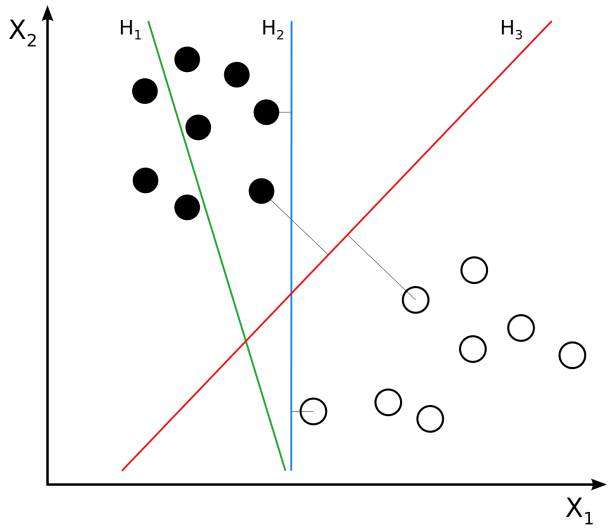

Figura 3.5: Possibili iperpiani (in questo caso rette) del SVM [7]: H1 non separa le classi. H2 ci riesce, ma solo con un piccolo margine. H3 le separa con il massimo margine.

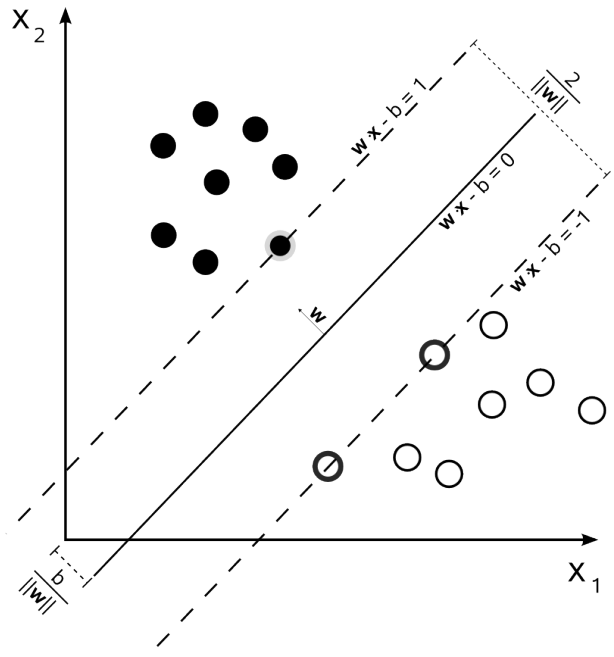

Figura 3.6: Esempio di funzionamento del SVM [6]: gli elementi sono tracciati nello spazio (in questo caso un piano), e una funzione si occupa di dividerlo in due zone cercando di massimizzare il gap tra le due categorie.

3.3 Reti neurali

Le **reti neurali** sono dei complessi sistemi di computazione che si ispirano alla struttura biologica del cervello. La loro caratteristica preponderante è che essi vengono addestrati a effettuare tasks (attività) tramite degli esempi, senza essere quindi programmati con delle regole specifiche per svolgere quel particolare task. In altre parole, non vengono programmate per eseguire una certa attività, ma addestrate (utilizzando un algoritmo di apprendimento automatico) mediante una serie di esempi della realtà da modellare. Le reti neurali possono essere usate in numerosi ambiti, come ad esempio nel **image recognition**, in cui esse vengono impiegate per identificare se le immagini contengono o meno degli elementi specifici (come la presenza di un "gatto"), dopo essere state precedentemente addestrate tramite altre immagini già manualmente catalogate (indicanti, in questo caso, la presenza o meno di un "gatto"). Il tutto avviene senza alcuna conoscenza a priori, bensì unicamente tramite l'apprendimento.
Da un punto di vista architetturale, la rete neurale è modello computazionale parallelo, costituito da numerose unità elaborative omogenee (dette nodi, o neuroni) fortemente interconnesse mediante collegamenti di varia intensità (che rappresentano le sinapsi del cervello biologico). L'attività della singola unità è semplice (funzione di trasferimento) e la potenza dei modello risiede nella configurazione delle connessioni (topologia e intensità). Partendo dalle unità di input, a cui vengono forniti i dati del problema da risolvere, la computazione si propaga in parallelo nella rete fino alle unità di output, che forniscono il risultato.
Le connessioni tra i neuroni sono tipicamente caratterizzate da pesi che vengono aggiustati durante la fase di apprendimento. Questi pesi hanno il compito di aumentare o diminuire la forza del segnale attraversante la connessione. I neuroni invece, sono tipicamente organizzati in vari livelli: in base al loro numero, possono cambiare le trasformazioni effettuate sull'input. I segnali viaggiano dal livello di input fino al livello di output, passando attraverso i livelli interni (detti di hidden).
L'approccio basato sulla rete neurale evita il collo di bottiglia dell'accesso alla memoria (come avviene nell'architettura tradizionale) ed è intrinsecamente parallelo: infatti integra memorizzazione (nelle connessioni) e computazione (in tante unità elaborative autonome), in analogia con l'intelligenza biologica.

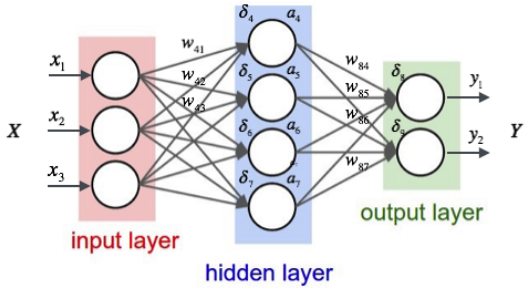

Figura 3.7: Esempio della struttura di una rete neurale [24].

Originariamente l'ambizioso obiettivo di questi sistemi era quello di risolvere i problemi nello stesso modo in cui il cervello umano avrebbe operato, ma col passare del tempo è stato lentamente abbandonato. Tuttavia, oggigiorno le reti neurali vengono usate per lo svolgimento di differenti tasks quali il riconoscimento dei segnali percettivi (voce ed immagini), diagnosi e gestione di apparati complessi in tempo reale, controllo dei movimenti di robot e veicoli autonomi, classificazione ed interpretazione di dati rumorosi, memoria associativa (accesso in tempo reale a grandi quantità di dati), ricostruzione di informazioni parziali o corrotte da rumore, soluzioni approssimate in tempo reale di problemi computazionalmente intrattabili, diagnostica medica e tanto altro.

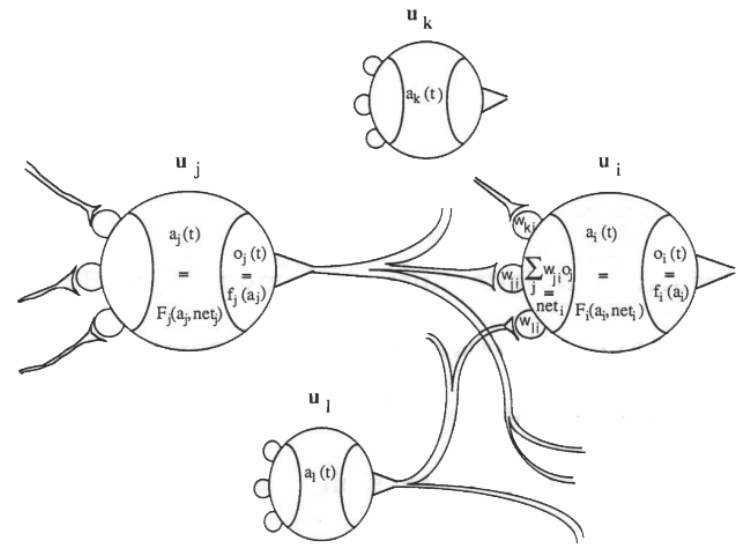

Figura 3.8: Componenti di una rete neurale [24].

Analizzando la lora struttura nel dettaglio, le reti neurali sono caratterizzate da:

- Un insieme di **unità elaborative**.
- Uno **stato di attivazione**, $a_j(t)$ nella Figura 3.8, dove j rappresenta la j-esima unità elaborativa.
- Una **funzione di output** f_i per ogni unità, $(f_j(a_j))$.
- Le **connessioni** fra le unità, ciascuna con un w_{ji}, indicante il peso sulla connessione dall'unità j-esima alla i-esima.
- Una **regola di propagazione** per propagare i valori di output attraverso la rete di connessioni
- Una **funzione di attivazione** F_i per combinare gli input con il valore di attivazione corrente e produrre un nuovo livello di attivazione. Ad esempio: $F_j(a_j, net_j)$ è la funzione di attivazione del nodo j dove net_j è l'input al nodo.
- Una **regola di apprendimento** per modificare le connessioni

Per quanto riguarda le **unità elaborative**, esse hanno una struttura semplice ed uniforme. Nello specifico, l'attività di una singola unità consiste nel ricevere un input da un insieme di unità (connesse in input) e calcolare un valore di output da inviare ad un altro insieme di unità (connesse in output). Come già detto, il sistema è strutturato in parallelo in quanto molte unità possono effettuare la loro computazione in modo concomitante. Generalmente vengono distinte tre tipologie di unità: di input, di output ed interne (hidden).
Come già ripetutamente affermato, le unità sono collegate tra loro tramite **connessioni**. In particolare, viene definita una matrice di connessioni W in cui ogni connessione w_{ji} è caratterizzata da una unità di partenza j, da una unità di arrivo i e da una forza della connessione che può essere:

- $w_{ji} > 0$: connessione eccitatoria.
- $w_{ji} < 0$: connessione inibitoria.

- $w_{ji} = 0$: connessione senza alcun contributo.

È importante specificare che è l'insieme delle connessioni a costituire la conoscenza dell'intera rete neurale e a determinare la sua risposta agli input. In altri termini, la memoria risiede nelle connessioni, le quali non sono programmate ma apprese automaticamente durante l'addestramento. Inoltre, più generalmente, vi possono essere diversi tipi di connessioni, quindi di conseguenza, diverse matrici di connessione W_a, W_b ecc..
Ritornando alle unità di elaborazione, ciascuna di esse è caratterizzata da un **valore di attivazione** al tempo t: $a_i(t)$ dove i indica l'unità u_i. Modelli diversi fanno assunzioni diverse circa i valori di attivazione che esse possono assumere: valori discreti, binari (attivo/inattivo), insieme ristretto o valori continui (limitati o illimitati).
Per quanto riguarda l'output delle unità elaborative, esse interagiscono trasmettendo segnali ai loro vicini (le unità connesse, ovvero il cui peso è diverso da 0). La forza dei loro segnali, e di conseguenza il grado di influenza esercitata sul vicini, dipende dal loro grado di attivazione. Ogni nodo u_i ha associata una funzione di output $f_i(a_i(t))$ che trasforma il valore corrente di attivazione $a_i(t)$ nel segnale in output $o_i(t)$:

$$o_i(t) = f_i(a_i(t))$$

È bene precisare però che in alcuni modelli il valore di output è uguale a quello di attivazione: f(x) = x. In altri invece, f corrisponde ad una funzione a soglia, in modo tale che un'unità non influenza le altre se la sua attivazione non supera un certo valore.
La modalità con cui invece gli output o(t) delle varie unità sono propagati attraverso le connessioni della rete per produrre i nuovi input, è detta **regola di propagazione**: se vi sono più tipi di connessioni diverse (eccitatorie, inibitorie, o nulle) la propagazione avviene indipendentemente per ogni tipo di connessione. Denominando con $net_{ai}(t)$ l'input di tipo "a" dell'unità i-esima, la regola spesso prevede semplicemente la somma degli input all'unità i-esima pesati rispetto all'intensità delle connessioni:

$$net_{ai} = \sum_j w_{aji} o_j$$

che in forma vettoriale diventa:

$$net_a = w_a o$$

Infine, la **funzione di attivazione** è una funzione, F, che prende il valore corrente di attivazione a(t) e il vettore net_k di input all'unità (dove k corrisponde al tipo di connessione) e produce un nuovo valore di attivazione:

$$a(t+1) = F(a(t), net_1(t), net_2(t), ...)$$

In genere F è una funzione a soglia, una funzione quasi-lineare (es. sigmoide) oppure una funzione stocastica.

3.4 Elaborazione dell'analisi lessicale

Dopo aver dato uno sguardo al funzionamento teorico dei classificatori e delle reti neurali, ai fini di una completa comprensione del progetto che verrà spiegato in seguito, è necessario introdurre un nuovo insieme di tecniche che si sta sviluppando sempre più velocemente che permette l'analisi del linguaggio naturale. Queste metodologie consentono l'estrazione automatica dal testo di informazioni utili e il suo significato. Ciò ha comportato la possibilità di ottenere automaticamente nuove informazioni che possano essere quindi sfruttate, nel nostro caso, nel campo del Data mining e del Machine Learning.
L'elaborazione dell'analisi lessicale (più comunemente nota come **Natural-Language processing, NLP**) è un'area dell'informatica e dell'intelligenza artificiale che si occupa delle interazioni tra i computers e il linguaggio naturale umano, e come i computers debbano essere programmati per poterlo processare ed analizzare. Il riconoscimento del parlato, la comprensione e generazione del linguaggio naturale sono alcune delle sfide che tipicamente coinvolgono il NLP.
Nonostante i numerosi passi avanti effettuati nel settore, l'intero processo è reso particolarmente complesso e difficile a causa degli aspetti intrinseci di ambiguità del linguaggio umano. Per questa ragione il processo di elaborazione viene spesso suddiviso in fasi diverse:

- **Analisi lessicale:** in cui avviene una scomposizione del testo e viene effettuato un processo in cui si prende in ingresso una sequenza di caratteri e si produce in uscita una sequenza di token (parole).

- **Analisi grammaticale:** effettuata su ogni parola nel testo.
- **Analisi sintattica:** organizzazione dei token in una struttura sintattica.
- **Analisi semantica:** assegnazione di un significato alla struttura sintattica e, quindi, all'espressione linguistica.

Volendo scendere più nel dettaglio, è bene fare una distinzione tra alcuni termini che vengono usati nel settore. La Linguistica Computazionale (**Computational Linguistics, CL**) è un'espressione spesso usata come sininonimo di NLP, sebbene in realtà la prima si focalizzi sulla teoria (scienza) mentre la seconda sulle applicazioni pratiche.
Nonostante l'enorme diffusione in questo campo del Machine Learning, il più tradizionale approccio al NLP è basato sulle regole linguistiche, formulate e implementate da linguisti o ingegneri della conoscenza.
I sistemi a regole e il machine learning hanno entrambi vantaggi e svantaggi. In generale, i primi sono ottimi per analisi linguistiche "dettagliate" come il parsing (analisi sintattica) approfondito, mentre il machine learning è molto efficace in casi di analisi "generali" come la classificazione automatica o il clustering. Volessimo paragonare il linguaggio a una città, e le frasi agli edifici, il machine learning è lo strumento adatto per vedere il panorama della città, mentre un sistema a regole ci consente di osservare ogni singolo palazzo.
In termini di qualità dei dati, i sistemi a regole sono una buona soluzione soprattutto per la *precision*, mentre il machine learning per la *recall*.

Passando all'analisi di quello che può essere effettuato con il NLP, si possono individuare quattro macrocategorie: l'analisi fatta sulla **sintassi** del linguaggio naturale, sulla sua **semantica**, sul **discorso**, o sul **linguaggio parlato**.
Per quanto riguarda l'analisi sintattica, i più famosi processi sono:

- **Lemmatizzazione:** è il processo algoritmico che determina il lemma di una parola basata sul suo reale significato. Al contrario dello stemming (che vedremo dopo), la lemmatizzazzione si basa sulla corretta identificazione della parte del parlato e del significato della parola nella frase, considerando non solo la singola frase, ma anche quelle vicine o addirittura l'intero documento.

- **Segmentazione Morfologica:** in cui avviene la separazione delle parole in singoli *morfemi* (la più piccola unità grammaticale del linguaggio). La difficoltà di questa attività dipende fortemente dalla complessità della morfologia (come ad esempio la struttura delle parole) della lingua che viene considerata. L'inglese, per esempio, ha una morfologia semplice e quindi risulta spesso possibile ignorare questa attività riuscendo a modellare tutte le possibili forme di una parola (ad esempio: eat, eats, ate, eaten, eating). In altre lingue, come l'italiano, ciò non è possibile perché ogni parola del dizionario può assumere un gran numero di forme diverse.

- **Etichettatura in categorie (Part-of-speech tagging):** a partire da una frase, si determina la cosidetta *part-of-speech* (ovvero la categoria che raggruppa le parole che hanno proprietà grammaticali simili) per ciascuna parola. Diverse parole, specie le più comuni, possono appartenere a più categorie. Ad esempio, "piano" può essere sia nome "Beethoven suona il piano", sia aggettivo "Vai più piano". Tipicamente, più la morfologia del linguaggio è semplice, più ambiguità possono sorgere.

- **Analisi (Parsing):** a partire da una specifica frase, determina uno dei tanti possibili alberi di analisi (parse tree, albero che rappresenta la struttura sintattica di una frase) che possono essere generati.

- **Scomposizione delle frasi:** a partire da un frammento di testo, si occupa di trovare i "confini" della frase. Spesso è sufficiente trovare i simboli di punteggiatura.

