ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Scienze
Corso di Laurea in Ingegneria e Scienze Informatiche

DESCRIPTIVE TEXT MINING IN AMBITO MEDICO: APPLICAZIONE DELLA METODOLOGIA POIROT

Elaborato in
Programmazione Di Applicazioni Data Intensive

Relatore
Prof. Gianluca Moro

Presentata da
Riccardo Battistini

Terza Sessione di Laurea
Anno Accademico 2020 – 2021

PAROLE CHIAVE

Machine Learning

Natural Language Processing

Descriptive Text Mining

POIROT

Transformer

You shall know a word
by the company it keeps.

JOHN RUPERT FIRTH

Introduzione

Poiché la nostra conoscenza collettiva continua ad essere digitalizzata e memorizzata, diventa più difficile trovare e scoprire ciò che stiamo cercando. Abbiamo bisogno di nuovi strumenti computazionali per aiutare a organizzare, rintracciare e comprendere queste vaste quantità di informazioni.

I modelli di linguaggio sono potenti strumenti che possono essere impiegati per estrarre conoscenza statisticamente significativa ed interpretabile tramite apprendimento non supervisionato. Modelli di linguaggio algebrici, come LSA, o probabilistici, come LDA, consentono di estrarre informazioni dal testo in modo automatico o interattivo e di gestire la sinonimia. Tramite modelli di linguaggio neurali, basati su reti neurali profonde, è possibile creare delle rappresentazioni del testo che catturino anche la polisemia. Questi modelli di linguaggio possono essere adattati a molti tipi di dati. Ad esempio permettono di identificare schemi in dati genetici e in immagini, oltre che in documenti testuali o nel codice sorgente.

Obiettivo della tesi L'obiettivo di questa tesi è impiegare una metodologia di *descriptive text mining* denominata "POIROT" [1] per analizzare i rapporti medici del dataset *Adverse Drug Reaction* (ADE) [2]. Si vogliono stabilire delle correlazioni significative che permettano di comprendere le ragioni per cui un determinato rapporto medico fornisca o meno informazioni relative a effetti collaterali dovuti all'assunzione di determinati farmaci.

Struttura della tesi La trattazione è stata articolata partendo da una panoramica del campo dell'apprendimento automatico nel capitolo 1. Si descrivono le principali applicazioni e le problematiche, oltre che i tipi di sistemi di apprendimento automatico. In seguito si espone la struttura di un tipico progetto di Machine Learning. Nel capitolo 2 si fornisce una panoramica del campo dell'elaborazione del linguaggio naturale e dell'Information Retrieval. In particolare si approfondisce la rappresentazione di dati testuali non strutturati in un formato che può essere fornito in input ad una macchina e si descrivono alcuni dei principali compiti che sono svolti per realizzare sistemi di elaborazione del

linguaggio naturale. Inoltre si descrivono i modelli di linguaggio LSA, pLSA e LDA. Nel capitolo 3 si analizzano gli strumenti messi a disposizione dal deep learning per l'elaborazione del linguaggio naturale. Si descrivono le reti neurali ricorrenti e le loro estensioni che impiegano meccanismi di memoria. Successivamente si analizza l'architettura del Transformer e alcuni dei modelli di linguaggio basati su questa architettura. Nel capitolo 4 si descrivono la metodologia e il dataset impiegati per realizzare il progetto oltre che il suo dominio di appartenenza. Nel capitolo 5 si mostrano le fasi in cui il progetto si articola. Dalla preelaborazione del testo fino alle visualizzazioni e alle relazioni estratte dai documenti.

Come introduzione preliminare e a supporto degli argomenti trattati nella tesi sono state consultate diverse fonti [3] [4] [5] [6].

Indice

Elenco delle figure

Capitolo 1

Apprendimento automatico

L'apprendimento automatico, o *Machine Learning*, fornisce metodi generali per estrarre modelli di conoscenza da un insiemi di dati. È applicabile a qualsiasi problema in cui esiste una quantità sufficiente di dati. Un modello di Machine Learning consiste in una funzione che associa ad ogni dato in input una previsione, una classe (valore discreto) o un valore numerico (valore continuo). Un modello è la specifica di una relazione matematica o probabilistica che esiste tra diverse variabili.

1.1 Tipi di sistemi di apprendimento

I sistemi di Machine Learning possono essere classificati in base a diversi fattori che sono illustrati di seguito.

1.1.1 Tipologie di dati in input

Si possono distinguere i sistemi in base al tipo di dati che ricevono durante l'addestramento. In particolare si possono impiegare *variabili strutturate*, come tabelle con domini noti in ogni attributo, oppure *variabili destrutturate*, come frasi, documenti, post ed email.

1.1.2 Tipologie di apprendimento

Si possono distinguere i sistemi in base al tipo di supervisione che ricevono durante l'addestramento. Di seguito si distinguono due principali categorie: l'apprendimento supervisionato e non supervisionato.

Apprendimento supervisionato Nei sistemi di apprendimento supervisionato si impiegano dei dataset di addestramento con i valori corretti da cui

apprendere, detti *etichette*. Esempi tipici di compiti che impiegano l'apprendimento supervisionato sono la classificazione, nel caso si voglia predire il valore di variabili discrete, o la regressione, nel caso si vogliano predire variabili continue.

Apprendimento non supervisionato Nei sistemi di apprendimento non supervisionato i dati di addestramento non sono etichettati. Il sistema apprende in modo autonomo. Alcuni dei principali algoritmi non supervisionati sono gli algoritmi di riduzione della dimensionalità e di clustering, oltre che di rilevazione delle anomalie. Ad esempio nel primo caso si può rappresentare una distribuzione complessa di dati non etichettati in uno spazio bidimensionale o tridimensionale evidenziandone le caratteristiche salienti. Nel secondo una possibile applicazione è in un sistema di rilevamento delle intrusioni su reti di calcolatori.

Esistono poi ulteriori tipologie di apprendimento, come l'apprendimento supervisionato e l'apprendimento per rinforzo che non saranno considerate in questa trattazione.

1.1.3 Batch Learning vs Online Learning

Si possono distinguere i sistemi in base alla loro capacità di apprendere in modo incrementale. In caso affermativo sono modelli di tipo *online learning* altrimenti sono di tipo *batch learning*.

Batch learning Un sistema di tipo batch learning per poter essere addestrato ed aggiornato richiede di impiegare l'intero dataset, sui vecchi e i nuovi dati. Ciò implica l'impiego di notevoli risorse computazionali e generalmente è svolto offline. In ambiente di produzione il sistema si limita ad applicare ciò che ha appreso senza imparare più nulla. Si dice in questo caso che il sistema è di tipo *offline learning*. In particolari applicazioni in cui è richiesto un riaddestramento frequente a causa di un flusso di nuovi dati continuo, come la previsione degli indici di borsa, un sistema di batch learning è del tutto inadeguato.

Online learning Un sistema di tipo online learning si addestra in modo incrementale a partire da piccoli gruppi di dati, detti *minibatch*. Ciascun passo di apprendimento si svolge rapidamente ed è poco costoso. In questo modo il sistema può apprendere da un flusso di dati continuo in entrata. Un vantaggio di questi sistemi sta nella possibilità di adattarsi autonomamente e in modo rapido ai cambiamenti. Inoltre si risparmia memoria dato che non è necessario mantenere i dati su cui il sistema si è già addestrato. Un altro vantaggio sta nella possibilità di impiegare dei dataset di addestramento che normalmente sarebbero troppo grandi per essere contenuti nella memoria principale di un

unico calcolatore. In quest'ultimo caso si parla di *out-of-core learning*. Uno dei principali limiti dell'online learning sta nella facilità con cui le prestazioni del sistema declinino se si forniscono dati di pessima qualità. Per mitigare questo problema si possono mantenere copie del modello funzionanti e si possono impiegare algoritmi di rilevamento delle anomalie.

1.1.4 Instance based vs Model based learning

Un altro modo con cui si differenziano i sistemi è nell'approccio adottato per la generalizzazione. Si distinguono sistemi di apprendimento di tipo *instance based*, o *memory based*, e *model based*.

Instance based learning In un sistema di apprendimento instance based si confrontano direttamente i dati di addestramento con i nuovi dati per formulare delle previsioni. Un sistema di questo tipo generalizza calcolando la distanza di una nuova istanza da istanze note tramite misure di similarità. Un esempio è il sistema di raccomandazione *user based nearest neighbor collaborative filtering* per fare previsioni in base alla similarità tra utenti.

Model based learning Un sistema di apprendimento model based cerca di apprendere degli schemi nei dati di addestramento impiegando un modello. Dopo aver scelto un modello lo si addestra in modo da configurare i parametri interni, si scelgono i relativi iperparametri e si formulano delle valutazioni.

1.2 Applicazioni

Il Machine Learning si presta ad essere applicato a problemi per i quali le soluzioni esistenti richiedono di formulare lunghi elenchi di regole. Un algoritmo di Machine Learning permette di semplificare il codice ed è più prestante degli approcci tradizionali.

1.3 Problematiche

In un progetto di Machine Learning si sceglie un algoritmo di apprendimento e dei dati su cui addestrarlo, validarlo e verificarlo. Perciò possono esserci problemi che riguardano gli algoritmi o i dati.

1.3.1 Quantità di dati insufficiente

Nel 2001 Banko e Brill [7] hanno mostrato che se si addestrano degli algoritmi di Machine Learning molto diversi fra loro per risolvere un problema di disambiguazione del linguaggio naturale, fornendo una quantità di dati sufficiente le prestazioni sono quasi identiche per tutti gli algoritmi. L'idea che investire sui dati piuttosto che sugli algoritmi sia importante, specialmente per i problemi più complessi, è stata riproposta anche da uno studio di Norvig nel 2009, in un articolo denominato "L'irragionevole efficacia dei dati" [8].

Dai problemi più semplici fino a quelli più complessi, sono necessarie dalle migliaia alle milioni di osservazioni per generare risultati accettabili. Questa è una delle ragioni per cui la possibilità di utilizzare dei modelli preaddestrati risulta particolarmente vantaggiosa.

1.3.2 Dati non rappresentativi

Un esempio di dataset con dati non rappresentativi si ha nel contesto di problemi di classificazione, quando la suddivisione di istanze tra classi è molto sbilanciata. In questo caso si verificano molti più errori di classificazione sulla classe meno rappresentata, con risultati spesso inaccettabili. Per gestire questa difficoltà si possono impiegare diversi accorgimenti come l'aumento del peso degli errori della classe meno numerosa, tramite un apposito iperparametro. In alternativa si può ricorrere al sovracampionamento, o *oversampling*, della classe meno numerosa o al sottocampionamento, o *undersampling*, della classe più numerosa.

Una delle tecniche che si può impiegare per l'oversampling è SMOTE, per la creazione di istanze artificiali. Se l'accuratezza a seguito di oversampling aumenta bisogna verificare che ciò non sia dovuto solo ai dati artificiali. Questa è una delle ragioni per cui non si può misurare l'affidabilità del modello solo a partire dall'accuratezza ma è necessario impiegare metriche come precisione e richiamo.

È fondamentale impiegare dei dataset che siano rappresentativi dei casi che si vogliono generalizzare. Se si acquisiscono campioni di dati troppo piccoli si incorre nel *sampling noise*, ovvero nel rumore dovuto alla presenza di dati non rappresentativi. Anche campioni troppo grandi possono non essere rappresentativi nel caso il metodo di campionamento non sia adeguato. In questo caso si ha a che fare con il *sampling bias*.

1.3.3 Dati di bassa qualità

Se un dataset ha molti outlier, molti errori e molto rumore, magari a causa di misurazioni di bassa qualità, sarà più difficile per il sistema rilevare degli

schemi e sarà meno probabile che abbia buone prestazioni.

Ad esempio nel caso di valori errati o di outlier si impiegano delle tecniche di pulizia per rimuoverli dal dataset. Nel caso di istanze con caratteristiche mancanti bisogna decidere se non considerare la caratteristica o se stabilire un criterio con cui rimpiazzare i valori mancanti.

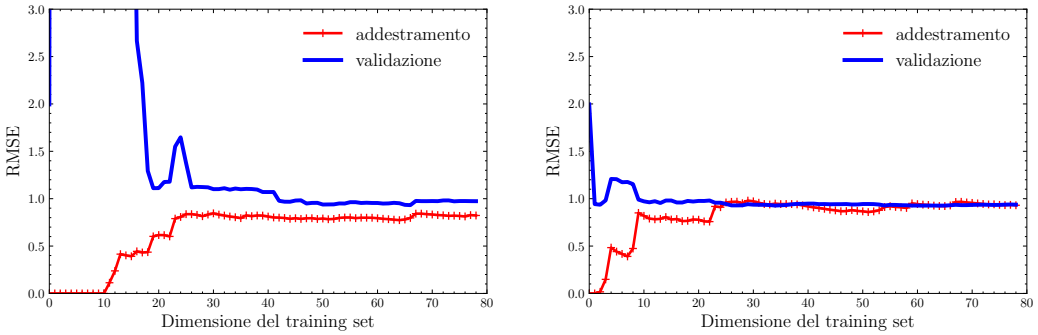
1.3.4 Caratteristiche irrilevanti

È necessario identificare le caratteristiche rilevanti in un dataset e rimuovere quelle irrilevanti tramite il processo di *feature engineering*, anche impiegando le tecniche accennate in sezione 1.5.

1.3.5 Overfitting e Underfitting

Un modello di machine learning può essere in *overfitting* o *underfitting*. Nel primo caso si ottiene un modello che effettua delle ottime previsioni sui dati con cui è addestrato ma non generalizza bene se applicato su nuovi dati. Ciò può implicare che il modello ha appreso anche il rumore contenuto nei dati oppure che potrebbe aver imparato a riconoscere input specifici piuttosto che i fattori che permettono realmente di prevedere un certo output. Modelli troppo complessi rispetto alla quantità di rumore nei dati d'addestramento tendenzialmente portano ad overfitting. Nel secondo caso si realizza un modello che non genera delle buone previsioni nemmeno sui dati d'addestramento.

Un altro modo per determinare se un modello è in overfitting o underfitting è tramite il grafico delle curve di apprendimento. Si tratta di visualizzazione delle prestazioni del modello sul training set e sul validation set in funzione delle dimensioni del training set.



(a) Esempio di un modello overfitting.

(b) Esempio di un modello underfitting.

Figura 1.1: Curve di apprendimento.

In figura 1.1b si osserva la curva di apprendimento tipica di un modello in underfitting. Inizialmente il modello apprende dall'aggiunta di nuovi dati al training set. In seguito si raggiunge un plateau e con l'aggiunta di nuovi dati l'errore medio non peggiora né migliora ed è elevato. Lo stesso accade durante la validazione. Inizialmente il modello non è in grado di generalizzare, in seguito apprende dai nuovi dati e infine raggiunge un punto di stallo, in cui l'errore medio non cambia anche aggiungendo nuovi dati ed è molto vicino all'errore in fase di addestramento.

Un modello in overfitting avrà una curva di apprendimento con caratteristiche simili a quella in figura 1.1a. In particolare l'errore in fase di addestramento è minore che in figura 1.1b e c'è una distanza significativa tra la curva di addestramento e quella di validazione. Inoltre si osserva che aumentando la quantità di dati impiegata per l'addestramento le due curve tendono ad avvicinarsi.

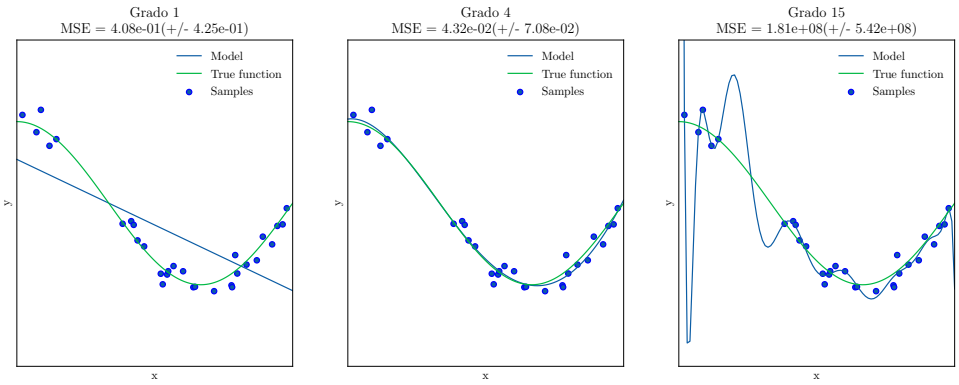


Figura 1.2: Esempio di modello di regressione rispettivamente underfitting (grado 1), fitting (grado 4) e overfitting (grado 15).
https://scikit-learn.org/stable/auto_examples/model_selection/plot_underfitting_overfitting.html

L'esempio in figura 1.2 mostra come l'underfitting e l'overfitting possano limitare notevolmente le prestazioni di un modello di Machine Learning. La quantità di dati a disposizione e la complessità del modello sono interdipendenti. Con più dati, generati anche attraverso tecniche di *data augmentation*, si può rimediare all'overfitting di un modello troppo complesso. Inoltre si possono ridurre i gradi di libertà del modello tramite la selezione di caratteristiche o la regolarizzazione. Per gestire l'underfitting si possono impiegare modelli più potenti, con più parametri, generare nuove caratteristiche tramite feature engineering e diminuire i vincoli sul modello, ad esempio riducendo l'iperparametro di regolarizzazione.

1.4 Processo di estrazione della conoscenza

Un progetto di Machine Learning consiste in un processo di estrazione della conoscenza. Questo processo è valido per ogni sistema di intelligenza artificiale basato sull'apprendimento. È costituito dalle seguenti fasi:

- raccolta dei dati, analisi esplorativa e analisi della loro qualità;
- trattamento di valori mancanti, normalizzazione, standardizzazione;
- selezione delle caratteristiche, ossia degli attributi del dataset rilevanti rispetto agli obiettivi;
- divisione dei dati in *training set*, *validation set* e *test set*;
- selezione dei modelli di conoscenza e loro addestramento;
- messa a punto dei modelli tarando gli iperparametri e confronto tra modelli;
- interpretazione dei migliori modelli di conoscenza estratti;
- schieramento, monitoraggio e mantenimento del sistema a partire dal modello scelto.

Per quanto riguarda la selezione dei modelli, secondo il teorema *No Free Lunch*, dimostrato nel 1996 da David Wolpert [9], se non si formulano ipotesi sui dati non è possibile stabilire a priori quale sia il modello migliore. Perciò l'unico modo per determinare il modello più adatto è di provarli tutti di volta in volta e scegliere il migliore. Nella pratica ciò non è possibile e ci si limita a formulare ipotesi ragionevoli sui dati e si valutano solo pochi modelli scelti ragionevolmente.

1.5 Selezione ed estrazione di caratteristiche

Il processo di creazione, selezione ed estrazione di caratteristiche da un dataset è detto *feature engineering*.

Creazione ed estrazione di caratteristiche Per aumentare il numero di caratteristiche a disposizione si può considerare ad esempio il quadrato o il logaritmo delle caratteristiche note. In aggiunta si possono combinare più caratteristiche, ad esempio considerando il loro prodotto. Considerare combinazioni di caratteristiche è anche uno dei possibili modi con cui capire come interagiscono tra loro.

Selezione di caratteristiche Nel caso si vogliano identificare e rimuovere caratteristiche irrilevanti si possono impiegare algoritmi di riduzione della dimensionalità o la regolarizzazione.

Tipi di caratteristiche In genere le caratteristiche che si estraggono dai dati sono numeriche, binarie o categoriche. A seconda dei modelli che si scelgono di impiegare può essere necessario alterare o non considerare determinate caratteristiche. Per gestire le caratteristiche non numeriche si può impiegare la codifica *one hot*. Ad esempio i modelli di regressione richiedono caratteristiche esclusivamente numeriche. Gli alberi di decisione al contrario possono gestire sia caratteristiche numeriche che categoriche.

1.6 Addestramento di un modello

L'addestramento di un modello di apprendimento può essere effettuato risolvendo un problema di ottimizzazione in modo da impostare il valore dei suoi parametri in base ai dati forniti in input. A questo scopo si può impiegare un algoritmo di ottimizzazione come la *discesa del gradiente*.

1.6.1 Algoritmo della discesa del gradiente

La discesa del gradiente è un metodo generale impiegato da modelli di apprendimento per determinare la combinazione di parametri nello *spazio dei parametri* che consente di formulare delle valutazioni che approssimano i dati di addestramento commettendo l'errore più piccolo possibile. Modelli complessi, formati da più parametri, richiederanno maggior tempo per determinare la combinazione ottimale.

La discesa del gradiente è un algoritmo iterativo. Durante l'inizializzazione si assegnano dei valori casuali al vettore dei parametri θ. Ad ogni iterazione si calcola il gradiente a partire dai dati di addestramento e si modifica il valore dei singoli parametri θ_i in modo da minimizzare la funzione di loss $\mathrm{f}(\theta)$, fino alla convergenza ad un punto di minimo. In altri termini ad ogni iterazione si calcola la derivata parziale $\frac{\partial}{\partial\theta_j}\mathrm{f}(\theta)$ rispetto a θ_j, definita in equazione (1.1).

$$\frac{\partial}{\partial\theta_j}\mathrm{f}(\theta) = \frac{2}{m}\sum_{i=1}^{m}\left(\theta^\top\mathbf{x}^{(i)} - y^{(i)}\right)x_j^{(i)}, \tag{1.1}$$

dove $\mathbf{x}^{(i)}$ indica l'i-esima istanza del dataset di addestramento $\mathbf{X}$ e $y^{(i)}$ l'i-esima etichetta. In questo modo si può determinare di quanto cambi il valore della funzione di costo alterando ciascun θ_i. Il vettore del gradiente $\nabla_\theta\mathrm{f}(\theta)$, definito in equazione (1.2), contiene tutte le derivate parziali della funzione di loss rispetto a ciascun θ_i.

$$\nabla_\theta \mathrm{f}(\theta) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} \mathrm{f}(\theta) \\ \frac{\partial}{\partial \theta_1} \mathrm{f}(\theta) \\ \vdots \\ \frac{\partial}{\partial \theta_n} \mathrm{f}(\theta) \end{pmatrix} \tag{1.2}$$

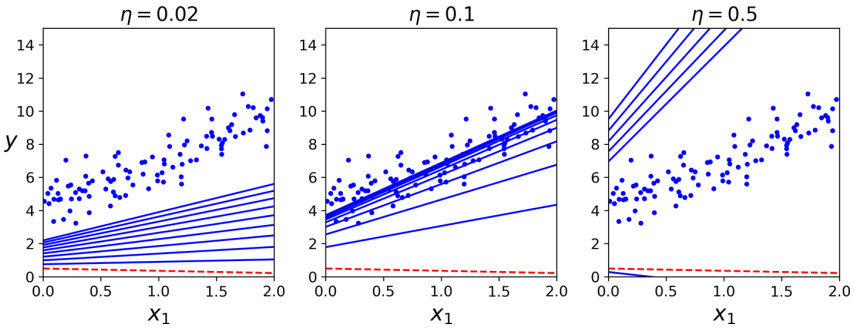


Figura 1.3: Discesa del gradiente con diverse lunghezze del passo di discesa η.

A partire dal gradiente $\nabla_\theta \mathrm{f}(\theta)$ si può calcolare il passo di discesa all'iterazione k, mostrato in equazione (1.3).

$$\theta^{(k+1)} = \theta^{(k)} - \eta \nabla_\theta \mathrm{f}(\theta) \tag{1.3}$$

dove $-\eta \nabla_\theta \mathrm{f}(\theta)$ indica la direzione di massima discesa ed η costituisce la lunghezza del passo di discesa, detto anche *step size* o *learning rate*. η è un iperparametro da impostare. Come si osserva in figura 1.3, se η è troppo piccolo l'algoritmo impiega troppe iterazioni per convergere mentre se è troppo grande l'algoritmo potrebbe essere più lento nel trovare il minimo per via di avanzamenti troppo ampi.

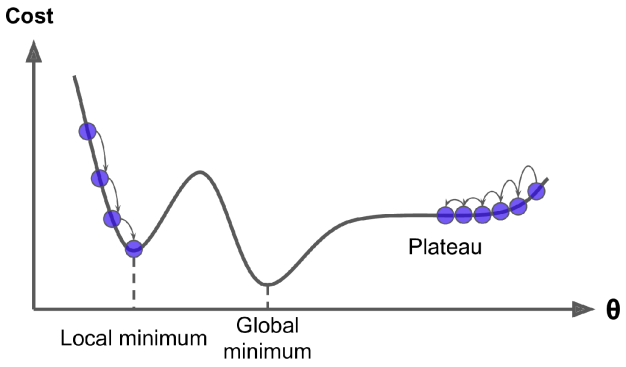


Figura 1.4: Ostacoli alla discesa del gradiente.
Fonte: Hands-On Machine Learning with Scikit-Learn [4, p. 176]

In figura 1.4 si osserva come la discesa del gradiente si possa bloccare in punti di minimo locale, se applicato a funzioni non convesse, o in plateau. Per evitare che ciò accada si può impostare una valore di tolleranza ε tramite un iperparametro. In altri termini si interrompe la discesa quando il vettore del gradiente diventa più piccolo di ε. Un altro fattore che può rallentare la velocità di convergenza è l'impiego di caratteristiche fuori scala. Per questa ragione si devono standardizzare i dati da fornire in input ai modelli di apprendimento che impiegano la discesa del gradiente.

Per determinare i valori ottimali di un iperparametro, come η, ε o il numero massimo di iterazioni, si può impiegare una tecnica detta *grid search*. La grid search effettua una ricerca esaustiva tra i possibili valori degli iperparametri forniti in modo da determinare la combinazione migliore. In alternativa si può effettuare una ricerca casuale indicando unicamente gli iperparametri da determinare e il numero di iterazioni a disposizione per effettuare la ricerca dei valori ottimali.

1.6.2 Tipologie di discesa del gradiente

In base al numero di istanze del training set $\mathbf{X}$ su cui è calcolato il gradiente $\nabla_\theta \mathrm{f}(\theta)$ ad ogni iterazione, si distinguono tre varianti della discesa del gradiente: la discesa del gradiente *batch*, la discesa del gradiente *stocastica* e la discesa del gradiente *minibatch*.

La discesa del gradiente batch impiega ad ogni iterazione tutti i valori del training set $\mathbf{X}$ per calcolare il gradiente. Ciò lo rende molto lento nel caso di dataset di addestramento grandi.

Algoritmo 1: Discesa del gradiente stocastica

Input: Training set $\mathbf{X} \in \mathbb{R}^{m \times n}$, learning rate η, model parameters θ
Output: Model parameters $\theta \in \mathbb{R}^n$

1 $\eta \leftarrow 0.1$; $\theta \sim \mathcal{N}(0, \sigma)$;
2 **repeat**
3 **for** *i in m* **do**
4 choose randomly $(\mathbf{x}_i, \mathbf{y}_i)$ from $(\mathbf{X}, \mathbf{y})$;
5 $\nabla_\theta \mathrm{MSE}(\theta) \leftarrow 2\mathbf{x}_i^\top(\mathbf{x}_i\theta - \mathbf{y}_i)$;
6 update η according to the learning schedule;
7 $\theta = \theta - \eta\nabla_\theta \mathrm{MSE}(\theta)$;
8 **end**
9 **until** *stopping criterion is not met*;

Nell'algoritmo 1 si può osservare il funzionamento della discesa del gradiente stocastica. Dopo aver inizializzato casualmente il vettore dei parametri θ (riga

1), per convenzione si itera sulle m istanze del dataset di addestramento (riga 3). Ogni m iterazioni sulle istanze del training set si completa un'*epoca*. Rispetto alla variante batch, ad ogni iterazione si calcola il gradiente a partire da una singola istanza di addestramento del dataset scelta tramite estrazione casuale senza rimpiazzo (righe 4-5). L'impiego di una singola istanza piuttosto che dell'intero dataset rende le singole iterazioni molto più rapide. La casualità nella scelta dell'istanza per il calcolo del gradiente può evitare il blocco dell'algoritmo in punti di minimo locale nel caso la funzione di loss sia irregolare. Perciò la discesa del gradiente stocastica ha maggiori probabilità di determinare l'ottimo globale rispetto alla discesa del gradiente batch. La casualità nella scelta ha anche lo svantaggio di non permettere mai alla discesa di terminare dato che continuerà ad oscillare attorno ad un punto di minimo. Una tecnica per mitigare questo problema consiste nel ridurre gradualmente la lunghezza del passo di discesa. Questo processo è analogo a quello dell'algoritmo di *tempra simulata*, ispirato al processo metallurgico della tempra, in cui si raffredda lentamente il metallo. La funzione che determina il tasso di decadimento del passo di discesa è detta *learning schedule* (riga 6). Se il decadimento è troppo rapido l'algoritmo potrebbe bloccarsi in un punto di minimo locale o nel percorso per raggiungere il minimo. Se il decadimento è troppo lento si potrebbe girare attorno al punto di minimo troppo a lungo e arrestarsi con una soluzione subottimale. Dopo aver calcolato il gradiente a partire da un'istanza casuale e aggiornato il passo di discesa si aggiorna il vettore dei parametri (riga 7). L'algoritmo prosegue fino al raggiungimento di un criterio di arresto (riga 9). Ad esempio si itera fino ad un certo numero di iterazioni massime o fino al raggiungimento da parte del gradiente di un valore inferiore alla tolleranza ε.

In figura 1.5 si osserva come la discesa del gradiente stocastica sia erratica rispetto alla più fluida discesa batch. Una via di mezzo è rappresentata dalla discesa del gradiente minibatch.

In una discesa di tipo minibatch si calcola il gradiente a partire da un sottoinsieme di istanze del training set scelto casualmente, detto minibatch. Il principale vantaggio della discesa del gradiente minibatch rispetto a quella stocastica sta nella possibilità di impiegare la moltiplicazione di matrici, ottimizzata a livello hardware sia su CPU che, soprattutto, su GPU. Questo algoritmo di ottimizzazione è alla base di alcuni dei principali algoritmi per l'addestramento di reti neurali.

Sia la discesa del gradiente stocastica che minibatch possono essere implementate come algoritmi out-of-core, in modo da poter essere applicati a dataset di addestramento molto grandi. Inoltre possono essere impiegati per l'apprendimento incrementale, visto anche il costo ridotto delle singole iterazioni. Poiché entrambi gli algoritmi ad ogni iterazione scelgono casualmente una o più istanze è necessario che le istanze del training set siano *identicamente distribuite*, in

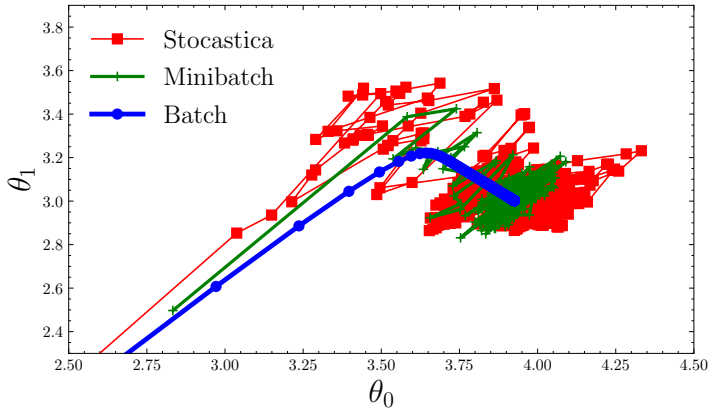


Figura 1.5: Esempio di percorsi della discesa del gradiente nello spazio dei parametri.

modo che i parametri tendano in media a convergere verso l'ottimo globale. Per far ciò è necessario le istanze del training set siano mescolate e non ordinate secondo una certa caratteristica ad esempio.

1.6.3 Early stopping

Per gestire la complessità di un modello addestrato con algoritmi iterativi come la discesa del gradiente si può impiegare una tecnica di regolarizzazione detta *early stopping*. In figura 1.6 si osserva come all'aumentare del numero di epoche dell'algoritmo, il modello apprende e sia l'RMSE sul training set che sul validation set diminuiscono. Non appena l'errore di validazione inizia ad aumentare l'algoritmo si arresta e si ottiene così il modello con l'errore di validazione minimo.

1.7 Validazione di un modello

Perché sia possibile stabilire su un modello generalizza bene è necessario osservare come si comporta a regime, ovvero fornendo nuovi dati diversi da quelli con cui è stato addestrato. Perciò bisogna distinguere tra *training set* e *test set* in modo da poter stimare l'errore di generalizzazione. Inoltre è necessario introdurre anche un *validation set*, o *development set*, per poter scegliere tra più modelli addestrati con diverse configurazioni di iperparametri. Questo perché quando si tarano gli iperparametri si modifica il modello sulla base degli errori che si ottengono nel validation set. Perciò scegliendo il migliore tendenzialmente si ottiene un modello tarato per funzionare bene solo sul training set o sul validation set, che diventa come un secondo training set.

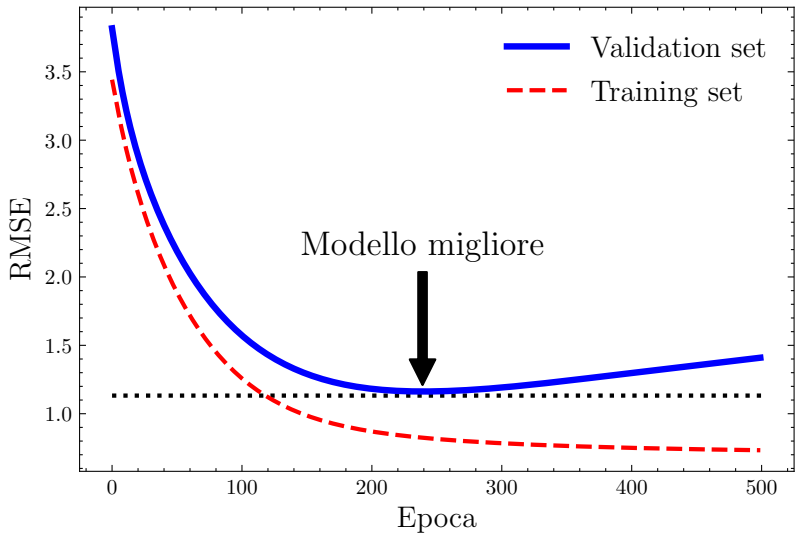


Figura 1.6: Regolarizzazione con tecnica *early stopping* applicando la discesa del gradiente batch.

Quindi si impiega il training set per addestrare ciascun modello, il validation set per tarare gli iperparametri e il test set per valutare il modello finale.

