ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Scienze
Corso di Laurea in Ingegneria e Scienze Informatiche

IDENTIFICAZIONE DI FARMACI E DISPOSITIVI MEDICI EQUIVALENTI CON TECNICHE DI NATURAL LANGUAGE PROCESSING E DEEP LEARNING

Elaborato in
Programmazione Di Applicazioni Data Intensive

Relatore
Prof. Gianluca Moro

Presentata da
Giacomo Montalti

Seconda Sessione di Laurea
Anno Accademico 2017 – 2018

PAROLE CHIAVE

Natural Language Processing

Machine Learning

Deep Neural Networks

Semantic Similarity Search

Python

A chiunque mi sia stato vicino,
e mi abbia aiutato a raggiungere questo traguardo.

Introduzione

L'intelligenza artificiale (AI) è recentemente diventata uno dei principali trend tecnologici strategici grazie ad avanzamenti importanti nella ricerca scientifica del settore e nell'enorme sviluppo delle capacità di calcolo dei sistemi hardware. E' una disciplina vastissima che risale all'epoca del Prof. Alan Turing, il quale ideò nel 1950 il famoso test, che prese poi il suo nome, per stabilire se una macchina è intelligente.

Negli ultimi dieci anni la ricerca scientifica in ambito AI ha fatto progressi che gli esperti ritengono superiori alle conoscenze accumulate in precedenza. Questi avanzamenti si sono tradotti in tecnologie software di pubblico dominio che hanno portato allo sviluppo di applicazioni eclatanti in quasi ogni ambito delle discipline scientifiche.

Una delle differenze più importanti tra i metodi di AI classici e quelli recenti è che questi ultimi sono fortemente basati sull'estrazione di conoscenza da masse di dati e maggiore è la disponibilità di dati, più la conoscenza estratta è accurata. Perciò la svolta è avvenuta grazie anche alla disponibilità di masse di dati, la cui generazione continua con ritmi sempre maggiori e impensabili solo fino a qualche anno fa. Infatti, assistiamo all'introduzione in sempre più settori e ambiti, di dispositivi elettronici e sensori di ogni genere. Ciò ha reso necessaria la ricerca di nuovi metodi per la gestione e l'elaborazione dei dati, le cui dimensioni hanno largamente superato le capacità umane di trattarli e analizzarli.

In particolare nell'ultimo periodo, ha iniziato a guadagnare molta popolarità una macro area di tecniche nota con il nome di **machine learning**.

I recenti successi del machine learning sono trasversali a numerosi settori tra cui *speech recognition* e *computer vision*, tanto per citarne due tra i più famosi.

L'utilizzo del machine learning può però portare a risultati non soddisfacenti quando si ha la necessità di gestire immagini o testi in forma naturale poiché l'attività di preparazione dei dati e di selezione delle variabili decisive per l'obiettivo finale richiede l'intervento di esperti ed è molto dispendiosa. In questi casi, infatti, si ottengono risultati generalmente più accurati e precisi attraverso tecniche diverse, appartenenti alla macro area di ricerca del **deep learning**

che ha tra le sue prerogative quella di riuscire ad estrarre autonomamente le variabili migliori direattamente dai dati grezzi.

Il deep learning è un campo relativamente giovane le cui potenzialità sono ancora tutte da esplorare, in grado di elaborare in maniera ancora più *approfondita* i dati, e sarà affrontato nel dettaglio all'interno di questo lavoro di tesi.

Questa tecnologia ha permesso di **migliorare drasticamente** i risultati raggiunti in passato in tantissimi settori, consentendo ad esempio lo sviluppo di auto a guida autonoma, assistenti virtuali in grado di comprendere una conversazione e di fornire risposte alle nostre domande o macchinari medicali capaci di identificare masse tumorali con una precisione maggiore rispetto a quella umana.

All'interno di questo elaborato verranno analizzati e sperimentati diversi approcci recenti in ambito *natural language processing* (NLP) e deep learning (DL), allo scopo di identificare prodotti medicali equivalenti dalla loro breve descrizione testuale destrutturata.

Il natural language processing è uno dei più complessi da trattare e si occupa di affrontare l'analisi lessicale e semantica del testo scritto. Essendo il testo destrutturato il tipo di dato maggiormente diffuso, negli ultimi anni si sono moltiplicate le ricerche e gli studi a riguardo. Sfortunatamente, i dati disponibili in questa forma presentano ambiguità e problematiche, che verranno discusse.

La tesi valuta e confronta diversi metodi NLP sopra menzionati, marcandone gli aspetti positivi e negativi, sulla base del problema sopra menzionato.