- **Stemming:** è il processo di riduzione delle parole nella loro forma base. Ad esempio in inglese "opening" viene ridotto a "open".

- **Segmentazione in parole:** si separano porzioni di testo in singole parole. In alcune lingue come l'inglese o l'italiano è veramente semplice perché sono solitamente separate da spazi. Al contrario, in lingue come il cinese e il giapponese, è necessaria una conoscenza del vocabolario e della morfologia delle parole.

- **Estrazione della terminologia:** in cui l'obiettivo è di estrarre automaticamente i termini rilevanti a partire da un testo.

I processi dell'**analisi semantica** invece sono:

- **Semantica lessicale:** si occupa di individuare il significato delle parole in un contesto, la semantica appunto.
- **Traduzione automatica (Machine translation):** permette la traduzione automatica di un testo da un linguaggio umano ad un altro. Questo è tipicamente uno dei compiti più difficili da implementare in quanto si richiede una conoscenza della grammatica, semantica e delle situazioni del mondo reale, in modo tale da tradurre correttamente.
- **Riconoscimento delle entità nominali (Named entity recognition, NER):** a partire da un flusso di testo, determina se una parola è un nome proprio o meno (come ad esempio nome di persona, di località, di un'organizzazione) ma può essere usata anche per l'identificazione di molte altre entità (Figura 3.9). È bene notare che non è sufficiente sfruttare le lettere maiuscole per effettuare una corretta identificazione. Infatti ad esempio, in molte lingue, le frasi iniziano sempre con la lettera maiuscola, e inoltre non tutte le parole di entità nominali ne fanno uso. Addirittura in altri casi, come le lingue orientali, non si fa minimamente uso del maiuscolo.
- **Generazione del linguaggio naturale (Natural language generation):** processo che si occupa della conversione dell'informazione da un linguaggio macchina al linguaggio naturale umano.
- **Comprensione del linguaggio naturale:** si occupa di convertire parti di testo in rapresentazioni più formali che risultano essere più semplici da manipolare da parte dei programmi informatici. In particolare, la comprensione del linguaggio naturale coinvolge l'identificazione della giusta semantica a partire da un insieme di possibili semantiche che possono essere derivate da una espressione del linguaggio naturale.

- **Riconoscimento ottico di caratteri (OCR):** determina il testo corrispondente a partire da un'immagine raffigurante del testo stampato.
- **Risposta a domande:** si occupa di determinare la risposta alle domande formulate attraverso il linguaggio umano. La difficoltà di questo processo dipende dal tipo di domanda, infatti possono essere formulate domande che possiedono una specifica risposta giusta (come "Quale è la capitale dell'Italia?") ma anche domande con più risposte possibili.
- **Riconoscimento delle implicazioni testuali:** Dati due frammenti di testo, si determina se vi è qualche implicazione tra i due (ad esempio se uno implica l'altro).
- **Estrazione di relazioni:** dato un frammento di testo, si identifica le relazioni tra le entità nominali (ad esempio chi è sposato con chi).
- **Sentiment analysis:** si occupa dell'estrazione di informazioni soggettive a partire da un set di documenti, spesso usando revisioni online per determinare la "polarità" di specifici oggetti. Tipicamente questo processo è usato per identificare i pareri dell'opinione pubblica, specialmente per scopi di marketing.
- **Segmentazione e riconoscimento di argomenti:** processo che si occupa della separazione di un frammento di testo in segmenti, ciascuno dei quali è collegato ad uno specifico argomento, che viene quindi identificato.
- **Disambiguazione del significato delle parole:** poiché molte parole hanno più di un significato, diventa necessario selezionare il significato che ha più senso nel contesto in cui si trova la parola.

Per quanto riguarda quelli sul **discorso** invece:

- **Riassunto automatico:** Si occupa della generazione di un riassunto di un frammento di testo. Spesso è usato per testi di cui è noto l'ambito.
- **Coreference resolution:** A partire da una frase o da un frammento di testo, si determina quali parole ("menzioni") si riferiscono allo stesso oggetto (entità).

Una delle più generali attività di questo processo include l'identificazione delle cosidette *bridging relationships* che coinvolgono espressioni di riferimento. Ad esempio, in una frase come "Lui entrò nella casa di Mario dalla porta principale", "la porta principale" è una espressione di riferimento e la "bridging relationship" da identificare è il fatto che la porta a cui si sta facendo riferimento è quella della casa di Mario.

- **Analisi discorsiva:** questo processo include un gran numero di attività diverse, tra le quali l'identificazione del discorso di un testo (ad esempio la natura del discorso e le relazioni tra le frasi) oppure il riconoscimento e la classificazione dei cosiddetti *speech acts*.

L'ultima macrocategoria riguarda i processi del **linguaggio parlato**:

- **Riconoscimento del linguaggio parlato:** a partire da una traccia audio in cui una o più persone parlano, si determina la rappresentazione testuale del parlato. Questo processo è l'opposto di quello che comporta la traduzione dal testo al parlato, ed è una delle attività più difficili. Infatti nel parlato naturale ci sono raramente delle pause tra parole successive, per questo motivo la segmentazione del parlato (vedere il processo successivo) è una delle attività necessarie per effettuare il riconoscimento. Inoltre in molte lingue parlate, i suoni possono essere anche ottenuti tramite la composizione di parole che si uniscono tra loro, per questo la conversione in singoli caratteri può risultare essere ancora più complicata.

- **Segmentazione del linguaggio parlato:** a partire da una traccia audio in cui una o più persone parlano, il discorso viene separato in parole.

- **Conversione da testo a parlato:** processo che si occupa della conversione di un testo scritto in parlato mediante l'utilizzo di suoni.

Tra tutte queste possibili applicazioni in cui può essere sfruttato il Natural Language Processing, per il progetto di tesi ci focalizzeremo sul processo di etichettatura (tagging) delle parole di un testo, così da effettuare un riconoscimento delle entità. Quello che come abbiamo già introdotto, prende il nome di **Named Entity Recognition**.

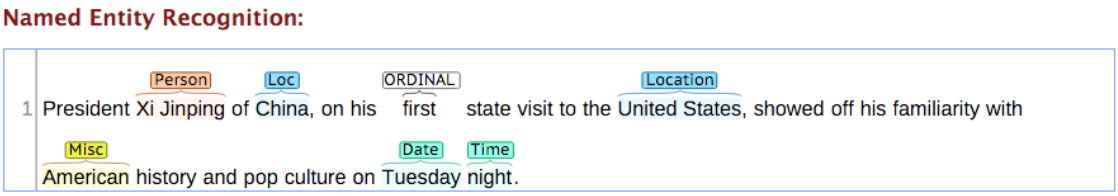

Figura 3.9: Esempio di funzionamento del processo di Named Entity Recognition [9].

Lo scopo infatti è quello di ottenere informazioni utili dal testo dei tweet che andremo ad analizzare, così da poter arricchire il numero di feature che il classificatore potrà utilizzare per il processo di classificazione di un tweet.

Capitolo 4

Strumenti, Librerie e Softwares

In questo capitolo verrà brevemente introdotta tutta la strumentazione usata nel corso della progettazione, per poter dare un quadro più pratico di quali tools e software sono stati effettivamente impiegati per i nostri scopi.
Tutta la fase di implementazione di codice ha previsto l'utilizzo di Python, linguaggio di programmazione ad alto livello, orientato agli oggetti, ed è considerato uno dei più diffusi nell'ambito del Data Science, insieme a R, grazie ai numerosi packages e funzionalità che essi mettono a disposizione.

4.1 Pandas: a python Data Analysis Library

Pandas [13](nome derivante da "Panel Data") è una libreria software scritta per Python, ed utilizzata per il trattamento ed analisi dei dati, mettendo a disposizione strutture dati e operazioni per la manipolazione di tabelle numeriche e time series. Tra le caratteristiche principali che sono fornite dalla libreria vi sono:

- DataFrame object per la manipolazione dei dati e indicizzazione integrata.
- Strumenti per la lettura e la scrittura di dati tra le strutture dati "in-memory" e i differenti formati di files.
- Allineamento dei dati e gestione integrata dei dati mancanti.
- Rimodellazione e pivoting dei data sets.

- Tagli basati sull'etichetta, indicizzazione efficente, ed estrazione di porzioni a partire da grandi data sets.
- Inserimento e cancellazione di colonne nelle strutture dati.
- Motore di Group By che permette le operazioni di taglio e le combinazioni sui data sets.
- Merging e Joining dei Data sets.
- Indicizzazione gerarchica per lavorare con dati ad alta dimensionalità in strutture dati a bassa dimensionalità.
- Funzionalità applicabili sui Time series, come la generazione dei dati, la conversione in frequenza, le statistiche su window mobile e data shifting.

Tra queste funzionalità elencate, le strutture dati più utilizzate nel corso della progettazione sono state quelle dei DataFrames, in quanto hanno permesso una semplice e performante manipolazione dei dataset (ottenuti dai file CSV).

4.2 Numpy

NumPy[14] è un'altra libreria per il linguaggio di programmazione Python, che aggiunge il supporto per grandi matrici e arrays multidimensionali, fornendo inoltre una grande collezione di funzioni matematiche di alto livello per poter operare su queste strutture. Tra le varie funzionalità vi sono:

- Potenti e performanti arrays N-dimensionali.
- Sofisticate funzionalità di broadcasting.
- Strumenti per l'integrazione con C/C++ e Fortran.
- Algebra lineare, trasformate di Fourier e capacità di generazione di numeri casuali.

Numpy è di fondamentale importanza per poter usare in maniera semplice ed intuitiva Pandas. I DataFrame infatti possono essere facilmente creati a partire da queste strutture.

4.3 Git e GitHub

Dal momento che l'intero progetto è basato sulla collaborazione di più persone, allo scopo di permettere la condivisione semplice ed efficente di tutto il materiale generato durante tutta la progettazione, abbiamo fatto uso dei ben noti servizi di Git e GitHub. **Git** [15] è un sistema di "controllo versione" usato per rilevare i cambiamenti nei files e per coordinare il lavoro tra più persone su questi stessi. Esso è principalmente usato per la gestione di codici sorgente durante lo sviluppo software, ma può essere usato per tener traccia di qualsiasi cambiamento di qualunque insieme di files. Essendo un sistema di controllo di revisione distribuito, come obiettivi ha quelli di garantire velocità, integrità dei dati e supporto per eventuali workflows non lineari.

Figura 4.1: Esempio di workflow generato usando Git.

GitHub[15] è un servizio di web-based hosting facente uso di Git per il "controllo versione". Esso, oltre ad aggiungere implementazioni proprie, offre tutte le funzionalità di Git - adattate per il contesto "remoto" - ed è quindi principalmente usato per la gestione di codici. Si occupa inoltre di fornire un controllo di accesso e funzionalità di collaborazione come il task management, bug tracking e feature requests per ogni progetto.

4.4 Twython

Twython, nel nostro caso la versione 3.6.0, è un Python wrapper (modulo software che ne "riveste" un altro) che permette l'utilizzo delle API di Twitter in maniera semplice

e compatta. Nel capitolo successivo verrà approfondito, in maniera dettagliata, il suo scopo e ruolo.

4.5 RapidMiner

RapidMiner è un software usato nell'ambito del Data Science che fornisce un ambiente integrato per il machine learning, il pre-processamento dei dati, deep learning, mining sui testi ed analisi predittive. È caratterizzato da una interfaccia grafica semplice ed intuitiva che permette la facile modellazione di tasks e processi volti all'analisi dei dati. Per questo motivo è usato sia per applicazioni commerciali e di business sia per fini di ricerca. Inoltre si occupa di fornire il supporto in tutte le fasi di un processo di machine learning, tra cui il preprocessing dei dati, la visualizzazione dei risultati e la validazione e l'ottimizzazione del modello.
Nel corso della progettazione, come vedremo nei capitoli successivi, abbiamo fatto uso di questo software per l'implementazione del classificatore, oltre che per il pre processing di alcune feature precedentemente estratte tramite Python.

4.6 Gensim

Gensim è un robusto strumento open source, implementato in Python, per realizzare e modellare uno spazio vettoriale, facente uso di NumPy, SciPy e opzionalmente Cython. Nello specifico, Gensim è stato designato per gestire grandi collezioni di testi, sfruttando algoritmi incrementali e quindi differenziandosi dagli altri pacchetti softwares più comuni. Esso include le implementazioni del TF-IDF, proiezioni casuali, Word2Vec, Doc2Vec e tante altre tecniche.
Nello specifico, nel nostro progetto di tesi, Gensim sarà fondamentale per poter trasformare i tweet in una rappresentazione vettoriale.

Capitolo 5

Acquisizione e preprocessamento dei tweet

5.1 I dati: acquisizione e caratteristiche

Come già precedentemente riportato nel capitolo introduttivo, la prima fase del progetto di tesi ha previsto la ricerca e l'acquisizione di dati che fossero coerenti ed inerenti al lavoro da svolgere. Lo scopo infatti è quello di utilizzare un algoritmo di machine learning per classificare i tweet e quindi filtrare (eliminare) le informazioni non rilevanti riguardo una situazione di crisi o emergenza. Per far ciò, l'intenzione è stata quella di utilizzare un approccio basato sull'apprendimento supervisionato che ha necessitato quindi dell'addestramento tramite l'impiego di un set di dati già classificato.
Dopo diverse ricerche sul web, siamo riusciti ad entrare in possesso di un insieme di dataset già etichettati, che riguardassero situazioni di crisi, fornito dalla repository di **CrisisLex** [10] per scopi di ricerca. In particolare, nel corso del nostro lavoro, abbiamo fatto utilizzo dei tweet relativi a crisi avvenute in località in cui si parlasse la lingua inglese, così da poter fare utilizzo del Natural Language Processing (vedere capitolo 3, sezione 3.4), che in altre lingue sfortunatamente non presenta tecniche così sviluppate. In particolare, i dataset ottenuti ed utilizzati per la sperimentazione sono stati riportati nella tabella seguente (Tabella 1).

Tabella 1: l'elenco dei dataset etichettati utilizzati

Crisi	Paese	Durata	Numero di tweet	Tipologia
2012 Colorado wildfire	USA	31 giorni	4,172	Naturale
2013 Queensland floods	Australia	19 giorni	1,223	Naturale
2013 Boston bombings	USA	60 giorni	1,001	Causa umana
2013 Alberta floods	Canada	25 giorni	5,887	Naturale
2013 Colorado floods	USA	21 giorni	1,778	Naturale
2013 Australia wildfires	Australia	21 giorni	1,982	Naturale
2013 Glasgow helicop. crash	UK	30 giorni	2,558	Causa umana
2013 LA Airport shootings	USA	12 giorni	2,730	Causa umana

Ciascuno di questi dataset è rappresentato sotto forma di file CSV, contenente diverse feature utili, oltre ovviamente al testo e all'etichetta già assegnata. In particolare per ogni tweet le informazioni riportate sono:

- **Tweet ID:** identificativo univoco del tweet che risulta essere fondamentale per poter accedere ad altri dati "nascosti" mediante l'impiego delle API di Twitter.
- **Tweet Text:** campo contenente il testo del tweet, che ogni utente scrive.
- **Information Source:** campo contente i dettagli su chi o cosa abbia generato quell'informazione (tweet). In altre parole, la fonte di informazione che ha portato alla scrittura del testo. Nello specifico, possono essere assunti i seguenti valori:
  - *Eyewitness:* il tweet è il risultato di una testimonianza oculare.
  - *Government:* l'informazione riportata nel tweet è frutto di qualche dichiarazione del governo.
  - *NGOs:* l'informazione è stata emessa da una organizzazione non governativa.
  - *Business:* il tweet è stato promulgato da qualche ente di business.
  - *Media:* l'informazione è stata emessa dai media.
  - *Outsiders:* tweet generato da qualsiasi altra fonte.
  - *Not applicable:* valore di default se il tweet non è stato etichettato, (vedere il campo Informativeness).

- **Information Type:** campo contenente i dettagli sul tipo di informazione contenuta nel tweet, e quindi la categoria. I valori che può assumere sono:
  - *Affected individuals:* tweet contente informazioni che riguardano individui, persone.
  - *Infrastructure and utilities:* tweet contente informazioni che riguardano infrastrutture e utenze.
  - *Donations and volunteering:* tweet contente informazioni che riguardano donazioni o manifestazioni di volontariato .
  - *Caution and advice:* tweet contente informazioni utili su come comportarsi e reagire.
  - *Sympathy and support:* tweet contente messaggi di supporto e di solidarietà.
  - *Other Useful Information:* tweet contente qualunque altro tipo di informazione utile.
  - *Not applicable:* valore di default se il tweet non è stato etichettato, (vedere il campo successivo).
- **Informativeness:** campo contenente dettagli sull'informatività rispetto a quella determinata situazione catastrofica. Può assumere i seguenti valori:
  - *Related and informative:* il tweet è relativo e rilevante per la specifica situazione di crisi.
  - *Related - but not informative::* il tweet è relativo alla crisi, ma non è rilevante in termini di utilità.
  - *Not related:* il tweet non è relativo alla crisi.
  - *Not applicable:* non è stata applicata alcuna etichettatura al tweet.