Per effettuare l'addestramento e la validazione di un modello si possono impiegare diverse tecniche di suddivisione dei dati come il *metodo Holdout*, la *K-Fold Cross Validation* o la *Nested Fold Cross Validation*.

1.7.1 Holdout

Il metodo di validazione Holdout prevede di dividere i dati a disposizione in due insiemi, secondo una proporzione predefinita ed effettuando un partizione casuale. Le proporzioni comunemente più usate sono 70/30, 66/33 e 50/50. Si tratta di uno dei metodi di validazione di un modello più semplici. Il training set è utilizzato per addestrare il modello di apprendimento, minimizzando l'errore su di esso. Dal training set si determinano i parametri del modello, come i parametri θ nel caso della regressione lineare multivariata o i pesi w nel caso di una rete neurale. Il validation set è usato dopo l'addestramento per verificare l'errore del modello su dati ignoti.

Per determinare la capacità di generalizzazione del modello si confrontano l'errore sul training set e sul validation set. Se i due errori sono simili si assume che il modello abbia generalizzato bene i dati. Nel caso il training set non sia un campione rappresentativo delle caratteristiche dei nuovi dati si può ripetere la suddivisione dei dati con un diverso seme casuale.

1.7.2 K fold Cross Validation

La K fold Cross Validation è un metodo di valutazione più sofisticato di holdout. Consiste nel suddividere l'insieme di dati iniziale in k sottoinsiemi disgiunti, detti *fold*. L'impiego dei fold si rende necessario quando il validation set è troppo piccolo o troppo grande. Nel primo caso le valutazioni del modello saranno di scarsa qualità, nel secondo si effettuerà un confronto fra modelli addestrati su un training set troppo piccolo rispetto al validation set. Ciò implica che la scelta che avviene tra modelli addestrati si basa sul modello che per primo è in grado di fornire delle valutazioni di qualità, anche se non sono le migliori possibili.

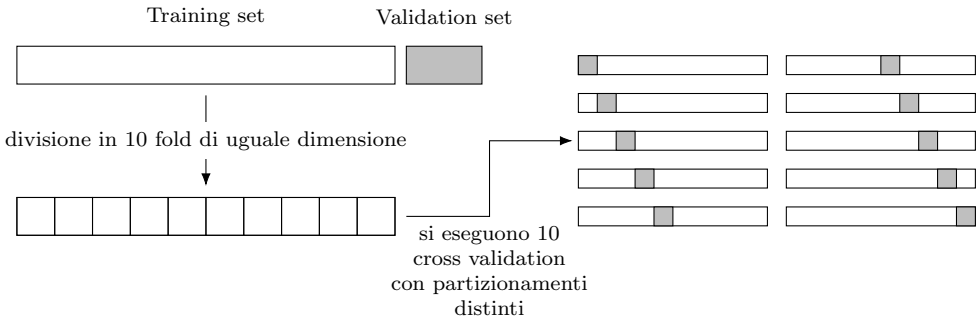


Figura 1.7: Esempio di K fold cross validation con $k = 10$.

In figura 1.7 si osserva il funzionamento della K fold cross validation. In ciascun fold un sottoinsieme è usato come validation set e gli altri $k - 1$ come training set. Si ripete il procedimento k volte con ciascuno dei k sottoinsiemi ottenuti. In questo modo si ottengono k modelli da valutare. L'accuratezza complessiva è data dalla media delle accuratezze dei k modelli. Dato che tendenzialmente questa tecnica genera dei punteggi di accuratezza ottimisti se il dataset non è troppo grande rispetto alle risorse computazionali disponibili si preferisce impiegare la *Nested Fold Cross Validation*. Per un'analisi delle problematiche legate alla K fold cross validation si rimanda allo studio di Cawley e Talbot [10].

Esiste una variante di questo modello, detta *K Fold Cross Validation stratificata*, che permette di evitare potenziali sbilanciamenti tra i singoli fold a causa di distribuzioni dei dati fortemente sbilanciate in un fold piuttosto che negli altri. Per distribuire i dati ad ogni fold si impiega il campionamento stratificato piuttosto che il campionamento casuale. Il dataset è diviso in sottogruppi, detti *strati*, e si campiona da ciascuno strato il giusto numero di istanze, in modo che la distribuzione dei dati nei fold sia rappresentativa della distribuzione dei dati nel dataset. Si applica ad esempio nei problemi

di classificazione per garantire che la distribuzione di classi in ciascun fold rispecchi la distribuzione nel dataset su cui si effettuano i campionamenti.

Il limite della K fold cross validation sta nelle risorse computazionali richieste. Più valori si vogliono verificare per tarare gli iperparametri maggiore è il tempo richiesto per addestrare il modello. Ad esempio se si vuole addestrare un modello di classificazione con un percettrone con $\alpha = [0.0001, 0.001, 0.01, 1]$ impiegando una K Fold Cross Validation stratificata a cinque fold sarà necessario effettuare venti cross validation dato che per ciascuno dei possibili valori di α saranno testati cinque distinti validation set. Per questa ragione in dataset di grandi dimensioni può essere conveniente usare un metodo holdout a tre vie, in modo da generare sia il training, che il validation che il test set.

1.7.3 Nested Fold Cross Validation

La Nested Fold Cross Validation è un metodo di valutazione di modelli di machine learning discusso approfonditamente da Varma e Simon nel 2006 [11]. Fu introdotto come possibile rimedio alle valutazione ottimistiche fornite dalla K fold cross validation, dovute all'impiego dello stesso validation set sia per tarare gli iperparametri che per valutare il modello. Ogni porzione di dati di training della K fold cross validation esterna è suddivisa nella cross fold validation interna per individuare gli iperparametri migliori. Gli iperparametri migliori si impiegano per addestrare e testare il modello nella relativa parte di validation esterna. in figura 1.8 è illustrato questo metodo di validazione. Come la K fold cross validation e il metodo holdout, non è un metodo per estrarre un modello predittivo migliore ma per stimare l'accuratezza a regime sui dati.

1.8 Valutazione di un modello

Per poter capire se un modello generalizza bene ed effettua previsioni di qualità si possono impiegare diversi strumenti statistici, a seconda delle diverse tipologie di problemi. Di seguito si è scelto di approfondire alcune delle metriche utilizzate in problemi di classificazione dato che saranno richiamate in seguito.

1.8.1 Correttezza della classificazione

In un modello di classificazione, dato un dataset etichettato, si possono individuare quattro possibili categorie in cui ogni campione si può trovare a seconda che sia stato commesso o meno un errore. Nel primo caso si hanno i veri positivi T_p e i veri negativi T_n, nel secondo caso si distinguono due tipi di errore. Un falso positivo F_p è un errore di tipo 1, un falso negativo F_n è

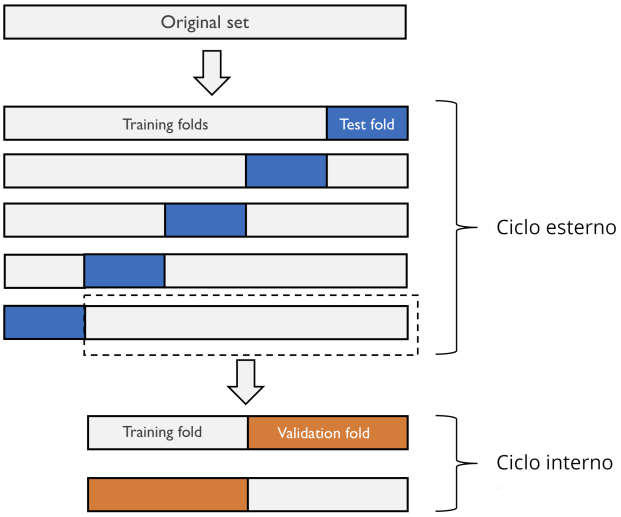


Figura 1.8: Esempio di Nested Fold cross validation. Nel ciclo esterno si stima l'accuratezza media sui dati. Nel ciclo interno si determinano gli iperparametri migliori.
Adattata a partire dalla figura 22 dell'articolo di Raschka [12]

un errore di tipo 2. In genere questi risultati si presentano in una *matrice di confusione*. Per valutare la qualità di un modello di classificazione si possono impiegare statistiche come l'accuratezza, la precisione, il richiamo, l'F1-score e la specificità. Di seguito si definiscono queste metriche e si forniscono esempi di potenziali campi di applicazione.

Definizione 1.8.1 (Accuratezza)**.** L'*accuratezza* è definita come la frazione di previsioni corrette sul totale, ovvero come:

$$\mathrm{ACC} = \frac{T_p + T_n}{T_p + F_p + F_n + T_n}$$

Definizione 1.8.2 (Precisione)**.** La *precisione* indica quanto sono accurate le previsioni positive. Può essere vista come una misura di esattezza. È definita come:

$$\mathrm{PR} = \frac{T_p}{T_p + F_p}$$

Definizione 1.8.3 (Richiamo)**.** Il *richiamo*, o *sensibilità*, misura quale frazione dei positivi identifica il modello. Può essere vista come una misura di completezza. È definito come:

$$\mathrm{RC} = \frac{T_p}{T_p + F_n}$$

Definizione 1.8.4 (F1-score)**.** L'*f1-score* è la media armonica di precisione e richiamo. È definita come:

$$F_1 = 2\frac{\mathrm{PR} \cdot \mathrm{RC}}{\mathrm{PR} + \mathrm{RC}}$$

Definizione 1.8.5 (Specificità)**.** La *specificità* rappresenta una misura di quanto un modello sia in grado di identificare i veri negativi. Se l'obiettivo di un modello è determinare accuratamente il numero di falsi positivi la specificità dovrà essere elevata. È definita come:

$$\mathrm{SP} = \frac{T_n}{T_n + F_p}$$

Per decidere quali tra queste misure impiegare bisogna considerare il dominio su cui il modello dovrà essere in grado di generalizzare. Ad esempio se si vuole realizzare un motore di ricerca si darà importanza a metriche come la precisione e il richiamo mentre sarà ignorata la specificità, visto che rispetto alle altre metriche non è importante conoscere la quantità di informazione irrilevante scartata dal motore di ricerca. Se si vuole realizzare un sistema che sia in grado di identificare rumori provenienti da armi da fuoco è importante determinare correttamente sia i falsi positivi che i falsi negativi. In questo caso si devono considerare tutti gli indicatori a disposizione, come precisione, richiamo e specificità.

1.8.2 Compromesso tra precisione e richiamo

In genere in ogni modello di classificazione si ha un compromesso tra precisione e richiamo. Un modello "prudente" tendenzialmente ha un'elevata precisione e un basso richiamo mentre un modello più "incauto" avrà una bassa precisione e un elevato richiamo. Non si può aumentare la precisione di un modello senza ridurre il richiamo e viceversa. Si prenda come esempio un modello di classificazione di pazienti positivi o negativi ad un test che considera un certo numero di fattori di rischio per distinguere gli infetti dai non. Se si modifica la soglia di decisione del modello in modo da considerare un numero crescente di fattori di rischio, si ha un incremento della precisione e un decremento del richiamo. La maggior precisione è dovuta alla maggior probabilità da parte degli individui con più fattori di rischio di essere infetti. Il minor richiamo è dovuto alla diminuzione del numero di pazienti infetti che

arriveranno alla soglia necessaria per poter considerare l'esito del test positivo. In questi casi la scelta del giusto valore di soglia richiede un compromesso. Si può visualizzare questo compromesso confrontando graficamente come precisione e richiamo possono variare modificando la soglia di decisione.

1.8.3 Curva ROC

La curva *receiver operating characteristic* (ROC) consente di valutare un modello di classificazione binario osservando come varia la sensibilità rispetto alla specificità.

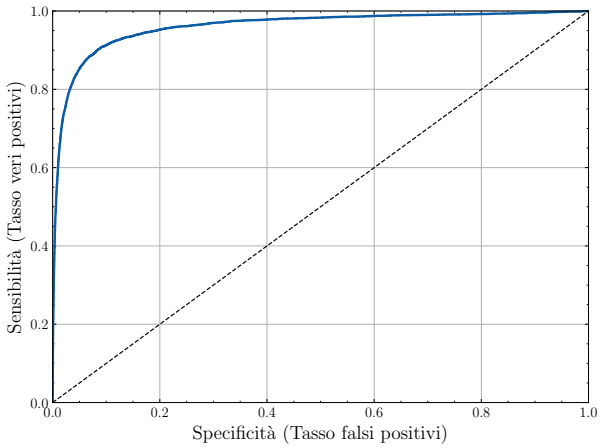


Figura 1.9: Esempio di curva ROC.

La linea diagonale tratteggiata in figura 1.9 rappresenta la curva ROC di un classificatore casuale. Un buon classificatore si avvicina alla parte in alto a sinistra del grafico stando il più possibile lontano dalla diagonale.

Un metodo per confrontare più modelli di classificazione consiste nel misurare l'area sottesa alla curva (AUC). Un classificatore perfetto ha una ROC AUC pari a uno mentre un classificatore casuale ha ROC AUC pari a 0.5.

1.8.4 Compromesso tra bias e varianza

Per valutare un modello si considera il suo errore di generalizzazione. L'errore di generalizzazione si può decomporre in tre tipi di errore: *bias*, *varianza* ed un *errore irriducibile* [13, p. 37]. L'errore irriducibile è la componente dell'errore di generalizzazione che tiene conto del rumore nei dati. L'unico modo di ridurre l'errore irriducibile è tramite pulizia dei dati. La varianza è dovuta all'estrema sensibilità di un modello di apprendimento a variazioni,

anche piccole, nei dati di addestramento. Il bias è dovuto ad ipotesi errate sui dati, come considerare lineare una distribuzione quadratica.

Sia bias che varianza esprimono ciò che accadrebbe se si riaddestrasse un modello più volte su diverse porzioni estratte casualmente di un dataset. Un modello in underfitting, che commette molti errori di previsione, avrà un elevato bias. Tuttavia scelti casualmente due insiemi di dati d'addestramento dovrebbero dare un risultato molto simile. In questo caso si dice che il modello ha una bassa varianza. D'altra parte un modello in overfitting avrà un basso bias e un'elevata varianza, dato che scelti due diverse porzioni di dati per l'addestramento si otterranno modelli molto diversi.

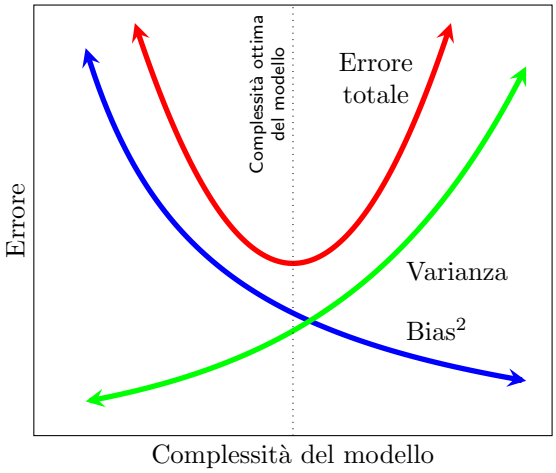


Figura 1.10: Bias e varianza in funzione della complessità del modello e dell'errore.

In figura 1.10 si può osservare come al crescere della complessità del modello si riduce il bias e aumenta la varianza. Maggiore è il numero di parametri che si aggiungono ad un modello più gestire la varianza diventa importante. Ciò si osserva in modelli con molti gradi di libertà, come in una regressione lineare con termini polinomiali. Nel caso si abbia un'elevata varianza si possono eliminare delle caratteristiche oppure acquisire più dati. Quando un modello ha un elevato bias l'aggiunta di nuove caratteristiche può migliorare le prestazioni del modello. Impiegare più dati non è d'aiuto.

Comprendere i concetti di bias e la varianza permette di capire il comportamento di un modello ma nella pratica si vuole minimizzare l'errore complessivo. Per far ciò è necessario minimizzare la somma degli errori di generalizzazione, tenendo presente che non esiste un metodo analitico per ottenere un modello avente un grado di complessità ottimale.

Capitolo 2

Elaborazione del linguaggio naturale

L'elaborazione del linguaggio naturale, o *Natural Language Processing* (NLP), è un ramo del Machine Learning applicato a dati destrutturati testuali e per processare il parlato.

Il campo dell'elaborazione del linguaggio naturale fu fondato da Alan Turing nel 1950, con lo sviluppo del test di Turing, introdotto nell'articolo denominato "Computing Machinery and Intelligence" [14]. Il test di Turing misura la capacità di una macchina di dimostrare una forma di intelligenza che sia indistinguibile da quella di un essere umano. Perché una macchina superi il test di Turing è necessario che sia in grado di generare delle risposte in modo che un interlocutore umano non sia in grado di affermare se quelle risposte provengano da una macchina o da un altro uomo.

L'idea proposta da Turing suscitò un intenso dibattito nel nascente campo dell'intelligenza artificiale e spinse i ricercatori a creare i primi modelli di elaborazione del linguaggio naturale. Ancora oggi si cerca un modo di sviluppare una macchina che superi il test.

2.1 Approcci all'elaborazione del linguaggio

Esistono tre approcci principali allo svolgimento di compiti di elaborazione del linguaggio naturale: basati su regole, basati su algoritmi di apprendimento automatico e su reti neurali.

Approccio basato su regole I sistemi basati su regole fanno affidamento su regole del linguaggio realizzate da linguisti e specialisti del settore. Queste regole possono essere formate da espressioni regolari. Sistemi di questo tipo

non hanno alcuna capacità di generalizzazione e richiedono molto lavoro anche per gestire pochi semplici casi. Rappresentano una delle prime tipologie di sistemi NLP realizzati e al giorno d'oggi sono impiegati unicamente per gestire casi particolari.

Approccio basato su apprendimento automatico I sistemi basati sugli algoritmi di apprendimento automatico impiegano meno regole e più dati. Il loro funzionamento si basa un approccio statistico in cui si stimano le distribuzioni di probabilità a partire da un grande corpus annotato. Gli specialisti di dominio eseguono estrazione e selezione di caratteristiche. In seguito si addestrano modelli per compiere singoli compiti di elaborazione del linguaggio naturale, come la classificazione del testo. Sistemi basati su algoritmi di apprendimento automatico generalizzano meglio e sono più facili da costruire e mantenere rispetto ai sistemi basati su regole.

Approccio basato su reti neurali I sistemi basati su reti neurali permettono di superare i limiti dei sistemi basati su algoritmi di apprendimento classici. Le reti neurali svolgono autonomamente la fase di selezione ed estrazione delle caratteristiche. Apprendono le caratteristiche più influenti tramite *representation learning*. I singoli termini sono rappresentati come vettori, detti *word embedding*, in uno spazio continuo. Per poter funzionare adeguatamente questi sistemi richiedono risorse computazionali significative e una grande quantità di dati, non necessariamente etichettati.

Di questi tre tipi di sistemi, quelli basati su reti neurali sono i più potenti, anche grazie all'impiego di reti neurali molto profonde. In ambito commerciale sono ancora molto usati i sistemi di elaborazione del linguaggio naturale che impiegano algoritmi di apprendimento automatico ma recentemente stanno avendo molto successo anche quelli basati sull'impiego di reti neurali profonde [5, p. 29].

2.2 Applicazioni

Alcune delle principali applicazioni di elaborazione del linguaggio naturale riguardano la correzione di errori grammaticali, la traduzione o il riassunto di testi, la generazione e la comprensione del testo, la formulazione di risposte a domande, il riconoscimento del parlato, i *chatbot* e i *voicebot*, la *sentiment analysis*, la generazione di testo e audio, l'estrazione di informazioni e la conversione del testo in parlato e del parlato in testo. Inoltre l'elaborazione del linguaggio naturale può essere applicata al campo dell'*Information Retrieval* (IR) per realizzare motori di ricerca sul web, ad esempio per siti di ecommerce,

ad uso personale, per la ricerca di email e documenti sul PC, e per realizzare sistemi di IR specializzati su un dominio, come quello medico o legale, ad uso interno in un'organizzazione.

Un'altra applicazione, oggetto della tesi, è il *Descriptive Text Mining*. Il *Text Mining* è il processo di estrazione automatica di informazioni di qualità a partire da dati testuali destrutturati. La qualità è definita a partire dalla rilevanza, dall'interesse e dalla novità delle informazioni estratte. Il text mining impiega l'elaborazione del linguaggio naturale per trasformare il testo in dati analizzabili in modo automatico e tecniche di information retrieval per il reperimento e la strutturazione dei dati.

Il descriptive text mining riguarda compiti di text mining che non richiedono predizione e classificazione dei dati ma scoperta di schemi e associazioni. In particolare in compiti di descriptive text mining si impiegano tecniche, come il clustering dei dati, per condurre un'analisi dettagliata che consenta ad esempio di determinare perché accadono determinati fenomeni che emergono dal testo in analisi.

2.3 Problematiche

L'elaborazione del linguaggio naturale impiega dati destrutturati, ovvero privi di un modello che ne descriva la semantica, al contrario di quanto si verifica nei problemi di regressione e classificazione in cui si impiegano dati strutturati. I dati testuali destrutturati hanno un elevato valore perché la conoscenza che si può ricavare da essi è maggiore che nei dati strutturati. Inoltre sono presenti in gran quantità, ma la loro elaborazione efficace e senza perdita di informazione è particolarmente complessa. Ci sono diverse difficoltà legate all'elaborazione del testo dovuta alla sinonimia, ovvero i termini possono avere significati diversi a seconda del contesto, e alla polisemia, ovvero più termini possono avere un significato identico o simile. Ulteriori difficoltà sono legate alla presenza di errori, come refusi o mancanza di punteggiature, e di elementi non previsti dalla lingua, come hashtag, emoticon o abbreviazioni. Inoltre il testo può essere scritto in un modo da non essere facilmente interpretabile. Per un calcolatore è difficile riconoscere l'ironia e la satira.

2.4 Compiti di elaborazione del linguaggio

Per poter realizzare le applicazioni accennate e affrontare le difficoltà insite nei dati testuali destrutturati è necessario effettuare una serie di compiti di elaborazione del linguaggio.

In generale si estraggono i singoli termini che compongono ciascuna frase di un documento in singole unità di significato, dette *segmenti*. In base alle applicazioni da realizzare si assegnano dei metadati a ciascun segmento, come la classe grammaticale di appartenenza e le sue relazioni con gli altri segmenti di una frase.

Segmentazione Si tratta della prima operazione compiuta generalmente su di un testo. Consiste nella segmentazione, o *tokenization* , del testo negli elementi sintattici di base. Si distingue tra *sentence tokenization*, se avviene a livello di singole frasi, *word tokenization* se avviene a livello di singole parole o *subword tokenization* se avviene a livello di singoli caratteri o gruppi di caratteri. In ogni caso si devono gestire anche altri elementi oltre alle parole come segni di punteggiatura, simboli e numeri. Inoltre esistono casi particolari come la gestione di parole unite da un trattino o di contrazioni e fusioni, ad esempio isn't e is n't. Per effettuare una segmentazione accurata in genere si integrano regole e modelli specifici della lingua e del dominio analizzati. La segmentazione sarà ulteriormente approfondita in sezione 2.5.

Etichettatura delle parti del discorso L'etichettatura delle parti del discorso, detta anche *Part of speech Tagging*, è una procedura attraverso cui si associa a ciascuna parola estratta tramite segmentazione del testo una classe grammaticale, detta *Part of Speech* (PoS). Questa operazione deve tenere conto del contesto di ogni parola, in quanto una stessa parola può avere diversi PoS. Il PoS è utile se si vogliono filtrare parole di tipo specifico oppure se elaborazioni successive dipendono da questa fase. Questo compito può non essere necessario, ad esempio nel caso in cui si vogliano estrarre le parole chiave di un documento indipendentemente dalla loro posizione nella frase.

In figura 2.1 si osserva un albero sintattico che evidenzia le parti del discorso di un frase di esempio. Per effettuare l'etichettatura si possono impiegare dei corpus detti *Treebank*, realizzati da linguisti. Uno dei primi grandi corpus è il Treebank Penn, pubblicato negli anni 90. Per una lista dei PoS impiegati nel Treebank Penn si rimanda a [15]. In genere i PoS tagset sono specifici per una lingua. Nel tentativo di proporre uno standard che fosse applicabile a più lingue sono stati proposti dei PoS tagset universali, come in [16].

Analisi delle dipendenze L'analisi delle dipendenze, detta anche *dependency parsing*, è la fase in cui si identificano le relazioni tra i singoli segmenti della frase. In questo modo è possibile determinare come è strutturato il testo nella frase. In figura 2.2 è visibile un esempio di annotazione delle relazioni tra i segmenti della frase fornita in input. Per una tassonomia delle relazioni di dipendenza universali si fa riferimento a [17].

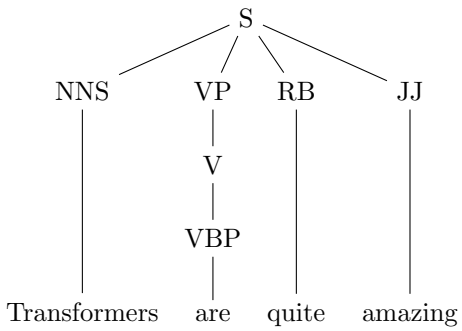


Figura 2.1: Albero sintattico in cui sono evidenziate le PoS identificate a partire dalla frase fornita in input.

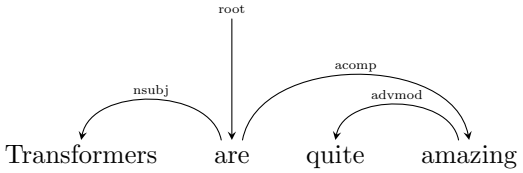


Figura 2.2: Grafo delle dipendenze per indicare le relazioni tra i segmenti della frase fornita in input.

Chunking Il *chunking* è il processo con cui si combinano più segmenti in un'unico segmento. In questo modo si possono raggruppare segmenti relativi allo stesso concetto come quelli riferiti al nome di una località o di una persona, in modo da poter lavorare su un insieme di segmenti ridotto e semplificare le fasi successive di analisi del testo.

Lemmatizzazione La lemmatizzazione consiste nel sostituire ciascuna parola con il suo lemma, ovvero con la sua forma base. La lemmatizzazione permette di normalizzare tutte le forme flesse in cui si può trovare una parola. Questo processo consente di raggruppare gruppi di termini simili, riducendo la dimensionalità dello spazio senza perdita di informazione rilevante e riducendo la dimensione del vocabolario. Gli algoritmi di lemmatizzazione sono complessi e richiedono una conoscenza approfondita della lingua e del dominio in analisi.

Stemming Lo *stemming* permette di ottenere a partire da ciascun parola la sua radice morfologica. Non è detto che la radice morfologica sia una parola di senso compiuto. Rispetto alla lemmatizzazione richiede algoritmi più semplici ed è più efficiente. Inoltre può individuare correlazioni tra lemmi simili, come nel caso di nome e verbo con la stessa radice. Ha lo svantaggio di unire termini non correlati ed è perciò più approssimativo. Può portare ad una perdita di

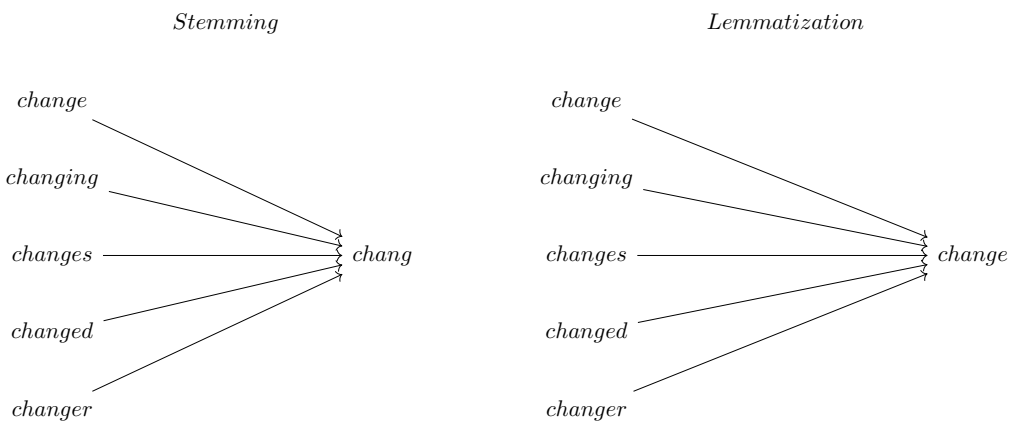


Figura 2.3: Confronto fra stemming e lemmatizzazione.

precisione. Ha efficacia variabile in base alla lingua a cui è applicato. In figura 2.3 si osserva come talvolta lo stemming dia come risultato delle parole senza senso. Per questa ragione in genere si preferisce impiegare la lemmatizzazione, anche se si tratta di un processo più costoso.

Casefolding Il *casefolding* consiste nel convertire tutte i segmenti in un documento o in una frase in minuscolo o in maiuscolo. Si tratta di un metodo basilare per uniformare segmenti che rappresentano la stessa parola anche se hanno forma diversa. Non sempre è una buona idea effettuare il casefolding. Ad esempio modelli particolarmente flessibili, come quelli che impiegano reti neurali profonde, possono essere in grado di catturare l'informazione legata all'impiego di maiuscole e minuscole.

Rimozione di stopword Le *stopword* sono le parole che considerate singolarmente non danno informazioni sulla semantica del testo. Alcuni esempi sono articoli, preposizioni e congiunzioni. In genere le stopword sono rimosse tramite un'apposita lista predefinita, detta *stoplist*. La stoplist cambia a seconda della lingua dei documenti. Non sempre conviene rimuovere le stop word. Ad esempio possono essere utili nel caso della sentiment analysis.

Finora sono stati descritti compiti che stanno alla base di applicazioni più complesse. Di seguito si riportano due compiti che possono essere essere fini a sé stessi oppure essere impiegati in applicazioni più complesse.

Named Entity Recognition La *named entity recognition* (NER) è il processo con cui si assegnano delle etichette a entità note, come nomi di persone, di organizzazioni, di località o date. In figura 2.4 è visibile un esempio di NER.

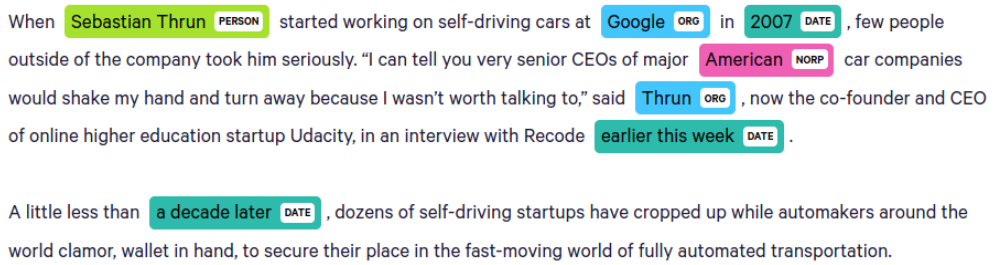


Figura 2.4: Esempio di named entity recognition.
https://explosion.ai/demos/displacy-ent

Entity linking L'*Entity linking* è il processo con cui si disambiguano le entità appoggiandosi a delle basi di dati esterne. La disambiguazione è difficile da eseguire dato che richiede di determinare il contesto in cui appare un termine e richiede la conoscenza della struttura delle singole frasi. Insieme alla NER è un processo che sta alla base delle applicazioni nel campo dell'Information Retrieval.

2.5 Segmentazione

La segmentazione è un processo ampiamente impiegato al di fuori del campo dell'elaborazione del linguaggio naturale. Ad esempio si tratta di un processo fondamentale per la realizzazione di compilatori. Si identificano le parole chiave, in modo da ottenere una sequenza di segmenti a partire dai caratteri ASCII/Unicode. Questi segmenti sono poi impiegati per costruire degli alberi sintattici.

Nel campo dell'elaborazione del linguaggio naturale la segmentazione genera dei vettori one hot. In input si fornisce del testo e in output si ottiene una rappresentazione vettoriale che sarà poi impiegata per generare matrici termini documenti o word embedding.

Esistono diversi approcci per la segmentazione basati sulla generazione di segmenti a partire da singoli caratteri o da singoli termini. Nel primo caso un termine è diviso in più segmenti, con potenziale perdita di informazione e il vantaggio di un insieme di segmenti, o *vocabolario*, più piccolo. Nel secondo caso si possono rappresentare i termini in modo accurato ma sarà necessario un vocabolario più grande. La dimensione del vocabolario determina le dimensioni della matrice di embedding e ha quindi un impatto sui costi di computazione.

Quando si effettua una segmentazione si vuole massimizzare la quantità di informazione di ciascun segmento e si vuole minimizzare la dimensione del vocabolario. Gli algoritmi di segmentazione più recenti impiegano una fase di addestramento preliminare per trovare il compromesso ottimale fra

segmentazione a livello di carattere e di termine. Uno dei fattori che può influire su questo compromesso è la lingua a partire da cui si effettua la segmentazione. Ad esempio nel cinese i singoli caratteri hanno più informazione che un carattere in una lingua latina.

La fase di addestramento preliminare permette di apprendere i criteri con cui dividere i termini. L'obiettivo è trovare del testo ricorrente nel corpus e acquisire l'informazione che apporta come un segmento. Se uno schema non si ripete abbastanza spesso non è incluso come segmento. Se un termine non è abbastanza frequente è scomposto in gruppi di caratteri, altrimenti è mantenuto come termine. Ad esempio dato il termine "annoyingly" ci si aspetta che sia relativamente raro e che si incontrino più spesso i sottotermini "annoying" e "ly". Perciò si scarta il termine intero e si includono nel vocabolario i sottotermini. Inoltre si considera che una segmentazione a livello di singoli caratteri permette al modello di elaborare termini mai incontrati scomponendoli nelle unità elementari che già conosce e di impiegare un vocabolario di dimensione ridotta grazie alla maggior capacità di generalizzazione. Ad esempio dati i termini "long", "longer", "longest" e "short", "shorter" e "shortest", si possono impiegare quattro segmenti, "long", "short", "xxer", "xxest" e non sei per catturare la stessa quantità d'informazione. Inoltre è possibile riconoscere nuovi termini come "sweeter" e "sweetest", partendo solo "sweet". In questo modo si riduce il numero di segmenti [UNK], relativi a parole non riconosciute dal modello in quanto non incluse nel vocabolario, ovvero *out of vocabulary* (OOV), e si ha un minor perdita d'informazione.

Nel caso si impieghi un corpus di documenti relativi ad un dominio specifico, medico o legale ad esempio, conviene addestrare un algoritmo di segmentazione apposito piuttosto che impiegare un modello generico. Alcuni tra i migliori programmi di segmentazione a disposizione sono *Byte pair encoding* [18], impiegato in GPT-2, *WordPiece* [19], impiegato in BERT, e *SentencePiece* [20].