Il lavoro di tesi è stato suddiviso nei seguenti capitoli:

- **Capitolo 1** - Analisi preliminare sulla struttura dei dati all'interno dei dataset contenenti;
- **Capitolo 2** - Panoramica sulle varie tecnologie esistenti in letteratura utilizzabili per il problema posto;
- **Capitolo 3** - Introduzione alla modellazione del problema con i primi approcci alle difficoltà riscontrate e alle relative soluzioni;
- **Capitolo 4** - Metodi di risoluzione dei problemi riscontrati e spiegazione dei diversi approcci utilizzati;
- **Capitolo 5** - Il lavoro realizzato e il codice impiegato.

Indice

Elenco delle figure

Capitolo 1

Analisi dei dati forniti

In questa parte vengono analizzati i dataset forniti e il modo in cui sono state strutturate le informazioni all'interno dei diversi file che le contengono. In questo modo viene fornita una visione più specifica del lavoro che verrà svolto e verranno definiti gli obiettivi del progetto.

1.1 Contesto applicativo

Il Servizio Sanitario Nazionale dispensa farmaci e articoli medici a tutti i principali centri ospedalieri della regione Emilia-Romagna. Questi vengono memorizzati e salvati tramite una stringa testuale che contiene, in una quantità ristretta di caratteri (al massimo 200), le principali parole chiave utilizzate per identificare e descrivere il prodotto in considerazione.

Dalla Regione Emilia Romagna sono stati forniti 2 documenti contenenti un insieme consistente di prodotti farmaceutici e di tipo medicale. Nel primo file sono presenti i dati descrittivi e identificativi, mentre nel secondo sono stati inseriti alcuni codici utilizzabili per identificare i prodotti equivalenti.

Non esistono campi secondari: tutte le parole chiave che possono servire all'identificazione del prodotto sono inserite in un unico campo testuale. Questo di per sé è già un problema, perché non è possibile decidere arbitrariamente quali analizzare o ignorare.

Non è ad esempio possibile decidere di isolare la posologia, il tipo di prodotto (compresse, capsule, effervescenti) o la quantità di principio attivo presente, poiché tutto contenuto in maniera indivisibile all'interno del campo `DESCRIZIONE`.

Oltre ad essere molto inefficiente sotto il profilo della struttura, questo metodo accentua le probabilità di commettere errori (sia di battitura che di distrazione) e di registrare righe duplicate. Per fare un esempio pratico:

`ASPIRINA RAPIDA 10CPRMAST500MG`
e
`ASPIRINA RAP 10 CPR MAST 500 MG`

Vengono riconosciute come due descrizioni differenti, in quanto l'unico modo che un computer ha a disposizione per constatare l'effettiva uguaglianza, è controllare carattere per carattere se i testi coincidono.

Al contrario un essere umano associa automaticamente *RAPIDA* e *RAP*, e le rimanenti sigle vengono identificate come equivalenti, ma suddivise da spazi.

I dati forniti contengono un **insieme molto consistente** di descrizioni di prodotti ospedalieri e sanitari, differenti uno dall'altro, che necessitano di essere accoppiati in base alle equivalenze reali esistenti tra loro.

L'obiettivo principale di questa tesi sarà proprio quello di trovare un metodo per far riconoscere le descrizioni equivalenti, nello stesso modo in cui è in grado di farlo una persona, in modo da poter affermare quali farmaci o articoli siano *equivalenti*, anche se associati a stringhe differenti.

Nell'esempio precedente quindi, un sistema abbastanza preciso dovrebbe essere abbastanza intelligente da etichettare entrambi i prodotti come **identici**, anche se il confronto per caratteri indichi l'opposto.

Vengono forniti 2 diversi documenti in formato `csv`, uno standard molto usato per la risoluzione di questo tipo di problemi, che permette di rappresentare i dati in formato tabellare con estrema facilità. La prima riga viene generalmente utilizzata per indicare le intestazioni delle colonne, dalla riga successiva vengono inseriti gli elementi, dividendo i campi che la compongono tramite un separatore (nel caso in questione il carattere `;`).

Nel primo file si trovano tutti i farmaci e gli articoli sanitari trattati dalla Regione Emilia Romagna, in aggiunta a diverse informazioni di tipo amministrativo (relative per lo più ai fornitori).

Nel secondo documento sono presenti un numero di righe (cioè elementi) molto più limitato, circa il 9% del primo, anche qui come nel precedente non sono specificate informazioni aggiuntive su farmaci e articoli, ma solo la descrizione e un **codice regionale**.