Tra le feature elencate, **Information Source** non sarà mai usata, in quanto non è una informazione che può essere raccolta in maniera automatica per i tweet futuri che dovranno essere classificati. Al contrario, vedremo che il valore di **Informativeness** sarà quello poi usato per poter effettuare il training del classificatore. Infatti, i tweet

con valore pari a "Related and informative" saranno quelli considerati rilevanti ai fini della crisi.
Alcuni esempi verosimili (semplificati per questioni di spazio) possono essere visti nella Tabella 2 riportata in seguito.

Tabella 2: esempi di riportanti le feature contenute nei CSV

Tweet ID	Tweet Text	Infor. Source	Information Type	Informativeness
2112326236	"It isn't .. "	Not labeled	Not labeled	Not related
2115574014	"RT @no.."	Media	Other Useful Info	Related and info
2119641982	"RT @Hu..	NGOs	Caution and advice	Related and info
2121970956	"@wide .."	Outsiders	Donations and Volun.	Related-not info

5.2 Estrazione dei metadati

5.2.1 Strategia e strumenti usati per l'estrazione

Come già affermato nella sezione precedente, il campo **Tweet ID** ha assunto un ruolo fondamentale per i nostri scopi, in quanto è stato necessario per poter accedere a tutte le informazioni relative (medatati) ad ogni tweet che sono presenti su Twitter. Per questa estrazione, abbiamo fatto utilizzo di **Python** linguaggio di programmazione ad alto livello, orientato agli oggetti, utilizzato anche per sviluppare applicazioni distribuite, computazione numerica, system testing e scripting. Grazie alla sua facilità di utilizzo, infatti, esso è uno dei linguaggi più sfruttati nell'ambito del Data Science, permettendo una rapida e intuitiva manipolazione dei dati. Per queste e altre ragioni (tra le quali la presenza di una comminity particolarmente attiva), sono stati sviluppati e resi disponibili numerosissimi packages che implementano utili funzionalità.
Tra questi packages vi è **Twython**[11] (nel nostro caso, la versione 3.6.0), un Python wrapper (modulo software che ne "riveste" un altro) che permette l'utilizzo delle **API di Twitter** in maniera semplice e compatta. Quest'ultime infatti, prevedono nativamente l'utilizzo del protocollo HTTP, che il wrapper invece esegue internamente.
Twython permette di effettuare query per ottenere informazioni sull'User, sulle liste di Twitter, sulle Timelines, sui messaggi diretti, e su qualsiasi altra funzionalità implemen-

tata dalle API. È importante sottolineare come Twitter lasci la possibilità di accedere alle sue informazioni solo facendo utilizzo delle proprie credenziali (registrando la propria applicazione sull'Application Management [12] di Twitter così da ottenere le keys e token necessari per utilizzare Twython), così da poter permettere alla piattaforma di monitorare le azioni e raccogliere i dati. Inoltre per gli utenti base, è stata stabilita una soglia massima di informazioni che possono essere scaricate ed estratte entro un certo intervallo di tempo. Per questo motivo abbiamo dovuto tenere conto degli eventuali timeout durante la scrittura del codice che si sarebbe dovuto occupare dell'utilizzo delle API.

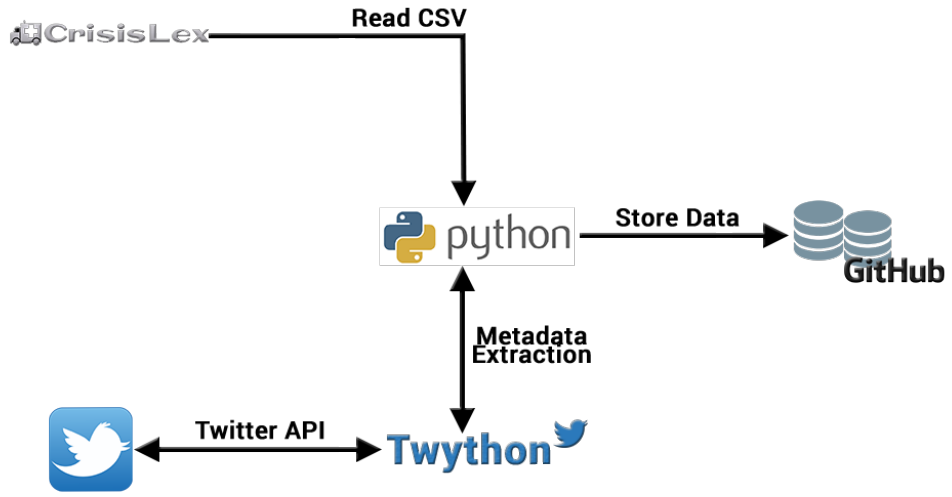

Figura 5.1: Schema raffigurante l'estrazione dei dati.

Nonostante tutte le dovute precauzioni, è stato comunque inevitabile perdere parte dei tweet durante l'estrazione, che nel corso del tempo sono stati cancellati, o resi privati dagli utenti.

5.2.2 Metadati estratti: caratteristiche e motivazioni

Consultando la documentazione fornita da Twitter, è possibile osservare come come sia possibile estrarre numerose informazioni, non solo sul tweet stesso, ma anche sull'utente che l'ha generato. Ovviamente, l'idea che non tutte potessero essere ritenute funzionali ed utili per i nostri scopi, ci ha spinto ad una analisi preventiva su quali metadati potessero essere utili per l'implementazione di un classificatore che dovesse riuscire a discriminare i tweet *rilevanti* da quelli *non rilevanti* in situazioni catastrofiche.
Il processo di scelta delle informazioni più adeguate, è partito dall'analisi dei risultati della ricerca *"A Language-agnostic Approach to Exact Informative tweet during Emergency Situations"* (già brevemente introdotto e spiegato nel Capitolo 2), per poi proseguire con l'aggiunta di ulteriori considerazioni e analisi proprie, frutto di un approccio **trial and error**, volto al raggiungimento dei migliori risultati possibili.
In questa sottosezione quindi verranno presentati tutti i metadati che sono stati selezionati come utili ai fini della nostra ricerca. Nello specifico, nella Tabella 3, è stato quindi riportato l'elenco completo di tutte le informazioni estratte, a partire dal Tweet ID, mediante le API di Twitter (richiamate tramite Twython, come spiegato nella sottosezione precedente). Queste feature potranno essere ottenute per qualsiasi nuovo tweet che dovrà essere poi successivamente classificato.
È opportuno specificare che, nel caso dei dataset già raccolti da CrisisLex ed usati come training e test set, queste informazioni si andranno ad aggiungere a quelle già possedute (che abbiamo elencato nella sezione 5.1): "Text", "Information Source", "nformation Type" e "Informativeness". Per questi tweet, ci si è inoltre occupati di verificare l'effettiva corrispondenza tra Tweet ID e testo, per essere sicuri di estrarre i corretti metadati per ciascun messaggio (aspetto non banale in quanto in altri dataset trovati sul web, ciò non è stato sempre garantito).

Tabella 3: elenco delle feature estratte (in aggiunta a quelle del tweet già presenti)

Feature	Description
Followers	*Numero di persone che seguono l'utente che ha scritto il tweet*
Followed	*Numero di persone seguite dall'l'utente che ha scritto il tweet*
GeoTagged	*Flag che indica se l'utente ha attivato la geolocalizzazione*
Totaltweet	*Numero totale dei tweet scritti dall'utente*
TwitterAge	*Età dell'utente, in giorni, rispetto al giorno di registrazione*
nHashTags	*Numero di hashtags contenuti all'interno del tweet*
nMentions	*Numero di menzioni effettuate all'interno del tweet*
nUrls	*Numero di urls presenti all'interno del tweet*
Verified	*flag che indica se l'utente è "verificato" o meno*
nRetweet	*Quante volte il tweet è stato retweettato*
nLikes	*Numero di likes che ha ricevuto il tweet*
Source	*Da dove è stato generato il tweet*
isRetweet	*flag che indica se il tweet è stato retweetato o meno*
CreationTime	*Data in cui è stato generato il tweet*
DeltaSeconds	*Attributo derivato: secondi passati dal primo tweet sull'evento*

In seguito sono state riportate le motivazioni per cui ciascuna di queste feature è stata scelta:

- **Followers** e **Followed:** indicano rispettivamente il numero di utenti che seguono e che sono seguiti dall'utente che ha generato il tweet. Sono entrambe informazioni utili che possono essere sfruttate dal classificatore per capire la rilevanza del tweet stesso. Ad esempio, un alto numero di Followers può lasciar dedurre che l'account in merito appartenga ad un ente noto ed importante, e che quindi il contenuto del tweet abbia più possibilità di essere un messaggio informativo. Inoltre, account con tanti followers tendono a generare anche un numero inferiore di tweet, rispetto alla media.

- **GeoTagged:** flag che indica se l'utente ha scelto di condividere la propria posizione. L'assunzione che può essere facilmente fatta, è che questo tipo di utenti risultano essere più affidabili rispetto agli altri. Nonostante ciò però, rischia di

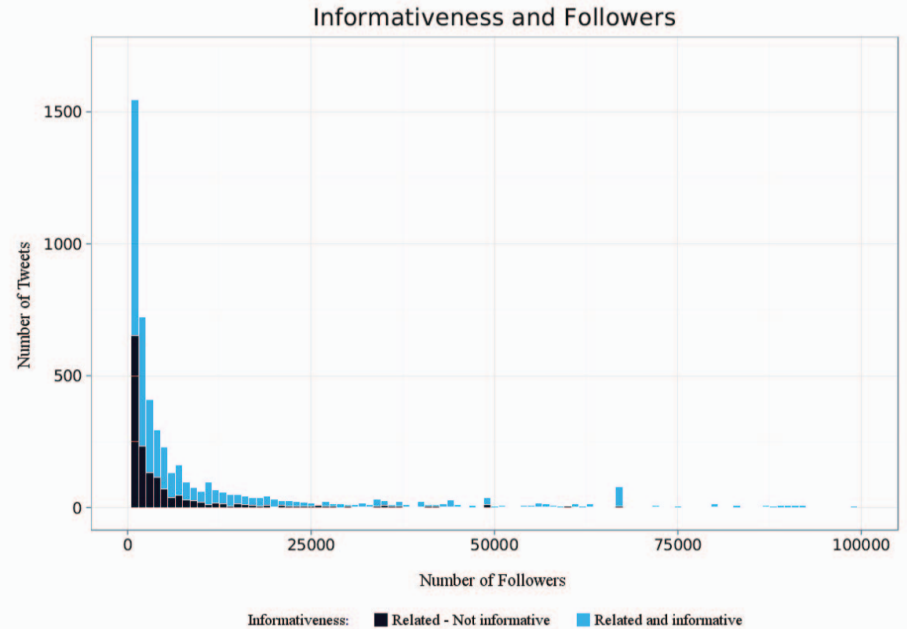

Figura 5.2: Come riportato dalle ricerche effettuate in altri lavori [3], il grafico mostra come all'aumentare dei followers, il numero di tweet diminuisce e la percentuale che essi siano rilevanti aumenta.

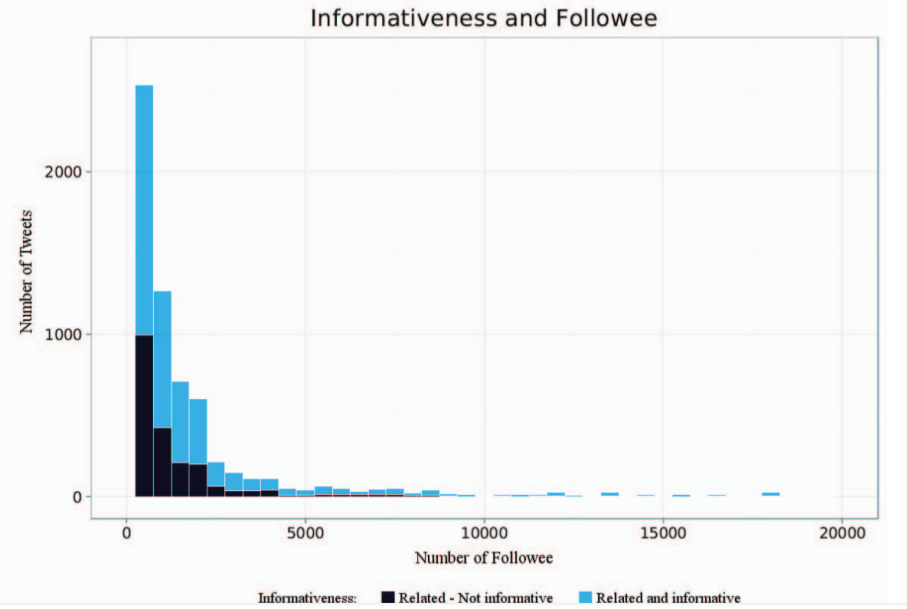

Figura 5.3: Il grafico mostra la percentuale di tweet rilevanti (informativi), al variare del numero dei followees. [3]

essere un attributo poco rilevante poiché la geolocalizzazione è attivata solo da una piccola percentuale degli utenti di Twitter.

- **Total tweet:** indica il numero di tweet totali scritti dall'utente a partire dal momento dell'iscrizione. È intuibile che, più alto è il valore, più alta sarà la reputazione dell'utente, che avrà quindi meno possibilità di essere uno spammer.

- **TwitterAge:** indica l'età dell'utente espressa in giorni a partire dal momento della sua registrazione. È un'altro attributo caratteristico in quanto account più datati possono essere considerati più affidabili rispetto ai nuovi che potrebbero essere stati creati per questioni di spam.

- **nHashTags**, **nUrls** e **nMentions:** indicano rispettivamente il numero di hashtags, di urls e di menzioni che sono contenuti all'interno del tweet. Un alto valore di tutti e tre questi parametri può rappresentare un cattivo segnale sull'importanza del messaggio contenuto. Infatti, gli spammers tendono tipicamente ad usare più hashtags relativi agli argomenti più in voga rispetto agli utenti normali, così da poter apparire più facilmente nei risultati delle ricerche Twitter, e quindi così da riuscire a guadagnare più attenzione. Inoltre, un'altro comportamento tipico, è quello di inserire nei tweet diversi link con lo scopo di attirare le persone su pagine esterne. Infine, un numero elevanto di menzioni, può anch'esso essere visto come un ulteriore modo da parte degli spammers di ottenere visibilità, specialmente delle persone note.

- **Verified:** è un attributo fornito da Twitter ad accounts speciali come governi, politici, media, giornalisti, o personaggi famosi. Se il campo assume valore di "True", significa che l'account è stato effettivamente autenticato e verificato. Come è facilmente intuibile, la percentuale di contenuto informativo è maggiore per gli utenti verificati.

- **nRetweet** e **isRetweet:** attributi che indicano rispettivamente il numero di retweet che ha subito il tweet stesso e semplicemente se è stato retweetato o meno (informazione che può essere facilmente dedotta dall'altra feature). È possibi-

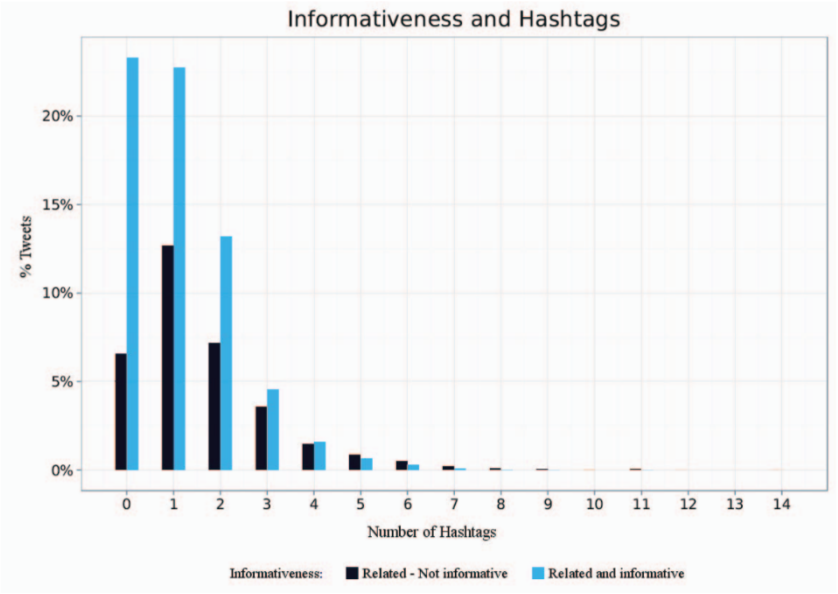

Figura 5.4: Come riportato dalle ricerche effettuate in altri lavori [3], il grafico mostra come effettivamente la rilevanza/informatività del tweet diminuisce con la crescita del numero di hashtags

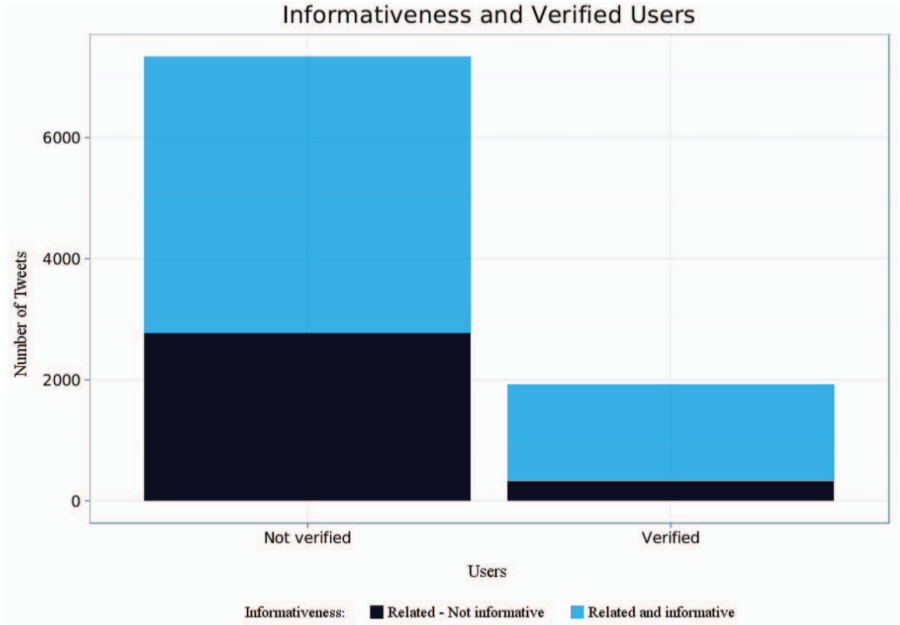

Figura 5.5: Il grafico mostra come effettivamente gli account verified riportano informazioni rilevanti [3]

le intuire che, messaggi molto retweettati possono rilevarsi essere informativi e rilevanti.