2.6 Information Retrieval

Una delle principali applicazioni del NLP è nel campo dell'Information Retrieval. Data una collezione di documenti, o *corpus*, si vuole realizzare un sistema che effettui *ad hoc retrieval*. In altri termini si vuole realizzare un sistema che restituisca i documenti più rilevanti a partire da un'interrogazione effettuata da un'utente. La *rilevanza* è definita in base alla capacità di un documento di soddisfare il *bisogno informativo* dell'utente, ovvero l'acquisizione di conoscenza su un determinato argomento. Per valutare l'efficacia di un sistema di IR si impiegano metriche come la precisione e il richiamo, già introdotte in sezione 1.8.1 nell'ambito dei problemi di classificazione.

2.6.1 Modello booleano standard

Uno dei più semplici sistemi di IR impiega operatori booleani standard, come AND, OR o NOT. Un sistema di IR di questo tipo è detto *Modello Booleano Standard*.

Nel modello booleano standard, a seguito di un'interrogazione, i documenti sono divisi tra rilevanti e non. I documenti rilevanti sono restituiti senza un ordine specifico. Ad un generico termine di ricerca si associa l'insieme di documenti che contiene quel termine. Si ottiene in questo modo l'insieme dei documenti rilevanti. Per gestire la combinazione dei termini si eseguono operazioni insiemistiche. Ad esempio A AND B, detta *congiunzione logica*, corrisponde all'intersezione tra l'insieme dei documenti rilevati per A e quelli per B. Ad A OR B , detta *disgiunzione logica*, corrisponde l'unione tra gli insiemi dei documenti.

Il modello booleano standard è semplice da implementare ma tende a restituire o troppi o troppo pochi documenti. Inoltre non permette di ordinare il risultato di un'interrogazione in modo significativo. In ogni caso la ricerca tramite operatori booleani è ancora oggi ampiamente impiegata, soprattutto per ricerche da esperti di dominio come medici e assistenti legali. Una delle ragioni è legata al grado di controllo e trasparenza superiore offerto dalle interrogazioni booleane rispetto alla ricerca a testo libero. Il dataset ADE impiegato per lo svolgimento del progetto è stato costruito tramite estrazione casuale di 3000 dei 30000 documenti ottenuti effettuando un'interrogazione booleana alla banca dati PubMed [2, p. 2].

2.6.2 Vector Space Model

Per rimediare ai limiti del modello booleano è necessario stabilire il grado di rilevanza di ciascun documento restituito dal risultato di una ricerca. A questo scopo si impiega un sistema in grado di associare un punteggio ad una coppia (interrogazione, documento) e si rappresentano documenti, termini e interrogazioni su un unico spazio vettoriale tramite il *Vector Space Model*.

In un sistema che impiega il vector space model si effettuano interrogazioni formate da testo libero. Si assegna un punteggio di rilevanza ai risultati della ricerca considerando le interrogazioni come insiemi di termini.

Un limite del vector space model sta nell'impossibilità di impiegare espressioni booleane. Ad esempio dati due termini non è possibile richiedere che appaiano entrambi o almeno uno nel risultato. Inoltre il vector space model effettua un'analisi lessicale dei termini ma non semantica.

2.6.3 Topic model

I *Topic model* consentono di superare i limiti del vector space model e del modello di ricerca booleana, che si limitano ad un'analisi lessicale dei termini.

I topic model forniscono delle relazioni semantiche tra interrogazioni, documenti e termini descrivendo ciascun tema come una distribuzione di termini pesata in modo probabilistico e modellando ciascun documento come una distribuzione su tutti i temi attraverso un processo generativo. Si basa sull'intuizione per cui in documenti che trattano di un certo tema determinati termini appaiono più spesso.

Gli algoritmi per il topic modeling cercano gruppi di termini simili. Sulla base delle statistiche relative alla presenza di termini in ciascun documento si possono determinare gli argomenti trattati in un documento e in quale proporzione questi argomenti siano trattati in ciascun documento. In questo modo si aggiunge un livello di astrazione tra il documento e i termini esatti presenti nel documento. I topic model permettono di recuperare i termini che l'autore di un documento avrebbe potuto usare in base ai termini che sono stati scelti. Alcuni dei topic model fondamentali saranno approfonditi in sezione 2.8.

L'impiego dei topic model permette di migliorare i sistemi di ad hoc retrieval, ad esempio impiegando un indice ottenuto a partire da *Latent Semantic Indexing* [21] o *Probabilistic Latent Semantic Indexing* [22]. In alternativa i topic model si possono impiegare per realizzare sistemi di ricerca che consentono di visualizzare e raffinare iterativamente una ricerca su un ampio corpus di documenti tramite restituzione di documenti che trattano temi di interesse.

2.7 Rappresentazione di dati testuali

Uno dei metodi più semplici per rappresentare una collezione in un formato che può essere impiegato dalle macchine è creare una *matrice di incidenza*, detta anche *matrice termini documenti*. L'elemento in posizione (t, d) è 1 se il documento d contiene il termine alla riga t, altrimenti è 0. La matrice avrà dimensioni $M \times N$, dove M indica il numero di termini e N il numero di documenti. In figura 2.5 è mostrato un esempio di matrice di incidenza.

2.7.1 Bag of Words e N-gram

Nel vector space model la matrice termini documenti è costituita da *Bag of Words* (BoW). Una BoW è una rappresentazione in forma di multiinsieme dei termini contenuti in un documento.

Il modello BoW è un caso specifico dei modelli *n-gram* con $n = 1$, visto che si considerano i singoli termini presenti nel testo. Un n-gram è una sequenza

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
Antony	1	1	0	0	0	1	
Brutus	1	1	0	1	0	0	
Caesar	1	1	0	1	1	1	
Calpurnia	0	1	0	0	0	0	
Cleopatra	1	0	0	0	0	0	
mercy	1	0	1	1	1	1	
worser	1	0	1	1	1	0	
...							

Figura 2.5: Illustrazione di una matrice termini documenti.
Fonte: Introduction to Information Retrieval [23, p. 4]

di n termini. Ad esempio per $n = 2$ si ha un bigram, per $n = 3$ un trigram. Bigram e trigram rappresentano un buon compromesso tra l'incremento della dimensione del vocabolario e la quantità d'informazione aggiuntiva che può essere catturata includendo termini composti.

Il modello BoW non permette di considerare l'ordine con cui si presentano i termini nel testo. Perciò il processo di conversione del testo in BoW, detto *vettorizzazione*, è un metodo di strutturazione del testo con perdita di informazione. In particolare la rappresentazione del testo tramite modello BoW non permette di gestire polisemia e sinonimia. In sezione 2.8 si osserverà che l'impiego di topic model consente di catturare relazioni di sinonimia tra i termini. Per realizzare delle rappresentazioni vettoriali del testo che tengano conto della polisemia si impiegano dei modelli di linguaggio neurali, approfonditi nel capitolo 3.

2.7.2 Term weighting

L'impiego di BoW nell'ambito del vector space model permette di associare ad un termine di un'interrogazione t un peso che indichi quanto è rilevante il documento d. Un primo criterio per assegnare un peso consiste nell'impiegare la frequenza di t in d. Il criterio è detto *term frequency*, si denota con $\mathrm{tf}_{t,d}$, ed è definito in equazione (2.1).

$$\mathrm{tf}_t = \log\big(1 + f(t,d)\big) \tag{2.1}$$

Dato che non tutti termini hanno la stessa importanza in un documento si impiega un ulteriore criterio per ridurre l'influenza dei termini meno importanti. Si misura la frequenza df_t di t nel corpus e si riduce il peso di t di un fattore che cresce con df_t. Il criterio è detto *inverse document frequency* ed è definito come in equazione (2.2).

$$\mathrm{idf}_t = \log\left(\frac{N}{df_t}\right), \tag{2.2}$$

dove N è il numero di documenti nella collezione. L'idf di un termine raro è elevato, l'idf di un termine frequente ha un valore basso.

Uno dei possibili modi per definire un peso da assegnare ai termini organizzati in BoW consiste nell'impiegare uno schema di *term weighting* che utilizzi sia tf che idf. Il tf è considerato un fattore locale, mentre idf è un fattore globale. Il tf.idf è definito come in equazione (2.3).

$$\mathrm{tf.idf}(t,d) = \log\big(1 + f(t,d)\big) \cdot \log\left(\frac{N}{df_t}\right) \tag{2.3}$$

Il $\mathrm{tf.idf}_{t,d}$ assegna ad un termine t nel documento d un peso che è maggiore se t ha molte occorrenze in un ristretto numero di documenti. Il peso sarà minore se il termine ha meno occorrenze in un documento oppure se è presente in molti documenti. Nel caso estremo il tf.idf di un termine presente in ogni documento è nullo.

Una variante del tf.idf che sarà impiegata per lo svolgimento del progetto, in accordo con la metodologia POIROT, utilizza l'inverso dell'entropia di Shannon come fattore locale. Il calcolo dell'entropia avviene sulla matrice termini documenti tdm. In equazione (2.4) è definita questa variante.

$$w_{t,d} = \log\big(1 + f(t,d)\big) \cdot \big(1 - \mathrm{shannonEntropy(tdm)}\big) \tag{2.4}$$

La scelta di uno schema di weighting ha molta importanza in quanto determina il modo in cui si costruirà una spiegazione dei fenomeni. Ciò accade perché lo schema di weighting determina la norma delle rappresentazioni vettoriali dei termini.

Esistono diversi altri schemi di term weighting. Per un confronto tra i diversi schemi e la loro efficacia si rimanda a [24].

2.7.3 Word Embedding

A partire dalla rappresentazione della matrice termini documenti tramite BoW e tf.idf, impiegando il vector space model, si ottiene una matrice sparsa avente una dimensione per termine. Per codificare il significato di un termine in un formato che sia meno costoso da memorizzare e manipolare, e che consenta di catturare sia le relazioni sintattiche che semantiche tra termini si impiega una rappresentazione vettoriale tramite word embedding.

Gli algoritmi di word embedding mappano i termini in input come vettori in output apprendendo la rappresentazione degli embedding in modo non supervisionato e realizzando una *rappresentazione distribuita* dei termini. Questo

perché ciascun termine è rappresentato da una distribuzione di pesi in uno spazio continuo. Il mapping effettuato da questi algoritmi consiste nel passaggio dallo matrice sparsa termini documenti ad una matrice densa di dimensione molto inferiore. In output si ottiene una *matrice di embedding*. Nel capitolo 3 si approfondiranno diversi modelli che possono essere impiegati per la generazione di embedding.

2.7.4 Misurare la similarità

Per formulare il punteggio di rilevanza di un documento $\mathbf{d}$ rispetto ad un'interrogazione $\mathbf{q}$ che contiene un insieme di termini $\mathbf{q} = \{t_1, \ldots, t_n\}$ si può impiegare l'equazione (2.5).

$$\text{Score}(\mathbf{q}, \mathbf{d}) = \sum_{t \in q} \text{tf.idf}_{t,d} \tag{2.5}$$

In alternativa, se si impiega il vector space model, si possono rappresentare documenti e termini come vettori nello stesso spazio. Perciò per stabilire quali siano i documenti più rilevanti si può utilizzare un prodotto scalare tra vettori per misurare la similarità tra i termini dell'interrogazione e i documenti. A questo scopo si può impiegare la *similarità coseno*, definita in equazione (2.6).

$$\text{sim}(\mathbf{q}, \mathbf{d}) = \frac{\mathbf{q} \cdot \mathbf{d}}{\|\mathbf{q}\| \, \|\mathbf{d}\|} \tag{2.6}$$

Per vettori con solo valori positivi la similarità coseno è compresa in $[0, 1]$. Altrimenti in $[-1, 1]$. Se $\text{sim}(\mathbf{q}, \mathbf{d}) = \cos(\theta) = 1$ allora i vettori hanno la stessa direzione e l'angolo θ compreso tra essi è pari a $0°$. Se $\text{sim}(\mathbf{q}, \mathbf{d}) = \cos(\theta) = 0$ allora i vettori sono perpendicolari l'uno all'altro.

La similarità coseno può essere impiegata anche per gli spazi distinti in cui si rappresentano documenti e termini tramite word embedding.

La ragione per cui si impiega la similarità coseno rispetto ad un semplice confronto del modulo dei vettori sta nella possibilità di normalizzare i vettori $\mathbf{q}$ e $\mathbf{d}$. Ciò permette di misurare la distribuzione relativa dei termini anche se in assoluto le frequenze potrebbero essere molto diverse a causa della lunghezza dei documenti.

Nel caso della polisemia la similarità coseno sarà maggiore della similarità reale dato che si attribuirà valore ad un termine fuori contesto. Si avrà perciò una restituzione di documenti irrilevanti e una conseguente diminuzione della precisione. Nel caso della sinonimia la similarità coseno sarà minore della similarità reale in quanto non sarà rilevata l'importanza di un termine associato ad uno rilevante. Perciò non si recupereranno documenti rilevanti e si avrà una diminuzione del richiamo.

2.8 Modelli di linguaggio

I *modelli di linguaggio* sono uno dei concetti fondamentali alla base dei topic model e della generazione di word embedding. Un modello di linguaggio è una funzione che prende in input una sequenza di termini e restituisce una distribuzione di probabilità di tutte le possibili parole che possono essere il termine successivo di quella sequenza, denotata con $p(w_1, w_2, \cdots, w_n)$.

I modelli di linguaggio formulano delle previsioni a partire dalla conoscenza della lingua e delle sue caratteristiche. Questa conoscenza è estratta dai dati, senza bisogno di etichettarli esplicitamente. Perciò è disponibile per l'addestramento un'enorme quantità di dati non etichettati. Questa è la ragione per cui i modelli di linguaggio costituiscono le fondamenta dei più potenti sistemi di elaborazione del linguaggio naturale. Nel capitolo 3 si approfondiranno i modelli di linguaggio n-gram.

Un modello di linguaggio unigram, ovvero che impiega l'assunzione BoW, assume che ciascun termine nella sequenza sia indipendente dagli altri. Il calcolo della probabilità avviene come indicato in equazione (2.7). I modelli di linguaggio che saranno approfonditi nella sezione successiva si basano sull'assunzione BoW.

$$p(w_1, w_2, \cdots, w_n) = p(w_1)p(w_2)\cdots p(w_n) \tag{2.7}$$

2.9 Topic Modeling

Il Topic modeling consiste nell'applicare un modello di linguaggio statistico a una collezione di documenti per far emergere le similarità semantiche tra termini e documenti in uno spazio vettoriale latente. Come modelli di linguaggio si possono impiegare LSA, pLSA e LDA.

Un modello di linguaggio applicato per il topic modeling è detto *topic model*. Nel caso di pLSA, LDA e modelli basati su LDA si impiega il termine topic model probabilistico [25].

Uno dei primi modelli ad essere proposti per il topic modeling è stata la *Latent Semantic Analysis* (LSA), nel 1998 [26]. In seguito fu proposta da Hofmann nel 1999 la *probabilistic Latent Semantic Analysis* (pLSA) [22]. Uno dei modelli più diffusi è la *Latent Dirichlet Allocation* (LDA) [27].

2.9.1 Latent Semantic Analysis

La tecnica LSA effettua un mapping della matrice termini documenti, eventualmente pesata, in uno spazio semantico a bassa dimensionalità latente. Il mapping avviene applicando la *Singular Value Decomposition* (SVD), una

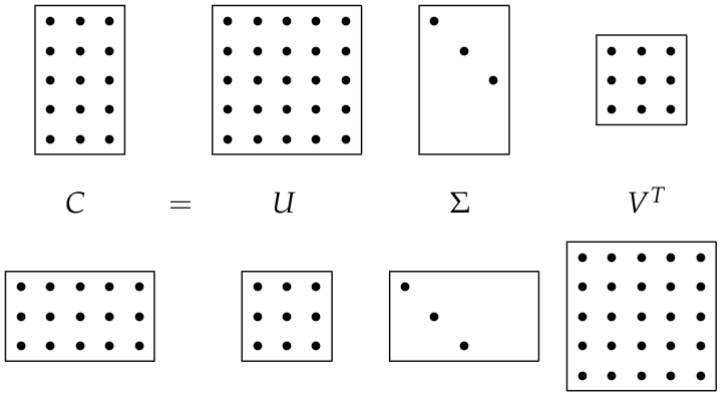


Figura 2.6: Illustrazione della singular value decomposition. In alto è mostrato il caso in cui la matrice C ha $t > d$, in basso il caso in cui $t < d$.

Fonte: Introduction to Information Retrieval [23, p. 42]

tecnica di fattorizzazione delle matrici. Applicando SVD si può risolvere il problema dell'approssimazione *low rank* scegliendo un rango k minore del rango della matrice termini documenti $\mathbf{C} \in \mathbb{R}^{t \times d}$. L'obiettivo è minimizzare la discrepanza $\mathbf{X} = \mathbf{C} - \mathbf{C}_k$ tra $\mathbf{C}$ e la matrice a dimensionalità ridotta $\mathbf{C}_k$ di rango k tramite la norma di Frobenius. La definizione della norma di Frobenius di una matrice $\mathbf{X} \in \mathbb{R}^{m \times n}$ è fornita in equazione (2.8).

$$\|\mathbf{X}\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |x_{ij}|^2} \tag{2.8}$$

Si ottiene come risultato la fattorizzazione di $\mathbf{C}$ nel prodotto di tre matrici, come mostrato in equazione (2.9) e in figura 2.6.

$$\underset{t \times d}{\mathbf{C}} = \underset{t \times k}{\mathbf{U}} \times \underset{k \times k}{\mathbf{\Sigma}} \times \underset{k \times d}{\mathbf{V}^\top} \tag{2.9}$$

Dove $\mathbf{U}$ contiene gli autovettori destri di $\mathbf{C}\mathbf{C}^\top$, ovvero la matrice quadrata che indica la similarità tra termini, e $\mathbf{V}$ contiene gli autovettori sinistri di $\mathbf{C}^\top\mathbf{C}$, ovvero la matrice quadrata che indica la similarità tra documenti.

Gli autovalori sono i valori della matrice diagonale $\mathbf{\Sigma}$. Sono disposti in ordine decrescente sulla diagonale e sono detti anche *valori singolari*. Ciascun autovalore indica la quantità d'informazione catturata da ciascun asse. In particolare ogni autovalore è la varianza dei dati sulla dimensione che rappresenta. I primi autovalori catturano una quantità di informazione non lineare rispetto

al numero dei vettori stessi. Bastano pochi vettori iniziali per catturare la maggior parte della variabilità nei dati. La distribuzione degli autovalori segue la legge di potenza, come si osserva in figura 4.2.

Il costo computazionale dell'applicazione di questa tecnica è pari a quello di applicare SVD ad una matrice $m \times n$, ovvero $O(\min(mn^2, m^2n))$. Impiegando tecniche per il calcolo approssimato della fattorizzazione SVD, come *Randomized SVD* [28], si possono ridurre i tempi di calcolo di diversi ordini di grandezza.

2.9.2 Probabilistic Latent Semantic Analysis

La tecnica pLSA impiega un approccio probabilistico piuttosto che l'utilizzo di SVD per effettuare il mapping in uno spazio a bassa dimensionalità. La scoperta di strutture tematiche nascoste in una collezione di documenti avviene fattorizzando la matrice termini documenti, realizzando una riduzione della dimensionalità non supervisionata.

L'obiettivo di pLSA è massimizzare la probabilità di predire $P(w, d)$ in ogni elemento della matrice termini documenti originale determinando i valori di alcuni parametri tramite stima della massima verosimiglianza. In equazione (2.10) sono mostrate due diverse formulazioni del problema. La prima è detta *asimmetrica*, mentre la seconda è *simmetrica*. La formulazione simmetrica è simile alla LSA. Dato che la pLSA è basata su distribuzioni di probabilità, le matrici risultanti dalla fattorizzazione sono non negative e normalizzate.

$$\begin{aligned} P(w, d) &= P(d) \sum_t P(t \mid d) P(w \mid t) \\ &= \sum_t P(t) P(d \mid t) P(w \mid t) \approx U \Sigma V^T \end{aligned} \tag{2.10}$$

2.9.3 Latent Dirichlet Allocation

LDA è un modello statistico generativo che rappresenta un'estensione di LSA. I documenti sono interpretati come una distribuzione di distribuzioni di temi e la distribuzione di temi si assume che abbia una distribuzione di Dirichlet sparsa. Intuitivamente ciò consente alla LDA di riflettere la proprietà dei documenti reali di trattare solo una frazione dei temi potenzialmente disponibili e dei temi di essere formati solo da un piccolo insieme di termini frequenti.

Un modello di LDA permette di rimediare alla facilità con cui pLSA va in overfitting in dataset di piccole dimensioni. Un modello di LDA si addestra per fornire in output le probabilità $P(t \mid d)$ e $P(w \mid t)$, in modo analogo alla formulazione asimmetrica di pLSA. Una forma simmetrica per LDA si può ricavare calcolando $P(t)$ e $P(d \mid t)$. $P(t)$ si ottiene tenendo conto dell'importanza

di ciascun termine nella distribuzione $P(t \mid d)$ di ciascun documento d ed è mostrata in equazione (2.11), mentre la seconda può essere derivata applicando la regola di Bayes, come mostrato in equazione (2.12).

$$P(t) = \frac{\sum_d P(t \mid d)}{|t|} \tag{2.11}$$

$$P(d \mid t) = \frac{P(t \mid d)P(d)}{P(c)} \tag{2.12}$$

2.10 Selezione di caratteristiche

La riduzione della dimensionalità è fondamentale in problemi che impiegano dati testuali. In una collezione di documenti si ha un numero di caratteristiche molto elevato, dovuto alla dimensione del vocabolario di termini impiegato. Per ridurre la dimensionalità dei dati si eliminano le caratteristiche meno rilevanti tramite la loro selezione. Ciò consente di rimuovere i termini che generano rumore.

Due delle principali misure impiegate per la selezione di caratteristiche sono la *mutua informazione* e il *test chiquadro*, approfondite di seguito.

2.10.1 Mutua informazione

La mutua informazione misura la reciproca dipendenza di un termine t da una classe c. Indica quanta informazione la presenza o assenza di un termine t contribuisca alla corretta classificazione di c. È definita come in equazione (2.13).

$$I(U;C) = \sum_{e_t \in \{1,0\}} \sum_{e_c \in \{1,0\}} P\left(U = e_t, C = e_c\right) \log_2 \frac{P\left(U = e_t, C = e_c\right)}{P\left(U = e_t\right) P\left(C = e_c\right)} \tag{2.13}$$

Dove U è la variabile casuale relativa al documento, può assumere il valore $e_t = 1$ se il documento contiene il termine t o $e_t = 0$ in caso contrario. C è la variabile casuale relativa alla classe, assume valore $e_c = 1$ se il documento è nella classe c o valore $e_c = 0$ viceversa.

Se la distribuzione di un termine in una classe è la stessa che nel resto della collezione allora $I(U, C) = 0$. Il valore della mutua informazione è massimo se un termine è un indicatore perfetto dell'appartenza di documento ad una determinata classe. Ciò significa che il termine può essere presente in un documento se e solo se il documento appartiene alla classe.

2.10.2 Test chiquadro

In statistica il test χ^2 è applicato per verificare l'indipendenza di due eventi A e B. Gli eventi sono considerati indipendenti se $P(AB) = P(A)P(B)$, o equivalentemente se $P(A \mid B) = P(A)$ e $P(B \mid A) = P(B)$. Nell'ambito della selezione delle caratteristiche i due eventi sono l'occorrenza di un termine t e l'occorrenza di una classe c. Il test chiquadro è definito in (2.14).

$$X^2(\mathbb{D}, t, c) = \sum_{e_t \in \{0,1\}} \sum_{e_c \in \{0,1\}} \frac{(N_{e_t e_c} - E_{e_t e_c})^2}{E_{e_t e_c}} \tag{2.14}$$

Dove N rappresenta le frequenze osservate, E le frequenze attese e χ^2 è la misura della divergenza delle frequenze N ed E nella collezione di documenti $\mathbb{D}$. e_t ed e_c sono definite come nel caso della mutua informazione. Se il valore di χ^2 è grande allora l'ipotesi di indipendenza, che implica frequenze non omogenee, deve essere respinta. Perciò le frequenze sono generate da una stessa distribuzione e l'occorrenza o l'assenza del termine rende l'occorrenza della classe più o meno probabile ed è quindi una caratteristica utile. Per stabilire il valore di soglia di χ^2 si osserva la distribuzione omonima con v gradi di libertà. Il numero di gradi di libertà dipende dal numero di righe i e colonne j che compongono la matrice delle frequenze osservate e attese. In particolare $v = (i-1)(j-1)$. A seconda del valore assunto da χ^2 si ha un corrispondente p-value che indica il grado di confidenza e permette di respingere o meno l'ipotesi nulla, secondo cui l'omogeneità tra le frequenze di E e N non è statisticamente significativa.

Il test chiquadro ha diversi limiti. È inadeguato con pochi valori osservati e con valori piccoli. In particolare non può essere applicato se il numero di dati osservati è minore di trenta e la loro somma è minore di duecento. Inoltre sono necessari metodi di correzione quando i dati osservati sono inferiori a cinquanta e quelli attesi minori di cinque.

Rispetto alla mutua informazione il metodo chiquadro può rifiutare l'ipotesi di indipendenza di un termine t anche se questo ha poca informazione riguardo la classe c se t è un termine raro. Perciò l'occorrenza di un termine raro in una classe è considerata statisticamente significativa. Dal punto di vista del metodo della mutua informazione un termine raro non ha molta informazione.

Sia il metodo chiquadro che la mutua informazione sono metodi golosi in quanto possono selezionare caratteristiche che non aggiungono alcuna informazione alle caratteristiche selezionate precedentemente. Nonostante ciò possa introdurre delle ridondanze che riducono l'accuratezza, l'impiego di metodi non golosi è raro a causa dei costi computazionali elevati.

Capitolo 3

Modelli di linguaggio neurali

Non è possibile comprendere in modo approfondito la semantica di un testo se non si tiene conto della sua natura sequenziale. I termini in una frase, in un paragrafo e in un documento sono scritti in sequenza e non basta considerare delle permutazioni casuali. I modelli di linguaggio introdotti nel capitolo precedente si basano sull'assunzione BoW e perciò non considerano l'ordine dei termini. In questo capitolo si studia come le reti neurali possono essere impiegate per realizzare modelli di linguaggio in grado di generare rappresentazioni vettoriali che tengano conto di sinonimia e polisemia.

3.1 Modelli di linguaggio n-gram

Una sequenza di testo formata da T segmenti $x_1, x_2, \ldots, x_T$ può essere considerata come una serie di osservazioni in t istanti temporali. Un modello di linguaggio ideale genera del testo in linguaggio naturale, estraendo un segmento x_t alla volta all'istante t da una distribuzione di probabilità condizionata $P(x_t \mid x_{t-1}, \ldots, x_1)$. In altri termini per generare del testo in modo sensato si condiziona il segmento da generare all'istante t a partire dai frammenti di testo generati precedentemente e si calcola la probabilità congiunta $P(x_1, x_2, \ldots, x_T)$.

Ad esempio in equazione (3.1) si effettua il calcolo della probabilità di una sequenza di quattro termini.

$$\begin{aligned} P(\text{transformers, are, quite, amazing}) = P(\text{transformers}) \\ P(\text{are} \mid \text{transformers}) P(\text{quite} \mid \text{transformers, are}) \\ P(\text{amazing} \mid \text{transformers, are, quite}) \end{aligned} \tag{3.1}$$

Nel caso generale si impiega l'equazione (3.2).

$$P(x_1, x_2, \ldots, x_T) = \prod_{t=1}^{T} P(x_t \mid x_{t-1}, \ldots, x_1) \tag{3.2}$$

Esistono due strategie principali per stimare $P(x_t \mid x_{t-1}, \ldots, x_1)$ in modo efficiente. La prima consiste nell'utilizzare un numero limitato di n osservazioni, considerando solo i $t - n$ segmenti precedenti nel calcolo della probabilità condizionata. Si ha in questo caso un modello *autoregressivo* in cui si effettua il calcolo di $P(x_t \mid x_{t-1}, \ldots, x_{t-n+1})$.

Un modello di linguaggio in cui la probabilità condizionata di un termine x_t all'istante t dipende unicamente dagli $n-1$ termini precedenti è detto modello n-gram. Per includere gli effetti di segmenti che si trovano prima del tempo $t-(n-1)$ nel calcolo della probabilità x_t è necessario incrementare n. Ciò determina una crescita esponenziale del numero di parametri. In particolare se si impiega un vocabolario V, sarebbe necessario memorizzare V^n parametri per gestire sequenze con $n-1$ segmenti di cui tenere conto.

Una seconda strategia per la stima di $P(x_t \mid x_{t-1}, \ldots, x_1)$ consiste nel mantenere uno storico delle precedenti osservazioni in una variabile h_t. Ad ogni istante temporale t si aggiornano sia x_t che h_t. La predizione x_t è data da $x_t = P(x_t \mid h_t)$. Dato che la variabile h_t non è mai osservata, il modello è detto *autoregressivo latente*.

Si possono realizzare modelli di linguaggio che impiegano una variabile latente tramite reti neurali specializzate.

3.2 Reti Neurali Feed Forward

Una *rete neurale artificiale*, o *rete neurale*, è un modello di apprendimento che si basa sul funzionamento del cervello. Una rete neurale è formata da un insieme di neuroni collegati tra loro. Ciascun neurone osserva gli output degli altri neuroni a lui collegati, effettua una computazione e in base al superamento di una certa soglia può attivarsi.

Una *rete feed forward* (FFN) è una rete neurale artificiale in cui le connessioni tra nodi non formano cicli. Si tratta del più semplice tipo di rete neurale, in cui l'informazione è elaborata in una sola direzione. Una rete feed forward può essere vista come un Perceptron o un Multi Layer Perceptron, a seconda che sia costituita unicamente da uno strato di input e uno di output o che abbia anche uno o più strati nascosti.

3.2.1 Perceptron

Il Perceptron è il progenitore delle prime reti neurali ed è stato proposto da Rosenblatt nel 1958 [29]. Il percettrone utilizza una funzione di attivazione lineare a soglia. Si tratta di un modello che converge solo se i dati sono separabili linearmente dato che è in grado di apprendere solo mapping lineari. Scegliendo adeguatamente i pesi il percettrone permette di risolvere problemi semplici come la creazione di porte AND, OR o NOT.

3.2.2 Multi Layer Perceptron

Il Multi Layer Perceptron (MLP) è una rete feed forward e totalmente connessa con almeno 3 strati, di cui almeno uno nascosto, che impiega funzioni di attivazione non lineari. Teoricamente, stando all'*universal approximation theorem*, un MLP può approssimare una qualsiasi funzione continua che mappa intervalli di numeri reali su un intervallo di numeri reali impiegando un unico strato nascosto. Una delle ragioni alla base della recente esplosione del Deep Learning è dovuta alla maggior efficienza delle reti neurali che impiegano più strati nascosti, visto che molti problemi complessi per poter essere risolti su un MLP con un unico strato nascosto richiederebbero un numero esponenziale di nodi.

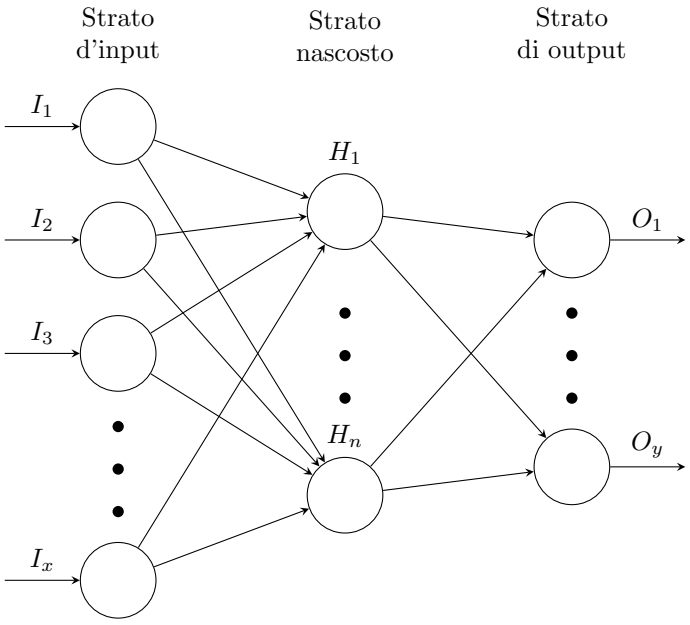


Figura 3.1: Esempio di Multi Layer Perceptron. Lo strato di input è formato da x nodi, lo strato nascosto da n nodi e lo strato di output da y nodi.

L'esempio in figura 3.1 rappresenta una rete a tre livelli avente x neuroni in input, n livelli nascosti e y neuroni in output. Il numero totale di pesi, includendo i bias, è pari a: $xn + ny + n + y$. in figura 3.2 è mostrato il flusso di informazione in un neurone. Gli input sono moltiplicati per i rispettivi pesi e sommati al bias. Il risultato è fornito in input ad una funzione di attivazione per introdurre delle non linearità e si ottiene in questo modo l'output del neurone.

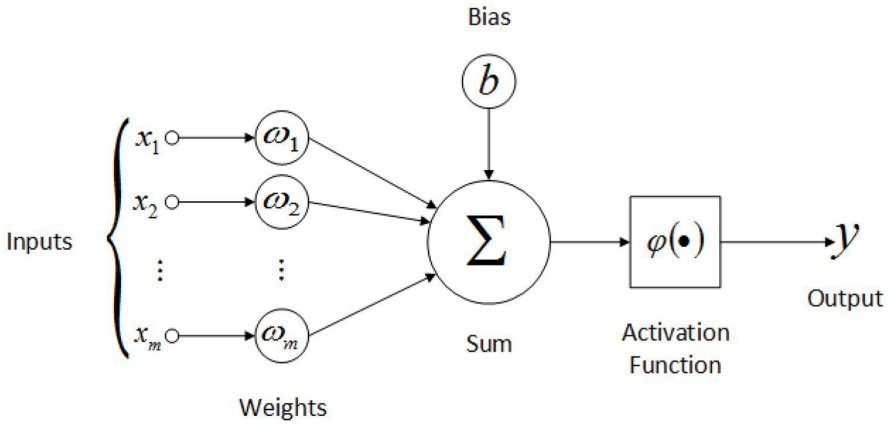


Figura 3.2: Flusso d'informazione in un singolo neurone.
Fonte: https://dzone.com/articles/the-artificial-neural-networks-handbook-part-4

Alla base del funzionamento di un MLP stanno due meccanismi: la *forward propagation* e la *backward propagation*. In fase di inferenza una rete neurale si limita a processare i dati forniti in input tramite propagazione in avanti. In fase di addestramento si impiega anche la retropropagazione.