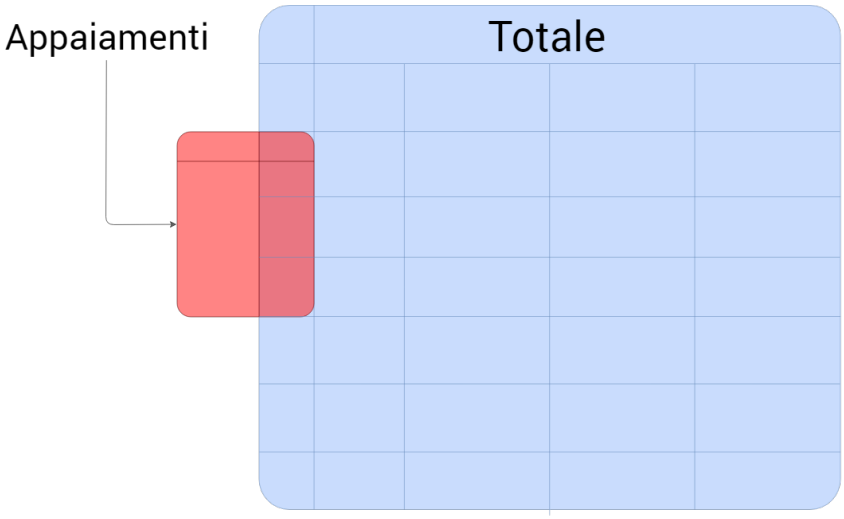


Figura 1.1: Rappresentazione grafica dei file di partenza.

Le descrizioni in questione sono le stesse che compaiono all'interno del file precedente, mentre il codice **permette di accoppiare gli elementi**: se due righe hanno lo stesso codice, le descrizioni associate (che saranno ovviamente differenti) devono essere considerate equivalenti.

Saranno discussi nel dettaglio delle prossime sezioni i contenuti di entrambi i file, così da poter comprendere completamente entrambi.

1.2 Documento completo

Il file `Totale.csv` comprende l'insieme di **tutti** gli elementi che dovranno essere accoppiati. Per ognuno dei 322.977 elementi che contiene, oltre alla stringa descrittiva specificata in precedenza, sono presenti informazioni che non verranno utilizzate direttamente nel progetto trattato, come ad esempio:

- Dati per l'identificazione interna
- Codici relativi a fornitore e produttore
- Identificatori per la ricerca in diverse banche dati
- Categorie inventariali, classificazioni e tipologia

Tutti questi campi possono essere (**e in molti casi lo sono**) non compilati. Ciò comporta anche una difficoltà aggiuntiva nel caso in cui questi dati volessero essere usati per qualunque tipo di operazione.

1.3 Documento per appaiamenti

Il file precedente però non ci da alcuna informazione relativa a tutti gli eventuali accoppiamenti. Questa informazione è invece compresa all'interno del secondo file, molto meno corposo del primo (poichè comprende 27.739 righe) e scarno anche sotto il profilo di attributi per ogni elemento. Infatti, ogni riga sarà composta solo quattro campi, di cui i primi tre copiati direttamente dalle informazioni presenti nel file principale.

Nello specifico abbiamo:

- **Codice regionale azienda** - identificatore dell'azienda sanitaria;
- **Codice** - identifica univocamente un articolo all'interno dell'anagrafica aziendale;
- **Descrizione** - la descrizione testuale del farmaco principale;
- **Codice Regionale** - un campo che indica univocamente quali siano gli elementi equivalenti.

Quest'ultimo campo è stato aggiunto a mano da un esperto del settore, che ha dichiarato equivalenti due o più elementi direttamente provenienti dal dataset originario, rimuovendo da questo tutti gli attributi presenti *in eccesso*.

1.4 Obiettivi

L'obiettivo principale di questa tesi, come già sottolineato, sarà sviluppare un sistema capace di etichettare due prodotti come equivalenti o meno, utilizzando i due dataset a disposizione nei modi più produttivi.

DESCRIZIONE
ASPIRINA CP.MG 500 OS aic004763037
ASPIRINA DOLORE E INFIAMM. MG.500 CPR
ASPIRINA RAP 10 CPR MAST 500 MG
ASPIRINA RAPIDA 10CPRMAST500MG
ASPIRINA RAPIDA 10CPRMAST500MG
ASPIRINA*AD 20CPR 0,5G(AFM)
ASPIRINA-03 10CPR 325MG
ASPIRINETTA 100MG 30CPR
ASPIRINETTA 100MG 30CPR -
ASPIRINETTA MG 100 CP MASTICABILI
ASPIRINETTA MG.100 CPR

Figura 1.2: Esempio di descrizioni contenute nel dataset.