- **nLikes:** indica il numero dei likes che il tweet ha ricevuto. Questo attributo, combinato con gli altri, può dare informazioni utili. Ad esempio, è improbabile che un tweet scritto da uno spammer riceva apprezzamenti.
- **Source:** indica la sorgente da cui è stato generato il tweet. Da questo attributo si può ad esempio rilevare se il messaggio è stato scritto da un dispositivo mobile o meno. Visto il grande numero di sorgenti esistenti, per evitare che l'attributo diventi troppo dispersivo, abbiamo effettuato una selezione di quelli più importanti - come può essere osservato nella Tabella 4 - , facendo infine collassare tutti i rimanenti in un altra grande macro categoria ("OtherSite").

Tabella 4: elenco delle sorgenti principali

Sorgente	**Descrizione**
MobileApp	*Categoria che raccoglie i tweet generati da app per smartphones*
Client Web	*tweet generati tramite pc o laptop*
Tablet	*Categoria che raccoglie i tweet scritti mediante tablets*
Developer	*tweet generati dagli sviluppatori*
TweetDeck	*tweet generati da TweetDeck*
twitterfeed	*tweet generati da twitterfeed*
Mobile Web	*tweet generati dal sito mobile.twitter*
Facebook	*tweet generati tramite Facebook*
Instagram	*tweet generati tramite Instagram*
nLikes	*Numero di likes che ha ricevuto il tweet*
Tweetbot	*tweet generati da Tweetbot*
UnofficialApps	*Categoria che raccoglie i tweet generati da terze applicazioni*
OtherSite	*Categoria che raccoglie i tweet rimanenti*

- **DeltaSeconds:** attributo derivato sfruttando **CreationTime**, su cui sono state applicate diverse conversioni. DeltaSeconds infatti rappresenta la differenza,

espressa in secondi, tra il momento in cui è stato generato il tweet in analisi e il tweet più vecchio collegato alla crisi (nel caso nel training set quindi, il tweet con valore di CreationTime più vecchio, tra quelli *Related and Informative* e *Related - but not informative*). Questa variabile risulta essere particolarmente importante perché si è notato come i tweet rilevanti di una crisi sono generalmente generati entro le prime ore dall'evento, specialmente per inaspettati, imprevedibili e instantanei eventi come i terremoti.

Sfortunatamente, altre featurers che si sarebbero potute rilevare utili per i nostri scopi, non sono state estratte a causa dei limiti posti da Twitter stesso (che permette l'accesso solo agli accounts "Enterprise", a pagamento).

Capitolo 6

Estrazione di caratteristiche ricavabili dal testo

Come può essere facilmente intuibile, a prescindere dai metadati estraibili, è dal testo che possono essere dedotte informazioni utili per capire la rilevanza o meno del tweet. Per questo motivo, nel corso dell'attività, ci siamo occupati di cercare ed applicare tecniche che, a partire dal messaggio scritto, potessero estrarre informazioni utili usabili da un eventuale classificatore. Ovviamente, nel voler usufruire del testo, si è costretti inevitabilmente a specializzarsi (e quindi vincolarsi) su una specifica lingua, così da poter riuscire ad avere risultati significativi. Per questa ragione, l'assunzione fatta è che i tweet siano tutti scritti in lingua inglese.
Nel corso del capitolo quindi, verranno discusse le tecniche tentate ed usate per poter "arricchire" l'insieme di caratteristiche sfruttabili nella classificazione.

6.1 La funzione TF-IDF

La funzione di peso **TF-IDF** (term frequency–inverse document frequency) è una funzione generalmente impiegata nell'information retrieval per misurare l'importanza di una parola rispetto ad un documento o ad una collezione di documenti, e nel nostro caso, rispetto ai messaggi contenuti nei tweet. Il valore di tale funzione cresce in maniera proporzionale al numero di volte in cui la parola è contenuta nel documento,

ma aumenta in maniera inversamente proporzionale con la frequenza della parola nella collezione. L'idea di base di questa tecnica è di dare più importanza alle parole che compaiono nel documento, ma che in generale sono poco frequenti.

6.1.1 Formulazione matematica

La funzione di peso può essere vista come la combinazione di due fattori: il primo è il numero delle parole presenti nel documento. Tipicamente, per evitare che siano privilegiati i documenti più lunghi, questo valore viene diviso per la lunghezza del documento stesso.

$$TF_{j,k} = \frac{n_{j,k}}{|d_k|}$$

Dove $n_{j,k}$ è il numero di occorrenze della parola j nel documento k, mentre il denominatore $|d_j|$ è la dimensione, espressa in numero di parole, del documento k.

Il secondo fattore della funzione invece indica l'importanza generale del termine j nella collezione:

$$IDF_j = log\frac{|D|}{|\{d : j \in d\}|}$$

dove il denominatore è il numero di documenti che contengono il termine i, mentre $|D|$ è il numero di documenti nella collezione.
In conclusione quindi:

$$(TF - IDF)_{j,k} = TF_{j,k} \times IDF_j$$

6.1.2 Esempio numerico

Considerando un tweet contenente 10 parole e nel quale la parola *"casa"* compare 2 volte. Il fattore TF per la parola *"casa"* è $\frac{2}{10}$. Assendo di avere ora 100 tweet nella collezione e *"casa"* compare in 10 di questi. $IDF = log\frac{100}{10} = 1$. Da questo possiamo calcolare il valore TF-IDF relativo al documento iniziale: $TF - IDF = 0.5 \times 1 = 0.5$.

6.1.3 Utilizzo nella classificazione

Vista la sua semplicità di implementazione, si è pensato quindi di introdurre la funzione di TF-IDF così da poter ottenere informazioni utili a partire dal testo, nella speranza che il classificatore da implementare potesse riuscire a discriminare la rilevanza in base ai valori ottenuti.

Nonostante ciò, dopo i primissimi tentativi, ci si è resi conto dell'inefficacia di questo approccio. Ad essere selezionati dai primi prototipi di classificatori, sono stati i termini specifici di certe crisi o situazioni di emergenza. Un esempio che rende ben visibile questa problematica è quello riportato in Figura 6.1, in cui è stata utilizzata una classificazione basata su Decision Tree, sul data set dei tweet riguardanti l'alluvione avvenuta in Colorado nel 2013. Come si può evincere, nell'albero compare il termine "Colorado" che rende il classificatore vincolato a questo specifico cataclisma.

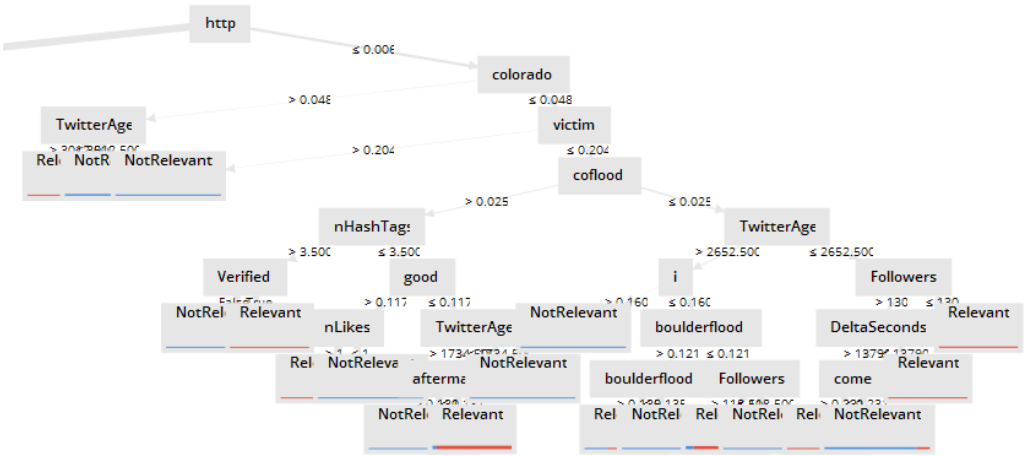

Figura 6.1: Sotto albero di un esempio di classificazione basata su Decisione Tree.

Per questo motivo, l'utilizzo del TF-IDF è stato abbandonato con l'intenzione di cercare una strategia più adatta ed "intelligente".

6.2 Natural Processing Language

Come già precedentemente affermato nel richiami teorici (Capitolo 3 della trattazione), nel corso degli ultimi decenni sono state sviluppate nuove tecniche che permettono l'analisi del linguaggio naturale, non soltanto da un punto di vista sintattico, bensì semantico (e non solo). Queste metodologie permettono l'estrazione automatica di utili informazioni sul testo stesso e il suo significato. Ciò ha comportato la possibilità di ottenere automaticamente nuove informazioni che possono essere quindi sfruttate, nel nostro caso, nel campo del Data mining e del Machine Learning. Tra queste vi è il **Natural Processing Language**.

Nel nostro scenario, l'idea è quella di sfruttare questa metodologia per riuscire ad ispezionare il contenuto del messaggio da un punto di vista semantico, individuando la presenza di alcuni concetti o entità (come ad esempio, la presenza di nomi propri, di località e molto altro). Nel concreto quindi, la strategia è quella di aggiungere dei flag ad ogni tweet - con valori binari: vero o falso - indicanti la presenza o meno di determinati "elementi" nel testo.

Il primo passo è stato quindi quello di trovare un software o package che implementasse all'interno un tagger che riuscisse ad etichettare le parole presenti nei testi. Dopo diverse ricerche ci siamo imbattuti nel ben noto **Stanford CoreNLP**, sotware scritto in Java, che fornisce un insieme di strumenti tecnologici per la comprensione del linguaggio naturale.

Ovviamente l'assunzione fatta è quella di dover trattare unicamente testi in lingua inglese (che è la più diffusa e per cui quindi le tecniche di NLP sono particolarmente performanti), visto che il NLP dipende fortemente dalla lingua utilizzata. Come nel caso delle API di Twitter, ci si è dovuti affidare ad un package di supporto per far uso di questi tools in Python: il **Natural Language Toolkit** [18]. **NLTK** è una piattaforma guida per la creazione di programmi scritti in Python in grado di manipolare il linguaggio umano. Tra le varie funzionalità, si occupa di fornire un modulo per interfacciarsi con i taggers dello Stanford.

6.2.1 CoreNLPNERTagger

Il **CoreNLPNERTagger** è un modulo che permette il riconoscimento delle cosiddette *named-entity* (in italiano noto come identificazione delle entità). Questo tagger è stato usato per l'analisi del testo di ciascun tweet. Le tipologie di entità riconosciute da questo modulo (sviluppato dallo Stanford) sono le seguenti:

- **Time:** etichetta assegnata a parole indicanti il concetto di tempo (ad esempio: 8 p.m).
- **Location:** etichetta assegnata a termini che indicano una località (ad esempio: Samar).
- **City:** etichetta assegnata a parole che indicano una città (ad esempio: Denver o Manila).
- **Country:** etichetta assegnata a parole che si riferiscono ad un paese (ad esempio: USA o Filippine).
- **State_Or_Province:** etichetta assegnata a parole, tipicamente abbreviate in sigle, che indicano uno stato o una provincia (ad esempio: AS o MT).
- **Nationality:** etichetta assegnata a parole che indicano una nazionalità (es: francese o filippino).
- **Title:** etichetta assegnata a parole che indicano un titolo (come sindaco o giornalista).
- **Criminal_Charge:** etichetta assegnata a parole che si riferiscono ad un capo di imputazione (ad esempio: bombardamento o treno).
- **Duration:** etichetta assegnata a parole che si riferiscono ad una durata (ad esempio: giorni o ore)
- **Ordinal:** etichetta assegnata a parole che indicano un numero ordinale (ad esempio: primo o ultimo).

- **Date:** etichetta assegnata a parole che si riferiscono ad una data (ad esempio: Domenica, domani, Dicembre).
- **Organizzazione:** etichetta assegnata a parole che si riferiscono ad una organizzazione (ad esempio: ONU, Google, NASA).
- **Cause_Of_Death:** etichetta assegnata a parole che esprimono una causa di decesso (ad esempio: indondazione, tempesta).
- **URL:** etichetta assegnata a parole che appartengono ad un url.
- **Money:** etichetta assegnata a parole che si riferiscono al denaro (ad esempio: $ o €).
- **Percent:** etichetta assegnata a parole che si riferiscono a percentuali.
- **Ideology:** etichetta assegnata a parole che si riferiscono ad una ideologia (ad esempio: separatista).
- **Number:** etichetta assegnata a numeri.
- **Religion:** etichetta assegnata a parole che si riferiscono ad una religione (ad esempio: Islam, Ebraismo).
- **Person:** etichetta assegnata a parole che si riferiscono a persone (ad esempio: David, Martha).
- **Misc:** etichetta che raccoglie un insieme disparato di termini.
- **Set:** etichetta assegnata a parole che si riferiscono ai giorni della settimana abbreviati (ad esempio: Wends, Sat in inglese).
- **O:** etichetta di default che raccoglie tutti i rimanenti termini.

Essendo il NLP una disciplina ancora in fase di sviluppo (che comunque ha mosso numerosi passi in avanti nel corso degli ultimi anni), la precisione ed efficenza del tagger stesso non sono ancora del tutto ottimali e esaustivamente veritieri.
Nonostante ciò, abbiamo comunque fatto uso di queste informazioni estratte a partire dal testo, per poter associare a ciascun tweet nuove caratteristiche da aggiungere alle

precedenti. In altri termini, sfruttando il tagger, abbiamo definito nuovi flag - assumenti valori di *vero* o *falso* - per ciascun tweet, così da aggiungere ulteriori informazioni che il classificatore potrà utilizzare per riuscire a discriminare la rilevanza e la non rilevanza del messaggio contenuto.
Come riportato in Tabella 5, ciascuno dei flag indica la presenza o meno di una determinata entità all'interno del tweet.

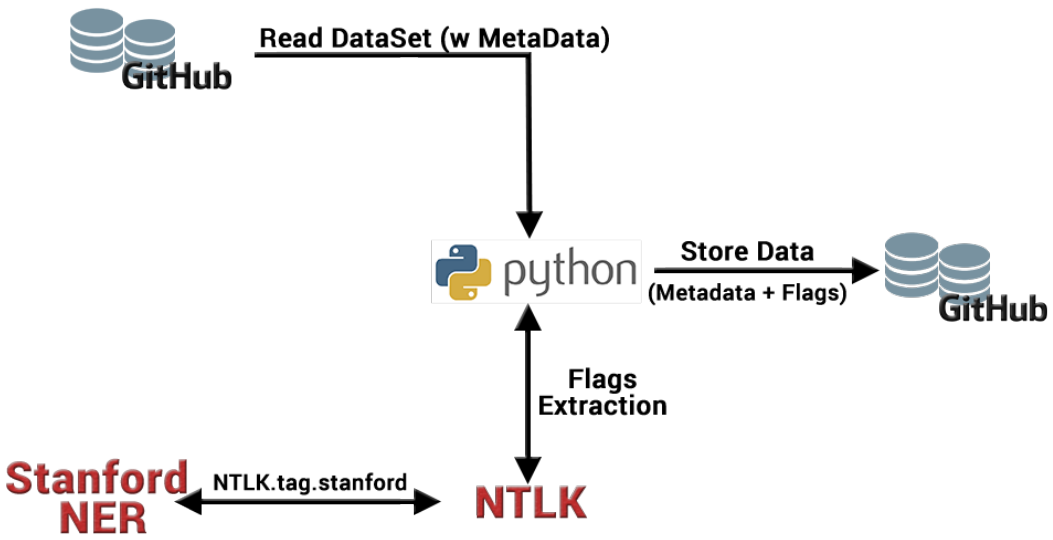

Figura 6.2: Schema mostrante il processo semplificato di estrazione dei flag.