Propagazione in avanti

Si considera una rete MLP con un solo strato nascosto. La funzione di attivazione si denota con ϕ. Dato un minibatch di input $\mathbf{X} \in \mathbb{R}^{n \times d}$, con n dimensione del batch e d numero di input, l'output dello strato nascosto $\mathbf{H} \in \mathbb{R}^{n \times h}$ è calcolato come in equazione (3.3).

$$\mathbf{H} = \phi\left(\mathbf{X}\mathbf{W}_{xh} + \mathbf{b}_h\right), \tag{3.3}$$

dove $\mathbf{W}_{xh} \in \mathbb{R}^{d \times h}$ sono i pesi, $\mathbf{b}_h \in \mathbb{R}^{1 \times h}$ è il bias e h è il numero di unità nascoste. A partire da $\mathbf{H}$ si effettua il calcolo dello strato di output come mostrato in equazione (3.4).

$$\mathbf{O} = \mathbf{H}\mathbf{W}_{hq} + \mathbf{b}_q, \tag{3.4}$$

dove $\mathbf{O} \in \mathbb{R}^{n \times q}$ è la variabile di output, $\mathbf{W}_{hq} \in \mathbb{R}^{h \times q}$ sono i pesi e $\mathbf{b}_q \in \mathbb{R}^{1 \times q}$ è il bias dello strato di output.

Retropropagazione

Per addestrare i parametri di una rete neurale, pesi e bias, si impiega un algoritmo di *backpropagation*, per ottimizzare l'errore della rete tramite discesa del gradiente stocastica minibatch. Si calcola il gradiente a partire dalle derivate parziali della funzione di loss rispetto a tutti i parametri della rete neurale. Per far ciò si applica la regola di derivazione della catena a partire dallo strato di output fino allo strato di input.

3.3 Algoritmi di word embedding

Alcuni dei principali algoritmi di embedding proposti negli ultimi anni sono *Word2Vec* [30], *Global Vector for Word Representation* (GloVe) [31] e fastText [32]. Di seguito si descrivono brevemente alcune delle principali caratteristiche di ciascun algoritmo. Per maggiori informazioni si rimanda agli studi citati.

3.3.1 Word2Vec

Word2Vec impiega una rete neurale feed forward addestrata con metodi basati su finestre locali di contesto per generare embedding di dimensioni prefissate. In particolare per l'addestramento si impiegano le architetture *Continuous Bag of Words* (CBOW) e Skip-Gram, mostrate in figura 3.3.

In CBOW si genera un word embedding a partire dal termine corrente considerando gli n termini futuri e gli n termini passati. Perciò data la frase di esempio "Lo studente è in ritardo per la lezione" e scegliendo come termine corrente "ritardo" si vuole effettuare la sua previsione a partire dal contesto dato dai termini vicini. Se si considera una finestra di contesto con $n = 2$ si useranno per la previsione i termini ["è", "in"] e ["per", "la"].

In Skip-Gram si cerca di prevedere il contesto dato dagli n termini passati e gli n termini futuri a partire dal termine corrente. Riprendendo l'esempio precedente in Skip-Gram, impiegando una finestra di contesto con $n = 2$, si cerca prevedere i termini ["è", "in"] e ["per", "la"] a partire dal termine corrente "ritardo". Si assume che il contesto del termine corrente sia formato da termini indipendentemente e identicamente distribuiti (i.i.d.). Si impiega perciò l'assunzione BoW.

In figura 3.4 è possibile osservare come gli embedding generati da Word2Vec siano in grado di apprendere la relazione tra stati e capitali in modo non supervisionato. In generale l'algoritmo è in grado di apprendere analogie tra termini in modo automatico. Un esempio noto, $\overrightarrow{\text{man}} - \overrightarrow{\text{woman}} \approx \overrightarrow{\text{king}} - \overrightarrow{\text{queen}}$ mostra, applicando delle operazioni algebriche agli embedding, che il modello è in grado di comprendere concetti che potrebbero essere associati al genere e alla

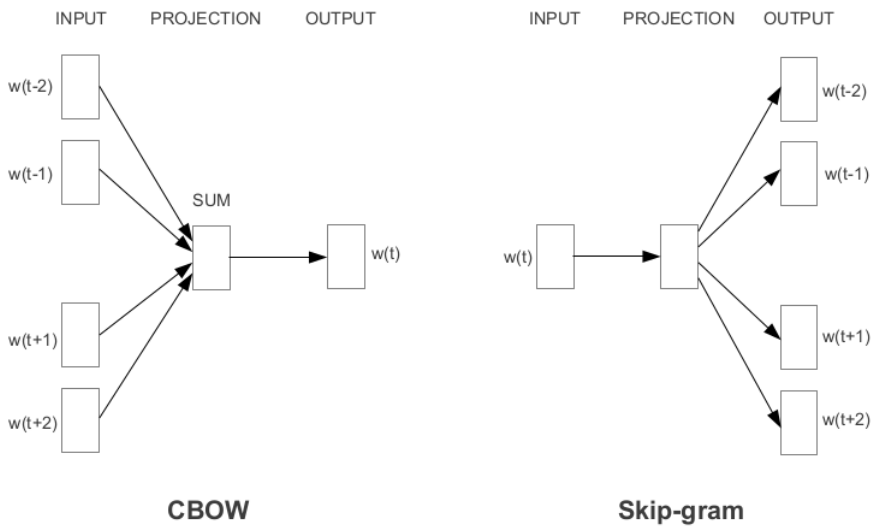


Figura 3.3: Illustrazione delle architetture CBOW e Skip-Gram. CBOW predice il termine corrente in base al contesto locale, ovvero al contesto ottenuto considerando i termini vicini a quello corrente. Skip-Gram predice i termini vicini a partire da un dato termine corrente.

Figura 1 dell'articolo Efficient Estimation of Word Representations in Vector Space [30]

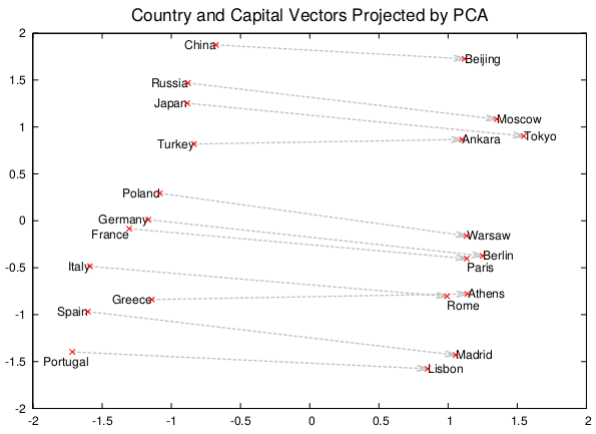


Figura 3.4: Illustrazione della proiezione bidimensionale ottenuta tramite PCA dei vettori del modello *Skip-Gram* a 1000 dimensioni su un dataset contenente i nomi degli stati e delle relative capitali. Si osserva che il modello apprende in modo automatico a partire da dati non etichettati le relazioni tra i termini.

Figura 1 dell'articolo Efficient Estimation of Word Representations in Vector Space [30]

regalità. Tuttavia l'apprendimento avviene in un modo che non permette di controllare la precisione con facilità e può portare a risultati sorprendenti. Ad esempio in [33] si osserva la seguente relazione tra embedding: $\overrightarrow{\text{man}} - \overrightarrow{\text{woman}} \approx \overrightarrow{\text{computer programmer}} - \overrightarrow{\text{homemaker}}$. In questo caso il modello ha appreso il sessismo implicito nel testo su cui è stato addestrato.

3.3.2 GloVe

Mentre Word2Vec è un modello predittivo in cui si estraggono embedding a partire da una rete neurale feed forward addestrata tramite CBOW o Skip-Gram, in GloVe si adotta un approccio che non comprende l'utilizzo di reti neurali.

GloVe impiega le informazioni statistiche globali riguardanti le co-occorrenze dei termini nel corpus. Si basa sull'utilizzo di una tecnica di fattorizzazione della matrice di co-occorrenza $\mathbf{X} \in \mathbb{R}^{|V|\times|V|}$, ovvero di una matrice quadrata di dimensioni pari al numero di termini nel vocabolario V in cui l'elemento in posizione (i, j) indica il numero di volte in cui il termine i occorre nel contesto di un termine j. In modo simile a quanto si osserva per la *Latent Semantic Analysis*, esposta in sezione 2.8, si genera una matrice di embedding in cui ciascuna riga contiene la rappresentazione vettoriale del termine corrispondente in uno spazio vettoriale a dimensionalità ridotta. Per effettuare la fattorizzazione si cerca di minimizzare una funzione che esprima l'errore con cui si ricostruisce $\mathbf{X}$ a partire dalla matrice di embedding generata. Questa funzione consente di determinare una rappresentazione a bassa dimensionalità che catturi la maggior parte della varianza dei dati nello spazio ad elevata dimensionalità di $\mathbf{X}$.

Gli embedding sono ottimizzati in modo che il prodotto scalare dei vettori corrispondenti a due termini sia uguale al logaritmo del numero di volte in cui i due termini appaiono uno vicino all'altro. La funzione obiettivo impiegata in GloVe è mostrata in equazione (3.5).

$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2, \tag{3.5}$$

dove w_i e b_i sono il vettore e il bias relativi al termine i e $\tilde{w}_j$ e $\tilde{b}_j$ sono il vettore e il bias relativi al termine j, X_{ij} è l'elemento (i, j) nella matrice di co-occorrenza e f è una funzione per il calcolo dei pesi che penalizza le co-occorrenze più rare o le più frequenti.

In genere gli embedding generati da GloVe hanno prestazioni simili a quelli ottenuti da Word2Vec configurando opportunamente gli iperparametri. Un vantaggio di GloVe rispetto a Word2Vec è la maggior facilità con cui si può parallelizzare l'implementazione e impiegare quindi corpus di grandi dimensioni.

3.3.3 fastText

fastText è un'estensione di Word2Vec che impiega una segmentazione a livello di carattere per costruire le rappresentazioni e l'addestramento di un modello skip-gram per apprendere gli embedding. In fastText gli embedding sono generati rappresentando ogni termine come un n-gram di caratteri. Ad esempio dato il termine "artificial" con $n = 3$ il termine è rappresentato in fastText come $\langle$ ar, art, rti, tif, ifi, fic, ici, ial, al $\rangle$, dove le parentesi angolate segnalano l'inizio e la fine di una parola. Sfruttando una segmentazione a livello di gruppi di n caratteri si ottengono i vantaggi già indicati in sezione 2.5, come la capacità di ridurre i termini OOV.

3.3.4 Embedding contestuali

Algoritmi di embedding come Word2Vec, GloVe e fastText generano word embedding indipendenti dal contesto. In altri termini forniscono in output un vettore per ciascun termine, combinando tutti i differenti significati assunti dal termine nei dati in input in un unico vettore. Ad esempio date due frasi presenti nel corpus "per fare il disegno serve una squadra" e "quella squadra sta vincendo", gli algoritmi Word2Vec e GloVe genererebbero un unico embedding per il termine "squadra" indipendentemente dal diverso significato nelle due frasi. A partire dal 2018 sono stati proposti diversi approcci per la generazione di embedding contestuali, ovvero di embedding che tengono conto della posizione dei singoli segmenti nelle sequenze in input. Riprendendo le frasi di esempio viste precedentemente, modelli in grado di generare embedding contestuali creeranno due embedding distinti per il termine "squadra". Il primo sarà più vicino a termini come "riga", mentre il secondo sarà più vicino a termini come "sport". Alcuni dei principali modelli in grado di generare embedding contestuali sono ELMo e i modelli basati su Transformer, che saranno approfonditi rispettivamente in sezione 3.4.8 e 3.5.

3.4 Reti Neurali Ricorrenti

Una rete neurale ricorrente (RNN) è una rete neurale artificiale in cui si mantiene uno stato latente impiegando connessioni cicliche tra nodi. Si contrappone alle reti feed forward, che formano un grafo aciclico. La struttura delle reti neurali ricorrenti le rende adatte all'elaborazione di sequenze e di serie temporali.

3.4.1 Struttura di una rete ricorrente

Si ipotizzi di avere un minibatch di input $\mathbf{X}_t \in \mathbb{R}^{n \times d}$ al tempo t. Per un minibatch di n istanze di lunghezza d di una sequenza, ciascuna riga di $\mathbf{X}_t$ corrisponde ad un campione al tempo t della sequenza. La variabile nascosta al tempo t si denota con $\mathbf{H}_t \in \mathbb{R}^{n \times h}$. Inoltre, a differenza di un MLP, si calcola ad ogni passo t anche la variabile nascosta $\mathbf{H}_{t-1}$ dall'istante temporale precedente e si introduce un nuovo insieme di pesi $\mathbf{W_{hh}} \in \mathbb{R}^{h \times h}$ per indicare come utilizzare la variabile nascosta del tempo $t-1$ al tempo t. La variabile nascosta nell'istante corrente si calcola a partire dall'input dell'istante corrente e dalla variabile nascosta dello stato precedente, come mostrato in equazione (3.6).

$$\mathbf{H}_t = \phi\left(\mathbf{X}_t \mathbf{W}_{xh} + \mathbf{H}_{t-1} \mathbf{W}_{hh} + \mathbf{b}_h\right) \tag{3.6}$$

Le variabili nascoste $\mathbf{H}_t$ e $\mathbf{H}_{t-1}$, appartenenti ad istanti temporali vicini, fungono da memoria della rete neurale all'istante t. Per questa ragione la variabile nascosta $\mathbf{H}_t$ è detta *stato nascosto*. Si osserva che il calcolo dello stato nascosto è ricorrente. Perciò le reti che impiegano uno stato nascosto sono dette reti neurali *ricorrenti*.

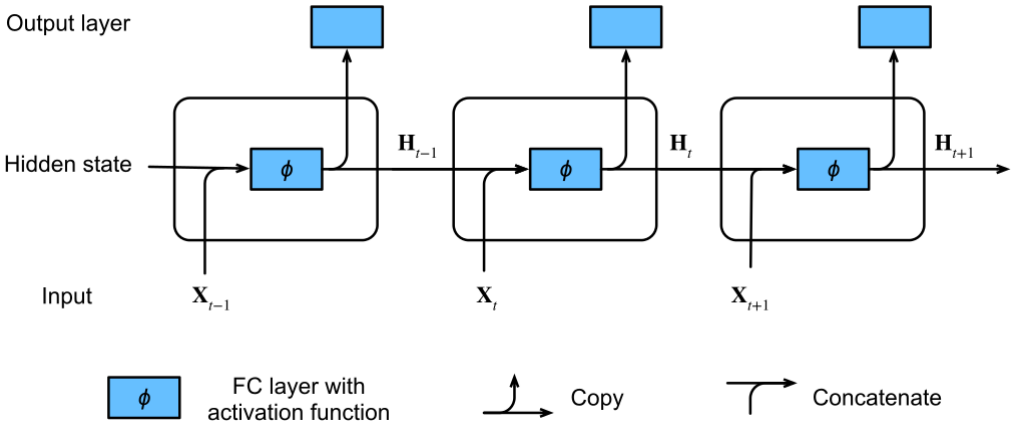


Figura 3.5: Illustrazione di una rete neurale ricorrente.
Fonte: Dive into Deep Learning [6, p. 321]

Come si vede in figura 3.5, ad ogni istante temporale t il calcolo dello stato nascosto comprende diverse fasi:

- Concatenazione degli input $\mathbf{X}_t$ al tempo t con lo stato nascosto $\mathbf{H}_{t-1}$.
- Calcolo di $\mathbf{H}_t$ a partire dalla concatenazione fornita in input ad uno strato totalmente connesso con funzione di attivazione ϕ. I parametri sono dati dalla concatenazione di $\mathbf{W}_{xh}$, $\mathbf{W}_{hh}$ e $\mathbf{b}_h$.

- Calcolo di $\mathbf{H}_{t+1}$ e dello strato di output $\mathbf{O}_t$ a partire da $\mathbf{H}_t$.

Il calcolo dello strato di output di una RNN è simile a quello nel MLP.

$$\mathbf{O}_t = \mathbf{H}_t \mathbf{W}_{hq} + \mathbf{b}_q$$

Si osserva che in una rete ricorrente l'input è fornito sequenzialmente in più passi temporali. Inoltre in istanti temporali distinti le reti neurali ricorrenti utilizzano sempre gli stessi pesi.

3.4.2 Addestramento di una rete ricorrente

Per addestrare una rete ricorrente tramite retropropagazione è necessario eseguire l'*unrolling in time*, come si vede in figura 3.7. Per effettuare lo "srotolamento' è necessario conoscere la lunghezza della sequenza a priori. La *backpropagation through time* consiste nell'addestrare una rete ricorrente *unrolled* come una rete neurale profonda con uno strato per ciascun istante temporale.

Così come nell'addestramento di FFN profonde, anche nel caso delle RNN profonde ci possono essere delle difficoltà legati ai fenomeni dell'*exploding gradient* e del *vanishing gradient*.

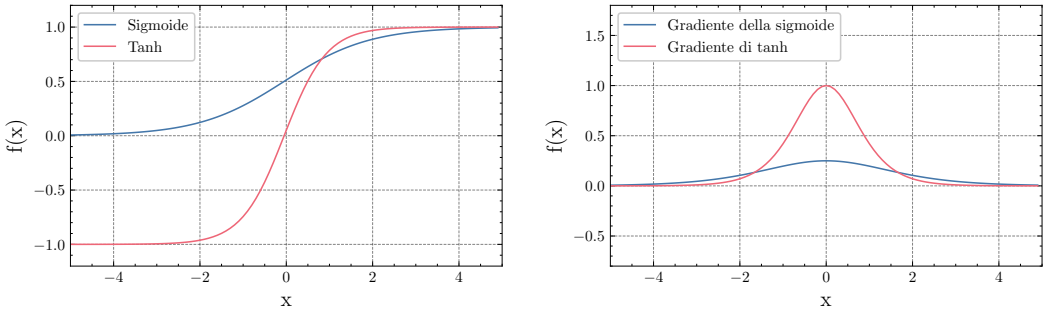


Figura 3.6: Illustrazione delle funzioni di attivazione sigmoide e tanh e dei relativi gradienti.

Scomparsa del gradiente Il problema della scomparsa del gradiente si osserva nell'addestramento di reti neurali profonde (DNN) che impiegano funzioni di attivazione lineari classiche, come la funzione logistica e la tangente iperbolica. In queste funzioni, come si vede in figura 3.6, si possono ottenere valori del gradiente molto piccoli. Quando si osserva la scomparsa del gradiente l'ultimo strato della DNN ha valori del gradiente vicini allo 0. Perciò i valori del gradiente ottenuti tramite retropropagazione saranno sempre più piccoli e

i primi strati nascosti non riceveranno nessun cambiamento significativo dei pesi. Ciò ha conseguenze disastrose; i primi strati non apprendono nulla e gli strati successivi non possono elaborare le caratteristiche identificate dai primi strati. Per mitigare il problema si possono impiegare funzioni di attivazione più moderne, come ReLU e GELU, oltre che una diversa inizializzazione dei pesi, come quella di Xavier [34].

Esplosione del gradiente Il problema dell'esplosione del gradiente si osserva nell'addestramento di reti neurali profonde quando si accumulano elevati valori del gradiente e si effettuano degli aggiornamenti dei pesi molto grandi. Ciò rende le DNN instabili e determina una degradazione della qualità del modello. Uno dei metodi con cui si gestisce l'esplosione del gradiente è il *gradient clipping*.

Gradient Clipping Per una sequenza di lunghezza T si calcola il gradiente su T intervalli temporali in un'iterazione. Perciò nella retropropagazione si effettua una catena di prodotti tra matrici di lunghezza $O(T)$. Per valori di T grandi si può incorrere in problemi di instabilità numerica legati all'esplosione o alla scomparsa del gradiente. Il gradient clipping consiste nel proiettare il gradiente $\mathbf{g}$ su una palla con raggio θ, come mostrato in equazione (3.7). Questo approccio ha il vantaggio di garantire che la norma del gradiente non sarà mai superiore a θ e che $\mathbf{g}'$ avrà la stessa direzione del gradiente originale $\mathbf{g}$. Inoltre si limita l'influenza che può avere un determinato minibatch, favorendo un maggior grado di stabilità durante l'aggiornamento tramite aggiornamenti del gradiente piccoli e controllati. Si riesce in questo modo a mitigare il problema dell'esplosione del gradiente.

$$\mathbf{g}' \leftarrow \min\left(1, \frac{\theta}{\|\mathbf{g}\|}\right)\mathbf{g} \tag{3.7}$$

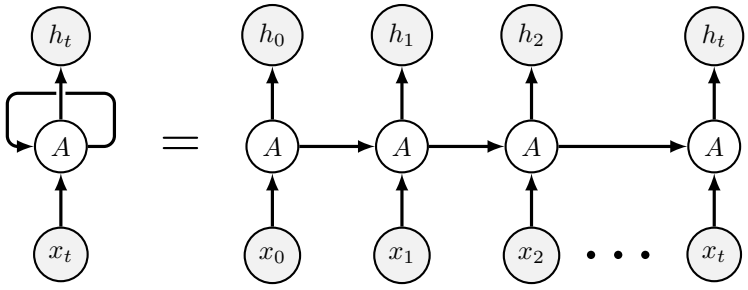


Figura 3.7: Struttura di una rete neurale ricorrente *rolled* a sinistra e *unrolled* a destra.

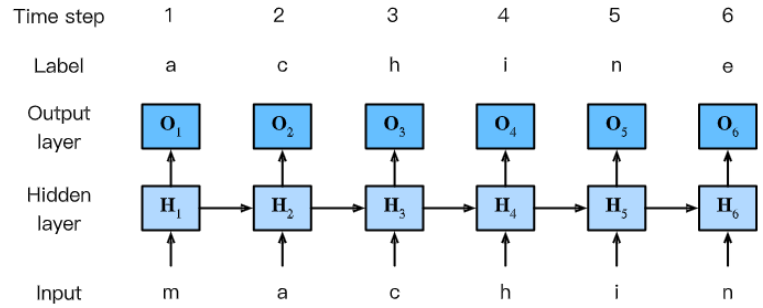


Figura 3.8: Un modello di linguaggio con segmenti definiti a livello di carattere basato su una rete neurale ricorrente. La sequenza in input è "machin", mentre la sequenza da prevedere è "achine".
Fonte: Dive into Deep Learning [6, p. 351]

3.4.3 Reti neurali ricorrenti a livello di carattere

Un modello di linguaggio che impiega una rete neurale ricorrente può essere visto come un modello di classificazione probabilistico che impara a predire una distribuzione di probabilità sul vocabolario, dato un contesto linguistico.

Il primo impiego di una rete neurale ricorrente come modello di linguaggio risale al 2000 [35]. Si può impiegare una RNN come modello di linguaggio a livello di carattere se si effettua una traslazione dei segmenti che compongono la sequenza in input a destra di uno. In questo modo si ottengono i segmenti che devono essere previsti con cui addestrare il modello. La segmentazione si effettua a livello di carattere. In figura 3.8 è mostrato un esempio di rete neurale ricorrente a livello di carattere.

Dato che il calcolo dello stato nascosto è ricorrente, al terzo istante temporale il calcolo dello strato di output è determinato dalla distribuzione di probabilità del carattere successivo generata a partire dall'input formato dai caratteri della sequenza "m" "a" e "c".

3.4.4 Limiti delle reti ricorrenti

I già citati problemi di instabilità numerica dovuti alla propagazione del gradiente in fase di addestramento si legano alla limitata capacità delle reti neurali ricorrenti di catturare dipendenze a lungo raggio nelle sequenze. In particolare, nel caso si incontri un'informazione utile tra le prime osservazioni, per fare in modo che si propaghi è necessario impiegare un valore del gradiente molto elevato. Se si incontra un'informazione poco utile perché sia possibile ignorarla è necessario impiegare un gradiente molto piccolo.

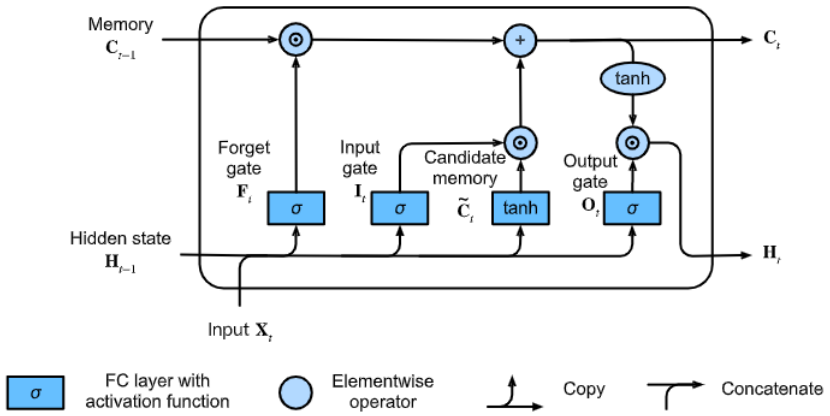


Figura 3.9: Illustrazione del flusso di dati in una cella di memoria di una rete LSTM.

Fonte: Dive into Deep Learning [6, p. 351]

Per mitigare questi problemi sono state proposte le varianti *gated* delle reti neurali ricorrenti che impiegano celle di base ampliate con dei gate per gestire lo stato nascosto. Alcuni dei principali tipi di reti ricorrenti con meccanismi di memoria sono LSTM e GRU.

3.4.5 Reti Long Short Term Memory

Per poter preservare l'informazione a lungo termine e ignorare selettivamente l'input a breve termine una delle prime estensioni delle reti ricorrenti proposta è il modello *Long Short Term Memory*, nel 1997 [36].

Questo modello introdusse per primo i concetti di cella di memoria e di gate, ispirati alle porte logiche dei computer. In una rete LSTM si impiegano tre tipologie di gate: il *forget gate*, l'*input gate* e l'*output gate*.

In figura 3.9 è mostrata una cella di memoria di una rete che impiega il modello LSTM e il flusso di dati che sarà analizzato di seguito. In input è fornito lo stato precedente $\mathbf{H}_{t-1} \in \mathbb{R}^{n \times h}$ e un minibatch $\mathbf{X}_t \in \mathbb{R}^{n \times d}$ all'istante t, con n numero di osservazioni, d numero di input di ciascuna osservazione e h numero di unità nascoste. I gate al tempo t sono definiti come in equazione (3.8).

$$\begin{aligned} \mathbf{I}_t &= \sigma\left(\mathbf{X}_t \mathbf{W}_{xi} + \mathbf{H}_{t-1} \mathbf{W}_{hi} + \mathbf{b}_i\right), \\ \mathbf{F}_t &= \sigma\left(\mathbf{X}_t \mathbf{W}_{xf} + \mathbf{H}_{t-1} \mathbf{W}_{hf} + \mathbf{b}_f\right), \\ \mathbf{O}_t &= \sigma\left(\mathbf{X}_t \mathbf{W}_{xo} + \mathbf{H}_{t-1} \mathbf{W}_{ho} + \mathbf{b}_o\right), \end{aligned} \tag{3.8}$$

dove $\mathbf{W}_{xi}, \mathbf{W}_{xf}, \mathbf{W}_{xo} \in \mathbb{R}^{d \times h}$ e $\mathbf{W}_{hi}, \mathbf{W}_{hf}, \mathbf{W}_{ho} \in \mathbb{R}^{h \times h}$ sono i pesi e $\mathbf{b}_i, \mathbf{b}_f, \mathbf{b}_o \in \mathbb{R}^{1 \times h}$ sono i bias.

Inizialmente si mantiene uno stato latente intermedio $\hat{\mathbf{C}}_t$, ottenuto a partire dagli input alla cella di memoria. Si applica la funzione di attivazione tanh perché il risultato stia nell'intervallo $(-1, 1)$. Il calcolo dello stato intermedio latente al tempo t è definito in equazione (3.9).

$$\tilde{\mathbf{C}}_t = \tanh\left(\mathbf{X}_t\mathbf{W}_{xc} + \mathbf{H}_{t-1}\mathbf{W}_{hc} + \mathbf{b}_c\right), \tag{3.9}$$

dove $\mathbf{W}_{xc} \in \mathbb{R}^{d\times h}$ e $\mathbf{W}_{hc} \in \mathbb{R}^{h\times h}$ sono i pesi e $\mathbf{b}_c \in \mathbb{R}^{1\times h}$ sono i bias.

Integrando i contributi dello stato latente intermedio e l'attivazione del forget gate e dell'input gate si ottiene un secondo stato intermedio $\hat{\mathbf{C}}_t$. Il calcolo di $\hat{\mathbf{C}}_t \in \mathbb{R}^{n\times h}$ è definito in equazione (3.10).

$$\mathbf{C}_t = \mathbf{F}_t \odot \mathbf{C}_{t-1} + \mathbf{I}_t \odot \tilde{\mathbf{C}}_t, \tag{3.10}$$

dove $\odot$ indica il prodotto di Hadamard, ovvero un'operazione che prende in input due matrici delle stesse dimensioni e produce un'altra matrice della stessa dimensione degli operandi, dove ogni elemento (i, j) è il prodotto degli elementi (i, j) delle due matrici originarie.

Se il forget gate ha valore prossimo a 1 e l'input gate ha valore vicino a 0, la cella di memoria precedente $\mathbf{C}_{t-1}$ sarà salvata e passata allo stato corrente. In questo modo si allevia il problema della scomparsa del gradiente ed è possibile tenere traccia di dipendenze a lungo raggio tra elementi in ciascuna delle sequenze.

A partire dall'integrazione delle informazioni estratte dalla memoria a lungo termine $\hat{\mathbf{C}}_t$ con l'output gate e la memoria della cella precedente $\mathbf{C}_{t-1}$ si calcola il nuovo stato nascosto $\mathbf{H}_t$ come mostrato in equazione (3.11).

$$\mathbf{H}_t = \mathbf{0}_t \odot \tanh\left(\mathbf{C}_t\right) \tag{3.11}$$

Se l'output gate ha valore vicino ad 1 si passa interamente la memoria, se l'output gate è vicino a 0 allora si mantiene l'informazione solo nella cella di memoria corrente e non si propaga ulteriormente.

3.4.6 Reti Gated Recurrent Unit

Introdotte nel 2014, le reti *Gated Recurrent Unit* [37] impiegano dei meccanismi per gestire l'aggiornamento e il reset dello stato nascosto tramite appositi gate. Rappresentano un'alternativa più efficiente e meno complessa delle reti LSTM.

Un *reset gate* permette di controllare quanta informazione dello stato nascosto precedente mantenere. Un *update gate* permette di controllare quanta parte del nuovo stato nascosto è ottenuta a partire dallo stato nascosto precedente.

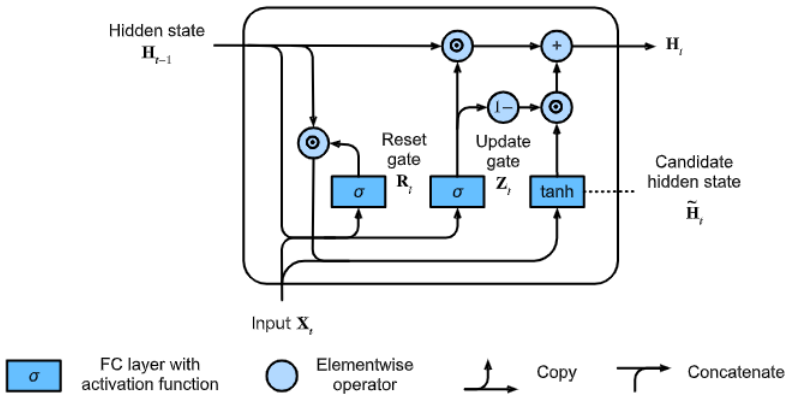


Figura 3.10: Illustrazione del flusso di dati in una cella di memoria di una rete GRU.

Fonte: Dive into Deep Learning [6, p. 344]

Rispetto alle reti LTSM il reset gate è simile al forget gate e l'update gate è simile all'unione dell'input gate e dell'output gate.

In figura 3.10 è mostrata una cella di memoria di una rete che impiega il modello GRU e il relativo flusso di dati che si analizza di seguito. In input alla cella di memoria sono forniti lo stato precedente $\mathbf{H}_{t-1} \in \mathbb{R}^{n \times h}$ e un minibatch all'istante t $\mathbf{X}_t \in \mathbb{R}^{n \times d}$, con n numero di osservazioni, d numero di input in ciascuna osservazione e h numero di unità nascoste. Gli output del reset gate $\mathbf{R}_t \in \mathbb{R}^{n \times h}$ e dell'update gate $\mathbf{Z}_t \in \mathbb{R}^{n \times h}$ sono dati dall'applicazione di due reti totalmente connesse con una funzione di attivazione sigmoide. Il calcolo dei due gate è definito in equazione (3.12)

$$\begin{aligned} \mathbf{R}_t &= \sigma\left(\mathbf{X}_t \mathbf{W}_{xr} + \mathbf{H}_{t-1} \mathbf{W}_{hr} + \mathbf{b}_r\right) \\ \mathbf{Z}_t &= \sigma\left(\mathbf{x}_t \mathbf{W}_{xz} + \mathbf{H}_{t-1} \mathbf{W}_{hz} + \mathbf{b}_z\right) \end{aligned} \tag{3.12}$$

dove $\mathbf{W}_{xr}, \mathbf{W}_{xz} \in \mathbb{R}^{d \times h}$ e $\mathbf{W}_{hr}, \mathbf{W}_{hz} \in \mathbb{R}^{h \times h}$ sono i pesi e $\mathbf{b}_r$, $\mathbf{b}_z \in \mathbb{R}^{1 \times h}$ sono i bias. Si impiega la funzione sigmoide in modo che i risultati stiano nell'intervallo $(0, 1)$.

Integrando il meccanismo di aggiornamento dello stato nascosto con il contributo del reset gate $\mathbf{R}_t$ si ottiene un stato nascosto intermedio $\hat{\mathbf{H}}_t \in \mathbb{R}^{n \times h}$.

$$\tilde{\mathbf{H}}_t = \tanh\left(\mathbf{X}_t \mathbf{W}_{xh} + \left(\mathbf{R}_t \odot \mathbf{H}_{t-1}\right) \mathbf{W}_{hh} + \mathbf{b}_h\right), \tag{3.13}$$

dove $\mathbf{W}_{xh} \in \mathbb{R}^{d \times h}$ e $\mathbf{W}_{hh} \in \mathbb{R}^{h \times h}$ sono i pesi e $\mathbf{b} \in \mathbb{R}^{1 \times h}$ sono i bias.