Verranno considerati e presi in considerazione diversi approcci esistenti al problema, usando metodi di **machine learnig** totalmente differenti, per testare quale dei due si adatti meglio al problema in causa.

Capitolo 2

Le tecnologie disponibili

In questa prima parte verranno descritte più nello specifico tutte le tecnologie che sono state solo accennate nell'introduzione, per rendere più chiari gli obiettivi e le metodologie di lavoro impiegate nei capitoli successivi.

2.1 Machine Learning

Il machine learning è una tecnica nata recentemente (negli ultimi decenni del 1900 circa) che fornisce un nuovo approccio ai problemi in cui l'obiettivo è stimare, predire o ipotizzare *"qualcosa"*.

Una delle definizioni più citate quando si parla di questa tecnologia, è quella proposta dal professor Tom Mitchell [1]:

> *Si dice che un programma apprende dall'esperienza E con riferimento ad alcune classi di compiti T e con misurazione della performance P, se le sue performance nel compito T, come misurato da P, migliorano con l'esperienza E.*

Cioè, riuscire a far eseguire un'attività a una macchina, che riesce in maniera autonoma a migliorare il modo in cui la esegue, usando l'esperienza che acquisisce nel farlo. La macchina quindi *impara ad eseguire un compito* in maniera sempre più accurata e precisa, senza essere stata programmata in maniera specifica a farlo.

Questo approccio rappresenta una completa innovazione nel modo in cui gli algoritmi vengono utilizzati per risolvere problemi, in quanto generalmente l'approccio classico ne prevedeva lo sviluppo sfruttando la conoscenza pregressa dell'autore o autori, mentre al contrario in questo modo la macchina impara da esempi e dati.

È inoltre molto importante il fatto che non esista un unico campo dove poterne sfruttare le potenzialità, poiché permette di adattarsi benissimo a un insieme molto vasto di problematiche. Per citarne qualcuna:

- Predizione di consumi;
- Categorizzazione di elementi;
- Previsione andamento meteorologico;
- Stima di costi.

E non solo. Al giorno d'oggi, infatti, questa tecnologia è impiegata negli ambiti e nei settori più disparati: sanità, sicurezza, management aziendale e previsione di fenomeni o eventi di carattere generale. Anche gli smartphone si affidano all'intelligenza artificiale per portare a termine i compiti più disparati (la capacità di predizione del testo durante la digitazione o le fotocamere in grado di applicare filtri personalizzati a ogni immagine).

L'avvento di questi metodi è stato sostenuto anche dalla crescente disponibilità di dati da poter analizzare, che come già accennato in precedenza negli anni si sono letteralmente moltiplicati, e complice l'aumento della disponibilità di risorse computazionali si sono create le condizioni ideali per lo sviluppo di nuovi studi e ricerche.

Per fare un ulteriore esempio di quanto questa tecnologia abbia avuto impieghi in casi del tutto reali e concreti, si prendano i **suggerimenti di prodotti simili** a quelli che si stanno visualizzando (ad esempio durante una sessione di shopping online). Non sono altro che dei sistemi che utilizzano il machine learning per trovare legami tra prodotti in maniera automatica, per poi successivamente consigliarne l'acquisto ai visitatori.

Figura 2.1: Esempio di un sistema di suggerimenti guidato dal machine learning.

Sistemi che ormai sono entrati nella quotidianità di tutti, e che silenziosamente permettono di fruire di un'esperienza d'uso maggiormente gradevole in grado di invogliare a passare più tempo sul sito o l'applicazione.

E per farsi un'idea di quanto siano importanti per le aziende questi meccanismi, basti pensare che nel caso di Amazon il 35% delle vendite viene generato proprio dai suggerimenti visualizzati all'interno delle pagine web.

2.1.1 Sviluppo di un modello

All'atto pratico, alla base del machine learning c'è un **modello**, il vero e proprio nucleo che compone il sistema che si vuole costruire.

Il modello deve riuscire ad approssimare nel modo migliore possibile la realtà descritta dai dati, trovando correlazioni non banali tra loro.

In base alla tipologia e all'organizzazione dei dati a disposizione, i problemi che si possono riscontrare durante lo sviluppo di un sistema di machine learning possono rientrare in una delle seguenti tipologie:

1. Apprendimento supervisionato;
2. Apprendimento non supervisionato;
3. Apprendimento per rinforzo.

2.1.2 Apprendimento supervisionato

Nell'**apprendimento supervisionato** i dati disponibili sono composti da due parti principali: una parte di input da affidare al modello che contiene uno o più campi descrittivi, che definiscono le caratteristiche di ogni elemento, e una parte in cui è specificato il valore in output reale.