Nello schema riportato (Figura 6.2) viene rappresentato il workflow del processo di estrazione dei flag facente uso di Python e del modulo fornito da **nltk** per accedere allo **Stanford NER**.

Tabella 5: elenco dei flag generati

Feature	Descrizione
hasLocation	*Se assume valore True, nel tweet è presente un termine etichettato come Location*
hasStateProvince	*Se assume valore True, nel tweet è presente un termine etichettato come State_Or_Province*
hasCountry	*Se assume valore True, nel tweet è presente un termine etichettato come Country*
hasCity	*Se assume valore True, nel tweet è presente un termine etichettato come City*
hasTime	*Se assume valore True, nel tweet è presente un termine etichettato come Time*
hasCriminal	*Se assume valore True, nel tweet è presente un termine etichettato come Criminal_Charge*
hasDuration	*Se assume valore True, nel tweet è presente un termine etichettato come Duration*
hasDate	*Se assume valore True, nel tweet è presente un termine etichettato come Date*
hasOrganization	*Se assume valore True, nel tweet è presente un termine etichettato come Organization*
hasDeath	Se assume valore True, nel tweet è presente un termine *etichettato come Cause_Of_Death*
hasMoney	*Se assume valore True, nel tweet è presente un termine etichettato come Money*
hasIdeology	*Se assume valore True, nel tweet è presente un termine etichettato come Ideology*
hasNumber	*Se assume valore True, nel tweet è presente un termine etichettato come Number*
hasReligion	*Se assume valore True, nel tweet è presente un termine etichettato come Religion*
hasPerson	*Se assume valore True, nel tweet è presente un termine etichettato come Person*
hasSet	*Se assume valore True, nel tweet è presente un termine etichettato come Set*

Anche in questo caso, per aggiungere ciascuna di queste nuove feature al dataset (già contenente dei metadati precedentemente estratti) si è fatto uso di Python.
Come vedremo in seguito, non tutti i flag aggiunti saranno poi effettivamente usati nel classificatore. Alcuni di essi infatti non risultano essere rilevanti ai fini del nostro obiettivo.

Capitolo 7

Classificazione basata su metadati e NLP flag

Nei capitoli precedenti abbiamo visto quali feature sono state estratte per ciascun tweet con l'obiettivo di ottenere informazioni utili da poter sfruttare per effettuare una corretta classificazione. In questo capitolo invece, ci si occuperà di spiegare come, a partire dai metadati estratti mediante le API, e dai flag generati sfruttando il Natural Language Processing, si sia definito un classificatore che sia in grado di discriminare i tweet rilevanti da quelli non rilevanti.

7.1 L'implementazione del classificatore mediante RapidMiner

Per poter implementare il classificatore, si è fatto uso del software RapidMiner (già introdotto nel Capitolo 4, dedicato agli strumenti usati nel progetto). Questo software, grazie alla sua semplice e performante interfaccia grafica, ci ha permesso di effettuare in maniera rapida, altre operazioni di preprocessing, senza dover fare uso di ulteriori programmi scritti in Python (che avrebbero ridotto la flessibilità che invece risulta essere necessaria per poter effettuare numerose prove e tentativi).
Come si evince dalla Figura 7.1, il primo passo è stato quello di creare un grande dataset non eccessivamente sbilanciato (evitando quindi di avere troppi tweet appartenenti ad

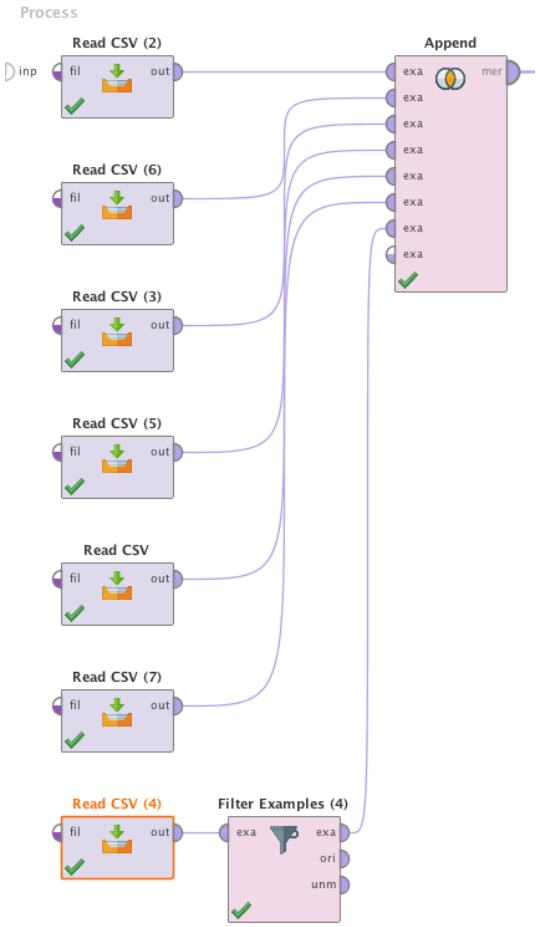

Figura 7.1: Parte I del processo di RapidMiner: unificazione dei dataset.

una sola delle categorie "rilevante" o "non rilevante") contenente tutti i metadati e flag estratti precedentemente. Per fare ciò sono stati uniti ben 7 dei 10 dataset originari (di cui uno opportunamente filtrato, estraendo solo i messaggi "non rilevanti", sempre per scopi di bilanciamento) così da raggiungere un insieme totale 4.480 tweet.
I successivi step invece hanno previsto:

- **Filtraggio dei tweet del Dataset:** per poter addestrare il classificatore, sono stati filtrati, quindi eliminati, tutti i tweet che non fossero relativi a qualche catastrofe, crisi o situazione di emergenza sulla base del valore presente nel campo *Informativeness*.

- **Generazione di attributi:** sono stati generati nuovi attributi binomiali a partire da feature con valori polinomiali. Si è infatti notato che questa sorta di "feature Subselection" abbia portato ad un miglioramento della classificazione stessa. Tra gli attributi generati, vi è anche la ClassLabel, ovvero l'etichetta di classe con valori "Relevant" e "Not Relevant" ottenuta a partire da Informativeness.
- **Selezione di attributi:** dopo un continuo processo trial-and-error, sono stati selezionati solo un sottoinsieme di feature che effettivamente sono risultate utili per la definizione del classificatore.
- **Definizione del ruolo:** l'attributo ClassLabel viene impostato come "etichetta" così da poter essere riconosciuto come tale dall'algoritmo di classificazione. Il valore di Tweet ID invece è impostato come identificativo (ID).

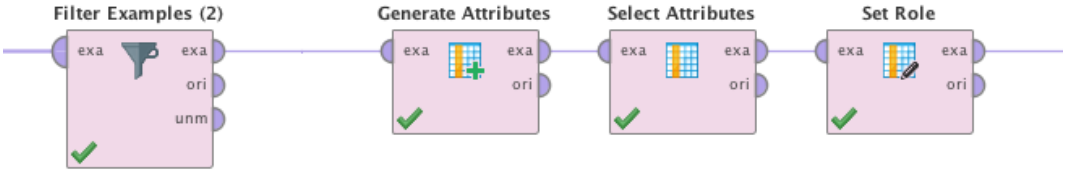

Figura 7.2: Parte II del processo di RapidMiner: preprocessing.

Nello specifico, la fase di generazione degli attributi ha previsto la creazione di feature a partire dai possibili valori assumibili dal campo *Source*, e il raggruppamento di tutti i flag NLP inerenti alla località (se il tweet contiene almeno una entità etichettata come Location, City, Country o State_Or_Province, la nuova feature assume valore di True, altrimenti False). Gli attributi generati sono visibili in Tabella 6.

Per quanto riguarda la selezione degli attributi, nella Figura 7.3 è stato riportato il filtraggio che è stato effettuato.

Tabella 6: Attributi generati

Nuovo Attributo	Descrizione
ClassLabel	*Generato a partire da Informativeness. Indica la classe*
HasPlace	*Attributo che ingloba tutti i flag sulla località*
MobileSource	*True se Source assume valore pari a MobileApp*
ClientSource	*True se Source assume valore pari a Client Web*
TabletSource	*True se Source assume valore pari a Tablet*
UnofficialSource	*True se Source assume valore pari a UnofficialApps*
MobWebSource	*True se Source assume valore pari a MobileWeb*
InstagramSource	*True se Source assume valore pari a Instagram*
FacebookSource	*True se Source assume valore pari a Facebook*
DeveloperSource	*True se Source assume valore pari a Developer*
TwitterDeckSource	*True se Source assume valore pari a TweetDeck*
TwitterFeedSource	*True se Source assume valore pari a twitterfeed*
TweetbotSource	*True se Source assume valore pari a Tweetbot*

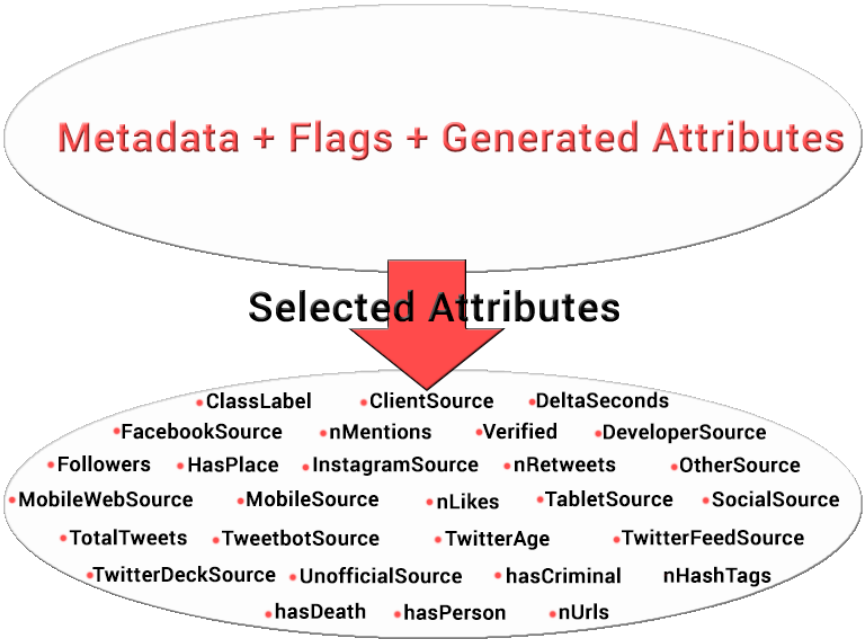

Figura 7.3: Parte II del processo di RapidMiner: selezione degli attributi.

7.1.1 Algoritmi di classificazione

Dopo aver effettuato tutte le operazioni di preprocessing sui dati, è stato finalmente possibile generare il modello utilizzabile per la classificazione facendo uso di uno tra i possibili algoritmi (che analizzeremo in seguito).
A prescindere dalla scelta effettuata però, un aspetto particolarmente importante che si è dovuto gestire è stato il problema dell'**overfitting**: uno dei rischi principali in cui si può incorrere durante la fase di addestramento di un classificatore. Conosciuto anche come *eccessivo adattamento*, l'overfitting è il fenomeno in cui un modello si adatta eccessivamente ai dati forniti per il training, funzionando quindi in maniera poco precisa con qualsiasi altro insieme di dati. Per questo motivo è importante gestire opportunamente questo rischio, evitando però di incorrere nel problema opposto, **l'underfitting**.
Numerose tecniche ed approcci sono stati definiti per poter gestire queste problematiche, tra le quali la **Cross Validation**. Essa è una tecnica di tipo statistico che viene utilizzata a partire da un training set di grandi dimensioni. Tra queste, in particolare, vi è la k-fold cross-validation che consiste nella suddivisione di tutto il dataset in k gruppi di uguale numerosità: ad ogni passo, il k-esimo gruppo del dataset viene usato come dataset per la validazione, mentre i gruppi restanti costituiscono il training dataset. In questo modo, si allena il modello per ognuna delle k parti, evitando quindi problemi sopracitati di overfitting, ma anche di campionamento asimmetrico (e quindi affetto da distorsione) del training dataset, tipico della suddivisione del dataset in due sole porzioni (ovvero di training e di validazione). In altri termini, nella k-fold cross validation si suddivide il training set (o campione) in parti di egual numero, si esclude iterativamente una parte alla volta e la si cerca di predire con le parti non escluse.
Nella scelta di come effettuare il campionamento (per la creazione dei folds), vari approcci possono essere utilizzati:

- **Campionamento stratificato:** in cui si divide il dataset in sottogruppi omogenei prima del campionamento. Ciascuno strato deve essere mutualmente e collettivamente esclusivo: ogni elemento del dataset deve essere assegnato ad un solo strato, e nessun elemento può essere escluso. Questo campionamento garantisce che la distribuzione della classe all'interno dei singoli sottogruppi, sia uguale a quella dell'intero dataset (ad esempio, nel caso della classificazione binomia-

le, vengono costruiti dei sottoinsiemi in cui ciascuno di essi contiene la stessa proporzione dei due valori assumibili dalla classe).

- **Campionamento lineare:** in cui si divide il dataset in partizioni senza cambiare l'ordine degli elementi.
- **Campionamento shuffled:** in cui si costruiscono sottoinsiemi casuali del dataset originario. Il tutto avviene in maniera totalmente casuale.
- **Automatico:** in cui viene applicato il campionamento stratificato per default. Se esso non risulta applicabile, viene usato quello shuffled.

Nel nostro caso è stato fatto uso di un **campionamento stratificato**, con un numero di folds pari a 10 (parametro fortemente influenzato dalla grandezza del dataset).

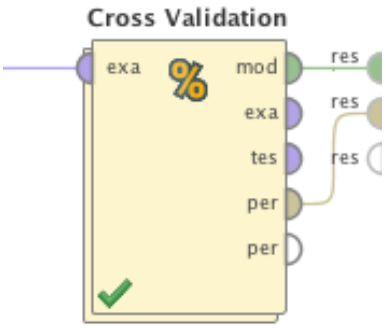

Figura 7.4: Parte III del processo di RapidMiner: Cross Validation.

Nella scelta del più appropriato algoritmo di classificazione invece, è stato tentato un approccio trial-and-error, in cui è stato selezionato quello con prestazioni migliori: il Decision Tree. Per determinare la valutazione, sono stati usati i seguenti parametri:

- **Accuratezza:** definita come il rapporto tra il numero di elementi correttamente classificati e il numero totale di elementi classificati.

- **Richiamo (r):** definito come il rapporto tra il numero di elementi correttamente assegnati ad una classe C, e il numero totale di elementi realmente appartenenti alla classe C.
- **Precisione (p):** definita come il rapporto tra il numero di elementi correttamente assegnati ad una classe C, e il numero i elementi assegnati alla classe C.

Dal punto di vista matematico:

$$\textbf{Accuracy} = \frac{True\ Positive + True\ Negative}{True\ Positive + False\ Positive + False\ Negative + True\ Negative}$$

$$\textbf{Recall} = \frac{True\ Positive\ for\ class\ C}{True\ Positive\ for\ class\ C + False\ Negative\ for\ class\ C}$$

$$\textbf{Precision} = \frac{TruePositive\ for\ class\ C}{True\ Positive\ for\ class\ C + False\ Positive\ for\ class\ C}$$

Seppur l'accuratezza sembra essere un parametro soddisfacente, se considerata da sola può risultare fuorviante in situazioni in cui c'è uno sbilanciamento su una delle due classi. Ad esempio se ci fossero 9900 tweet appartenenti alla classe "rilevante" e solo 100 appartenenti alla classe "non rilevante", se si classificasse ogni tweet come "rilevante", il classificatore avrebbe un'accuratezza del 99% seppur non rilevando alcun elemento della classe "non rilevante".
Per questo motivo sono stati considerati anche il richiamo e la precisione, che inoltre possono essere combinati nella cosiddetta **F-measure** definita come segue:

$$\textbf{F-measure} = \frac{2 \cdot Recall \cdot Precision}{Recall + Precision}$$

In seguito, nella Tabella 7, sono stati riportati i risultati di alcuni degli algoritmi tentati, con i corrispettivi parametri appena nominati: **Ri** e **P** sono le sigle usate per Richiamo e Precisione, mentre **R** e **NR** sono le classi Relevant e Not Relevant. Nonostante dai dati raccolti il classificatore bayesiano risulti avere performances migliori, ad essere scelto è stato il Decision Tree. Questo perché il Naive Bayes non può essere migliorato in quanto non presenta alcun parametro da modificare, risultando quindi già ottimale di

default. Al contrario il Decision Tree può essere fortemente *parametrizzato*, riuscendo a raggiungere performances più ottimali (la scelta dei valori dei parametri sarà affrontata nel paragrafo successivo).
È importante quindi sottolineare come i valori dell'albero di decisione in Tabella 7 non siano quelli finali, in quanto successivamente sono state effettuate una serie di ottimizzazioni sui parametri stessi dell'algoritmo.

Tabella 7: Confronto tra performances.