Per ogni elemento di $\mathbf{R}_t$ vicino a 0, lo stato nascosto intermedio è il risultato dell'applicazione di un MLP con $\mathbf{X}_t$ come input. Si effettua il reset di ogni traccia di stato nascosto precedente. Il risultato $\hat{\mathbf{H}}_t$ è integrato con l'update gate $\mathbf{Z}_t$. Ciò determina quanta parte del nuovo stato nascosto $\mathbf{H}_t \in \mathbb{R}^{n \times h}$ è

ottenuta a partire dallo stato precedente $\mathbf{H}_{t-1}$ o dallo stato nascosto intermedio $\hat{\mathbf{H}}_t$. Il calcolo del nuovo stato nascosto è definito in equazione (3.14).

$$\mathbf{H}_t = \mathbf{Z}_t \odot \mathbf{H}_{t-1} + (1 - \mathbf{Z}_t) \odot \tilde{\mathbf{H}}_t \tag{3.14}$$

Se $\mathbf{Z}_t$ è vicino ad 1, si mantiene lo stato precedente e si ignora l'informazione fornita da $\mathbf{X}_t$. Se $\mathbf{Z}_t$ è vicino a 0, il nuovo stato $\mathbf{H}_t$ è vicino allo stato intermedio $\hat{\mathbf{H}}_t$.

L'impiego di un reset gate e di un update gate permettono di gestire i problemi di instabilità numerica del gradiente e di mantenere una memoria a lungo termine. In particolare i reset gate catturano le dipendenze a breve termine e gli update gate le dipendenze a lungo termine.

3.4.7 Reti ricorrenti bidirezionali

Le reti neurali ricorrenti bidirezionali, introdotte nel 1997 [38], permettono di impiegare l'informazione proveniente da entrambi gli estremi di una sequenza per stimare l'output.

In figura 3.11 è mostrata l'architettura di una RNN bidirezionale. Al tempo t, è dato un minibatch di input $\mathbf{X} \in \mathbb{R}^{n \times d}$, con n numero di osservazioni in input e d lunghezza di ciascuna osservazione. ϕ denota la funzione di attivazione. Ci sono due stati nascosti al tempo t, uno in avanti $\overrightarrow{\mathbf{H}}_t \in \mathbb{R}^{n \times h}$ e uno all'indietro $\overleftarrow{\mathbf{H}}_t \in \mathbb{R}^{n \times h}$, dove h è il numero di unità nascoste. In equazione (3.15) si definisce come sono effettuati gli aggiornamenti dei due stati nascosti.

$$\begin{aligned} \overrightarrow{\mathbf{H}}_t &= \phi\left(\mathbf{X}_t \mathbf{W}_{xh}^{(f)} + \overrightarrow{\mathbf{H}}_{t-1} \mathbf{W}_{hh}^{(f)} + \mathbf{b}_h^{(f)}\right), \\ \overleftarrow{\mathbf{H}}_t &= \phi\left(\mathbf{X}_t \mathbf{W}_{xh}^{(b)} + \overleftarrow{\mathbf{H}}_{t+1} \mathbf{W}_{hh}^{(b)} + \mathbf{b}_h^{(b)}\right), \end{aligned} \tag{3.15}$$

dove i pesi $\mathbf{W}_{xh}^{(f)} \in \mathbb{R}^{d \times h}$ e $\mathbf{W}_{hh}^{(f)} \in \mathbf{R}^{h \times h}$ e il bias $\mathbf{b}_h^{(f)} \in \mathbb{R}^{1 \times h}$ sono i parametri del modello riferiti allo stato nascosto in avanti mentre le matrici dei pesi $\mathbf{W}_{xh}^{(b)} \in \mathbb{R}^{d \times h}$, $\mathbf{W}_{hh}^{(b)} \in \mathbb{R}^{h \times h}$ e il bias $\mathbf{b}_h^{(f)} \in \mathbb{R}^{1 \times h}$ sono i parametri del modello riferiti allo stato nascosto all'indietro.

In seguito si concatenano gli stati nascosti $\overrightarrow{\mathbf{H}}_t$ e $\overleftarrow{\mathbf{H}}_t$ in modo da ottenere uno stato nascosto $\mathbf{H}_t \in \mathbb{R}^{n \times 2h}$ da fornire in input allo strato di output. Lo strato di output $\mathbf{O}_t \in \mathbb{R}^{n \times q}$, con q numero di output, si calcola come in equazione (3.16).

$$\mathbf{O}_t = \mathbf{H}_t \mathbf{W}_{hq} + \mathbf{b}_q \tag{3.16}$$

La matrice dei pesi $\mathbf{W}_{hq} \in \mathbb{R}^{2h \times q}$ e il bias $\mathbf{b}_q \in \mathbb{R}^{1 \times q}$ sono i parametri del modello per lo strato di output. Si osserva che le due direzioni possono avere un numero differente di unità nascoste.

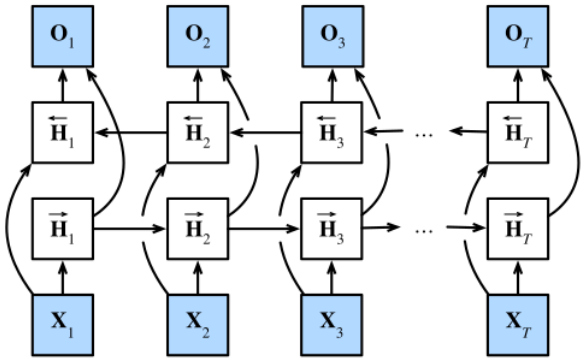


Figura 3.11: Architettura di una rete ricorrente ricorrente bidirezionale.

Le reti ricorrenti bidirezionali sono estremamente lente dato che si formano catene di dipendenze molto lunghe per il calcolo del gradiente durante la retropropagazione. Per questa ragione sono usate in poche applicazioni che includono la named entity recognition, la generazione di embedding e compiti di *masked language modeling* (trattato in sezione 3.5.3).

3.4.8 ELMo

Embeddings from Language Models [39] (ELMo) è uno dei primi modelli di linguaggio neurali per la generazione di embedding contestuali. Piuttosto che usare embedding statici per ciascun termine, ovvero a dimensioni fisse, come in Word2Vec o GloVe, ELMo elabora l'intera frase prima di assegnare a ciascun termine un embedding. Ciò avviene grazie all'impiego di una rete bidirezionale LSTM preaddestrata su un compito di modellazione del linguaggio. Perciò la rete neurale è addestrata a predire il termine successivo in una sequenza di termini. Ciò significa che ELMo effettua apprendimento non supervisionato ed ha accesso ad una vasta quantità di dati per l'addestramento.

ELMo è uno dei primi modelli a rendere disponibili le potenzialità del transfer learning nell'ambito del NLP. La LSTM di ELMo prima si preaddestra su un dataset massivo nella lingua di uno specifico dataset, e in seguito può essere usato come componente per altri modelli che necessitano di gestire il linguaggio.

Dato che ELMo impiega una rete bidirezionale LSTM il suo modello di linguaggio non crea embedding contestuali solo a partire dal termine successivo ma anche a partire dal termine precedente. Un esempio dei vettori coinvolti nella creazione di un embedding in ELMo si osserva in figura 3.12. In particolare, come si osserva in figura 3.13, gli embedding contestuali sono creati

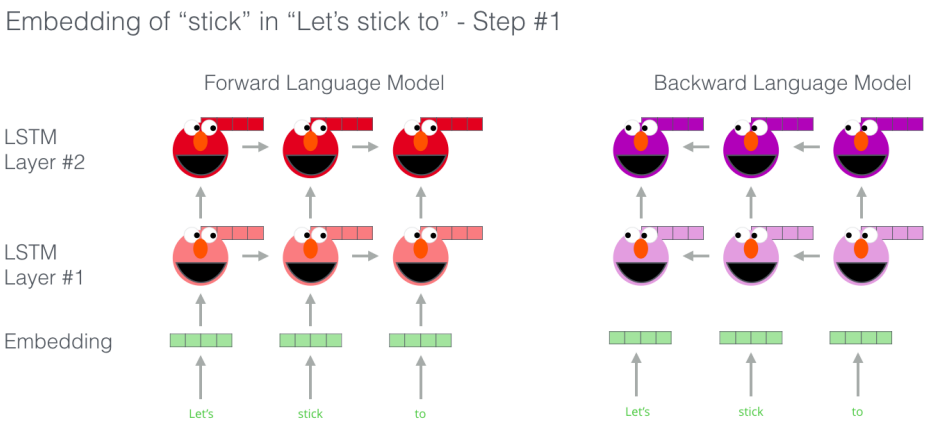


Figura 3.12: Esempio di generazione di un embedding contestuale per il termine “stick” a partire dagli embedding statici per la sequenza “Let’s stick to”.

Fonte: The Illustrated BERT, ELMo and co. [40]

concatenando ed effettuando una somma pesata degli stati nascosti provenienti dai due modelli di linguaggio e degli embedding iniziali.

3.4.9 Limiti delle RNN con meccanismi di memoria

L’impiego di reti ricorrenti con meccanismi di memoria permette di mitigare le problematiche riscontrate in fase di addestramento e legate alla limitata capacità di modellare dipendenze a lungo raggio ma non le risolvono completamente e hanno un costo di addestramento elevato. Per questa ragione sono state studiate delle varianti che impiegano il *meccanismo di attenzione*. Tra queste spiccano i Transformer, reti neurali che non impiegano reti ricorrenti e si basano interamente sul meccanismo di attenzione. I Transformer si sono dimostrati più adatti delle reti ricorrenti con meccanismi di memoria nel modellare dipendenze a lungo raggio e più efficienti grazie alla possibilità di parallelizzazione offerta.

3.5 Transformer

Il Transformer [41] è un modello di deep learning ampiamente impiegato in campi come l’elaborazione del linguaggio naturale e la visione artificiale. Originariamente fu introdotto nel 2017 per risolvere un problema di traduzione automatica. In seguito è stato dimostrato che modelli basati su transformer preaddestrati possono raggiungere lo stato dell’arte in diversi compiti.

Dato il loro successo, negli ultimi anni sono state presentate numerose varianti che migliorano il transformer originale, specializzandolo per eseguire

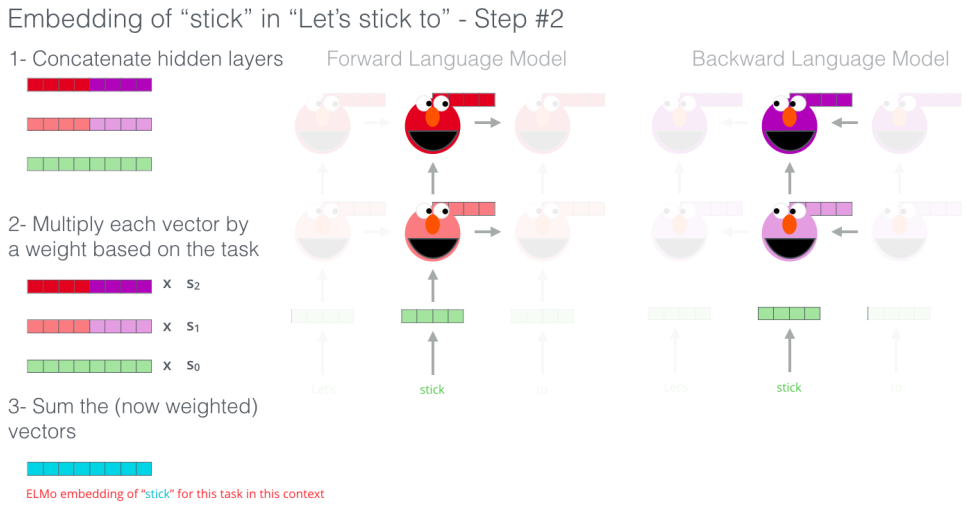


Figura 3.13: Esempio di generazione di un embedding contestuale in ELMo per il termine “stick”. Si effettuano concatenazione e somma pesata degli embedding statici in input e degli stati nascosti provenienti dai due modelli di linguaggio in avanti e all’indietro.
Fonte: The Illustrated BERT, ELMo and co. [40]

determinati compiti oppure rendendo il modello più efficiente o migliorando la sua capacità di generalizzazione. Di seguito si analizza l’architettura del transformer e si descrivono alcuni dei principali modelli di linguaggio basati su questa architettura. Per una tassonomia dei transformer introdotti in letteratura negli ultimi anni si rimanda agli studi di Tay et al. [42] del 2020 e di Lin et al. [43] del 2021.

3.5.1 Architettura del Transformer

In figura 3.15 è mostrata l’architettura del transformer [41]. Si tratta di un modello *sequence to sequence* costituito da un *encoder* e da un *decoder*, ciascuno formato da una pila di N blocchi identici. Ciascun blocco encoder è formato da un modulo di *multi head self attention* e da una rete neurale feed forward con codifica posizionale. Per costruire un modello più profondo si impiegano connessioni residue [44] sia nell’input che nell’output di ciascun modulo e uno strato di normalizzazione [45]. Si applica dropout [46] all’output di ciascun modulo prima di sommarlo all’input del modulo e applicare la normalizzazione. Nei blocchi decoder è inserito un modulo di multi head self attention aggiuntivo. Questo modulo impiega una maschera per evitare che ciascuna posizione possa prestare attenzione a posizioni successive. Perciò è detto *masked multi head attention*.

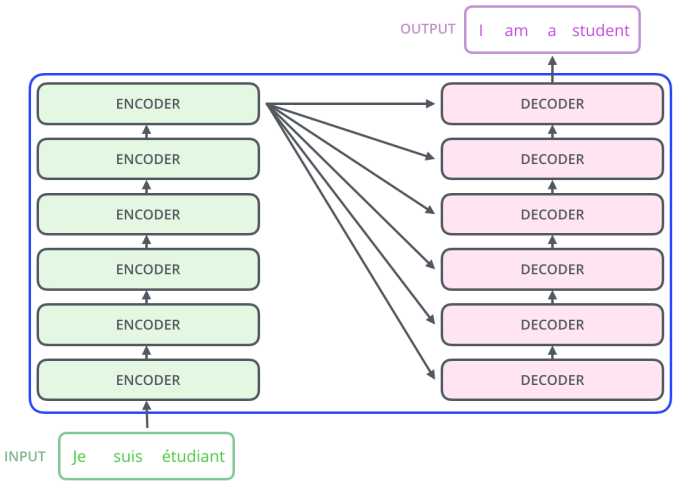


Figura 3.14: Illustrazione del flusso di informazione tra gli encoder, i decoder e la pila di encoder e decoder.
Fonte: The Illustrated Transformer [47]

In figura 3.14 si può osservare come si trasmette l'informazione tra i codificatori, i decodificatori e le rispettive pile. Il modulo di multi head attention del decodificatore i-esimo effettua il calcolo della multi head attention sull'output della pila di codificatori. Inoltre ciascun codificatore i-esimo invia il proprio output al codificatore i+1-esimo e lo stesso accade per i decodificatori. Il primo codificatore riceve in input degli embedding con codifica posizionale, detti *input embedding*. Il primo decodificatore riceve in input degli *output embedding* con applicata una traslazione e l'ultimo decodificatore invia l'output del modello allo strato lineare. Ai logit ottenuti come output dello strato lineare si applica la funzione softmax in modo che il risultato possa essere interpretato come probabilità associata ad un indice. Se si impiegano sequenze ottenute da testo l'indice a cui è associata la probabilità maggiore può essere visto come un termine del vocabolario.

Moduli di attenzione

Si impiega il meccanismo di attenzione basato sul modello Interrogazione Chiave Valore (QKV). Perciò per il calcolo dell'attenzione si impiegano tre tipologie di vettori, detti *interrogazioni*, *chiavi* e *valori*. A partire da ciascun *positional embedding* x_i della sequenza in input, ottenuto applicando la codifica posizionale agli input e output embedding, si crea un vettore di interrogazione q_i, uno di chiave k_i e uno di valore v_i moltiplicando l'embedding per la corrispondente tipologia di matrice ottenuta in fase di addestramento, $\mathbf{W}^{(q)}$, $\mathbf{W}^{(k)}$ o $\mathbf{W}^{(v)}$. Il calcolo della self attention è effettuato efficientemente su più segmenti

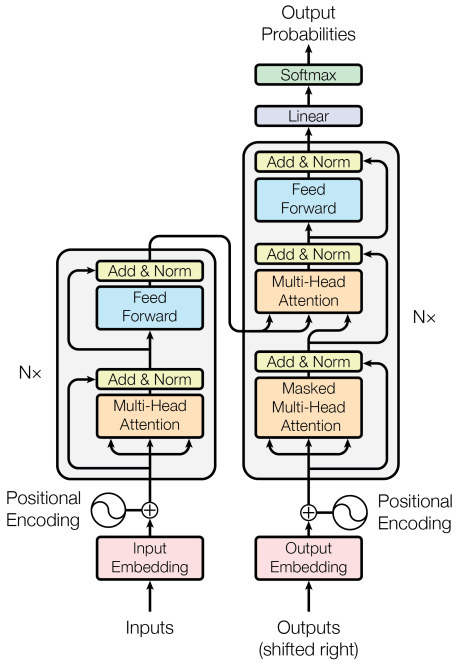
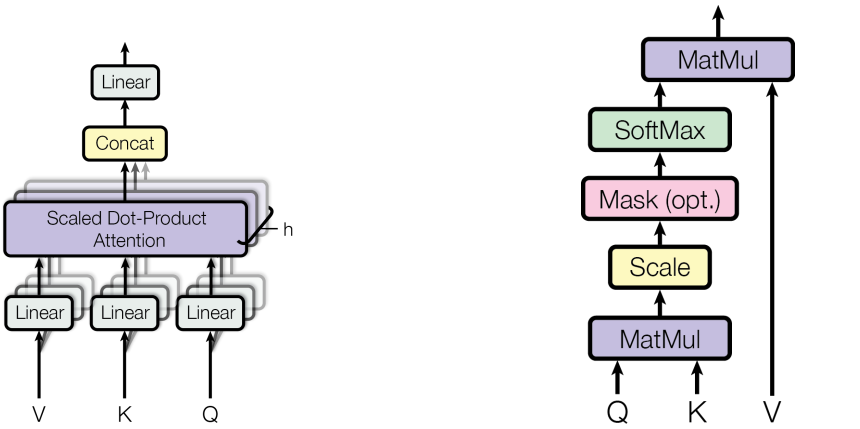


Figura 3.15: Illustrazione dell'architettura del Transformer originale.
Figura 1 dell'articolo "Attention is all you need" [41]

impiegando un prodotto tra matrici. Le matrici si ottengono concatenando verticalmente sia gli embedding che i vettori interrogazione, chiave e valore. Si ottengono perciò la matrice degli embedding posizionali $\mathbf{X} \in \mathbb{R}^{n \times d}$, la matrice delle interrogazioni $\mathbf{Q} \in \mathbb{R}^{n \times d}$, la matrice delle chiavi $\mathbf{K} \in \mathbb{R}^{m \times d}$ e la matrice dei valori $\mathbf{V} \in \mathbb{R}^{m \times v}$, con n numero di segmenti delle sequenze in input, d numero di dimensioni degli embedding, m numero di coppie chiave valore e v lunghezza dei vettori valore.

Le matrici interrogazione, chiave e valore sono combinate per ottenere una rappresentazione intermedia del codificatore. Questa rappresentazione è ottenuta normalizzando l'output del modulo di multi head attention ed è poi fornita in input alla rete feed forward.

I moduli di multi head attention impiegano la *scaled dot product attention* per il calcolo della self attention. La scaled dot product attention è definita in equazione (3.17) ed è illustrata in figura 3.16b. Si calcola il prodotto scalare dell'interrogazione con tutte le chiavi in modo da ottenere i punteggi che determinano come distribuire l'attenzione sui termini. Si divide il risultato per il fattore di normalizzazione $\sqrt{d}$ e si applica la funzione softmax. Il vettore ottenuto è moltiplicato per $\mathbf{V}$ in modo da definire l'attenzione come una media pesata di valori. Si ottiene la matrice di attenzione $A \in \mathbb{R}^{n \times d}$.



(a) Multi head attention.

(b) Scaled Dot Product Attention.

Figura 3.16: L'applicazione del meccanismo di attenzione avviene in parallelo su più strati (3.16a). Per il calcolo dell'attenzione si impiega la Scaled Dot Product Attention (3.16b).

Figura 2 dell'articolo "Attention is all you need" [41]

$$\mathbf{A}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}}\right)\mathbf{V} \tag{3.17}$$

Ipotizzando che tutti gli elementi dell'interrogazione e della chiave siano variabili indipendenti con media zero e varianza unitaria il prodotto scalare dei due vettori di lunghezza d ha varianza d. Per mantenere una varianza unitaria indipendentemente dalla lunghezza del vettore si impiega il fattore di normalizzazione $\sqrt{d}$. In questo modo si evita l'instabilità numerica associata a valori del gradiente molto piccoli nel caso si abbia un valore elevato di d.

La multi head attention rappresenta un raffinamento della self attention. Permette al modello di combinare la conoscenza acquisita dal prestare attenzione a diversi segmenti di una sequenza. Ciò consente ad esempio di modellare sia dipendenze a corto raggio che a lungo raggio. Nella multi head attention interrogazioni, chiavi e valori sono trasformati con h proiezioni lineari apprese indipendentemente. In equazione (3.18) si definisce come si calcola ciascuna attention head. In equazione (3.19) si definisce il calcolo della multi head attention a partire dalle singole attention head.

$$\mathbf{h}_i = \text{ScaledDotProductAttention}\left(\mathbf{W}_i^{(q)}\mathbf{q}, \mathbf{W}_i^{(k)}\mathbf{k}, \mathbf{W}_i^{(v)}\mathbf{v}\right) \tag{3.18}$$

Per ottenere un'unica matrice da fornire in input alla rete feed forward si concatenano le rappresentazioni intermedie $\mathbf{A}^{(h)}$ ottenute dal calcolo della self attention in ciascuno degli h spazi e si moltiplica il risultato per una matrice dei

pesi ottenuta tramite addestramento del modello, come mostrato in equazione (3.19).

$$\text{MultiHeadAttn}(Q, K, V) = \text{Concat}\,(h_1, \cdots, h_H)\, W^O, \tag{3.19}$$

Si ottiene in questo modo una matrice $\mathbf{Z}$ che cattura le informazioni fornite da tutte le attention head. Ciascuna attention head può prestare attenzione a diverse porzioni dell'input e il modello può quindi apprendere funzioni più articolate rispetto alla semplice media pesata che si ottiene calcolando la self attention su un unico spazio.

Infine alla matrice $\mathbf{Z}$ si applica uno strato di normalizzazione come mostrato in equazione (3.20).

$$\mathbf{Z}' = \text{LayerNorm}(\mathbf{Z} + \mathbf{X}) \tag{3.20}$$

Dove LayerNorm$(\cdot)$ indica l'applicazione dello strato di normalizzazione e $\mathbf{X}$ è la matrice dell'input dello strato di attenzione, ovvero degli embedding posizionali o dell'output $\mathbf{Z}''$ ottenuto da altri codificatori o decodificatori. $\mathbf{Z}''$ si ottiene applicando l'equazione (3.21).

$$\mathbf{Z}'' = \text{LayerNorm}(\text{FFN}(\mathbf{Z}') + \mathbf{Z}') \tag{3.21}$$

Dove FFN indica l'output dell'elaborazione di $\mathbf{Z}'$ da parte della rete neurale feed forward nel codificatore o nel decodificatore. Il calcolo di FFN$(\mathbf{Z}')$ avviene come mostrato in equazione (3.22).

$$\text{FFN}\,(\mathbf{H}') = \text{ReLU}\left(\mathbf{H}'\mathbf{W}^1 + \mathbf{b}^1\right)\mathbf{W}^2 + \mathbf{b}^2, \tag{3.22}$$

dove $\mathbf{W}^1 \in \mathbb{R}^{D_m \times D_f}$, $\mathbf{W}^2 \in \mathbb{R}^{D_f \times D_m}$, $\mathbf{b}^1 \in \mathbb{R}^{D_f}$ e $\mathbf{b}^2 \in \mathbb{R}^{D_m}$ sono i parametri addestrabili della rete. In genere la dimensione intermedia D_f è maggiore di D_m.

La multi head attention è impiegata nel transformer in tre diversi modi:

- Il decodificatore applica la self attention alle chiavi e ai valori provenienti dall'output del codificatore e alle interrogazioni fornite dal precedente decodificatore.
- Il codificatore contiene strati di self attention. Chiavi, valori e interrogazioni provengono tutti dall'output del precedente strato nel codificatore;
- Il decodificatore contiene strati di self attention. La multi head attention in questo caso è *masked*, in quanto si applica una maschera all'input della softmax in modo che ogni segmento possa essere relazionato solo a quelli che si trovano prima di lui nella sequenza. In questo modo si mantiene la proprietà di auto regressione.

Codifica posizionale

Al contrario delle reti ricorrenti, in cui i segmenti di una sequenza sono elaborati in modo ricorsivo uno alla volta, la self attention abbandona le operazioni sequenziali in favore di un approccio parallelizzabile. Per utilizzare l'informazione associata all'ordine delle sequenze si impiega una codifica posizionale per codificare le informazioni sulla posizione dei segmenti nella sequenza in modo assoluto o relativo.

Nel Transformer originale per determinare l'ordine con cui i singoli segmenti appaiono nella sequenza in input si impiega una codifica fissa tramite le funzioni periodiche seno e coseno. Ipotizzando che l'input $\mathbf{X} \in \mathbb{R}^{n \times d}$ sia formato da embedding d dimensionali per gli n segmenti di una sequenza, l'output della codifica posizionale $\mathbf{X} + \mathbf{P}$, con $\mathbf{P} \in \mathbb{R}^{n \times d}$ matrice degli embedding posizionali, è dato dalle equazioni (3.23).

$$\begin{aligned} \mathrm{p_{i,2j}} &= \sin\left(\frac{i}{10000^{2j/d}}\right), \\ \mathrm{p_{i,2j+1}} &= \cos\left(\frac{i}{10000^{2j/d}}\right), \end{aligned} \tag{3.23}$$

dove $p_{i,2j}$ indica l'elemento nella riga i-esima e nella colonna $(2j)$-esima di $\mathbf{P}$ e $p_{i,2j+1}$ l'elemento nella riga i-esima e nella colonna $(2j+1)$-esima.

Limiti del Transformer

I Transformer, nonostante i vantaggi offerti rispetto alle RNN con meccanismi di attenzione, hanno un grosso limite legato al costo computazionale quadratico e alla relativa ridotta capacità di elaborazione di sequenze di testo. Negli ultimi anni sono state proposte molte varianti basate su quest'architettura in grado di ridurre i costi computazionali, consentire di processare sequenze più lunghe e svolgere compiti di elaborazione del linguaggio naturale specifici ridefinendo frequentemente lo stato dell'arte. In particolare l'attenzione dei ricercatori si è concentrata sulla realizzazione di modelli con più parametri e addestrati su dataset di dimensioni maggiori e di modelli più leggeri e meno costosi dal punto di vista computazionale.

3.5.2 Modelli GPT

La famiglia di modelli *Generative Pre-Trained Transformers*, è stata sviluppata nel corso di due anni, a partire dal 2018, da OpenAI. I modelli di questa famiglia, GPT-1 [48], GPT-2 [49] e GPT-3 [50], sono specializzati nell'effettuare diversi compiti di NLP nell'ambito dello *zero shot learning*.

Zero Shot Learning

Lo zero shot learning consiste in una serie metodi di apprendimento per la risoluzione di un compito, non necessariamente di NLP, su cui il modello non è mai stato addestrato.

Ad esempio data la recensione di un prodotto si vogliono identificare delle categorie a cui fa riferimento il testo senza che queste appartengano a classi note. In questo caso si effettua un compito di *classificazione zero shot*. Un esempio intuitivo di zero shot learning è descrivere una zebra ad un persona che non ne ha mai vista una tramite i concetti di cavallo e strisce.

Esistono varianti dello zero shot learning, dette *one shot learning* e *few shot learning*, in cui si forniscono rispettivamente uno o pochi esempi etichettati. Un esempio di few shot learning si ha con gli algoritmi di rilevamento facciali per l'autenticazione su smartphone.

GPT-1

GPT impiega una struttura basata in larga parte sul Transformer, formata unicamente da una pila di 12 decoder e moduli di masked self-attention. Si tratta di un modello autoregressivo. Per la generazione degli embedding da fornire in input al modello si impiega l'algoritmo Byte Pair Encoding.

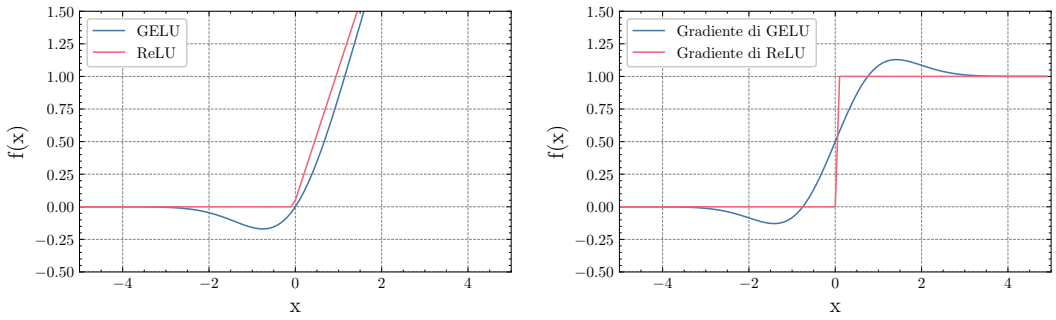


Figura 3.17: Illustrazione delle funzioni di attivazione ReLU e GELU e dei relativi gradienti.

Come funzione di attivazione utilizza *Gaussian Error Linear Unit* (GELU). GELU è ampiamente impiegata in molti transformer, tra cui i modelli della famiglia GPT, BERT e ALBERT. In figura 3.17 si propone un confronto tra le funzioni di attivazione ReLU e GELU. Si osserva che GELU permette di mitigare il problema della scomparsa del gradiente che si manifesta con la funzione logistica o tanh e anche di alleviare il problema della morte dei neuroni

che si riscontra con ReLU. In equazione (3.24) è mostrata l'approssimazione della funzione GELU.

$$\text{GELU}(x) = 0.5x\left(1 + \tanh\left(\sqrt{2/\pi}(x + 0.044715x^3)\right)\right) \tag{3.24}$$

GPT fu uno dei primi modelli basati su Transformer a proporre un processo di addestramento in due fasi. Inizialmente si effettua una fase di preaddestramento non supervisionato di un modello di linguaggio generativo seguita da una fase di messa a punto tramite risoluzione di un problema supervisionato su un compito specifico, come la sentiment analysis.

Nella prima fase si impiega la funzione obiettivo standard dei modelli di linguaggio, mostrata in equazione (3.25).

$$L_1(T) = \sum_i \log P\left(t_i \mid t_{i-k}, \ldots, t_{i-1}; \theta\right), \tag{3.25}$$

dove T è l'insieme di segmenti nei dati non supervisionati $\{t_1, \ldots, t_n\}$, k è la dimensione della finestra di contesto e θ sono i parametri di una rete neurale addestrata con discesa del gradiente minibatch.

GPT fu il primo modello a mostrare le potenzialità dell'utilizzo di modelli preaddestrati e come l'architettura dei Transformer facilita il transfer learning.

GPT-2

GPT-2 è stato rilasciato nel 2019 e ha migliorato significativamente le prestazioni di GPT. Ciò è avvenuto impiegando un dataset di addestramento più grande, formato da 40 GB di testo estratto dal web, e aumentando di più di dieci volte il numero di parametri impiegati (1.5 miliardi).

Il modello di seconda generazione è stato rilasciato in quattro versioni, GPT-2 SMALL, GPT-2 MEDIUM, GPT-2 LARGE e GPT-2 EXTRA LARGE. Si osserva che, in modo analogo a quanto si vedrà con BERT, aumentare il numero di parametri del modello migliora effettivamente le sue prestazioni.

GPT-3

Nel tentativo di costruire un modello di linguaggio più potente che migliorasse lo stato dell'arte in compiti zero shot, one shot o few shot nel 2019 fu rilasciato GPT-3.

GPT-3 impiega 175 miliardi di parametri, un numero più di 100 volte superiore al numero di parametri utilizzati in GPT-2. Le prestazioni ottenute dal modello spinsero OpenAI a non renderne pubblico il codice sorgente e Microsoft ad acquisirne una licenza esclusiva di utilizzo a seguito di un ingente

investimento. Nel tempo sono state sviluppate alternative open source a GPT-3 come GPT Neo e GPT-J.

Modelli di linguaggio di grandi dimensioni, come GPT-3, sono in grado di effettuare *in-context learning*, ovvero di identificare degli schemi nei dati durante l'addestramento come modello di linguaggio. In particolare nell'ambito di un problema zero shot, one shot o few shot il modello è in grado di associare alla descrizione del compito da svolgere e agli eventuali esempi forniti in input gli schemi che fanno parte della conoscenza appresa dal modello. Nell'articolo in cui è stato introdotto GPT-3 si osserva che l'efficacia dell'in-context learning aumenta con le dimensioni del modello.

La quarta generazione di GPT si basa sulla stessa idea di incrementare il numero di parametri e le risorse computazionali richieste per aumentare le prestazioni del modello. OpenAI afferma che GPT-4 avrà 100 trilioni di parametri, ovvero una dimensione 500 volte superiore al predecessore.

3.5.3 BERT

BERT [51] è stato rilasciato nel 2018 da Google ed ha segnato l'inizio di una nuova era per il campo dell'elaborazione del linguaggio naturale. Ha ridefinito lo stato dell'arte per quanto riguarda lo svolgimento di molti compiti di NLP in benchmark come GLUE [52] e SQuAD [53]. BERT si basa sia sull'architettura del Transformer che su altri modelli che impiegano reti ricorrenti con meccanismi di memoria, tra cui ELMo.

Architettura

BERT è costituito da una pila di encoder Transformer. Nelle due varianti in cui è stato rilasciato, BERT-Base e BERT-Large, si impiegano rispettivamente 12 encoder e 24 encoder. Negli encoder si impiegano FFN di maggiori dimensioni e più attention head rispetto al transformer originale. Nel primo caso si hanno 768 unità nascoste e 12 attention head e nel secondo 1024 unità nascoste e 16 attention head.