Si prenda come esempio seguente dataset (di tipo supervisionato), utilizzato per identificare la specie di iris partendo dai dati generici del fiore:

Lung. sepalo	**Larg. sepalo**	**Lung. petalo**	**Larg. petalo**	**Specie**
5.7	4.4	1.5	0.4	*Setosa*
7.7	3.8	6.7	2.2	*Virginica*
5.4	3.7	1.5	0.2	*Setosa*
7.2	3.6	6.1	2.5	*Virginica*
6.0	3.4	4.5	1.6	*Versicolor*
6.2	3.4	5.4	2.3	*Virginica*
5.0	3.3	1.4	0.2	*Setosa*
6.3	3.3	4.7	1.6	*Versicolor*
6.7	3.3	5.7	2.1	*Virginica*

Tabella 2.1: Esempio di dati in ingresso con le caratteristiche di diversi fiori.

Le prime 4 colonne compongono l'insieme di dati da passare al modello per far si che riesca ad elaborare tutte le correlazioni esistenti, mentre l'ultima colonna contiene il valore che ci si aspetta in output dopo l'elaborazione.

Il modello, avendo disponibile il risultato corretto da predire in uscita, può effettuare una predizione, e comportarsi di conseguenza in base alla differenza tra risultato atteso e risultato predetto.

Quindi ciclicamente, un sistema di machine learning supervisionato segue questo schema:

- Predizione in base ai dati di input;
- Controllo della differenza tra risultati attesi e predetti (**errore del modello**);
- Miglioramento del modello in base agli errori commessi.

Questo processo è definito **addestramento** del modello, e viene eseguito finché l'errore ottenuto non raggiunge un livello abbastanza basso da poter considerare il modello valido. Verrà affrontato nella sezione 2.1.5 il modo in cui il modello riesce a migliorare le proprie predizioni.

I modelli supervisionati sono generalmente usati per 2 diversi tipi di problemi:

- **Classificazione** - Come nell'esempio della tabella 2.1 relativa alla specie corretta dell'iris, abbinare l'elemento alla categoria corretta;
- **Regressione** - Predire un valore numerico, al posto di una categoria (come ad esempio il prezzo di un immobile), partendo dalle caratteristiche dell'elemento.

2.1.3 Apprendimento non supervisionato

Nell'**apprendimento non supervisionato** i dati disponibili **non** contengono alcun tipo di indicazione sulla bontà della previsione. Questo tipo di informazione è detta anche *non etichettata*,

Ovviamente i dati con il valore richiesto in output sono molto più facili da reperire, poiché non è necessaria l'analisi di una persona umana (il dataset d'esempio precedente ha richiesto l'identificazione della specie corretta per ogni fiore presente).

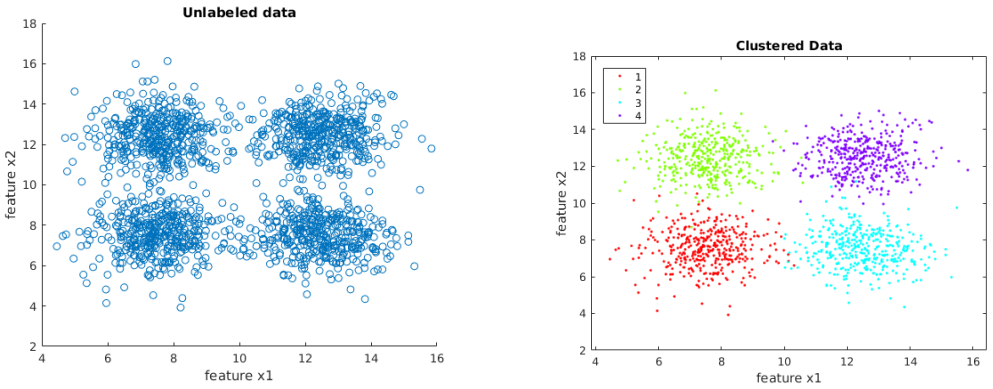


Figura 2.2: Rappresentazione grafica di un modello non supervisionato.
Fonte: https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/unsupervised_learning.html

Non avendo alcun tipo di *indicazione* sul valore che si aspetta in uscita, **non esiste un metodo generico per controllare l'errore** sviluppato dal modello.