Algoritmo	**Accuratezza**	**Ri per R**	**Ri per NR**	**P per R**	**P per NR**
Decision Tree	67.59%	99.9%	0.62%	67.57%	75%
Random Forest	67.54%	99.97%	0.34%	67.52%	83.33%
Naive Bayes	68.57%	70.81%	63.92%	80.27%	51.38%
SVM (Ker. M.)	62.90%	53.87%	81.62%	85.86%	46.05%

Riassumento quindi, l'ultima fase del processo, ha così previsto l'utilizzo dell'algoritmo di classificazione più appropriato, facendo uso della Cross Validation per evitare i rischi di overfitting.

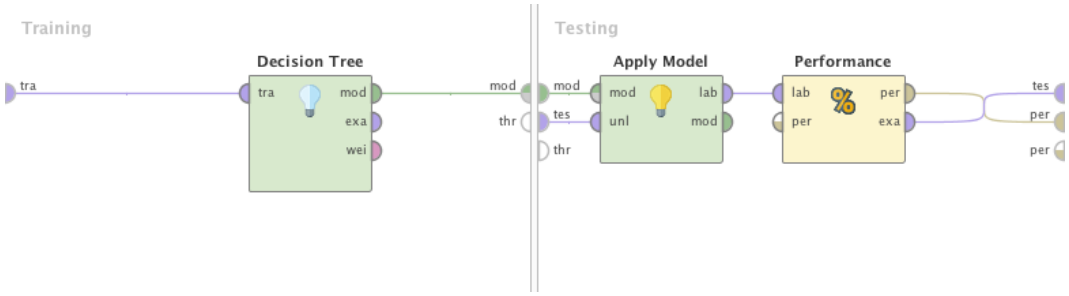

Figura 7.5: Parte III del processo di RapidMiner, interno della Cross Validation: generazione ed applicazione del modello.

7.1.2 Ottimizzazione dei parametri

Come precedentemente accennato, una fase fondamentale dell'implementazione del classificatore, è stata l'ottimizzazione dei parametri, fondamentale per ottenere, dagli algoritmi di classificazione, le migliori performances possibili.
Fortunamente, RapidMiner mette a disposizione un potentissimo strumento che seleziona i valori migliori assumibili dai parametri che si vogliono ottimizzare: l'**Optimizer Grid**.

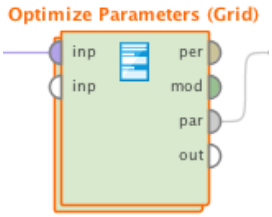

Figura 7.6: Optimizer Grid: all'interno vengono posti tutti i blocks che devono essere ottimizzati.

È da qui emerso quindi, che è il Decision Tree a raggiungere performances migliori, se impostato con i seguenti valori:

- **Criterion:** gini index (per la valutazione dell'impurità).
- **Maximal Depth:** 9 (profondità massima raggiungibile dall'albero).
- **Apply pruning:** false (non viene applicato alcun pruning).
- **Apply prepruning:** false (non viene applicato alcun prepruning).

7.2 Risultati

In questa sezione verranno presentati i risultati raggiunti facendo utilizzo del classificatore appena descritto.
Nella Figura 7.7 è stata riportata la **matrice di confusione** contenente i valori di accuratezza, richiamo e precisione raggiunti dal Decision Tree applicato tramite Cross Validation sul dataset precedentemente preprocessato, ottenuto dall'unione di 7 dataset

di input. Come si può osservare, l'accuratezza riesce a superare abbondantemente il 70%. Inoltre, aspetto non di poco conto, il richiamo è maggiore proprio sulla classe più "importante", ovvero la "Relevant": dovendo scegliere infatti, è più importante considerare un tweet come informativo, anche se non lo è realmente, piuttosto che il caso contrario.

accuracy: 73.28% +/- 1.19% (mikro: 73.28%)

	true Relevant	true NotRelevant	class precision
pred. Relevant	2607	782	76.93%
pred. NotRelevant	415	676	61.96%
class recall	86.27%	46.36%	

Figura 7.7: Matrice di confusione del Decision Tree, applicando la cross validation.

Per poter valutare con ancora più precisione la correttezza ed efficacia del classificatore ottenuto, esso è stato testato su ciascuno dei tre rimanenti dataset non utilizzati per il training. Come si può osservare dalle matrici di confusione riportate in seguito, il classificatore ottiene performances che oscillano da poco meno del 70% raggiungendo quasi il 90%. Questa variazione nei risultati è fortemente influenzata dalla distribuzione della classe nel dataset che deve essere predetto, visto che il modello è più propenso a classificare i tweet come "Relevant" piuttosto che "Not Relevant".

accuracy: 74.19%

	true NotRelevant	true Relevant	class precision
pred. NotRelevant	108	92	54.00%
pred. Relevant	114	484	80.94%
class recall	48.65%	84.03%	

Figura 7.8: Matrice di confusione del classificatore applicato per predirre i tweet dell'inondazione del 2013 in Queensland.

accuracy: 68.33%

	true NotRelevant	true Relevant	class precision
pred. NotRelevant	198	77	72.00%
pred. Relevant	133	255	65.72%
class recall	59.82%	76.81%	

Figura 7.9: Matrice di confusione del classificatore applicato per predirre i tweet dell'esplosione del 2013 in Texas.

accuracy: 86.37%

	true Relevant	true NotRelevant	class precision
pred. Relevant	561	32	94.60%
pred. NotRelevant	63	41	39.42%
class recall	89.90%	56.16%	

Figura 7.10: Matrice di confusione del classificatore applicato per predirre i tweet dell'incidente del 2013 a New York.

Per maggiore completezza, in seguito sono riportati i risultati parziali della classificazione facente utilizzo dei soli metadati o dei flag NLP, così da rendere ben evidenti i risultati intermedi, che poi sono stati effettivamente migliorati tramite l'utilizzo combinato dei due approcci.

accuracy: 67.01% +/- 2.03% (mikro: 67.01%)

	true Relevant	true NotRelevant	class precision
pred. Relevant	2956	1412	67.67%
pred. NotRelevant	66	46	41.07%
class recall	97.82%	3.16%	

Figura 7.11: Matrice di confusione del classificatore facente uso solo dei flag NLP.

accuracy: 72.77% +/- 1.90% (mikro: 72.77%)

	true Relevant	true NotRelevant	class precision
pred. Relevant	2565	763	77.07%
pred. NotRelevant	457	695	60.33%
class recall	84.88%	47.67%	

Figura 7.12: Matrice di confusione del classificatore facente uso solo dei metadati.

Capitolo 8

Ulteriore approccio: classificazione basata su Doc2Vec

Nelle sezioni precedenti è stata esaustivamente affrontata l'implementazione di un classificatore basato su metadati e flag NLP. In questo capitolo, invece, verrà introdotto un approccio completamente differente avente però sempre gli stessi fini di classificazione (discriminazione dei tweet rilevanti da quelli non rilevanti).

8.1 Classificazione basata su testo reso in forma vettoriale

L'idea pensata è stata quella di effettuare una classificazione basata unicamente sul testo del tweet. Come già precedentemente discusso però, il testo nella sua forma base, risulta essere troppo "grezzo" e quindi inutilizzabile.
Per questa ragione il messaggio sotto forma di "stringhe" viene tipicamente processato secondo varie strategie, per ottenere rappresentazioni più *potenti* e *compatte*: una di queste è la trasformazione in vettori.
Ovviamente, questa conversione vettoriale deve essere effettuata secondo una certa logica, cercando di mantenere, all'interno dei vettori, tutto il prezioso contenuto informativo che è espresso nel testo, riuscendo ad esempio a "codificare" l'informazione di

somiglianza e diversità tra i vari messaggi: **Doc2Vec** è uno tra i metodi più diffusi che riesce a garantire tutti questi aspetti.
In questa fase del progetto di tesi quindi, ci si è occupati di definire ed implementare un classificatore che operasse sui tweet opportunamente resi in forma vettoriale mediante il metodo Doc2Vec.

8.2 Word2Vec

Prima di inoltrarsi nella spiegazione di come effettivamente questo approccio sia stato utilizzato per gli scopi da noi prefissati, è necessario fornire un breve background teorico. **Doc2Vec** in realtà è un metodo derivante dal più noto **Word2Vec** [19] che consiste in un gruppo di modelli che sono usati per effettuare del **word embedding**, ovvero una traduzione di parole o frasi in vettori composti da numeri reali. Questi modelli non sono altro che delle reti neurali a due livelli che sono addestrate (tramite un approccio non supervisionato) per ricostruire i contesti linguistici delle parole. Word2Vec prende come input un grande frammento di testo e costruisce uno spazio vettoriale - tipicamente di diverse centinaia di dimensioni - in cui ogni parola è univocamente assegnata ad un corrispondente vettore nello spazio, seguendo un certo criterio. Infatti i vettori (rappresentanti le parole) vengono posizionati nello spazio cosicché le parole che risultino "simili" all'interno del frammento di testo, siano collocate vicine tra loro nello spazio stesso.
L'idea infatti è quella di riuscire ad incapsulare relazioni differenti tra le parole, come ad esempio sinonimi, contrari o analogie.
Word2Vec fa uso di un *trucco* tipicamente usato nel machine learning: addestrare una semplice rete neurale - con un singolo livello di nodi di hidden - ad effettuare un certo **task**, che però poi non sarà quello per cui verrà effettivamente usata la rete neurale. Un esempio noto è quello che riguarda la compressione di un vettore: si fa uso dell'apprendimento non supervisionato per addestrare la rete neurale a comprimere (nel livello di hidden) un vettore dato come input e a decomprimerlo nuovamente (nel livello di ouput) riottenendo l'originale. Dopo l'addestramento, il livello di output (ovvero lo step di decompressione) può essere scartato mantenendo solo i livelli di input e di hidden.

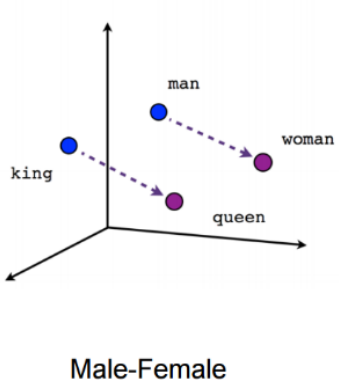

Figura 8.1: Esempio famoso [21]: king e queen è come man e woman.

Analogamente, nel caso del Word2Vec, il vero obiettivo sarà quello di imparare i pesi del livello di hidden, così da usarli come codifica di ogni singola parola (word vector). Nello specifico, esistono due metodologie per poter creare il modello che permette la codifica delle parole in vettori:

- **Skip-Gram model:** in cui si prendono coppie di parole dal testo (secondo un certo criterio che vedremo dopo) e si addestra una rete neurale con un livello di nodi di hidden sulla base del finto task in cui, a partire da una parola in input, la rete restituisce la distribuzione di probabilità delle parole vicine (all'interno del testo) all'input. In altre parole, la rete restituirà alte probabilità per le parole che tipicamente compaiono "vicine" alla parola in input. Essendo un finto task però, ad essere realmente importanti saranno i pesi tra i nodi di input e i nodi di hidden che saranno usati come word embedding. Quindi se lo strato di hidden possiede 300 neuroni, la rete restituirà un vettore di dimensione pari a 300, per ciscuna parola.

- **A continuous bag of words (CBOW):** anch'essa fa uso di una rete neurale con un livello di nodi di hidden. Il finto task in questo caso è basato sul predire una parola centrale, a partire da un insieme di parole di input (appartenenti allo stesso contesto). Anche qui, saranno poi i pesi tra il livello di input e di hidden ad essere usati come word embeddings per i termini dati in ingresso alla rete.

La seguente figura mostra come, a prescindere dall'approccio usato (e quindi dal fake task scelto), la codifica delle parole è rappresentata dalle righe della matrice dei pesi tra il nodo di input e di hidden. Nella sottosezione successiva analizzeremo bene il perché.

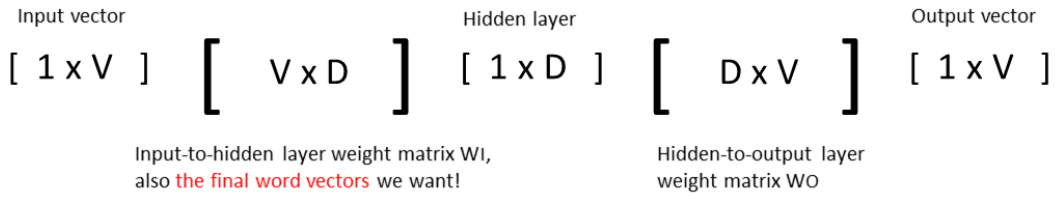

Figura 8.2: I prodotti che vengono effettuati all'interno della rete neurale.

8.2.1 Word2Vec: Skip Gram Model

Per poter permettere una migliore comprensione di come effettivamente funzioni il Word2Vec, in questa sottosezione verrà spiegato, nel dettaglio, come funziona l'approccio basato su Skip Gram [20] (che tipicamente restituisce risultati più accurati per grandi dataset rispetto al CBOW).

Il finto task che viene usato, in questo caso, per addestrare la rete, è il seguente: a partire da una specifica parola nel mezzo di una frase (detta parola di input), scegliendo casualmente una parola "nelle sue vicinanze", la rete dovrà restituire una probabilità per ciascuna parola del "vocabolario" (insieme di tutte le parole ottenute dai documenti di training) che essa sia effettivamente la parola "vicina" che abbiamo casualmente selezionato. Poiché il termine "vicina" è troppo generico, nella pratica si fa effettivamente uso di una **window size** per delimitare il concetto di vicinanza (ad esempio, con una window size pari a 5, si intendono le 5 parole prima e le 5 dopo rispetto alla parola in input). Quindi, le probabilità che verranno restituite in output, si riferiranno a quanto risulti plausibile trovare ciascuna parola del vocabolario nelle vicinanze della nostra parola di input. Ad esempio, in seguito all'addestramento, dando la parola "Sovietica", le probabilità di output saranno più alte per parole come "Unione" e "Russia" rispetto ad altri termini non correlati, come "pasta" o "koala".

Per addestrare la rete neurale a compiere questo task, si fa uso di coppie di parole ottenute dai documenti che vengono usati per il training. L'esempio sottostante mostra come verrebbero selezionate le coppie di parole a partire dalla frase "The quick brown fox jumps over the lazy dog", facendo uso di una window size di 2 elementi. La parola in blu è quella data come input.

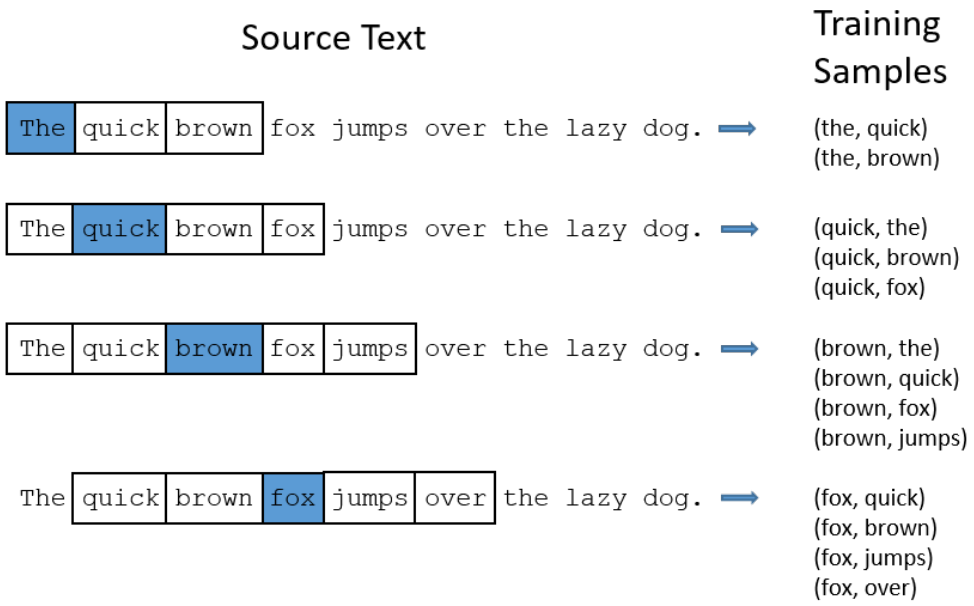

Figura 8.3: Esempio [20] di come a partire da un "testo", si generino le coppie di parole per il training, rispettando la window size prefissata.

La rete quindi verrà addestrata in base al numero di volte in cui ogni coppia viene usata. Quindi, per esempio, è probabile che la rete riceva più esempi di training di ("Sovietica","Unione") che di ("Sovietica","Canguro"). Una volta che la fase di training sarà conclusa, dando la parola "Sovietica" come input, l'output mostrerà una probabilità maggiore per "Unione" o "Russia" che per "Canguro".
Per semplicità, fino ad ora, per spiegare i concetti, si è fatto uso delle parole sotto forma di stringhe che, però, non possono essere usate dalla rete neurale. È quindi necessario un modo per poterle rappresentare: per questa ragione si costruisce un vocabolario di parole ottenute dai documenti usati come training (ad esempio 10.000 parole univoche). Un esempio di codifica per la rappresentazione delle parole è mediante un vettore **one-**

hot (gruppo di bits tutti posti a 0 tranne uno). Quindi, nel caso di 10.000 parole univoche, il vettore one-hot avrà 10.000 componenti: per ogni parola del vocabolario, ci sarà un solo bit a 1.