Come nel transformer originale i singoli termini in input attraversano la pila. Prima si applica la self attention e i risultati sono forniti ad una rete neurale feed forward e poi passati all'encoder successivo.

BERT si differenzia dal transformer nel modo in cui è impiegato l'output del modello. Per effettuare la classificazione si fornisce l'output del modello in input ad un classificatore. Nel caso di BERT BASE si utilizza una FFN con un unico strato formato da 768 unità nascoste. I logit ottenuti applicando il classificatore sono convertiti in probabilità di appartenenza ad una determinata classe tramite applicazione della softmax.

Figura 3.18: Illustrazione dei differenti approcci alla modellazione del linguaggio. Un approccio autoregressivo effettua predizioni a partire da un unico lato. L'approccio autoencoder effettua la predizione di un termine impiegando tutti gli altri termini nella sequenza.

Preaddestramento

BERT è un modello *autoencoder* in quanto genera rappresentazioni contestuali bidirezionali ricostruendo un'input corrotto. Modelli *autoregressivi* generano rappresentazioni contestuali unidirezionali, tenendo conto del contesto solo a destra o a sinistra. In figura 3.18 sono mostrati entrambi gli approcci. Per ottenere un risultato simile ad ELMo, che realizza rappresentazioni condizionate dal contesto in entrambi i sensi impiegando reti bidirezionali, BERT utilizza un metodo di addestramento che fa uso di maschere, detto *masked language modeling* (MLM).

Nel MLM si maschera un certo numero di termini nelle sequenze in input e si chiede al modello di predire i termini mascherati. Ad esempio, si supponga di avere in input la frase "Lo studente era in ritardo per la ⟨MASK⟩". Il modello deve assegnare al termine "lezione" la maggior probabilità di essere il termine mascherato e deve ignorare gli altri modelli. In BERT una frazione dei termini mascherati

BERT può essere impiegato per la *feature extraction*. In altri termini, se impiegato nell'ambito del NLP, permette di estrarre e combinare dei word

embedding a partire da uno o più strati nascosti. A seconda degli strati nascosti scelti si possono ottenere embedding più o meno indipendenti dal contesto. Gli embedding in input al modello sono non contestuali. Se si scelgono i primi strati si ottengono embedding contestuali più adeguati a compiti in cui è importante la sintassi del testo. Se si scelgono gli ultimi strati si possono estrarre embedding contestuali più adeguati a compiti in cui è necessario preservare la semantica del testo.

BERT richiede l'impiego di segmenti specifici per preparare l'input a seconda del compito di addestramento. In particolare se si addestra su un compito di classificazione richiede il segmento [CLS] mentre se si impiega un compito che richiede di fornire in input due frasi, come la *Next Sentence Prediction*, richiede l'impiego del separatore di frasi [SEP]. Per maggiori informazioni sulla Next Sentence Prediction si rimanda a [51].

BERT ed ELMo

ELMo impiega una rete neurale bidirezionale LSTM preaddestrata e BERT impiega l'architettura Transformer. Entrambi i modelli si addestrano su un compito di modellazione del linguaggio. Rispetto a quest'ultimo modello ELMo è un modello che rappresenta gli input a livello di carattere e apprende gli embedding a livello di termine mentre in BERT sia la rappresentazione in input che gli embedding appresi sono a livello di gruppi di caratteri. Entrambi i modelli sono in grado di limitare i termini OOV. Come nel caso di fastText, anche BERT è in grado di generare un vocabolario di dimensioni ridotte rispetto agli altri algoritmi che non impiegano segmentazione a livello di carattere o gruppi di caratteri per generare gli embedding.

3.5.4 BigBird

Bigbird è un meccanismo di attenzione sparsa, introdotto in letteratura nel 2020 [54], che permette di approssimare un meccanismo di attenzione piena e rendere più efficiente l'elaborazione di sequenze.

La ragione per cui si cerca di effettuare un'approssimazione della matrice dell'attenzione sta nel costo quadratico $\theta(n^2)$ in termini di tempo e di memoria richiesto dalla maggior parte dei modelli basati su Transformer per calcolarla. In particolare senza approssimare la matrice dell'attenzione diventa particolarmente oneroso fornire in input sequenze con più di 512 elementi.

BigBird impiega una meccanismo detto *block sparse attention* piuttosto che la normale attenzione, come quella in BERT, e può gestire sequenze di lunghezza massima pari a 4096 segmenti con un costo computazionale molto minore rispetto ad altre architetture che impiegano una matrice dell'attenzione

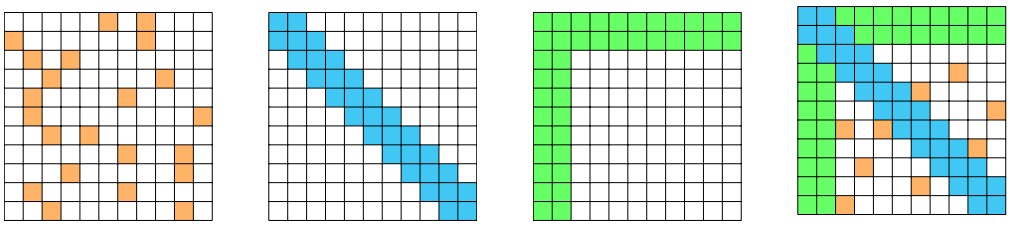
densa come BERT. BigBird rappresenta lo stato dell'arte in tutti i compiti che impiegano sequenze molto lunghe come il riassunto del testo e le risposte a domande con contesti lunghi. In ogni caso l'obiettivo di BigBird è impiegare un meccanismo di attenzione più efficiente e non migliore di BERT.

In un trasformer BERT ogni segmento della frase in input è messo in relazione con tutti gli altri. In un transformer BigBird si selezionano solo dei segmenti chiave impiegando diversi criteri. Il primo è detto *attenzione scorrevole* e consiste nel dare importanza ai segmenti che si trovano nelle immediate vicinanze del segmento in analisi. In aggiunta si introducono i *segmenti globali*, ovvero segmenti che sono messi in relazione con tutti gli altri, e *segmenti casuali*, ovvero segmenti che trasferiscono informazione ad altri segmenti. L'impiego dei segmenti casuali può ridurre il costo di trasmissione dell'informazione da un segmento all'altro. Impiegando questi tre meccanismi di attenzione è possibile individuare un sottoinsieme di segmenti chiave a partire dalla sequenza fornita in input.

In quest'ottica un meccanismo di attenzione piena può essere visto come un criterio che considera tutti i segmenti in input come segmenti globali.

Su GPU è possibile implementare efficientemente le computazioni richieste da BERT con la moltiplicazione tra matrici dense ma la moltiplicazione tra matrici sparse, richiesta da BigBird, è più difficoltosa da realizzare. Per gestire il problema si impiega la *block sparse attention*.

In un normale meccanismo di attenzione l'informazione può fluire in modo diretto tra un segmento e l'altro all'interno di un'unico strato. In un meccanismo di attenzione sparsa a blocchi è necessario che l'informazione fluisca attraverso più di un nodo per raggiungere determinati segmenti. Di conseguenza potrebbe essere necessario impiegare più strati per catturare tutta l'informazione presente nella sequenza. Per garantire che l'informazione fluisca rapidamente si impiegano i meccanismi di attenzione scorrevole, globale e casuale. In figura 3.19 è illustrata graficamente una matrice di attenzione sparsa a partire dall'impiego dei tre meccanismi visti.



(a) Attenzione casuale (b) Attenzione scorrevole (c) Attenzione globale (d) Big Bird

Figura 3.19: Componenti della una matrice di attenzione sparsa in BigBird.

Strategie di addestramento di BigBird

Esistono diverse strategie con cui addestrare un modello BigBird. Si distinguono *Internal Trasformer Construction* (ITC) ed *Extended Transformer Construction* (ETC). In ETC non si impiegano segmenti casuali o se ne impiega un numero ridotto, e si inseriscono più segmenti globali. Il ragionamento che sta alla base di questa strategia è che se ci sono abbastanza segmenti globali ci saranno sufficienti percorsi brevi attraverso cui può fluire l'informazione. In ITC si impiegano tutti e tre i meccanismi di attenzione visti. ITC ha un costo computazionale minore dato che ci saranno meno segmenti globali e sarà comunque in grado di catturare la maggior parte dell'informazione impiegando l'attenzione casuale. D'altra parte ETC è particolarmente efficace in compiti in cui è utile avere molti segmenti globali, come nella formulazione di risposte alle domande.

3.5.5 ALBERT

ALBERT [55], introdotto da un team di ricercatori di Google e Toyota nel 2019, è un modello che si pone come successore più leggero ed efficiente di BERT.

L'aumento delle dimensioni del modello porta ad un miglioramento delle sue prestazioni. Questa constatazione è stata osservata in BERT e portata alle estreme conseguenze nella famiglia di modelli GPT. Tuttavia ciò ha portato a toccare i limiti della memoria a disposizione di GPU e TPU, a tempi di addestramento elevati e ad una degradazione inattesa dei modelli. Una delle ragioni per cui i tempi di addestramento aumentano è dovuta all'overhead nella comunicazione direttamente proporzionale al numero di parametri del modello che si riscontra nell'addestramento distribuito. Riguardo alla degradazione dei modelli gli ideatori di BERT hanno dimostrato che esiste un punto di saturazione per BERT, addestrando un proprio modello BERT-xlarge con 1.27 miliardi di parametri e mostrando come le prestazioni del modello degradino rispetto alla variante BERT-large di 337 milioni di parametri proposto originariamente dai creatori di BERT.

ALBERT introduce tre importanti cambiamenti: *parametrizzazione fattorizzata degli embedding*, *condivisione dei parametri tra più strati* e *addestramento tramite Sentence Order Prediction* (SOP). Queste novità riducono il tempo di addestramento di BERT e hanno ridefinito lo stato dell'arte nei benchmark GLUE, RACE [56] e SQuAD 2.0 [57]. In particolare i ricercatori hanno creato due modelli, ALBERT-large, con 57 milioni di parametri, e una propria versione di BERT-large, con 337 milioni di parametri, e hanno dimostrato che si ottengono prestazioni migliori con ALBERT-large.

Factorized Embedding Parameterization

BERT impiega WordPiece per la generazione degli embedding da fornire in input al modello. Si tratta di rappresentazioni indipendenti dal contesto. Gli embedding contestuali si ottengono come output degli strati nascosti del modello. Perciò in BERT la dimensione degli embedding E è legata alla dimensione dello strato nascosto H. In applicazioni di NLP è necessario costruire un vocabolario di elevate dimensioni V. Perciò il numero di parametri H è direttamente proporzionale a VE. È stato osservato che solo una minima parte di questi parametri sono aggiornati in fase di addestramento. ALBERT divide i parametri di embedding in due matrici più piccole. Il numero di parametri si riduce da $O(VH)$ a $O(VE + EH)$.

Cross Layer Parameter Sharing

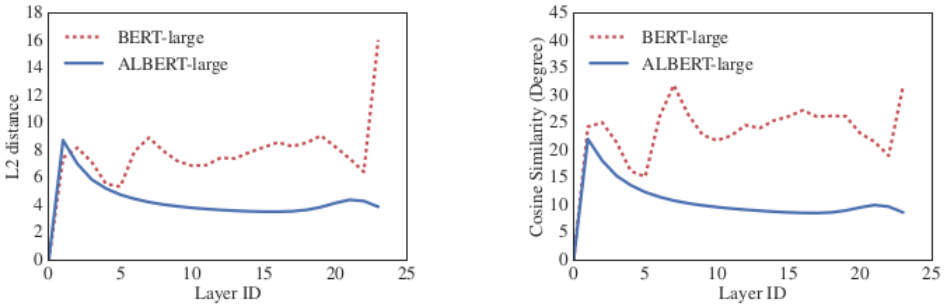


Figura 3.20: Misura della distanza euclidea e della similarità coseno espressa in gradi degli input e output embedding di ciascuno strato di BERT-Large e ALBERT-Large.
Figura 2 dell'articolo di Lan et al. [55]

La condivisione dei parametri tra gli strati FFN e i moduli di attenzione consente di ridurre il numero di parametri complessivo. In questo modo i parametri della rete sono più stabili e si ottengono transizioni da strato a strato più fluide. Ciò si osserva in figura 3.20, in cui si misurano la distanza euclidea e la similarità coseno degli input e output embedding di ciascuno strato di BERT-large e ALBERT-large.

Preaddestramento

Per il preaddestramento ALBERT impiega il MLM, con un numero massimo di maschere pari a 3. Piuttosto che utilizzare la NSP ALBERT impiega una loss per la SOP. Si impiegano due frasi estratte dallo stesso documento.

ALBERT deve determinare se le frasi sono nell'ordine giusto oppure se sono state scambiate.

3.6 Transfer Learning

Un vantaggio dei sistemi NLP basati su reti neurali sta nella possibilità di impiegare il *transfer learning*. In particolare nel caso di algoritmi di embedding come Word2Vec, GloVe e fastText si impiega solo la matrice di embedding generata dai modelli e non i modelli stessi. Modelli preaddestrati come ELMo o i Transformer possono essere utilizzati per estrarre degli embedding non contestuali, in modo analogo agli algoritmi di embedding meno recenti, oppure possono essere messi a punto per svolgere un compito specifico in modo da generare embedding contestuali rispetto al dataset su cui effettuare l'addestramento. Tramite il transfer learning si possono ottenere dei word embedding o dei modelli di linguaggio tarati su un particolare dominio applicativo rapidamente e ad un costo computazionale ridotto. Sia i word embedding che i modelli preaddestrati beneficiano della conoscenza estratta da corpus che è costoso e difficile costruire e che in genere sono molto più vasti di quelli impiegati per la successiva fase di messa a punto per lo svolgimento di un compito specifico.

3.7 Limiti dei modelli di linguaggio neurali

I modelli di linguaggio neurali hanno raggiunto lo stato dell'arte in molti compiti di elaborazione del linguaggio naturale. Tuttavia questi modelli richiedono dataset etichettati di grandi dimensioni, possono essere ingannati con input avversari [58] e non sono concepiti per spiegare la conoscenza appresa. Quest'ultimo è un grosso limite, specialmente in ambiti come quello medico, in cui le decisioni prese da un modello possono avere conseguenze potenzialmente dannose. La necessità di disporre di modelli interpretabili, ovvero più trasparenti e affidabili, ha portato alla nascita di un nuovo campo di ricerca, denominato "Explainable AI" (XAI). Un esempio di applicazione di XAI nel campo dell'elaborazione del linguaggio naturale si ha nello studio di Liu et al. [59].

Inoltre i modelli di linguaggio neurali generano embedding ad elevata dimensionalità che non si prestano ad essere visualizzati in spazi bidimensionali o tridimensionali. È necessario applicare algoritmi di riduzione della dimensionalità e clustering, al contrario di quanto accade con i modelli di linguaggio algebrici e probabilistici descritti in sezione 2.8.

Un altro aspetto da considerare riguarda le questioni etiche sollevate da questi modelli. Ad esempio la capacità di generazione del linguaggio raggiunta

dall'intelligenza artificiale grazie a modelli come GPT-3 può essere utilizzata per diffondere disinformazione e come strumento di propaganda. Inoltre gli embedding generati dai modelli di linguaggio neurali possono essere soggetti a bias di tipo razziale, religioso e di genere [33] [60].

Infine bisogna considerare che i modelli di deep learning hanno un consumo in termini di risorse computazionali ed energetico straordinario che rende la ricerca alla portata di poche grandi organizzazioni ed ha un impatto negativo dal punto di vista ecologico [61] [62].

Capitolo 4

Analisi e modellazione del progetto

A partire dalle tecnologie illustrate nei precedenti capitoli, di seguito si descrivono la metodologia, il dataset e la struttura del progetto.

4.1 Metodologia

Il progetto si basa sulla metodologia denominata *PhenOmena ExplanatIon fROm Text* [63] [1] (POIROT). POIROT appartiene all'area del Descriptive Text Mining ed è una metodologia non supervisionata che ha come obiettivo l'estrazione di spiegazioni di fenomeni di interesse a partire dal testo. In figura 4.1 è mostrata l'architettura di POIROT.

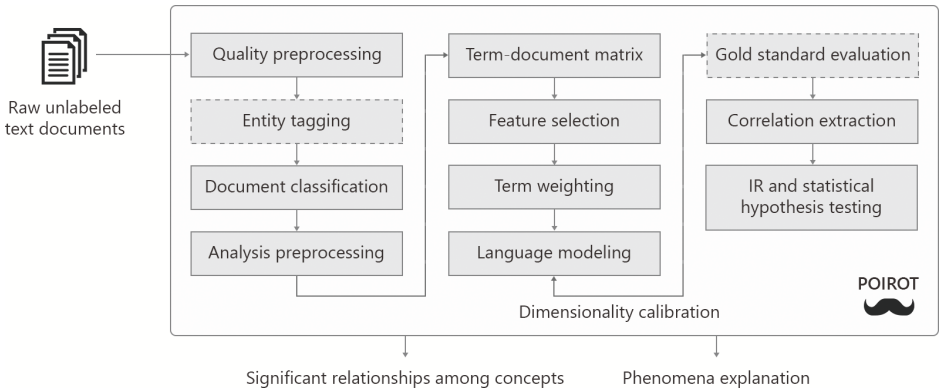


Figura 4.1: Illustrazione dell'architettura di POIROT. I contorni tratteggiati indicano i moduli opzionali.
Figura 1 tratta da [1]

Questa metodologia può essere applicata a domini totalmente differenti, da una raccolta di rapporti di incidenti aerei per estrarre informazioni sulle principali cause ad una raccolta di recensioni per comprendere le ragioni legate ad opinioni negative.

POIROT può essere impiegato sia per l'estrazione interattiva della conoscenza che tramite un processo automatico. Di seguito si riassumono brevemente le principali fasi in cui si articola l'analisi di un corpus con POIROT.

4.1.1 Preelaborazione del testo

In questa fase si preparano i dati per l'analisi, impiegando le tecniche viste in sezione 2.4. Si eliminano le stopword, i segni di punteggiatura, i numeri e gli spazi. Si applicano inoltre lemmatizzazione e casefolding. La segmentazione avviene a livello di termini, impiegando unigram.

Se necessario si effettua una fase preventiva di pulizia del testo. Ad esempio con dati provenienti da reti sociali può essere necessario gestire lo slang o l'allungamento dei termini, oltre che hashtag ed emoticon.

4.1.2 Costruzione della matrice termini documenti

Si estrae dal corpus una matrice termini documenti ad elevata dimensionalità. A partire dalla matrice termini documenti ottenuta, in cui l'elemento in posizione (i, j) indica il numero di volte in cui il termine i appare nel documento j, si effettua selezione di caratteristiche. Uno dei modi con cui si può effettuare la selezione consiste nel rimuovere dal vocabolario tutti i termini con una frequenza nei documenti inferiore all'uno percento del totale dei documenti del corpus. Infine si applica la variante del tf.idf impiegata in POIROT, in modo da ottenere una matrice termini documenti più piccola di quella originale e pesata.

4.1.3 Language modeling

In questa fase si impiega un modello di linguaggio per far emergere le similarità semantiche tra termini e documenti in uno spazio vettoriale latente. Si possono impiegare modelli di linguaggio algebrici, come LSA, probabilistici, come pLSA o LDA, o neurali, come BERT.

4.1.4 Visualizzazione di termini e documenti

A partire dallo spazio latente ottenuto tramite un modello di linguaggio si possono visualizzare le distribuzioni di termini e documenti in uno spazio bidi-

mensionale. Si può inoltre comprendere l'efficacia del modello ed eventualmente si possono rieseguire le fasi precedenti applicando delle correzioni.

Per ottenere una rappresentazione bidimensionale di documenti e termini può essere utile applicare tecniche di riduzione della dimensionalità e di clustering, come t-SNE [64], specialmente nel caso si impieghino modelli di linguaggio neurali. Nel caso di LSA, pLSA e LDA ciò non è necessario in quanto si possono ottenere rappresentazioni bidimensionali scegliendo due assi latenti a partire da $\mathbf{\Sigma}_k$. Questo perché la scelta dei primi assi permette di approssimare a sufficienza la distribuzione dei dati nello spazio latente.

Studiando la distribuzione dei documenti nelle rappresentazioni create durante l'analisi si possono identificare dei cluster formati da concentrazioni anomale di documenti, rispetto alla distribuzione nel corpus originale, appartenenti ad una determinata classe. A partire da queste concentrazioni anomale di documenti si possono determinare i termini più rilevanti per formulare spiegazioni dei fenomeni in analisi.

4.1.5 Scelta della dimensionalità

Per proseguire l'analisi si impiega un numero di dimensioni maggiori in modo da evitare una potenziale perdita di informazione. Nella scelta del numero di dimensioni, nel caso di LSA, pLSA o LDA, bisogna considerare che se k è troppo piccolo si ottengono delle false similitudini mentre se k è troppo grande ciò che è realmente simile sarà modellato come diverso. In questo caso la differenza tra ciò che è visto come diverso è frutto del caso, ovvero dovuta unicamente al rumore.

Il numero di dimensioni k dello spazio semantico latente può essere scelto in modo ottimale sia in modo analitico che osservando i *knee point*. Il punto di knee ottimale indica che la quantità di informazione catturata includendo nuove dimensioni non cresce più rapidamente e quindi non rende più vantaggioso considerare nuove dimensioni. Dal punto di vista geometrico la scelta del knee point ottimale coincide con il minimo locale del raggio di curvatura della funzione che interpola l'iperbole formata dalla distribuzione a legge di potenza dei valori singolari. La curvatura di una funzione $y = f(x)$ è definita come $c = y''/(1 + (y')^2)^{3/2}$. Un esempio di scelta del numero di valori singolari in base ai knee point si ha in figura 4.3. In questo caso si potrebbe selezionare per primo il sesto knee point in quanto rappresenta il primo minimo locale.

In genere si effettuano dei test con più punti di minimo locale per scegliere quello potenzialmente ottimo. Un modo per condurre questi test consiste nel verificare la coerenza semantica rispetto ad una interrogazione dei primi N documenti simili ad essa.

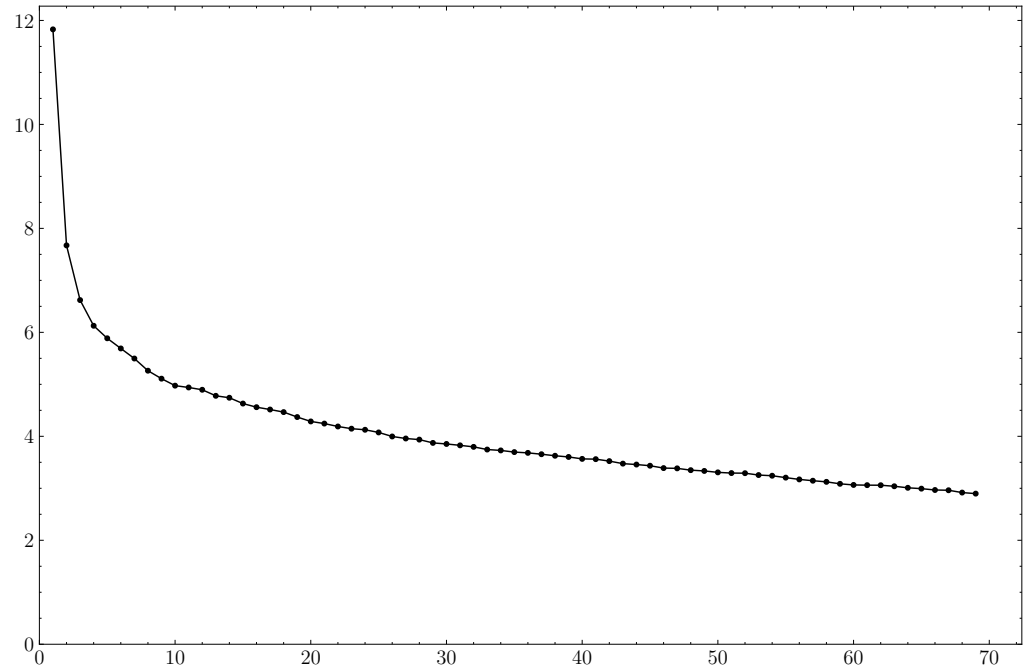


Figura 4.2: Illustrazione di una possibile distribuzione a legge di potenza dei valori singolari in seguito all'applicazione di SVD.

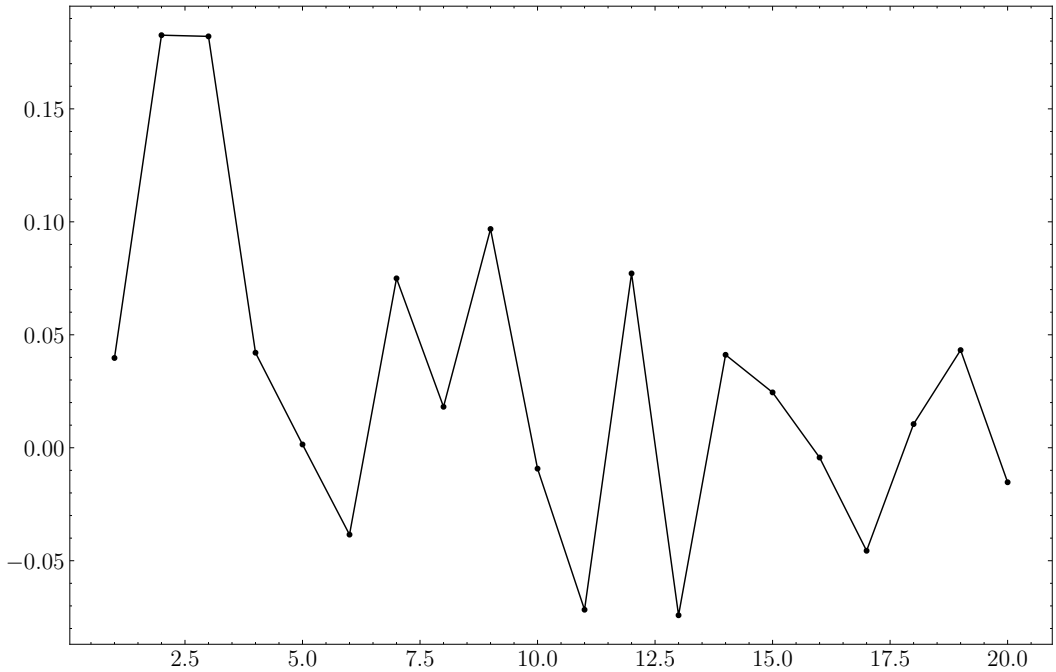


Figura 4.3: Illustrazione dei knee point ottenuti calcolando la curvatura dell'iperbole in figura 4.2.

4.1.6 Similarità nello spazio trasformato

Una caratteristica comune ai modelli di linguaggio LSA, pLSA e LDA è la possibilità di effettuare il *fold in* di nuovi documenti. Il *folding* consiste nell'aggiunta di nuovi vettori allo spazio trasformato ottenuto tramite uno dei modelli di linguaggio indicati senza ricostruirlo. In particolare si vuole aggiungere il vettore $\mathbf{q}$, formato da un insieme di termini C, allo spazio latente rappresentato da $\mathbf{V}_k$. Dato che $\mathbf{V} = \mathbf{C}^\top \mathbf{U}\boldsymbol{\Sigma}^{-1}$, allora $\mathbf{q}_k = \mathbf{q}^\top \mathbf{U}_k \boldsymbol{\Sigma}_k^{-1}$. La posizione di $\mathbf{q}$ nello spazio semantico latente è data da $\mathbf{q}^\top \mathbf{U}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Sigma} = \mathbf{q}^\top \mathbf{U}$.

Applicando la similarità coseno alla matrice termini documenti ricostruita $\mathbf{C}_k = \mathbf{U}\boldsymbol{\Sigma}_k\mathbf{V}^\top$ si distinguono cinque casi distinti a seconda che si voglia calcolare la similarità coseno tra documenti, tra termini, tra un documento e un termine tra un'interrogazione e un documento e tra un'interrogazione e un termine.

Nel primo caso, la similarità tra due documenti v_i e v_j si calcola come mostrato in equazione (4.1), a partire da $\mathbf{C}_k^\top \mathbf{C}_k$. La similarità tra i termini u_i e u_j è definita a partire da $\mathbf{C}_k\mathbf{C}_k^\top$ come in equazione (4.2).

$$\text{sim}(v_i\boldsymbol{\Sigma}_k, v_j\boldsymbol{\Sigma}_k) = \frac{(v_i\boldsymbol{\Sigma}_k)(v_j\boldsymbol{\Sigma}_k)^\top}{(\|v_i\boldsymbol{\Sigma}_k\|\|v_j\boldsymbol{\Sigma}_k\|)} \tag{4.1}$$

$$\text{sim}(u_i\boldsymbol{\Sigma}_k, u_j\boldsymbol{\Sigma}_k) = \frac{(u_i\boldsymbol{\Sigma}_k)(u_j\boldsymbol{\Sigma}_k)^\top}{(\|u_i\boldsymbol{\Sigma}_k\|\|u_j\boldsymbol{\Sigma}_k\|)} \tag{4.2}$$

La similarità tra un termine u_i e il documento v_j è definita in equazione (4.3).

$$\text{sim}(u_i\sqrt{\boldsymbol{\Sigma}_k}, \sqrt{\boldsymbol{\Sigma}_k}v_j) \tag{4.3}$$

La similarità tra un'interrogazione q e un documento richiede di trasformare l'insieme di termini in q in un nuovo documento q_k. Ciò avviene effettuando il folding dell'interrogazione in uno spazio trasformato. Per il calcolo della similarità si ottengono le due formulazioni equivalenti mostrate in equazione (4.4).

$$\text{sim}(q_k\boldsymbol{\Sigma}_k, v_i\boldsymbol{\Sigma}_k) = \text{sim}(q^\top U_k, v_i\boldsymbol{\Sigma}_k) \tag{4.4}$$

Per il calcolo della similarità tra un'interrogazione q e un termine u_i si considera la similarità tra u_i e l'interrogazione come nuovo documento nello spazio trasformato q_k. Si ottiene la definizione mostrata in equazione (4.5).

$$\text{sim}\left(u_i\sqrt{\boldsymbol{\Sigma}_k}, \frac{1}{\sqrt{\boldsymbol{\Sigma}_k}}U_k^\top q\right) \tag{4.5}$$

Se si applica la similarità coseno allo spazio trasformato è possibile osservare relazioni tra termini e documenti che saranno più vicini nella rappresentazione

dello spazio trasformato rispetto allo spazio originale. Si osserva che la similarità coseno nel modello LSA ha valori in $[-1, 1]$ mentre in pLSA e LDA ha valori in $[0, 1]$ e può quindi essere interpretata come probabilità.

4.1.7 Costruzione di spiegazioni dei fenomeni

Si formula una spiegazione probabilistica del fenomeno in analisi in modo incrementale. Si costruisce un'interrogazione passo dopo passo tramite folding. La spiegazione ottenuta tramite questo processo è formata dall'insieme di termini che meglio caratterizza il fenomeno nello spazio semantico latente.

Fold in dell'interrogazione Per prima cosa si effettua il fold-in dell'interrogazione nello spazio trasformato. Un'interrogazione può essere vista come un documento artificiale formato da più termini. Oltre ad effettuare il folding è necessario che si effettuino tutte le operazioni compiute sui documenti del corpus, come la pulizia e la preelaborazione del testo oltre che l'applicazione di uno schema di weighting e la normalizzazione.

Similarità semantica In seguito si effettua il calcolo della similarità semantica tra termini, documenti e interrogazioni. Si impiegano le definizioni di similarità coseno fornite in sezione 4.1.6.

Scelta del primo termine In base ad un'analisi della rappresentazione bidimensionale delle dimensioni latenti e ai risultati di similarità ottenuti si identifica come primo termine quello che si trova in una posizione centrale nel cluster che si sceglie di analizzare e che ha norma maggiore, ovvero una maggiore rilevanza. Questo primo termine, tramite folding, permette di iniziare a costruire l'interrogazione. Per dimostrare formalmente la correlazione tra l'interrogazione e la classe che rappresenta il fenomeno si impiegano il test chiquadro, descritto in sezione 2.10.2, e la R-precision. La R-precision consiste nell'applicare il test chiquadro ad un sottoinsieme di R documenti restituiti dall'interrogazione. Se R è pari al numero di istanze della classe c in analisi allora si considerano tutti i possibili documenti che possono essere restituiti dall'interrogazione. Per stabilire la significatività statistica di un fenomeno si impiega il p-value ottenuto dalla tabella della distribuzione χ^2 tra q e c con un grado di libertà. Si impiega un unico grado di libertà visto che la matrice di contingenza chiquadro è 2x2. Per stabilire se respingere o meno l'ipotesi nulla si impiega un valore di soglia per il p-value, pari ad esempio a 0.005.

Costruzione di una spiegazione tramite raffinamento iterativo Dopo aver verificato formalmente la rilevanza del primo termine scelto considerando

la classe con concentrazione anomala si procede con l'arricchimento della spiegazione aggiungendo nuovi termini. Si scelgono i termini da aggiungere, tra quelli con maggiore similarità e norma più elevata, che minimizzano il p-value. In altri termini la scelta avviene in modo da costruire l'interrogazione più significativa tra quelle realizzate fino a quel momento. Il processo si ripete iterativamente, fino a quando il p-value non scende sotto un valore di soglia specificato. Il processo di arricchimento alterna ricerche di correlazioni tra termini e interrogazione e tra documenti e interrogazione. La spiegazione ottenuta da questo processo è composta da un serie di termini facilmente interpretabile. In figura 4.4 è un mostrato un esempio del processo iterativo di costruzione di una spiegazione.

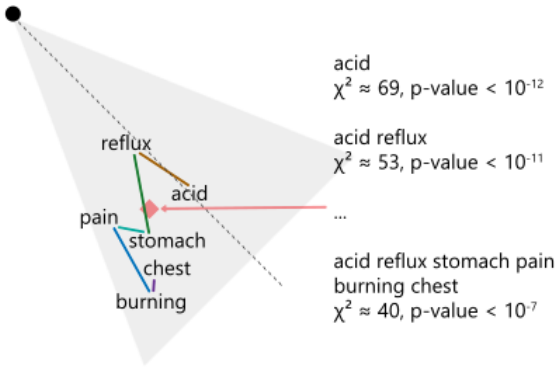


Figura 4.4: Esempio di spiegazione di un fenomeno. Si ha un'interrogazione composta da sei termini. Si osserva che con l'aggiunta di ciascun termine si arricchisce l'interrogazione con termini semanticamente vicini e la correlazione con la classe tende a diminuire.
Figura 7 tratta da [1]

4.1.8 Valutazione del modello

POIROT propone un approccio formale per la valutazione delle prestazioni dei modelli di classificazione che consente di valutare la correttezza della classificazione e la significatività delle correlazioni riscontrate tramite test chiquadro. A partire dal modello migliore identificato si effettua un'analisi locale, ovvero a partire da un sottoinsieme dei documenti, in modo da definire spiegazioni del fenomeno di interesse.