Generalmente questo approccio viene utilizzato nei problemi di **clustering**, quando cioè è necessario classificare un insieme di elementi senza però a priori avere le possibili categorie.

Il fatto di non sapere in partenza le diverse categorie, risulta però essere in molti casi vantaggioso, poiché il modello riesce ad analizzare e cogliere legami spesso *invisibili* alla mente umana.

2.1.4 Apprendimento per rinforzo

L'**l'apprendimento per rinforzo** è una tipologia utilizzata quando è necessario che il nostro modello impari relazionandosi con l'ambiente in cui si trova: non sfrutta dataset pregressi, ma le mutazioni che riesce a captare.

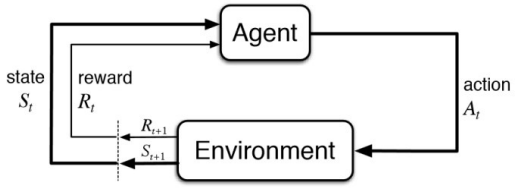


Figura 2.3: Modello di base nell'apprendimento di rinforzo.
Fonte: https://www.kdnuggets.com/images/reinforcement-learning-fig1-700.jpg

Viene utilizzato in casi molto particolari, generalmente facendo eseguire un compito a un modello in maniera ciclica. Dopo tantissimi tentativi (anche

svariati milioni) il sistema risulta esser in grado di svolgere operazioni complesse con un'accuratezza maggiore di quella umana.

Ad esempio, l'apprendimento per rinforzo è stato utilizzato in maniera soddisfacente per insegnare a un modello senza alcuna conoscenza di base a giocare a scacchi, affinando la propria tecnica partita dopo partita, arrivando a competere (e vincere) anche contro campioni mondiali.

2.1.5 Discesa del gradiente

Il metodo della discesa del gradiente è la base del machine learning, ciò che permette al modello di migliorarsi iterazione dopo iterazione.

In matematica viene utilizzato per trovare i punti di minimo e di massimo all'interno di **una funzione**, procedendo a step graduali.

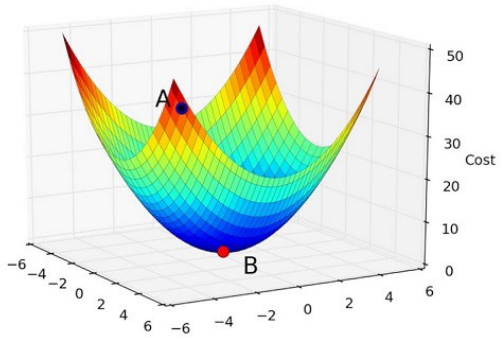


Figura 2.4: Discesa del gradiente dal punto A al punto B visualizzabile graficamente.
Fonte: https://medium.com/abdullah-al-imran/intuition-of-gradient-descent-for-machine-learning-49e1b6b89c8b

Nello stesso modo, quando nel processo di addestramento del modello vengono generate la predizioni, è possibile sviluppare una funzione che associa ad ognuna l'errore generato.

Si aggiunge quindi *una dimensione* alla funzione del modello, così da poter applicare la discesa del gradiente e minimizzare l'errore generato dalle predizioni future.

Nel grafico in figura 2.4 ad esempio, le dimensioni x e y che compongono il piano sono quelle relative ai dati di input, mentre la terza dimensione, la z, viene aggiunta per associare l'errore prodotto dal modello.

Per ridurre l'errore quindi, è necessario:

- Calcolare la funzione d'errore e il suo valore attuale ;
- Calcolare il gradiente della funzione nel punto, quindi *la direzione* da seguire per raggiungere il punto di minimo;

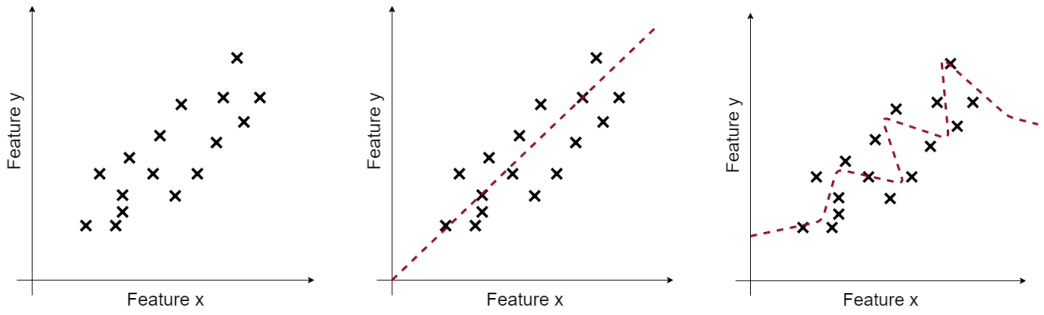
- Sottrarre al punto iniziale un vettore proporzionale al gradiente;
- Ricominciare da capo finché non viene raggiunto il punto obiettivo.