L'output della rete invece, è costituito da un singolo vettore (sempre con 10.000 componenti) contenente, per ogni termine del nostro vocabolario, la probabilità che esso sia la parola vicina a quella fornita come input. Nella Figura 8.4 viene riportata quindi la struttura della rete neurale: come input viene fornito un vettore one-hot codificante la parola "ants".

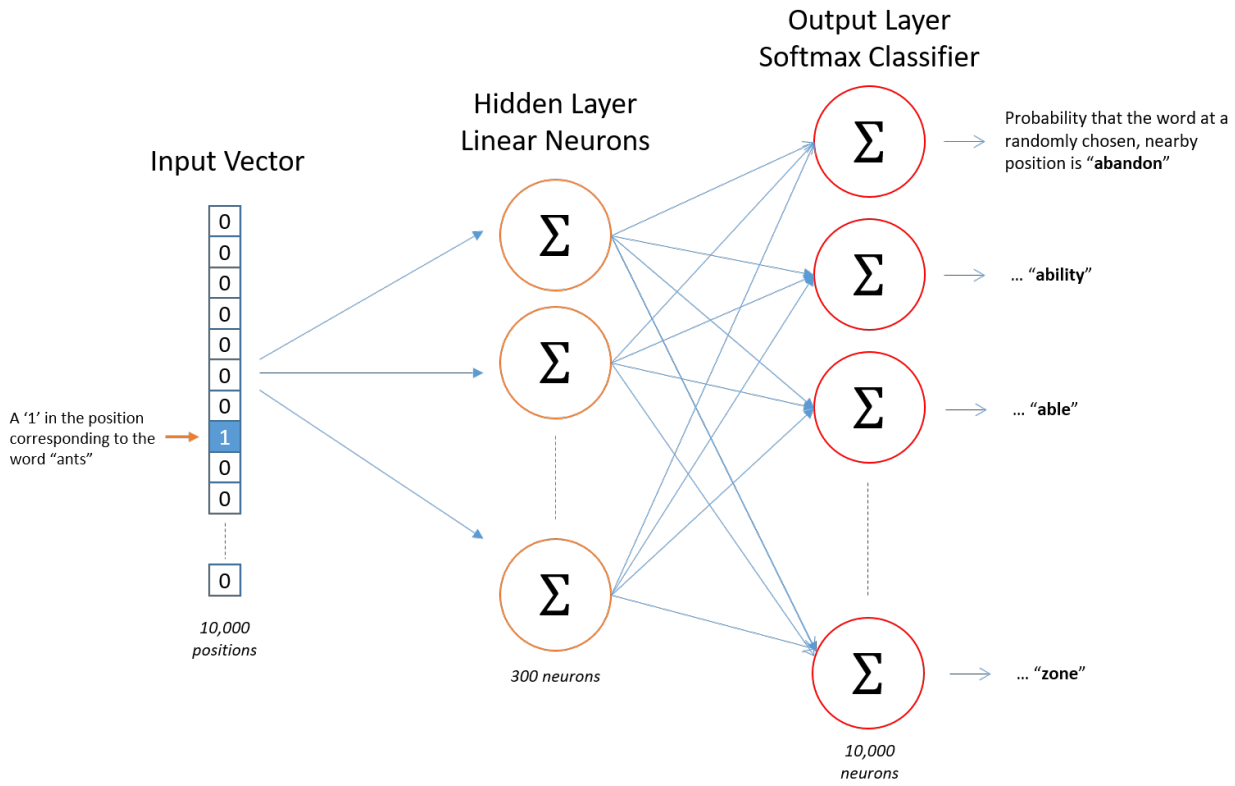

Figura 8.4: Rappresentazione della rete neurale [20].

Nello specifico, i nodi di output fanno uso della funzione di softmax mentre nei neuroni del livello di hidden non viene usata alcuna funzione di attivazione. Nella fase di addestramento della rete con le coppie di parole, l'input è un vettore di one-hot rappresentante la parola di input, e l'output di training è anch'esso un vettore di one-hot rappresentante la parola di output. Finito il training però, fornendo una parola come

input, il vettore di output sarà invece la distribuzione di probabilità di tutti i termini del vocabolario (quindi sarà caratterizzato da valori reali, non da vettori one-hot)
Un aspetto importante da considerare durante l'implementazione della rete neurale, è il numero di nodi di hidden (tipicamente qualche centinaia), che rappresenterà poi l'effettiva "lunghezza" del word embedding (i vettori con cui saranno codificate le parole). Prendendo come esempio Google, che fa uso di 300 neuroni, il livello di hidden sarà di conseguenza rappresentato da una matrice dei pesi con 10.000 righe (una per ogni parola del vocabolario) e 300 colonne (per ogni nodo di hidden). Saranno proprio le righe di questa matrice ad essere i word vectors cercati (ovvero le rappresentazioni vettoriali delle parole, Figura 8.5).

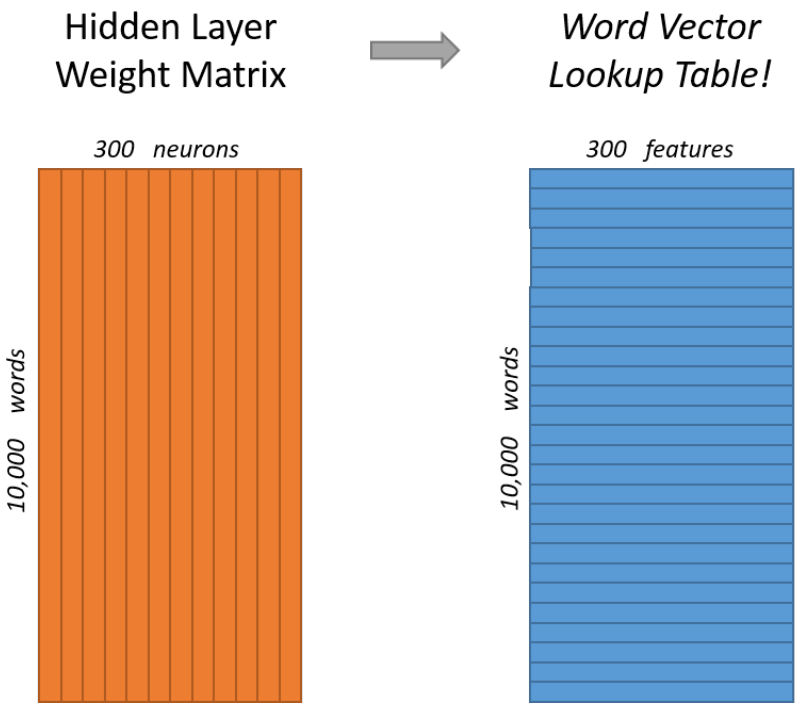

Figura 8.5: Matrice dei pesi del livello di hidden [20].

Il vero obiettivo finale di tutto questo processo, quindi, è in realtà quello di *ricavare* ("imparare") la matrice dei pesi del livello di hidden. Perciò il livello di output potrà essere scartato una volta ottenuta la matrice.
Analizzando nel dettaglio il funzionamento della rete, si può notare come la moltipli-

cazione tra un vettore con tutti i valori a 0 (tranne un termine solo) e una matrice, porterà all'effettiva selezione di una sola riga della matrice dei pesi, corrispondente all'unico valore di "1" del vettore (1 x 10,000 vettore one-hot per 10,000 x 300 matrice dei pesi).

$$[0 \quad 0 \quad 0 \quad 1 \quad 0] \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ 10 & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = [10 \quad 12 \quad 19]$$

Figura 8.6: Prodotto tra vettore e matrice: avendo solo un bit a 1, si estrae una sola riga della matrice [20].

In altre parole, si può facilmente dedurre come il livello di hidden di questo modello (rete neurale) in realtà operi semplicemente come una **tabella di lookup**: l'output del livello di hidden è semplicemente il word vector (la rappresentazione vettoriale) della parola data in input.

Nonostante il livello di output non sia rilevante rispetto agli obiettivi prefissati dalla rete, è bene riportare il suo funzionamento, per capire meglio da un punto teorico i risultati ottenuti. Riprendendo l'esempio precedente, dopo aver attraversato i nodi di hidden, in ingresso al livello di output verrà fornito il word vector (rappresentazione vettoriale) corrispondente alla parola "ants" (data in input alla rete neurale tramite la sua codifica one-hot). Come precedentemente affermato, nel livello di output viene fatto uso della funzione di softmax, ed ogni neurone di ouput (uno per ciascuna parola del vocabolario) produrrà un output tra 0 e 1, rappresentante la probabilità che quel termine sia la parola vicina all'input. Ovviamente la somma di tutti questi output sarà pari 1. Nello specifico, ogni neurone di output ha un vettore dei pesi che viene moltiplicato con il word vector ottenuto dal livello di hidden. Al risultato viene successivamente applicata la funzione exp(x). Infine, per avere la somma degli output pari ad 1, si divide il risultato per la somma di tutti i risultati di tutti e 10.000 nodi di output. In seguito (Figura 8.4) vi è l'illustrazione del calcolo risultato del nodo di output corrispondente alla parola "car".

È importante notare che la rete neurale non sa nulla dell'offset tra la parola di output

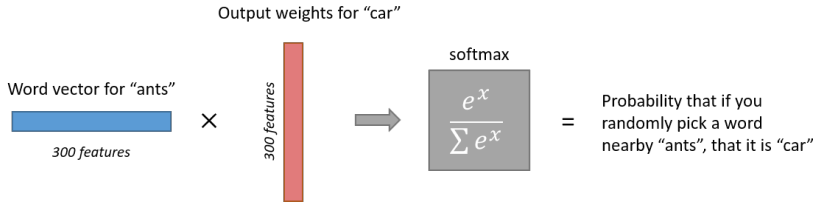

Figura 8.7: Prodotto tra il word vector rappresentante la parola "ants" e il vettore dei pesi della parola "car" [20].

e la relativa parola di input. In altri termini, tutte le parole all'interno della window size considerata, vengono trattate alla stessa maniera rispetto alla parola di input. Quindi non vengono imparate differenti insiemi di probabilità in base alla lontananza dall'input. Per comprendere l'implicazione, si osservi il seguente esempio: si ipotizzi che nei documenti usati per l'addestramento, ogni singola occorrenza della parola "York" sia preceduta dalla parola "New". Da ciò si potrebbe intuire che ci sia il 100% di probabilità che "New" sia vicina a "York". Invece, se si considerassero le dieci parole nelle vicinanze di "York" e si pescasse casualmente una di esse, la probabilità che essa sia "New" non sarebbe affatto 100%, poiché si potrebbe pescare un'altra delle parole nelle vicinanze.

Infine, in conclusione, l'intuizione che può essere fatta è che se due parole appartengono a "contesti" molto simili (ovvero che sono entrambe circondate dalle stesse parole), allora il nostro modello restituirà risultati analoghi per entrambi i termini. Per restituire output somiglianti, i word vectors dovranno essere necessariamente simili. Per questo motivo, se due parole appartengono a contesti analoghi, allora la rete dovrà imparare simili word vectors per questi due termini. Quindi, sinonimi come "astuto" o "furbo", avranno contesti similari, così come termini che sono collegati tra loro (ad esempio "motore" e "trasmissione"). Inoltre, la rete sarà in grado di gestire anche lo stemming, poiché parole come "dog" e "dogs" apparterranno a contesti analoghi.

8.2.2 Word2Vec: CBOW

Come già è stato accennato, oltre all'approccio Skip-Gram, esiste una tecnica simile ma concettualmente opposta che prende il nome di **Continuous bag of words (CBOW)** che verrà spiegata più rapidamente, in quanto si basa sui concetti analoghi. Il finto

task utilizzato in questo approccio è quello di addestrare la rete neurale a determinare la parola centrale a partire da un determinato contesto (insieme di parole). Anche in questo caso, ad essere effettivamente usata sarà poi la matrice dei pesi tra il livello di input e di hidden, rappresentante le parole codificate (word vectors).

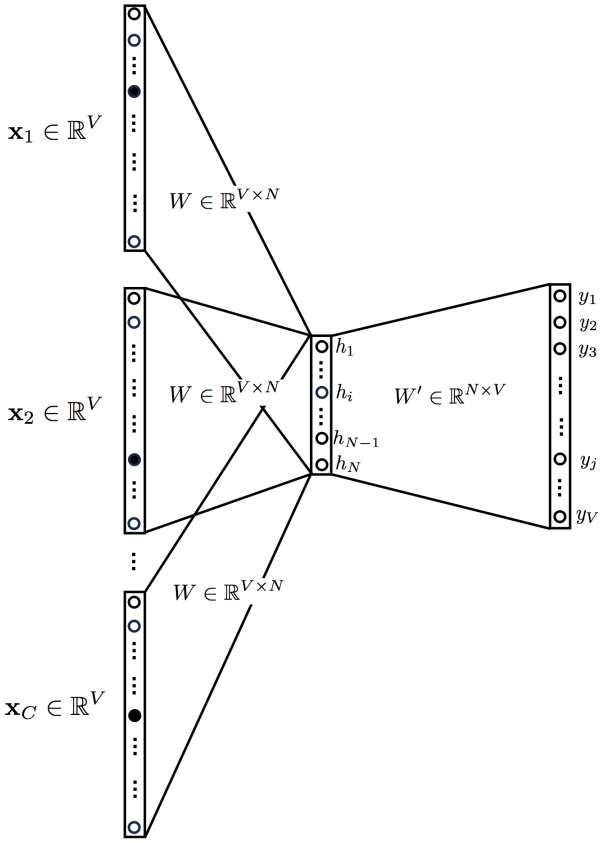

Figura 8.8: Rete neurale [20] nell'approccio CBOW.

La Figura 8.8 riporta la struttura della rete neurale, molto simile a quella dell'approccio skip-gram. È bene però sottolineare, come nella letteratura, vengano raffigurati più vettori in ingresso alla rete, per indicare come vengano effettivamente date più parole come input della rete neurale. Poiché ciascun termine è codificato come un vettore one-hot, nella pratica il vettore di ingresso sarà solo uno avente per ciascuna parola codificata, il corrispondente valore ad 1 (ad esempio [010101] corrispondente alle 3 parole codificate come [010000], [000100] e [000001]). Anche per il modello skip-gram, si fa uso di una rappresentazione simile, in cui come output si hanno un insieme di

vettori one-hot, ciascuno corrispondente ad una delle parole "vicine" al termine dato come input (quindi appartenenti allo stesso "contesto"). Questi vettori infatti possono essere facilmente ricavati, prendendo i C termini di output che possiedono la probabilità più alta (ad esempio: con vettore [0.9 0.1 0.35 0.14 0.7] e con C = 2, si possono ricavare [1 0 0 0 0] e [0 0 0 0 1], corrispondenti a 2 parole codificate).
Tornando alla figura precedente, come si può facilmente osservare, il livello di input è quindi caratterizzato da vettori one-hot $\{x_1...x_C\}$ codificanti parole (appartenenti allo stesso "contesto") all'interno della **window size C** e ad un vocabolario di **dimensione V**. Il livello di hidden invece è un vettore N-dimensionale identificato come **h**. Per quanto riguarda il livello di ouput, esso riporta la probabilità per ciascuna parola y_i del vocabolario, di essere il termine centrale rispetto al "contesto" dato come input: prendendo il nodo y_m con probabilità maggiore, il vettore one-hot corrispondente sarà quello con tutti 0 e con valore 1 nell'm-esima componente. Infine W e W' rappresentano le matrici dei pesi: terminato l'addestramento, le righe di **W** saranno i word vectors corrispondenti a ciascuna parola del vocabolario.

8.3 Doc2Vec: principi analoghi

In maniera analoga agli scopi del Word2Vec, l'obiettivo del Doc2Vec è di creare una rappresentazione numerica di un intero documento (o paragrafo, nel nostro caso di un tweet) a prescindere dalla sua lunghezza. Il principio usato è semplice ed intelligente: si fa uso del modello word2vec e si aggiunge un altro vettore, detto **Paragraph ID**. Quindi dopo aver addestrato la rete neurale, si avranno oltre che i word vectors (la rappresentazione vettoriale delle parole) anche un document vector (la rappresentazione vettoriale del documento).
Anche nel caso del Doc2Vec esistono due metodologie analoghe rispettivamente al CBOW e lo Skip-Grim:

- **Distributed Memory version of Paragraph Vector (PV-DM):** che agisce come una memoria che ricorda cosa manca esattamente all'interno del contesto in questione o come l'argomento del paragrafo. Mentre i word vectors rappresentano

il concetto di una parola, il document vector intende rappresentare il concetto di un documento.

- **Distributed Bag of Words version of Paragraph Vector (PV-DBOW):** che risulta essere più veloce e che consuma meno memoria (contrariamente a quanto accade allo skip-gram nel word2vec) in quanto non c'è alcun bisogno di salvarsi i word vectors (rappresentazioni vettoriali delle parole). Dopo l'addestramento infatti, basta fornire il paragraph ID (detto anche tag) per ricevere il document vector.