L'approccio di valutazione proposto da POIROT si basa sull'impiego di una serie di *gold standard*. I gold standard permettono di comprendere dei fenomeni a partire da correlazioni provenienti da fonti come esperti di dominio e dalla letteratura scientifica ad esempio. I gold standard rappresentano delle

correlazioni tra due documenti artificiali, ovvero tra due insiemi di termini. Si impiegano correlazioni positive che corrispondono a verità note e negative se corrispondo a falsità note. Per valutare l'efficacia del modello si effettua il folding delle due interrogazioni relative a ciascun gold standard e si applica il test chiquadro tra i vettori, per verificare che il p-value sia inferiore a un certo valore di soglia per le correlazioni positive o superiore ad un valore di soglia nel caso di correlazioni negative. In questo modo si possono applicare metriche come l'F1 score e si può creare una matrice di confusione in cui i veri positivi sono le correlazioni positive correttamente identificate, i veri negativi il numero di correlazioni negative correttamente identificate, i falsi positivi il numero di correlazioni negative a cui è assegnato erroneamente un p-value superiore ad un valore di soglia, i falsi negativi il numero di correlazioni positive a cui è stato erroneamente assegnato un p-value inferiore ad un valore di soglia.

4.2 Dataset e Dominio

In questo progetto si applicherà POIROT estrarre informazioni di carattere medico statisticamente significative a partire dal dataset ADE [2] in cui sono raccolti gli abstract di articoli scientifici appartenenti al database MeSH della banca dati PubMed. Sono presenti 23516 istanze con due caratteristiche, del testo in linguaggio naturale in inglese e una variabile categorica che indica se il testo contiene informazioni relative a effetti collaterali dovuti all'assunzione di medicinali.

A partire dal corpus di rapporti medici si possono estrarre informazioni di diverso genere. Ad esempio riguardanti gli effetti collaterali legati all'assunzione di determinati medicinali o anche le complicazioni associate ad interventi chirurgici. Queste informazioni provengono direttamente dalla letteratura scientifica e permettono di verificare l'efficacia di modelli di estrazione automatica di informazioni. La conoscenza che può essere estratta da questi dati può essere vista come un insieme di correlazioni più o meno statisticamente significative che possono essere selezionate, estratte e filtrate.

Capitolo 5

Sviluppo del progetto

In questo capitolo si descrivono le librerie e strumenti impiegati, il lavoro svolto sul dataset e i risultati ottenuti tramite i modelli di linguaggio indicati nei capitoli precedenti.

5.1 Librerie e strumenti

Per realizzare il progetto si impiegano i notebook di Jupyter [65], tramite l'interfaccia e le risorse rese disponibili da Google Colab, e i linguaggi di programmazione Python e R. Come riferimento si impiega un'implementazione in R di POIROT [1] resa disponibile dai suoi autori. Per la costruzione dello spazio semantico LSA si utilizza l'implementazione di Scikit-learn di randomized svd. [66]. Pandas [67], Numpy [68] e Scipy [69] sono impiegati per la manipolazione dei dati e il calcolo di misure statistiche e Matplotlib [70], SciencePlots [71] e Plotly [72] per realizzare visualizzazioni interattive facilmente esplorabili. Per la fase di preelaborazione si impiegano spaCy [73] e ScispaCy [74].

5.2 Struttura del progetto

Per l'analisi del corpus ADE si effettua una fase di preelaborazione del testo e di selezione di caratteristiche svolta manualmente, con successiva applicazione di LSA. Si fornisce in input una matrice termini documenti pesata con tf.idf e si sperimentano diverse combinazioni di knee point e dimensioni dello spazio semantico latente. Si realizzano alcune spiegazioni di esempio dei fenomeni di interesse in modo da mostrare le potenzialità del modello.

[1] https://github.com/unibodatascience/POIROT

Di seguito si riportano le fasi di applicazione di POIROT con LSA. Il codice sorgente è liberamente disponibile online [2].

5.3 Costruzione matrice termini documenti

La preelaborazione del testo e la costruzione della matrice termini documenti avvengono nello script `preprocessing_ade.py`. Per la costruzione della matrice termini documenti pesata si impiegano la classe `CountVectorizer` e la funzione `spacy_tokenizer`. In questa funzione si definisce la strategia adottata per la preelaborazione del testo nella forma di un generatore che filtra determinati segmenti.

```
excluded = [
#     "tok2vec",
#     "tagger",
#     "parser",
    "ner",
#     "attribute_ruler",
#     "lemmatizer"
]
nlp = spacy.load("en_core_sci_sm", exclude=excluded)
```

Per l'impiego di spaCy bisogna caricare i componenti necessari della *pipeline* relativa alla lingua inglese. L'impiego di ScispaCy permette di utilizzare delle regole specifiche di segmentazione aggiuntive adeguate al dominio scientifico e biomedico.

```
def spacy_tokenizer(sentence):
    sentence = nlp(sentence)
    tokens = [token.lemma_.lower() for token in sentence
          if token.is_stop == False
          and token.is_punct == False
          and len(token.text) > 1
          and token.is_ascii == True
          and token.is_alpha == True
          and token.is_quote == False
          and token.is_space == False
          and token.like_num == False
          and token.like_url == False]
    return tokens
```

[2] https://github.com/Da3dalu2/thesis-notebooks

```
vect = CountVectorizer(
    strip_accents = "ascii",
    tokenizer = spacy_tokenizer,
    min_df = 0.002
)
dtm = vect.fit_transform(corpus)
tdm = dtm.transpose()
```

Si osserva che si eliminano le stopword, i segni di punteggiatura, i segmenti con meno di due caratteri, i termini non ascii, i termini che non sono caratteri alfabetici e gli spazi. Inoltre si eliminano i segmenti che somigliano a numeri, come "10.9" o "dieci" o che somigliano a URL. Si applicano lemmatizzazione e casefolding ai segmenti rimasti.

Il `CountVectorizer` effettua inoltre la normalizzazione dei caratteri ascii, ad esempio rimuovendo gli accenti. Il parametro `min_df` permette di effettuare selezione di feature, ovvero di inserire nel vocabolario solo i termini con una frequenza nei documenti superiore ad un valore di soglia.

Si effettua il salvataggio della matrice termini documenti e del vocabolario su disco.

```
sp.save_npz("tdm.npz", tdm)

file_name = 'vocabulary.sav'
pickle.dump(vocabulary, open(file_name, 'wb'))
loaded_model = pickle.load(open(file_name, 'rb'))

file_name = 'doc_classes.sav'
df["class"].to_pickle(file_name)
doc_classes = pd.read_pickle(file_name)
```

5.4 Caricamento dei dati

Si effettua il caricamento della matrice termini documenti, del vocabolario e del corpus in memoria. Si eliminano i documenti duplicati in quanto non sono rilevanti per il tipo di analisi che si vuole condurre. Perciò il dataset è ora formato da 19464 istanze.

```
dataset_url = url
dataset_name = "ade_causes.csv"

if IN_COLAB:
```

```
    data_path = "./"
else:
    plt.style.use('science')
    data_path = "../data/"

if not path.exists(data_path + dataset_name):
    urlretrieve(dataset_url, dataset_name)

df = pd.read_csv(
        data_path + dataset_name,
        dtype={"text" : "string", "label" : "category"}
)
df.drop_duplicates(subset ="text", keep = False, inplace = True)

df.label = df.label.cat.rename_categories(
    {"1" : "Related",
     "0" : "Not related"}
)

dataset_name = "tdm_ade.npz"

if not path.exists(data_path + dataset_name):
    dataset_url = base_url + dataset_name
    urlretrieve(dataset_url, dataset_name)

tdm = sp.load_npz(data_path + dataset_name)
tdm = tdm.todense()
b = (1 - entropy(tdm, axis = 1))
b = np.array(b)

a = np.log(tdm + 1)
a = np.array(a)

wtdm = (a.T * b).T
wtdm = sp.csr_matrix(wtdm)

file_name = 'vocabulary_ade.sav'

if not path.exists(data_path + file_name):
    dataset_url = base_url + file_name
    urlretrieve(dataset_url, file_name)
```

```
vocabulary = pickle.load(open(data_path + file_name, 'rb'))
```

A partire dalla matrice termini documenti caricata, in cui l'elemento in posizione (i, j) indica il numero di volte in cui il termine i appare nel documento j, si applica la variante del tf.idf impiegata in POIROT, tramite la funzione `entropy`.

```
def entropy(m, axis = 0):
    m = np.asarray(m)
    p = 1.0 * m / np.sum(m, axis = axis, keepdims = True)
    ndocs = m.shape[axis]
    vec = (p * np.log(
        p,
        out = np.zeros_like(p),
        where = (p!=0))) / np.log(ndocs)
    entropy = - np.sum(vec, axis = axis)
    return entropy

# adding a nonzero scalar to a sparse matrix is not supported
tdm = tdm.todense()

a = np.log(tdm + 1)
a = np.array(a)
b = (1 - entropy(tdm, axis = 1))

wtdm = (a.T * b).T
wtdm = csr_matrix(wtdm)
```

Si ottiene in questo modo una matrice sparsa termini documenti pesata tramite la variante del tf.idf che utilizza come fattore locale l'entropia di Shannon.

5.5 Applicazione della LSA

Si effettua la creazione dello spazio LSA e il calcolo delle matrici $\mathbf{U}$, $\mathbf{\Sigma}$ e $\mathbf{V}^\top$ tramite Randomized SVD con $k = 150$. Inoltre si calcolano le due matrici di similarità semantica `tls` e `dls`, rispettivamente per i termini e i documenti. In questo modo si effettua *Latent Semantic Mapping*, ovvero si portano termini e documenti in uno spazio comune. Tramite misure di similarità, come la similarità coseno, è possibile calcolare la vicinanza tra due punti dello spazio.

```
U, s, Vh = randomized_svd(
    wtdm,
    n_components = k,
    random_state = 5)
Vh = Vh.transpose()
tls = np.matmul(U, np.diag(s))
dls = np.matmul(Vh, np.diag(s))
```

5.6 Visualizzazione dello spazio LSA

Si identificano potenziali correlazioni tra documenti e termini tramite un'analisi qualitativa. Esempi di rappresentazioni bidimensionali ottenute durante questa fase, considerando diversi assi latenti, si hanno in figura 5.3 e 5.4. Si impiegano le funzioni `plot_lsa2D` e `plot_lsa2D_termdoc`.

```
def plot_lsa2D(x, y, terms=None):
    fig = go.Figure()
    if terms is None:
        trace = go.Scatter(
            x=x,
            y=y,
            mode="markers+text",
            name="Markers and Text")
    else:
        trace = go.Scatter(
                    x=x,
                    y=y,
                    mode="markers+text",
                    name="Markers and Text",
                    text=terms,

                    textposition="bottom center",
                    textfont=dict(
                        family="sans serif",
                        size=9))

    fig.add_trace(trace)
    fig.update_layout(
        height=800,
        template = "none"
    )
```

```
    fig.update_traces(marker=dict(size=3,))
    return fig

def plot_lsa2D_termdoc(
    classes,
    d1,
    d2,
    t1,
    t2,
    ord_terms):

    dfc = pd.DataFrame(
        {"Incident" : classes,
         "lsa1" : d1,
         "lsa2" : d2}
    )

    fig = plot_lsa2D(t1, t2, ord_terms)
    fig1 = px.scatter(
        dfc,
        x = "lsa1",
        y = "lsa2",
        color="Incident",
        color_discrete_map=
        {"Not related": "lightgrey",
         "Related": "black"},
        template = "none",
        height = 800
    )

    fig.update_traces(name = "Termini", marker_color="blue")
    fig = go.Figure(data = fig1.data + fig.data)
    fig.update_traces(marker=dict(size=3,))
    fig.update_layout(
        height = 800,
        template = "none"
    )
    fig.update_traces(marker=dict(size=3,))

    return fig
```

Si osserva la distribuzione di documenti che contengono o meno informazioni

relative ad ADE quantitativamente, tramite un grafico a torta e osservando la distribuzione dei documenti ottenuta considerando i primi due assi latenti dello spazio LSA costruito. Si determina che 16599 documenti non contengono informazioni significative riguardo ad effetti collaterali. I restanti 2865 documenti hanno informazioni significative. Come si osserva in figura 5.1 la maggior parte dei documenti non forniscono informazioni su effetti collaterali a seguito dell'assunzione di medicinali. In figura 5.2 si osserva una distribuzione di documenti non rilevanti fortemente sbilanciata, con una concentrazione significativa nel primo e nel quarto quadrante. Si osserva una concentrazione di documenti relativi ad ADE nel secondo e terzo quadrante.

```
df.label.value_counts()

ade_class = list(df.label.unique())
fig = go.Figure(
    data = [go.Pie(labels = ade_class,
            values = df.label.value_counts(),
            marker = dict(colors = px.colors.qualitative.Bold),
            hole = .3)]
)
fig.show()
```

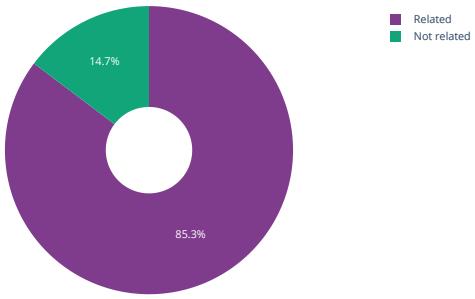


Figura 5.1: Grafico a torta della distribuzione delle tipologie di rapporti medici.

Perché sia possibile visualizzare termini e documenti senza differenze di scala si normalizzano i valori.

```
tlsn = norm(tls, 2, axis = 1)
dlsn = norm(dls, 2, axis = 1)
tlsn = tls / tlsn[:,None]
dlsn = dls / dlsn[:,None]
```

Realizzare grafici di proiezioni ottenute su dimensioni diverse permette di individuare più cluster di termini correlati. A partire da concentrazioni inaspettate di documenti relativi ad effetti collaterali, osservando i termini che si trovano vicini a queste concentrazioni, si possono identificare le cause che determinano gli ADE.

5.7 Costruzione di spiegazioni dei fenomeni

Per costruire le interrogazioni e confrontare la similarità tra interrogazioni, documenti e termini si impiegano le funzioni `makequery` e `get_similar`.

```
def get_similar(m, q, docs, threshold = 0, topn = 10):
    if threshold >= 1.0 or threshold < 0:
        raise("Threshold must be in (0,1]")

    docs = np.asarray(docs)
    docs = docs.reshape(docs.shape[0], 1)
    q = np.asarray(q)
    q = q.reshape(1, m.shape[1])
    query2doc = cosine_similarity(q, m)
    query2doc = query2doc.T

    mask = query2doc > threshold
    query2doc_masked = query2doc[mask]
    docs_masked = docs[mask]

    docs_idx = [idx for idx, elem in enumerate(mask) if elem == 1]

    # use a structured array
    dtype = [("id", int), ("doc", docs.dtype), ("sim_val", float)]
    ranks = [(z, y, x) for x, y, z in
        zip(query2doc_masked, docs_masked, docs_idx)]
    ranks = np.array(ranks, dtype = dtype)
    ranks = np.sort(ranks, order = "sim_val")
    ranks = ranks[::-1]  # docs in decreasing order
```

```
    nranks = len(ranks)
    if topn < nranks:
        ranks = ranks[:topn] # show only first top n matches
    return list(ranks)

def plot_knee_points(s):
    xlen = 20
    fordiff = lambda x : x[1:len(x)] - x[0:(len(x) - 1)]
    skd = fordiff(s)
    skdd = fordiff(skd)
    skcurv = skdd[0:xlen] / np.power(
        1 + np.power(skd[0:xlen], 2), 1.5)

    plt.figure(figsize=(12, 8))
    plt.plot(np.linspace(1, xlen, xlen), skcurv, "k.-")
    return plt

def makequery(qterms, tdm, U, s, vocabulary):
    query = np.zeros(tdm.shape[0])
    # one hot query
    query[[vocabulary.get(term) for term in qterms]] = 1
    # weighted query
    wquery = np.log(query + 1) * (1 - entropy(tdm, axis = 1))
    # folded query
    qls = np.matmul(wquery.T, U)
    # cosine similary query term
    dk = np.matmul(qls, np.diag(np.power(s, -1)))
    dksrs = np.matmul(dk, np.diag(np.sqrt(s)))
    return qterms, query, wquery, qls, dk, dksrs
```

Per stabilire la significatività statistica della spiegazione individuata si impiegano le funzioni `term_analysis`, `get_matching_docs` e `chisquare_lsa`, `get_query_result`.

```
def term_analysis(
    terms,
    doc_classes,
    without_word_in,
    with_word_in,
    without_word_out,
    with_word_out,
    quiet=False):
```

```
    get_term_doc_df = \
        lambda data, terms, doc_classes : pd.DataFrame(
            data,
            index = [f"not {terms}", f"{terms}"],
            columns = [f"not {doc_classes}", f"{doc_classes}"]
        )

    terms = "_".join(terms)
    doc_classes = "_".join(doc_classes)

    observed = [
        [without_word_out, without_word_in],
        [with_word_out, with_word_in]
    ]
    observed = get_term_doc_df(observed, terms, doc_classes)

    chi2, p, ddof, expected = chi2_contingency(
        observed,
        correction = False
    )
    expected = get_term_doc_df(expected, terms, doc_classes)

    if quiet is False:
        print(f"Pearson's Chi-squared test with {ddof} ddof")
        print("-------------------------------------\n")
        print("Observed frequencies")
        print(tabulate(observed, headers='keys', tablefmt='psql'))
        print("\nExpected frequencies")
        print(tabulate(expected, headers='keys', tablefmt='psql'))
        print(f"\nX-squared: \t{chi2}")
        print(f"p-value \t{p}")

    return chi2, p

def get_matching_docs(a, b):
    a = np.asarray(a)
    b = np.asarray(b)
    return np.isin(a, b).sum()
```

```
def chisquare_lsa(
    qclasses,
    qterms,
    query_matching_idxs,
    classes,
    dls_rows,
    quiet=False):

    destroyed_idx = np.asarray(classes == qclasses[0])
    doc_mask = destroyed_idx != 0

    query_mask = np.zeros(dls_rows)
    query_mask[query_matching_idxs] = 1

    mask = np.logical_and(doc_mask, query_mask)
    total = doc_mask.sum()
    with_word_in = mask.sum()
    without_word_in = total - with_word_in

    doc_mask = ~doc_mask
    mask = np.logical_and(doc_mask, query_mask)
    total = doc_mask.sum()
    with_word_out = mask.sum()
    without_word_out = total - with_word_out

    return term_analysis(
        qterms,
        qclasses,
        without_word_in,
        with_word_in,
        without_word_out,
        with_word_out,
        quiet=quiet
    )
def get_query_result(
    docs,
    ord_terms,
    dls,
    tls,
    qls,
    tlsn,
```

```
    r=482,
    low=0.2,
    high=0.85):

    # get matching docs
    docs_ranked = get_similar(
        dls,
        qls,
        docs,
        threshold = low,
        topn = r
    )

    # get matching terms
    terms_ranked = get_similar(
        tls[:,0:6],
        qls[0:6],
        ord_terms,
        threshold = low,
        topn = r
    )
    terms_ranked = [rank for rank in terms_ranked if rank[2] > high]
    sim_scores_reduced_idx = [rank[0] for rank in terms_ranked]
    norms = tlsn[sim_scores_reduced_idx]

    return docs_ranked, terms_ranked, norms
```

Per stabilire un valore ottimale del numero di dimensioni k da impiegare durante l'analisi si sceglie un knee point a partire dalla funzione di curvatura dei valori singolari. Si impiega la funzione `plot_knee`. Il grafico è visibile in figura 5.5.

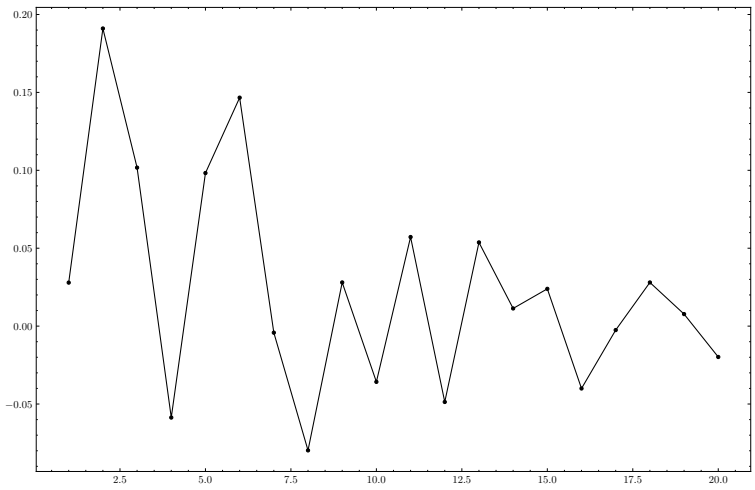


Figura 5.5: Illustrazione dei knee point ottenuti calcolando la curvatura dell'iperbole relativa ai valori singolari dello spazio LSA.

A partire dalla concentrazione di documenti relativi ad ADE in alto a destra, individuata in figura 5.4, si sceglie come primo termine rappresentativo di questa classe di documenti "acute". Si costruisce un'interrogazione con questo termine, avente norma particolarmente elevata, e si osservano i risultati ottenuti.

```
knee = 4
qterms = ["acute", "leukaemia"]
_, _, _, qls, _, _ = makequery(qterms, tdm, U, s, vocabulary);
qlsn = norm(qls, 2)
qlsn = qls / qlsn
m = dls[:,0:knee]
q = qls[0:knee]
docs = df.text

ranks = get_similar(
    m,
    q,
    docs,
    threshold = 0.2,
    topn = 482
)

query_matching_idxs = [rank[0] for rank in ranks]
sim_scores = [rank[2] for rank in ranks]
[rank[1] for rank in ranks[:10]]
```

Impiegando un punto di knee pari a 4, ovvero scegliendo il primo punto di minimo locale, si riesce a ridurre significativamente la perdita di informazione e ad ottenere risultati coerenti con l'interrogazione. Di seguito si riportano i primi dieci documenti restituiti dall'interrogazione.

```
['Acute lymphoblastic leukemia in a patient with chronic
cyanoacrylate exposure.',
 'Indolent aspergillus arthritis complicating fludarabine-based
 non-myeloablative stem cell transplantation.',
 'Secondary acute myeloid leukemia after etoposide therapy for
 haemophagocytic lymphohistiocytosis.',
 'L-asparaginase is a key component of the antileukemic therapy
 in children with acute lymphoblastic leukemia (ALL).',
 'It is estimated that 3-10% of children with acute lymphoblastic
 leukemia (ALL) experience acute, transient neurotoxicity during
 induction chemotherapy.',
 'MK-0457, a novel kinase inhibitor, is active in patients with
 chronic myeloid leukemia or acute lymphocytic leukemia with the
 T315I BCR-ABL mutation.',
 'Pulmonary veno-occlusive disease accompanied by microangiopathic
 hemolytic anemia 1 year after a second bone marrow transplantation
 for acute lymphoblastic leukemia.',
 'All-trans retinoic acid and short-time, high-dose cytarabine in
 two children with acute promyelocytic leukemia.',
 'L-asparaginase is a critical component in the treatment of acute
 lymphoblastic leukemia in children.',
 'Steroid-induced alterations of mood and behavior in children
 during treatment for acute lymphoblastic leukemia.']
```

Si osserva che esiste una correlazione oggettiva tra l'interrogazione e i documenti che riportano ADE nello spazio semantico latente. In questo caso i documenti considerati potrebbero anche non contenere i due termini, ma termini semanticamente correlati ad essi.

Per determinare se esiste una correlazione tra "acute leukemia" e i rapporti medici in cui si menzionano ADE si impiega un modello di ricerca semantico basato su ranking. Si fissa il numero massimo di risultati che possono essere restituiti dal modello tramite R-precision. Si sceglie come valore di r 2865 dato che questo è il numero di rapporti medici relativo ad ADE.

Di seguito si riporta procedimento interattivo impiegato per costruire iterativamente l'interrogazione formata dai termini "diabetes", "methylprednisolone", "warfarin" e "hypersensitivity".

```
docs = df.text
qterms = ["diabetes", "methylprednisolone"]
_, _, _, qls, _, _ = makequery(qterms, tdm, U, s, vocabulary);

docs_ranked, terms_ranked, norms = \
    get_query_result(
        docs,
        ord_terms,
        dls[:,0:knee],
        tls[:,0:knee],
        qls[0:knee],
        tlsn,
        high=0.8
    )
docs_matching_query = [rank[0] for rank in docs_ranked]
ranked_terms = [(rank[2], rank[1], norm) for rank, norm in
    zip(terms_ranked, norms)]
ranked_terms.sort(key=lambda y: y[2], reverse=True)
ranked_terms
```

```
[(0.9059624529322615, 'de', 3.098790777476597),
 (0.9292057721174412, 'drug', 3.0459141251617408),
 (0.9922276379095303, 'methylprednisolone', 2.374447038693702),
 (0.8379993969718977, 'warfarin', 1.688969480185153),
 (0.9813112545639446, 'damage', 1.1992031445607383),
 (0.938456443929159, 'hypersensitivity', 0.8301021848884711),
 (0.8333314892914753, 'dose', 0.5701440761985425),
 (0.9828879061925067, 'tumour', 0.43828679056423603),
 (0.8052882771527794, 'recovery', 0.41068290238415284),
 (0.81410590207735, 'believe', 0.3663996603749401),
 (0.8089623659401485, 'complex', 0.3628651822624118),
 (0.810826747024739, 'discontinuation', 0.3551777355519015),
 (0.8171443731999761, 'intervention', 0.25522527946995366)]
```

```
qclasses = ["Related"]
dls_rows = dls.shape[0]
classes = df["label"]

chisquare_lsa(
    qclasses,
    qterms,
    docs_matching_query,
```

```
    classes,
    dls_rows
);

Pearson's Chi-squared test with 1 ddof
--------------------------------------

Observed frequencies
+---------------------+---------------+-----------+
|                     |   not Related |   Related |
|---------------------+---------------+-----------|
| not matching query  |         16151 |      2831 |
| matching query      |           448 |        34 |
+---------------------+---------------+-----------+

Expected frequencies
+---------------------+---------------+-----------+
|                     |   not Related |   Related |
|---------------------+---------------+-----------|
| not matching query  |     16187.9   | 2794.05   |
| matching query      |       411.052 |   70.9479 |
+---------------------+---------------+-----------+

X-squared:      23.135577154316614
p-value         1.509713405643772e-06
```

```
docs = df.text
qterms = ["diabetes", "methylprednisolone", "warfarin"]
_, _, _, qls, _, _ = makequery(qterms, tdm, U, s, vocabulary);

docs_ranked1, terms_ranked1, norms1 = \
    get_query_result(
        docs,
        ord_terms,
        dls[:,0:knee],
        tls[:,0:knee],
        qls[0:knee],
        tlsn,
        high=0.85
    )
docs_matching_query1 = [rank[0] for rank in docs_ranked1]
ranked_terms = [(rank[2], rank[1], norm) for rank, norm in
```

```
    zip(terms_ranked1, norms1)]
ranked_terms.sort(key=lambda y: y[2], reverse=True)
ranked_terms
```

```
[(0.9503359392348931, 'de', 3.098790777476597),
 (0.8999089685598747, 'drug', 3.0459141251617408),
 (0.9910590490074458, 'methylprednisolone', 2.374447038693702),
 (0.9115272833620914, 'warfarin', 1.688969480185153),
 (0.9843489251333316, 'damage', 1.1992031445607383),
 (0.9553757083444822, 'hypersensitivity', 0.8301021848884711),
 (0.9884755997218097, 'tumour', 0.43828679056423603),
 (0.8751701940574174, 'believe', 0.3663996603749401),
 (0.8689879802683115, 'complex', 0.3628651822624118),
 (0.8726550057307292, 'discontinuation', 0.3551777355519015),
 (0.8585312474213622, 'persist', 0.2674752578048155),
 (0.8606685445379948, 'intervention', 0.25522527946995366)]
```

```
qclasses = ["Related"]
dls_rows = dls.shape[0]
classes = df["label"]
```

```
chisquare_lsa(
    qclasses,
    qterms,
    docs_matching_query1,
    classes,
    dls_rows
);
```

```
Pearson's Chi-squared test with 1 ddof
--------------------------------------

Observed frequencies
+---------------------+---------------+-----------+
|                     |   not Related |   Related |
|---------------------+---------------+-----------|
| not matching query  |         16151 |      2831 |
| matching query      |           448 |        34 |
+---------------------+---------------+-----------+

Expected frequencies
```

```
+--------------------+--------------+-----------+
|                    |   not Related |   Related |
|--------------------+--------------+-----------|
| not matching query |     16187.9   | 2794.05   |
| matching query     |       411.052 |   70.9479 |
+--------------------+--------------+-----------+

X-squared:      23.135577154316614
p-value         1.509713405643772e-06
```

```
get_matching_docs(docs_matching_query, docs_matching_query1)
```

```
2458
```

```
docs = df.text
qterms =
    ["diabetes", "methylprednisolone", "warfarin", "hypersensitivity"]
_, _, _, qls, _, _ = makequery(qterms, tdm, U, s, vocabulary);

docs_ranked2, terms_ranked2, norms2 = \
    get_query_result(
        docs,
        ord_terms,
        dls[:,0:knee],
        tls[:,0:knee],
        qls[0:knee],
        tlsn,
        high=0.85
    )
docs_matching_query2 = [rank[0] for rank in docs_ranked2]
ranked_terms = [(rank[2], rank[1], norm) for rank, norm in
    zip(terms_ranked2, norms2)]
ranked_terms.sort(key=lambda y: y[2], reverse=True)
ranked_terms
```

```
[(0.9509150811168253, 'de', 3.098790777476597),
 (0.9110822249961386, 'drug', 3.0459141251617408),
 (0.9933860669268725, 'methylprednisolone', 2.374447038693702),
 (0.9093594959367148, 'warfarin', 1.688969480185153),
 (0.9881024508599245, 'damage', 1.1992031445607383),
 (0.960960088656737, 'hypersensitivity', 0.8301021848884711),
 (0.991707457423017, 'tumour', 0.43828679056423603),
```

```
 (0.852346843453029, 'recovery', 0.41068290238415284),
 (0.876638915337093, 'believe', 0.3663996603749401),
 (0.8717601521780104, 'complex', 0.3628651822624118),
 (0.8751164573980224, 'discontinuation', 0.3551777355519015),
 (0.8619041345504364, 'persist', 0.2674752578048155),
 (0.867064782774706, 'intervention', 0.25522527946995366)]

qclasses = ["Related"]
dls_rows = dls.shape[0]
classes = df["label"]

chisquare_lsa(
    qclasses,
    qterms,
    docs_matching_query2,
    classes,
    dls_rows
);

Pearson's Chi-squared test with 1 ddof
--------------------------------------

Observed frequencies
+---------------------+---------------+-----------+
|                     |   not Related |   Related |
|---------------------+---------------+-----------|
| not matching query  |         16149 |      2833 |
| matching query      |           450 |        32 |
+---------------------+---------------+-----------+

Expected frequencies
+---------------------+---------------+-----------+
|                     |   not Related |   Related |
|---------------------+---------------+-----------|
| not matching query  |     16187.9   | 2794.05   |
| matching query      |       411.052 |   70.9479 |
+---------------------+---------------+-----------+

X-squared:          25.70803645099662
p-value         3.9716969291666143e-07
```

```
get_matching_docs(docs_matching_query, docs_matching_query2)
```

```
2421
```

Una possibile spiegazione che può essere estratta dall'interrogazione è quella riportata di seguito. Il warfarin è un medicinale al quale si può essere estremamente ipersensibili. Il methylprednisolone è un medicinale che può indurre il diabete in pazienti a rischio e può aggravare le condizioni di chi è già diabetico. Infatti l'assunzione di methylprednisolone può aumentare i livelli di glucosio e perciò può essere dannosa nel caso si abbia il diabete. Il warfarin (Coumadin) è un mecidinale che può provocare effetti collaterali se utilizzato insieme a farmaci contro il diabete, anche per la sua funzione anticoagulante. Provoca una drastica diminuzione del livello di glucosio nel sangue.

Altri esempi di interrogazioni non relative ad ADE sono disponibili nel repository online.