Questo approccio affonda le proprie radici nel campo della statistica, dove viene chiamato **regressione**. Si occupa di analizzare un insieme di dati che possono essere rappresentati da una funzione (lineare, polinomiale, o di altri gradi) e cercare di approssimare al meglio le variabili dipendenti e indipendenti.

2.1.6 Validazione del modello

Avendo un insieme di dati su cui addestrare il modello, non vengono utilizzati tutti per la fase di training, ma al contrario si suddivide in due (o tre) sottogruppi differenti e non sovrapposti.

Questo modo di approcciare i problemi è detto **hold-out**, e prevede la divisione dei dati in due insiemi secondo una proporzione nota (generalmente 30%-70%). Sull'insieme più grande verrà addestrato il modello, mentre il restante più piccolo verrà utilizzato per controllare la validità dell'addestramento.



(a) Dati grezzi rappresentati sul piano cartesiano.

(b) Regressione lineare eseguita correttamente.

(c) Modello con overfitting.

Figura 2.5

Il rischio, infatti, è quello di addestrare **troppo** un modello, in grado di produrre risultati accurati al 100% solo sui dati su cui viene sviluppato.

Al contrario, un buon sistema di machine learning dovrebbe essere il più generale possibile, ed è per questo che questa metodologia utilizza il gruppo di dati più piccolo, detto **validation set**, per controllare il comportamento del modello dopo l'addestramento:

- Se i risultati ottenuti nelle predizioni sono molto buoni solo sul training set ma non sul validation, allora c'è overfitting;
- Se su entrambi gli insiemi il modello si comporta *circa* nello stesso modo, allora l'addestramento risulta valido.

2.2 Deep Learning

Il machine learnig, però, mostra alcune limitazioni nella precisione delle previsioni sviluppate, in particolar modo se:

- Il tipo di dato che deve gestire è di tipo immagine o testuale (in particolar modo con testi naturali);
- La quantità di dati disponibili per il training non è così abbondante.

Inoltre, per poter utilizzare il machine learning è necessario **selezionare le feature manualmente**, e ciò richiede delle competenze non banali.

Al fine di affrontare queste problematiche, negli ultimi anni una branca del machine learning ha guadagnato sempre più popolarità nel campo dell'intelligenza artificiale: il **deep learning**.

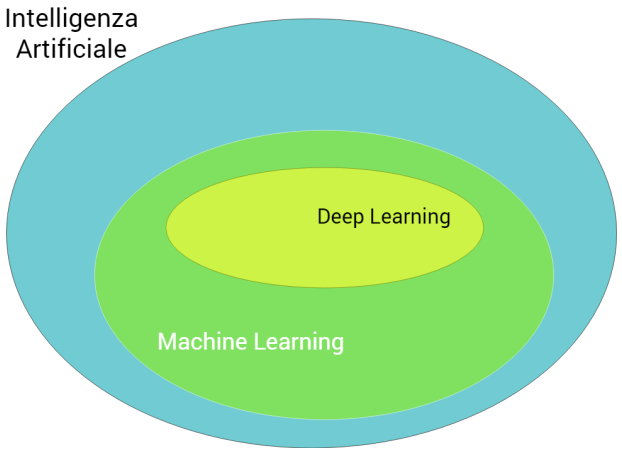


Figura 2.6: Gerarchia delle tecnologie di intelligenza artificiale.

Questa disciplina trova fondamento e ispirazione nella struttura dei neuroni all'interno del cervello umano, dove non esiste un unico punto di ingresso nel quale elaborare i dati in entrata, ma al contrario esistono molti più nodi, suddivisi in diversi strati (da qui il motivo della parola *deep*) che collaborano tra loro per raggiungere risultati estremamente accurati.

Nello stesso modo, all'interno delle **reti neurali** sviluppate tramite deep learning non si ritrova *un solo modello* da addestrare e da gestire, ma una fitta rete di *neuroni* raggruppati all'interno di strati.

Dopo aver eseguito l'addestramento sui dati di training, queste riescono ad eseguire i compiti più disparati, con una **precisione maggiore** di quella raggiungibile tramite machine learning: categorizzare immagini, riconoscere

lettere e numeri scritti a mano, e nei casi più avanzati anche a generare immagini realistiche o testi di senso compiuto.