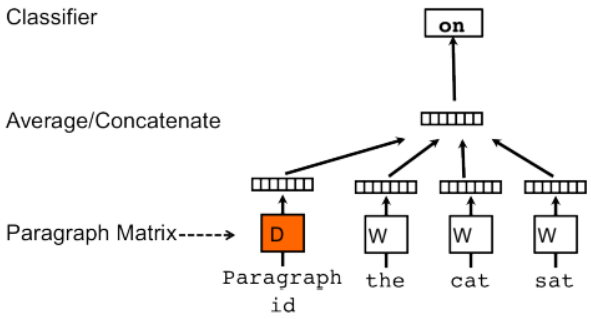

Figura 8.9: Modello PV-DM [21].

Si può notare nella Figura 8.9, come il PV-DM prevede l'aggiunta del Paragraph ID a quello che è il modello CBOW del word2vec, il quale di base si occupa di determinare la parola centrale a partire da un contesto (insieme di parole).

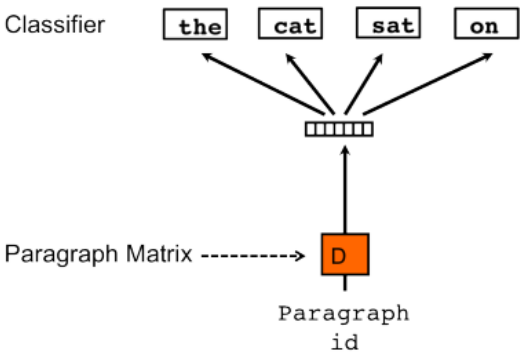

Figura 8.10: Modello PV-DBOW [21].

Tra i due approcci il PV-DM raggiunge performances migliori sebbene risulti essere più oneroso.

8.4 Utilizzo del Doc2Vec nella classificazione dei tweet

Dopo aver approfondito i principi e il funzionamento del Doc2Vec, in questa sezione verrà effettivamente spiegato come esso possa essere usato per i nostri fini di classificazione. L'idea di base è quella di convertire il testo dei tweet in **document vectors** (rappresentazioni vettoriali) tramite il modello (rete neurale addestrata) del Doc2Vec così da avere una rappresentazione che riesca a *catturare* le analogie e similitudini tra i vari messaggi (in questo contesto, quindi, i documenti saranno i tweet).
I passi che si dovrebbero eseguire, sono i seguenti:

- **Creazione del vocabolario:** una volta selezionati i documenti di training, tutte le parole presenti all'interno di essi devono essere codificate in forma vettoriale (one-hot), per poter essere usate come input della rete neurale, creando così un vocabolario di termini.
- **Addestramento della rete neurale (modello):** tramite l'utilizzo di un finto task (tra quelli precedentemente citati, ad esempio PV-DM). A fine addestramento per ogni tweet sarà stato definito il suo vettore corrispondente.
- **Classificazione dei tweet:**
  - *Addestramento del classificatore:* scelto uno specifico algoritmo di classificazione standard (come SVM o Linear Regression), il classificatore viene addestrato facendo utilizzo di un training set composto dai tweet sotto forma di document vectors, di cui sono note le loro etichette di classe ("Rilevante" o "Non Rilevante").
  - *Verifica del classificatore:* tramite il test set (sempre reso sotto forma vettoriale) si controlla l'accuratezza del classificatore precedentemente addestrato.

Ovviamente per fare uso di questo classificatore, sarà necessario convertire i futuri tweet - che dovranno essere predetti - in forma vettoriale, tramite il modello Doc2Vec.

Per poter costruire un modello sufficientemente adeguato (ovvero addestrare la rete neurale in maniera appropriata) è necessaria una grande mole di dati, contenente un gran numero di termini, così da poter costruire un vocabolario di lunghezza adeguata. Questo è fondamentale per poter permettere di ottenere una precisa codifica dei tweet futuri che devranno essere predetti. Infatti, questi nuovi messaggi dovranno essere **inferiti** sfruttando il modello precedentemente addestrato, che quindi deve contenere un grande vocabolario così da possedere tutti (o quasi) i termini usati all'interno dei : durante l'**inferenza**, le parole non note (non presenti nel vocabolario) sono semplicemente ignorate, rendendo, inevitabilmente, la conversione in vettore, meno precisa.
Per questa ragione, nella pratica, non avendo grandi risorse di calcolo da permettere la creazione di un modello ad hoc e completo, si è fatto uso di uno precedentemente addestrato con una grande mole di testi (English Wikipedia DBOW [22]), da cui poi sono state inferite le rappresentazioni vettoriali del training e test set.
Dal punto di vista pratico, anche in questa implementazione si è fatto uso di Python, e della ben nota libreria **Gensim** [23]. Nonostante però i suoi continui aggiornamenti, si è dovuto fare utilizzo della versione 0.13.4.1 per usufruire del modello già precedentemente addestrato (tramite approccio DBOW) disponibile su rete.

8.5 Risultati

Come è avvenuto per la classificazione mediante metadati e NLP flag, i 10 dataset di CrisisLex sono stati divisi in training e test set. Entrambe le collezioni sono state poi trasformate in vettori, facendo utilizzo del modello precedentemente addestrato.
Sono stati testati quindi, come algoritmi di classificazione, il Support Vector Machine e la Regressione Lineare, con i seguenti risultati:

Confronto tra performances (N° di training = numero di dataset usati come training set)

Algoritmo	N° training	N° testing	Accuratezza
Linear Regression	7	3	74.41%
Linear Regression	9	1	74.79%
Linear Regression	3	7	74.12%
SVM	7	3	63.40%
SVM	9	1	71.36%
SVM	3	7	67.51%

Come si può facilmente notare, la regressione lineare raggiunge un'accuratezza superiore al 74%, avendo quindi performances comparabili (se non addirittura superiori) alla classificazione basata sui metadati e NLP flag.

A questo punto può essere interessante effettuare i paragoni con una classificazione basata solamente sull'utilizzo del TF-IDF - già precedentemente introdotto e spiegato nei capitoli precedenti - appartenente alla famiglia delle tecniche conosciute come bag of words. Dando un rapido sguardo ai risultati, a prima vista si potrebbe dedurre che la tecnica TF-IDF possa garantire ottime performances. In realtà, analizzando nello specifico il contesto, si può notare come questi risultati siano fuorvianti: questa tecnica richiede 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.

Nella figura seguente sono state quindi riportate le performance del TF-IDF valutate

utilizzando la cross validation. Come già precedentemente affermato però, con nuovi dataset da predire si dovrebbe riaddestrare l'intero classificatore.

accuracy: 79.44% +/- 1.58% (mikro: 79.44%)

	true Related - but not informative	true Related and informative	class precision
pred. Related - but not informative	928	143	86.65%
pred. Related and informative	1122	3959	77.92%
class recall	45.27%	96.51%	

Figura 8.11: Performance del classificatore basato su TF-IDF utilizzando la cross validation.

Conclusioni e sviluppi futuri

Nel corso della progettazione sono stati quindi presentati diversi approcci con l'obiettivo di creare un classificatore capace di ottenere buone performances nella distinzione tra i tweets **rilevanti** e quelli **non rilevanti**:

- **Classificazione mediante metadati e NLP flags**: che ha mostrato performances buone ma anche variabili in base alla conformazione del dataset da etichettare e alla reale presenza dei tweets rilevanti.
- **Classificazione dei tweets resi in forma vettoriale tramite Doc2Vec**: che si è dimostrato essere molto più stabile rispetto al precedente, con performances efficaci.

Nonostante i risultati raggiunti, i due classificatori hanno potenzialmente diversi margini di miglioramento. Per quanto riguarda il primo approccio, ad esempio, si è notato che il tagger per il Name Entity Recognition fornito dallo Stanford mostra alcune imprecisioni. Essendo però il settore del NLP in costante sviluppo, in futuro potranno essere messi a disposizione tools più performanti e precisi, che potranno essere sfruttati per perfezionare il tagging dei termini presenti nei tweets. Oltre a ciò, altre migliorie potrebbero essere introdotte facendo uso di altri metadati (come **quote_count** o **reply_count**, indicanti rispettivamente, il numero di volte in cui tweet è stato citato, e quante volte il tweet riceve una risposta), che Twitter fornisce solo agli utenti Enterprise (account ottenibili tramite un pagamento).
Nel caso della classificazione mediante tweets sotto forma di document vectors, il passo successivo potrebbe essere quello di implementare un modello ad hoc facendo utilizzo di una grande mole di testi di training, tenendo però conto della necessità di possedere

una grande potenza di calcolo. Anche volendo usufruire di un modello precedentemente addestrato, si potrebbero ottenere diversi miglioramenti facendo uso di una rete neurale che si basi sull'approccio PV-DM che è tipicamente ritenuto più preciso e performante rispetto al DBOW.

Ringraziamenti

Questa tesi non rappresenta solo il frutto del lavoro durato qualche mese, bensì è la conclusione di un periodo della mia vita che è durato ben 24 anni: quello della crescita e della spensieratezza. E se c'è una cosa che ormai ho capito, è che non smetterò mai di incontrare persone meravigliose sul mio percorso. Per questo motivo, i ringraziamenti sono rivolti a chiunque abbia partecipato (anche indirettamente) a rendermi la persona che sono oggi.

Ringrazio i miei vecchi compagni di scuola, e le vecchie amicizie nate tra quei banchi: sebbene la vita ci abbia fatti momentaneamente allontanare, vi porterò sempre nel cuore e son sicuro che in futuro ci sarà modo di vederci più spesso. Vorrei fare qualche nome, ma rischierei di dimenticarmi qualcuno, quindi preferisco evitare.

Ringrazio i miei compagni universitari di Roma (in primis Matteo, Gabriele e Claudia), che sono stati fondamentali nel mio percorso di crescita professionale ed umano.

Ovviamente, una menzione speciale va a tutte le persone conosciute a Torino e che in soli due anni sono riuscite a diventare delle figure importanti della mia vita. Grazie a voi sono riuscito a mettermi in gioco ed affrontare, non senza difficoltà, questa esperienza di vita che porterò sempre nel cuore. Grazie ai miei conquilini Renzo e Paolo, che non mi hanno mai fatto provare il senso di solitudine: grazie a voi mi sono sentito a casa. Grazie ai miei amici universitari: Vincenzo detto Uccio, Mattia detto MiCStu, Nico detto EdwardManiDiNico per il , Federico il Cucciolo, Giusteppe Anima&Corpo (non riesco a ricordare il suo vero nome), l'imbuto (e il suo fido compagno, Toppariello) e Piero detto Metapod (Conte ImissU). Siete stato l'ossigeno necessario per non intossicarmi dall'inquinamento della città. Menzione speciale, va fatta per il mio compagno, amico e fratello torinese: Francesco (detto il Colonnello SantoPardo), con cui ho condiviso

questi 2 anni e mezzo dal primo all'ultimo istante, affrontando insieme ogni momento felice e difficile; dai primi istanti che sono sembrati insuperabili e tremendamente tosti, agli ultimi momenti, in cui ancora non mi capacito di come tutto sia passato così in fretta. Grazie per aver riempito la monotona vita torinese con il tuo aspirabriciole, l'Ajax, e le tue risate strampalate. Rimarrai sempre il mio KFC brother insieme ad Asso e a Rick (Il Gestore). La laurea insieme è semplicemente il modo migliore per poter concludere questa magnifica esperienza.

Ringrazio il professore Paolo Garza, mio relatore, non solo per il suo ruolo all'interno della tesi (in cui è stato fondamentale il suo assiduo supporto e la sua sapiente guida), bensì anche per la persona che vi è dietro: sempre disponibile, gentile, simpatico e soprattutto in grado di trasmettere la propria passione ai suoi studenti.

Continuo ringraziando tutto il team del "pranzo a sacco" che ha avuto un ruolo chiave in questi ultimi anni: siete stati la mia fonte di serenità e totale spensieratezza, insomma la mia *Arcadia*. Grazie a (faccio i nomi, altrimenti Emiliano mi mette il muso per giorni, rigorosamente in ordine alfabetico): Alberto (detto Dabbro, "scoperto" recentemente, ma che sembra essere l'amico di una vita), Annalisa (detta Hacker, presenza ormai costante, che dietro il suo cinismo e razionalità nasconde grande affetto e simpatia), Checco (detto il Doctor, punto di riferimento e simbolo di pace e serenità), Emiliano (detto #Transenna, amico e cugino di infanzia dal cuore enorme), Emilio (detto eSilio, fedele e leale compagno di avventura, con cui affrontare qualsiasi tipo di discorso), Giorgio (detto Lupo, sinonimo di carisma, simbolo di adrenalina e fomento), Nicola (detto Nigooola, incarnazione della simpatia e di umorismo low cost, quello che piace a me), Noemi (detta Noemma o Otto, paladina dell'allegria, similmente al re Mida, tutto ciò che tocca diventa gioioso) e Paolo (detto Pissi, sangue del mio sangue, amico e cugino con cui sono cresciuto e con cui crescerò). Un grazie speciale va anche a tutti i "cugini" acquisiti che non ho nominato (non odiatemi, già si è trasformata in una lista della spesa).

Infine, ringrazio la mia famiglia, che mi ha dato il supporto morale, economico e tutto l'amore necessario per affrontare questa sfida lontano da casa. Grazie a mio padre e madre, noi figli non avremmo saputo chiedere di meglio, anche se a volte non siamo in grado di dimostrarlo. Grazie a mia sorella Giulia e al suo futuro sposo Antonio, e a

mio fratello Claudio e la sua splendida famiglia (Rossana e Davide).
Ovviamente concludo ringraziando lei, Luciana, che è stata sempre presente in ogni momento della mia vita, da 3 anni a questa parte. Ormai sei il mio punto di riferimento, e sai già quanto sia impossibile scrivere in poche righe quanto tu sia importante per me. Sei la persona più bella che abbia mai avuto la fortuna di incontrare. Senza di te nulla di tutto ciò che ho fatto avrebbe avuto la stessa bellezza. Grazie davvero.
Onestamente non so cosa mi riserverà il futuro, ma grazie al supporto di voi persone a me care, sono sicuro che riuscirò ad affrontare nuove situazioni e sfide: si chiude un capitolo importante della mia vita, ma grazie a voi sono pronto a girare pagina.

Bibliografia

[1] International Communication Union: *ICT Facts and Figures, 2013*

http://www.itu.int/en/ITU-D/Statistics/Documents/factsICTFactsFigures2013-e.pdf

[2] Muhammad Imran, Carlos Castillo, Ji Lucas, Patrick Meier, Sarah Vieweg: *AIDR: Artificial Intelligence for Disaster Response*

http://chato.cl/papers/demo_2014_aidr_artificial_intelligence_disaster_response.pdf

[3] Jacopo Longhini, Claudio Rossi, Claudio Casetti, Federico Angaramo: *A Language-agnostic Approach to Exact Informative Tweets during Emergency Situations*

https://ieeexplore.ieee.org/document/8258372/

[4] Figura presa dal website: *Intelligenza Artificiale*

http://www.intelligenzaartificiale.it/data-mining/

[5] Figura presa dal website: *DataBase and Data Mining Group del Politecnico di Torino*

http://dbdmg.polito.it/wordpress/teaching

[6] Cyc, figura presa dal website: *Wikipedia: Support Vector Machines*

https://commons.wikimedia.org/w/index.php?curid=3566688

[7] ZackWeinberg, figura presa dal website: *Wikipedia: Support Vector Machines*

https://commons.wikimedia.org/w/index.php?curid=22877598

[8] Software per il linguaggio naturale: *Stanford CoreNLP*

`https://stanfordnlp.github.io/CoreNLP/index.html#human-languages-supported`

[9] Figura presa dal website: *Stanford CoreNLP*

`https://stanfordnlp.github.io/CoreNLP/index.html`

[10] Una repository di dati e strumenti relativi a crisi *CrisisLex.org*

`http://www.crisislex.org/`

[11] Python Wrapper per Twitter API: *Twython*

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

[12] Acquisizione per l'accesso ai dati di Twitter: *Application Management for Twitter*

`https://apps.twitter.com/app/15236493`

[13] Python Library: *Pandas*

`https://pandas.pydata.org/`

[14] Python Library: *Numpy*

`http://www.numpy.org/`

[15] Servizi di condivisione files: *Git and GitHub*

`https://git-scm.com/`

`https://github.com/`

[16] Repository contente buona parte del materiale del progetto

`https://github.com/FlavioC182/Crisis-tweets-analysis`

[17] Piattaforma per il processamento ed analisi dei dati: *RapidMiner 8.2*

`https://rapidminer.com/`

[18] Piattaforma per interfcciamento con StanfordNLP: *Natural Language Toolkit*

`https://www.nltk.org/`

[19] Wikipedia: *Introduzione al Word2Vec*

`https://en.wikipedia.org/wiki/Word2vec`

[20] Chris McCormick site: *Word2Vec Skip Gram Model*

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

[21] Gidi Shperber site: *A gentle introduction to Doc2Vec*

`https://medium.com/scaleabout/a-gentle-introduction-to-doc2vec-db3e8c0cce5e`

[22] Modello Doc2Vec: *English Wikipedia DBOW*

`https://ibm.ent.box.com/s/3f160t4xpuya9an935k84ig465gvymm2`

[23] Libreria Python per Doc2Vec: *Gensim*

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

[24] Struttura rete neurale: *Creating a Neural Network*

`https://medium.com/@curiousily/tensorflow-for-hackers-part-iv-neural-network-fro`