La costruzione di interrogazioni a partire dalla scelta del primo termine può essere automatizzata. Di seguito si propone un possibile approccio basato su quello proposto nell'ambito di POIROT. Si minimizza il p-value e si prosegue fino a quando il p-value dell'interrogazione costruita fino a quel momento non scende sotto un certo valore di tolleranza `tol` oppure se la lista di termini tra cui scegliere per espandere l'interrogazione si esaurisce.

```
knee = 7
r = 482
tol = 0.49
tlsn = norm(tls[:,0:knee], 2, axis = 1)
docs = df.text
qclasses = ["Related"]
dls_rows = dls.shape[0]
classes = df["label"]
qterms = ["liver"]
_, _, _, qls, _, _ = makequery(
    qterms, tdm, U, s, vocabulary);

docs_ranked, terms_ranked, norms = \
    get_query_result(
        docs,
        ord_terms,
        dls[:,0:knee],
        tls[:,0:knee],
        qls[0:knee],
        tlsn,
```

```
        high=0.85,
        r=r
    )
docs_matching_query = [rank[0] for rank in docs_ranked]

chi2, p = chisquare_lsa(
    qclasses,
    qterms,
    docs_matching_query,
    classes,
    dls_rows,
    quiet=True
)
current_p_value = p
current_terms_ranked = terms_ranked
exit = False

while True:
    terms_list = []
    for term in current_terms_ranked:

        if term[1] not in qterms:
            tmpq = qterms + [term[1]]
            _, _, _, qls, _, _ = makequery(
                    tmpq, tdm, U, s, vocabulary);

            docs_ranked, terms_ranked, norms = \
            get_query_result(
                docs,
                ord_terms,
                dls[:,0:knee],
                tls[:,0:knee],
                qls[0:knee],
                tlsn,
                high=0.9,
                r=r
            )
            docs_matching_query =
                [rank[0] for rank in docs_ranked]

            chi2, p = chisquare_lsa(
```

```
                qclasses,
                tmpq,
                docs_matching_query,
                classes,
                dls_rows,
                quiet=True
            )
            terms_list.append((term[1], p, terms_ranked))

    if len(terms_list) == 0:
        exit = True
    else:
        terms_list.sort(key=lambda y: y[1])
        print(terms_list[0][0] + " p: " + str(terms_list[0][1]))
        qterms.append(terms_list[0][0])
        current_p_value = terms_list[0][1]
        current_terms_ranked = terms_list[0][2]

    if current_p_value > tol or exit == True:
        break;
```

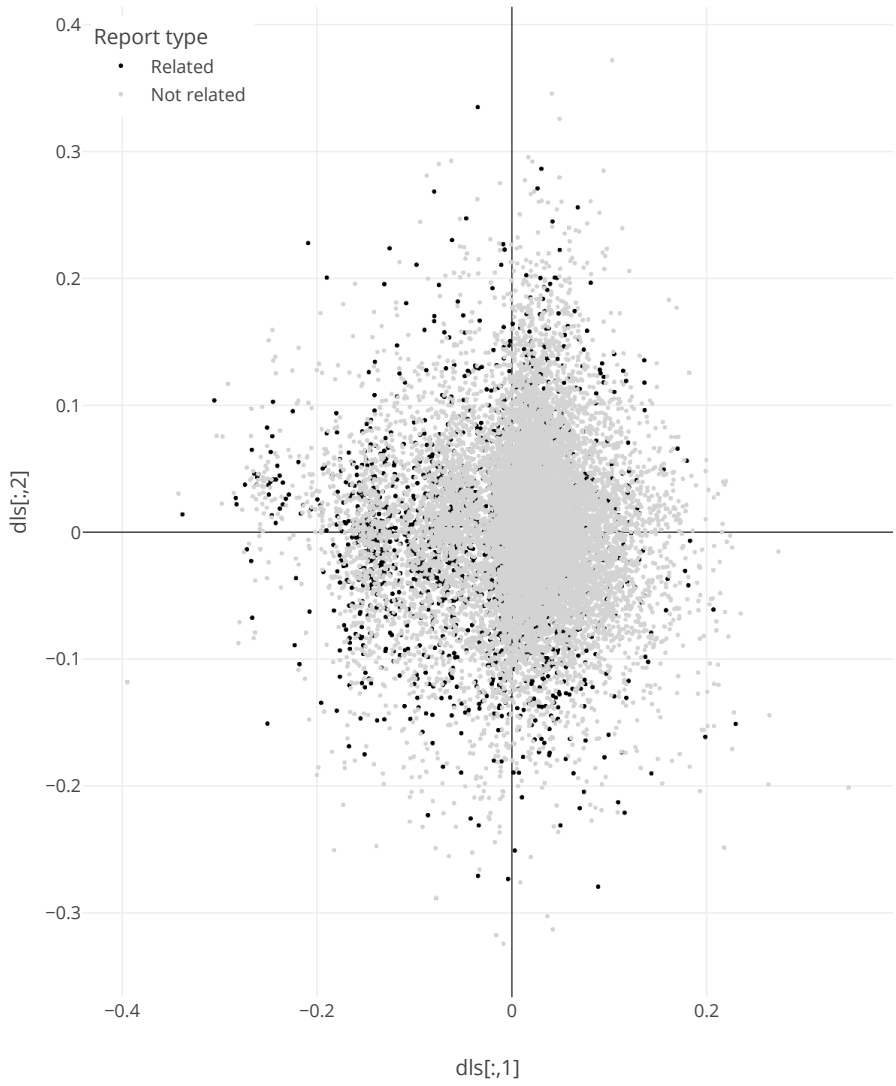


Figura 5.2: Rappresentazione bidimensionale dello spazio LSA ottenuta considerando la seconda e la terza dimensione latente. Sono rappresentati, normalizzati i documenti. I documenti che contengono informazioni relative ad effetti collaterali sono colorati in nero, altrimenti sono grigio chiaro.

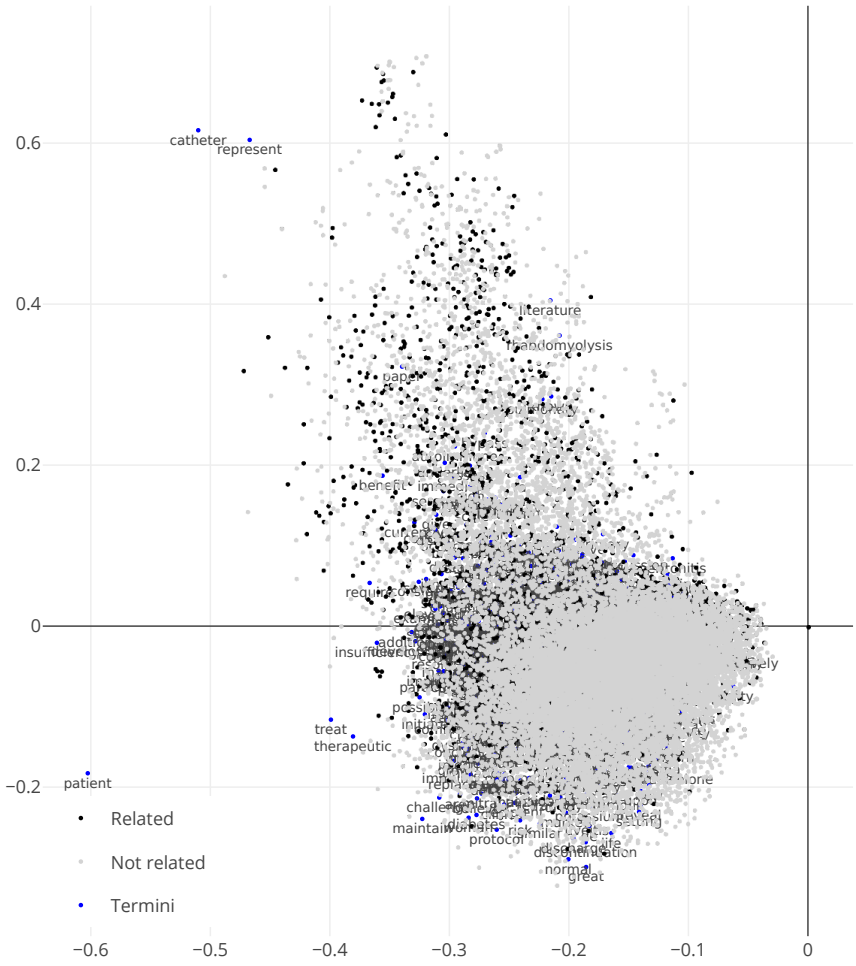


Figura 5.3: Rappresentazione bidimensionale dello spazio LSA ottenuta considerando la prima e la seconda dimensione latente. Sono rappresentati, normalizzati, sia i termini che i singoli documenti. I documenti che contengono informazioni relative ad effetti collaterali sono colorati in nero, altrimenti sono grigio chiaro.

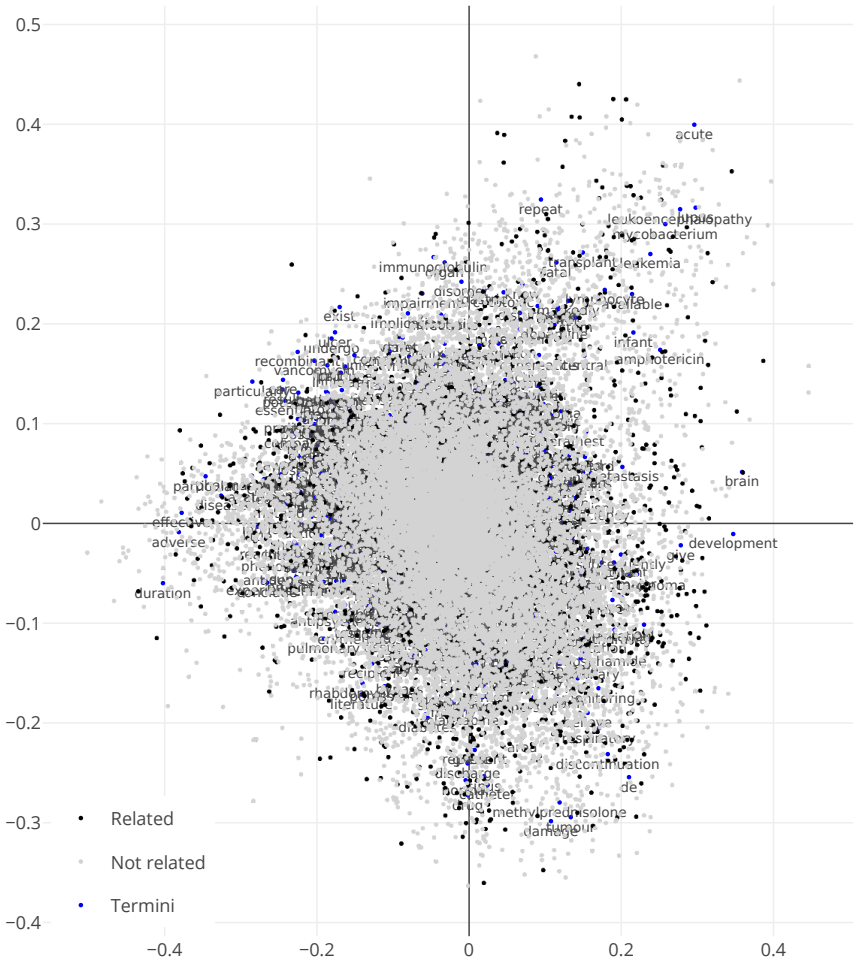


Figura 5.4: Rappresentazione bidimensionale dello spazio LSA ottenuta considerando la terza e la quarta dimensione latente. Sono rappresentati, normalizzati, sia i termini che i singoli documenti. I documenti che contengono informazioni relative ad effetti collaterali sono colorati in nero, altrimenti sono grigio chiaro.

Conclusioni e sviluppi futuri

Nel corso di questa tesi è stato possibile approfondire il funzionamento di una metodologia di Descriptive Text Mining, POIROT, per l'apprendimento di conoscenza interpretabile e statisticamente significativa.

Sono state inizialmente approfondite alcune della basi teoriche del Machine Learning e del Natural Language Processing. In seguito si sono analizzati i diversi modi in cui si è cercato di rappresentare il testo in linguaggio naturale, in un formato che potesse essere elaborato da una macchina e che permettesse di catturare informazioni sia di tipo sintattico che semantico.

A partire dalla conoscenza acquisita sul funzionamento di POIROT e dalle basi teoriche riportate nei primi tre capitoli è stato realizzato un progetto che permettesse di applicare la metodologia ideata ad un nuovo dataset relativo ad ADE. Inoltre sono stati realizzati dei notebook in Python a partire dall'esercitazione proposta dal Prof. Moro nel corso di Data Mining e dall'analisi condotta sui pazienti dell'Acalasia Esofagea in [1]. A partire da questi esempi è stato possibile sperimentare l'utilizzo di una metodologia che può essere utilizzata per ricavare informazioni utili a medici e ricercatori.

Gli sviluppi futuri comprendono l'impiego di modelli di linguaggio neurali in modo da generare embedding contestuali e la sperimentazione di diverse combinazioni di embedding. Inoltre si potrebbero impiegare la named entity recognition e l'entity linking per creare reti semantiche e per osservare come si evolve la conoscenza nel tempo.

Ringraziamenti

Il primo ringraziamento va al relatore di questo lavoro, il Prof. Gianluca Moro, che ha reso possibile tutto ciò e ha acceso il mio personale interesse nei confronti di questa splendida disciplina. Un ringraziamento ancora più grande va alla mia famiglia.

Bibliografia

[1] Giacomo Frisoni e Gianluca Moro. «Phenomena Explanation from Text: Unsupervised Learning of Interpretable and Statistically Significant Knowledge». In: *Data Management Technologies and Applications - 9th International Conference, DATA 2020, Virtual Event, July 7-9, 2020, Revised Selected Papers*. A cura di Slimane Hammoudi, Christoph Quix e Jorge Bernardino. Vol. 1446. Communications in Computer and Information Science. Springer, 2020, pp. 293–318. DOI: 10.1007/978-3-030-83014-4_14. URL: https://doi.org/10.1007/978-3-030-83014-4%5C_14 (cit. alle pp. vii, 73, 79, 107).

[2] Harsha Gurulingappa et al. «Development of a benchmark corpus to support the automatic extraction of drug-related adverse effects from medical case reports». In: *J. Biomed. Informatics* 45.5 (2012), pp. 885–892. DOI: 10.1016/j.jbi.2012.04.008. URL: https://doi.org/10.1016/j.jbi.2012.04.008 (cit. alle pp. vii, 29, 80).

[3] Joel Grus. *Data science from scratch: first principles with Python*. en. 1st. Sebastopol, CA: O'Reilly Media, 2015. ISBN: 978-1-4919-0142-7 (cit. a p. viii).

[4] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. English. 2nd. Beijing China ; Sebastopol, CA: O'Reilly Media, ott. 2019. ISBN: 978-1-4920-3264-9 (cit. alle pp. viii, 9).

[5] Ankur A. Patel e Ajay Uppili Arasanipalai. *Applied Natural Language Processing in the Enterprise: Teaching Machines to Read, Write, and Understand*. English. 1st. Beijing Boston Farnham Sebastopol Tokyo: O'Reilly Media, giu. 2021. ISBN: 978-1-4920-6257-8 (cit. alle pp. viii, 22).

[6] Aston Zhang et al. «Dive into Deep Learning». In: *CoRR* abs/2106.11342 (2021). arXiv: 2106.11342. URL: https://arxiv.org/abs/2106.11342 (cit. alle pp. viii, 47, 50, 51, 53).

[7] Michele Banko e Eric Brill. «Scaling to Very Very Large Corpora for Natural Language Disambiguation». In: *Association for Computational Linguistic, 39th Annual Meeting and 10th Conference of the European Chapter, Proceedings of the Conference, July 9-11, 2001, Toulouse, France*. Morgan Kaufmann Publishers, 2001, pp. 26–33. DOI: 10.3115/1073012.1073017. URL: https://aclanthology.org/P01-1005/ (cit. a p. 4).

[8] Alon Y. Halevy, Peter Norvig e Fernando Pereira. «The Unreasonable Effectiveness of Data». In: *IEEE Intell. Syst.* 24.2 (2009), pp. 8–12. DOI: 10.1109/MIS.2009.36. URL: https://doi.org/10.1109/MIS.2009.36 (cit. a p. 4).

[9] David H. Wolpert. «The Lack of A Priori Distinctions Between Learning Algorithms». In: *Neural Comput.* 8.7 (1996), pp. 1341–1390. DOI: 10.1162/neco.1996.8.7.1341. URL: https://doi.org/10.1162/neco.1996.8.7.1341 (cit. a p. 7).

[10] Gavin C. Cawley e Nicola L. C. Talbot. «On Over-fitting in Model Selection and Subsequent Selection Bias in Performance Evaluation». In: *J. Mach. Learn. Res.* 11 (2010), pp. 2079–2107. URL: http://portal.acm.org/citation.cfm?id=1859921 (cit. a p. 14).

[11] Sudhir Varma e Richard Simon. «Bias in error estimation when using cross-validation for model selection». In: *BMC Bioinform.* 7 (2006), p. 91. DOI: 10.1186/1471-2105-7-91. URL: https://doi.org/10.1186/1471-2105-7-91 (cit. a p. 15).

[12] Sebastian Raschka. «Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning». In: *CoRR* abs/1811.12808 (2018). arXiv: 1811.12808. URL: http://arxiv.org/abs/1811.12808 (cit. a p. 16).

[13] Trevor Hastie, Robert Tibshirani e Jerome H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, 2nd Edition*. Springer Series in Statistics. Springer, 2009. ISBN: 9780387848570. DOI: 10.1007/978-0-387-84858-7. URL: https://doi.org/10.1007/978-0-387-84858-7 (cit. a p. 18).

[14] Alan M. Turing. «Computing Machinery and Intelligence». In: *The Philosophy of Artificial Intelligence*. A cura di Margaret A. Boden. Oxford readings in philosophy. Oxford University Press, 1990, pp. 40–66 (cit. a p. 21).

[15] Mitchell P. Marcus, Beatrice Santorini e Mary Ann Marcinkiewicz. «Building a Large Annotated Corpus of English: The Penn Treebank». In: *Comput. Linguistics* 19.2 (1993), pp. 313–330 (cit. a p. 24).

[16] Slav Petrov, Dipanjan Das e Ryan T. McDonald. «A Universal Part-of-Speech Tagset». In: *Proceedings of the Eighth International Conference on Language Resources and Evaluation, LREC 2012, Istanbul, Turkey, May 23-25, 2012*. A cura di Nicoletta Calzolari et al. European Language Resources Association (ELRA), 2012, pp. 2089–2096. URL: http://www.lrec-conf.org/proceedings/lrec2012/summaries/274.html (cit. a p. 24).

[17] Marie-Catherine de Marneffe et al. «Universal Stanford dependencies: A cross-linguistic typology». In: *Proceedings of the Ninth International Conference on Language Resources and Evaluation, LREC 2014, Reykjavik, Iceland, May 26-31, 2014*. A cura di Nicoletta Calzolari et al. European Language Resources Association (ELRA), 2014, pp. 4585–4592. URL: http://www.lrec-conf.org/proceedings/lrec2014/summaries/1062.html (cit. a p. 24).

[18] Rico Sennrich, Barry Haddow e Alexandra Birch. «Neural Machine Translation of Rare Words with Subword Units». In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics, 2016. DOI: 10.18653/v1/p16-1162. URL: https://doi.org/10.18653/v1/p16-1162 (cit. a p. 28).

[19] Yonghui Wu et al. «Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation». In: *CoRR* abs/1609.08144 (2016). arXiv: 1609.08144. URL: http://arxiv.org/abs/1609.08144 (cit. a p. 28).

[20] Taku Kudo e John Richardson. «SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing». In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, EMNLP 2018: System Demonstrations, Brussels, Belgium, October 31 - November 4, 2018*. A cura di Eduardo Blanco e Wei Lu. Association for Computational Linguistics, 2018, pp. 66–71. DOI: 10.18653/v1/d18-2012. URL: https://doi.org/10.18653/v1/d18-2012 (cit. a p. 28).

[21] Scott C. Deerwester et al. «Indexing by Latent Semantic Analysis». In: *J. Am. Soc. Inf. Sci.* 41.6 (1990), pp. 391–407. DOI: 10.1002/(SICI)1097-4571(199009)41:6\<391::AID-ASI1\>3.0.CO;2-9. URL: https://doi.org/10.1002/(SICI)1097-4571(199009)41:6%5C%3C391::AID-ASI1%5C%3E3.0.CO;2-9 (cit. a p. 30).

[22] Thomas Hofmann. «Probabilistic Latent Semantic Indexing». In: *SIGIR '99: Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, August 15-19, 1999, Berkeley, CA, USA*. A cura di Fredric C. Gey, Marti A. Hearst e Richard M. Tong. ACM, 1999, pp. 50–57. DOI: 10.1145/312624.312649. URL: https://doi.org/10.1145/312624.312649 (cit. alle pp. 30, 34).

[23] Iraklis A. Klampanos. «Manning Christopher, Prabhakar Raghavan, Hinrich Schütze: Introduction to information retrieval». In: *Inf. Retr.* 12.5 (2009), pp. 609–612. DOI: 10.1007/s10791-009-9096-x. URL: https://doi.org/10.1007/s10791-009-9096-x (cit. alle pp. 31, 35).

[24] Giacomo Domeniconi et al. «A Comparison of Term Weighting Schemes for Text Classification and Sentiment Analysis with a Supervised Variant of tf.idf». In: *Data Management Technologies and Applications - 4th International Conference, DATA 2015, Colmar, France, July 20-22, 2015, Revised Selected Papers*. A cura di Markus Helfert et al. Vol. 584. Communications in Computer and Information Science. Springer, 2015, pp. 39–58. DOI: 10.1007/978-3-319-30162-4_4. URL: https://doi.org/10.1007/978-3-319-30162-4%5C_4 (cit. a p. 32).

[25] David M. Blei. «Probabilistic topic models». In: *Commun. ACM* 55.4 (2012), pp. 77–84. DOI: 10.1145/2133806.2133826. URL: http://doi.acm.org/10.1145/2133806.2133826 (cit. a p. 34).

[26] Christos H. Papadimitriou et al. «Latent Semantic Indexing: A Probabilistic Analysis». In: *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 1-3, 1998, Seattle, Washington, USA*. A cura di Alberto O. Mendelzon e Jan Paredaens. ACM Press, 1998, pp. 159–168. DOI: 10.1145/275487.275505. URL: https://doi.org/10.1145/275487.275505 (cit. a p. 34).

[27] David M. Blei, Andrew Y. Ng e Michael I. Jordan. «Latent Dirichlet Allocation». In: *J. Mach. Learn. Res.* 3 (2003), pp. 993–1022. URL: http://jmlr.org/papers/v3/blei03a.html (cit. a p. 34).

[28] Nathan Halko, Per-Gunnar Martinsson e Joel A. Tropp. «Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions». In: *SIAM Rev.* 53.2 (2011), pp. 217–288. DOI: 10.1137/090771806. URL: https://doi.org/10.1137/090771806 (cit. a p. 36).

[29] F. Rosenblatt. «The perceptron: a probabilistic model for information storage and organization in the brain». eng. In: *Psychological Review* 65.6 (nov. 1958), pp. 386–408. ISSN: 0033-295X. DOI: 10/fg6wr5 (cit. a p. 41).

[30] Tomás Mikolov et al. «Efficient Estimation of Word Representations in Vector Space». In: *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*. A cura di Yoshua Bengio e Yann LeCun. 2013. URL: http://arxiv.org/abs/1301.3781 (cit. alle pp. 43, 44).

[31] Jeffrey Pennington, Richard Socher e Christopher D. Manning. «Glove: Global Vectors for Word Representation». In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*. A cura di Alessandro Moschitti, Bo Pang e Walter Daelemans. ACL, 2014, pp. 1532–1543. DOI: 10.3115/v1/d14-1162. URL: https://doi.org/10.3115/v1/d14-1162 (cit. a p. 43).

[32] Piotr Bojanowski et al. «Enriching Word Vectors with Subword Information». In: *Trans. Assoc. Comput. Linguistics* 5 (2017), pp. 135–146. URL: https://transacl.org/ojs/index.php/tacl/article/view/999 (cit. a p. 43).

[33] Tolga Bolukbasi et al. «Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings». In: *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*. A cura di Daniel D. Lee et al. 2016, pp. 4349–4357. URL: https://proceedings.neurips.cc/paper/2016/hash/a486cd07e4ac3d270571622f4f316ec5-Abstract.html (cit. alle pp. 45, 72).

[34] Xavier Glorot e Yoshua Bengio. «Understanding the difficulty of training deep feedforward neural networks». In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010*. A cura di Yee Whye Teh e D. Mike Titterington. Vol. 9. JMLR Proceedings. JMLR.org, 2010, pp. 249–256. URL: http://proceedings.mlr.press/v9/glorot10a.html (cit. a p. 49).

[35] Yoshua Bengio, Réjean Ducharme e Pascal Vincent. «A Neural Probabilistic Language Model». In: *Advances in Neural Information Processing Systems 13, Papers from Neural Information Processing Systems (NIPS) 2000, Denver, CO, USA*. A cura di Todd K. Leen, Thomas G. Dietterich e Volker Tresp. MIT Press, 2000, pp. 932–938. URL: https://proceedings.neurips.cc/paper/2000/hash/728f206c2a01bf572b5940d7d9a8fa4c-Abstract.html (cit. a p. 50).

[36] Sepp Hochreiter e Jürgen Schmidhuber. «Long Short-Term Memory». In: *Neural Comput.* 9.8 (1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735. URL: https://doi.org/10.1162/neco.1997.9.8.1735 (cit. a p. 51).

[37] Kyunghyun Cho et al. «On the Properties of Neural Machine Translation: Encoder-Decoder Approaches». In: *Proceedings of SSST@EMNLP 2014, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation, Doha, Qatar, 25 October 2014*. A cura di Dekai Wu et al. Association for Computational Linguistics, 2014, pp. 103–111. DOI: 10.3115/v1/W14-4012. URL: https://aclanthology.org/W14-4012/ (cit. a p. 52).

[38] Mike Schuster e Kuldip K. Paliwal. «Bidirectional recurrent neural networks». In: *IEEE Trans. Signal Process.* 45.11 (1997), pp. 2673–2681. DOI: 10.1109/78.650093. URL: https://doi.org/10.1109/78.650093 (cit. a p. 54).

[39] Matthew E. Peters et al. «Deep Contextualized Word Representations». In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*. A cura di Marilyn A. Walker, Heng Ji e Amanda Stent. Association for Computational Linguistics, 2018, pp. 2227–2237. DOI: 10.18653/v1/n18-1202. URL: https://doi.org/10.18653/v1/n18-1202 (cit. a p. 55).

[40] Jay Alammar. *The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning)*. Dic. 2018. URL: https://jalammar.github.io/illustrated-bert/ (visitato il 08/11/2021) (cit. alle pp. 56, 57).

[41] Ashish Vaswani et al. «Attention is All you Need». In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. A cura di Isabelle Guyon et al. 2017, pp. 5998–6008. URL: https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html (cit. alle pp. 56, 57, 59, 60).

[42] Yi Tay et al. «Efficient Transformers: A Survey». In: *CoRR* abs/2009.06732 (2020). arXiv: 2009.06732. URL: https://arxiv.org/abs/2009.06732 (cit. a p. 57).

[43] Tianyang Lin et al. «A Survey of Transformers». In: *CoRR* abs/2106.04554 (2021). arXiv: 2106.04554. URL: https://arxiv.org/abs/2106.04554 (cit. a p. 57).

[44] Kaiming He et al. «Deep Residual Learning for Image Recognition». In: *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90. URL: https://doi.org/10.1109/CVPR.2016.90 (cit. a p. 57).

[45] Lei Jimmy Ba, Jamie Ryan Kiros e Geoffrey E. Hinton. «Layer Normalization». In: *CoRR* abs/1607.06450 (2016). arXiv: 1607.06450. URL: http://arxiv.org/abs/1607.06450 (cit. a p. 57).

[46] Nitish Srivastava et al. «Dropout: a simple way to prevent neural networks from overfitting». In: *J. Mach. Learn. Res.* 15.1 (2014), pp. 1929–1958. URL: http://dl.acm.org/citation.cfm?id=2670313 (cit. a p. 57).

[47] Jay Alammar. *The Illustrated Transformer*. Giu. 2018. URL: https://jalammar.github.io/illustrated-transformer/ (visitato il 06/11/2021) (cit. a p. 58).

[48] Alec Radford et al. «Improving Language Understanding by Generative Pre-Training». In: (2018) (cit. a p. 62).

[49] Alec Radford et al. «Language Models are Unsupervised Multitask Learners». en. In: *OpenAI blog* 1.8 (2019), p. 9 (cit. a p. 62).

[50] Tom B. Brown et al. «Language Models are Few-Shot Learners». In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. A cura di Hugo Larochelle et al. 2020. URL: https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html (cit. a p. 62).

[51] Jacob Devlin et al. «BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding». In: *CoRR* abs/1810.04805 (2018). arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805 (cit. alle pp. 65, 67).

[52] Alex Wang et al. «GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding». In: *Proceedings of the Workshop: Analyzing and Interpreting Neural Networks for NLP, BlackboxNLP@EMNLP 2018, Brussels, Belgium, November 1, 2018*. A cura di Tal Linzen, Grzegorz Chrupala e Afra Alishahi. Association for Computational Linguistics, 2018, pp. 353–355. DOI: 10.18653/v1/w18-5446. URL: https://doi.org/10.18653/v1/w18-5446 (cit. a p. 65).

[53] Pranav Rajpurkar et al. «SQuAD: 100, 000+ Questions for Machine Comprehension of Text». In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*. A cura di Jian Su, Xavier Carreras e Kevin Duh. The Association for Computational Linguistics, 2016, pp. 2383–2392. DOI: 10.18653/v1/d16-1264. URL: https://doi.org/10.18653/v1/d16-1264 (cit. a p. 65).

[54] Manzil Zaheer et al. «Big Bird: Transformers for Longer Sequences». In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. A cura di Hugo Larochelle et al. 2020. URL: https://proceedings.neurips.cc/paper/2020/hash/c8512d142a2d849725f31a9a7a361ab9-Abstract.html (cit. a p. 67).

[55] Zhenzhong Lan et al. «ALBERT: A Lite BERT for Self-supervised Learning of Language Representations». In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL: https://openreview.net/forum?id=H1eA7AEtvS (cit. alle pp. 69, 70).

[56] Guokun Lai et al. «RACE: Large-scale ReAding Comprehension Dataset From Examinations». In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*. A cura di Martha Palmer, Rebecca Hwa e Sebastian Riedel. Association for Computational Linguistics, 2017, pp. 785–794. DOI: 10.18653/v1/d17-1082. URL: https://doi.org/10.18653/v1/d17-1082 (cit. a p. 69).

[57] Pranav Rajpurkar, Robin Jia e Percy Liang. «Know What You Don't Know: Unanswerable Questions for SQuAD». In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 2: Short Papers*. A cura di Iryna Gurevych e Yusuke Miyao. Association for Computational Linguistics, 2018, pp. 784–789. DOI: 10.18653/v1/P18-2124. URL: https://aclanthology.org/P18-2124/ (cit. a p. 69).

[58] Robin Jia e Percy Liang. «Adversarial Examples for Evaluating Reading Comprehension Systems». In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*. A cura di Martha Palmer, Rebecca Hwa e Sebastian Riedel. Association for Computational Linguistics, 2017, pp. 2021–2031. DOI: 10.18653/v1/d17-1215. URL: https://doi.org/10.18653/v1/d17-1215 (cit. a p. 71).

[59] Hui Liu, Qingyu Yin e William Yang Wang. «Towards Explainable NLP: A Generative Explanation Framework for Text Classification». In: *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*. A cura di Anna Korhonen, David R. Traum e Llui's Màrquez. Association for Computational Linguistics, 2019, pp. 5570–5581. DOI: 10.18653/v1/p19-1560. URL: https://doi.org/10.18653/v1/p19-1560 (cit. a p. 71).

[60] Abubakar Abid, Maheen Farooqi e James Zou. «Persistent Anti-Muslim Bias in Large Language Models». In: *AIES '21: AAAI/ACM Conference on AI, Ethics, and Society, Virtual Event, USA, May 19-21, 2021*. A cura di Marion Fourcade et al. ACM, 2021, pp. 298–306. DOI: 10.1145/3461702.3462624. URL: https://doi.org/10.1145/3461702.3462624 (cit. a p. 72).

[61] Roy Schwartz et al. «Green AI». In: *Commun. ACM* 63.12 (2020), pp. 54–63. DOI: 10.1145/3381831. URL: https://doi.org/10.1145/3381831 (cit. a p. 72).

[62] Emma Strubell, Ananya Ganesh e Andrew McCallum. «Energy and Policy Considerations for Deep Learning in NLP». In: *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*. A cura di Anna Korhonen, David R. Traum e Llui's Márquez. Association for Computational Linguistics, 2019, pp. 3645–3650. DOI: 10.18653/v1/p19-1355. URL: https://doi.org/10.18653/v1/p19-1355 (cit. a p. 72).

[63] Giacomo Frisoni, Gianluca Moro e Antonella Carbonaro. «Learning Interpretable and Statistically Significant Knowledge from Unlabeled Corpora of Social Text Messages: A Novel Methodology of Descriptive Text Mining». In: *Proceedings of the 9th International Conference on Data Science, Technology and Applications, DATA 2020, Lieusaint, Paris, France, July 7-9, 2020*. A cura di Slimane Hammoudi, Christoph Quix e Jorge Bernardino. SciTePress, 2020, pp. 121–132. DOI: 10.5220/0009892001210132. URL: https://doi.org/10.5220/0009892001210132 (cit. a p. 73).

[64] Laurens Van der Maaten e Geoffrey Hinton. «Visualizing data using t-SNE». In: *Journal of Machine Learning Research* 9 (nov. 2008), pp. 2579–2605. URL: http://www.jmlr.org/papers/v9/vandermaaten08a.html (cit. a p. 75).

[65] Thomas Kluyver et al. «Jupyter Notebooks - a publishing format for reproducible computational workflows». In: *Positioning and Power in Academic Publishing: Players, Agents and Agendas*. A cura di Fernando Loizides e Birgit Scmidt. Netherlands: IOS Press, 2016, pp. 87–90. URL: https://eprints.soton.ac.uk/403913/ (cit. a p. 81).

[66] Fabian Pedregosa et al. «Scikit-learn: Machine Learning in Python». In: *J. Mach. Learn. Res.* 12 (2011), pp. 2825–2830. URL: http://dl.acm.org/citation.cfm?id=2078195 (cit. a p. 81).

[67] The pandas development team. *pandas-dev/pandas: Pandas*. Ver. 1.3.4. Feb. 2020. DOI: 10.5281/zenodo.3509134. URL: https://doi.org/10.5281/zenodo.3509134 (cit. a p. 81).

[68] Charles R. Harris et al. «Array programming with NumPy». In: *Nature* 585.7825 (set. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2 (cit. a p. 81).

[69] Pauli Virtanen et al. «SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python». In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2 (cit. a p. 81).

[70] J. D. Hunter. «Matplotlib: A 2D graphics environment». In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55 (cit. a p. 81).

[71] John D. Garrett. «garrettj403/SciencePlots». Ver. 1.0.9. In: (set. 2021). DOI: 10.5281/zenodo.4106649. URL: http://doi.org/10.5281/zenodo.4106649 (cit. a p. 81).

[72] Plotly Technologies Inc. *Collaborative data science*. 2015. URL: https://plot.ly (cit. a p. 81).

[73] Matthew Honnibal et al. «spaCy: Industrial-strength Natural Language Processing in Python». In: (2020). DOI: 10.5281/zenodo.1212303 (cit. a p. 81).

[74] Mark Neumann et al. «ScispaCy: Fast and Robust Models for Biomedical Natural Language Processing». In: *Proceedings of the 18th BioNLP Workshop and Shared Task*. Florence, Italy: Association for Computational Linguistics, ago. 2019, pp. 319–327. DOI: 10.18653/v1/W19-5034. eprint: arXiv:1902.07669. URL: https://www.aclweb.org/anthology/W19-5034 (cit. a p. 81).