La divisione in strati dei neuroni consente di gestire in maniera progressiva l'astrazione dei problemi da affrontare: nei livelli più bassi si riescono a riconoscere aspetti e pattern semplici, mentre negli strati più avanzati si gestiscono gli elementi di più alto livello, così da comprendere e individuare caratteristiche sempre più complesse.

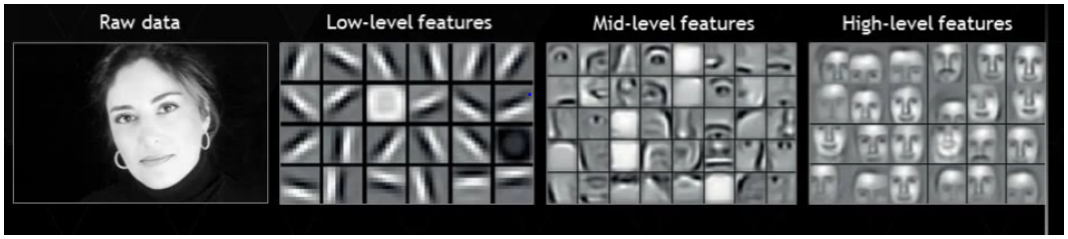


Figura 2.7: Diverse astrazioni trattate nelle reti neurali.
Fonte: https://www.analyticsvidhya.com/blog/2017/04/comparison-between-deep-learning-machine-learning/

Questo nuovo approccio ha consentito nel tempo di conseguire risultati impressionanti per precisione e tempi richiesti per il calcolo.

L'esempio più cristallino di quanto una rete neurale sia molto più accurata rispetto a metodi tradizionali, è l'applicazione a problemi riguardanti le immagini (uno su tutti riconoscimento e la categorizzazione dei soggetti) o il riconoscimento della grafia.

Prendendo come esempio il popolare dataset MNIST, contenente svariate migliaia di immagini raffiguranti cifre, utilizzato (generalmente come punto di riferimento in questo ambito), il tasso di errore migliore mai fatto registrare è attualmente dello **0.23%** [8], ed è stato raggiunto proprio da una rete neurale.

2.2.1 Struttura

Come detto, un sistema di deep learning si sviluppa in una struttura molto più complessa e articolata rispetto a un modello di machine learning.

Si creano degli strati di neuroni, elementi fondamentali che altro non sono che dei semplici algoritmi ai quali, dato in input un vettore di n elementi, generano un risultato in output generalmente compreso tra 0 e 1.

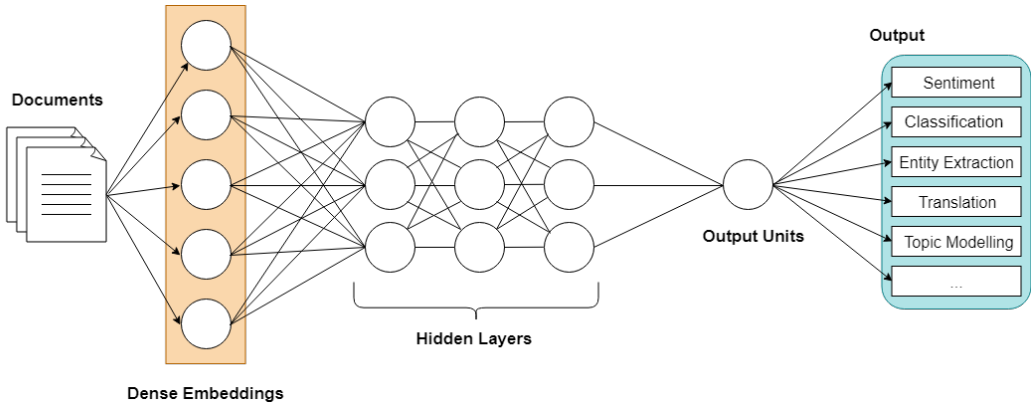


Figura 2.8: Struttura tipica di una rete neurale.

Gli strati sono tra loro collegati, e le informazioni in uscita di uno sono quelle in entrata del successivo. Ogni livello rappresenta un grado di astrazione differente, nel quale vengono gestiti aspetti differenti del dato in input, come in figura 2.7. Nel livello di output viene invece elaborata l'informazione di alto livello richiesta.

Ogni singolo neurone, riceve in input un insieme di valori all'interno di un vettore (detto in certi casi *tensore*) ed elabora internamente il valore y da ritornare in output.

Il neurone non fa altro che eseguire la regressione per minimizzare l'errore, e raggiungere risultati sempre più precisi.

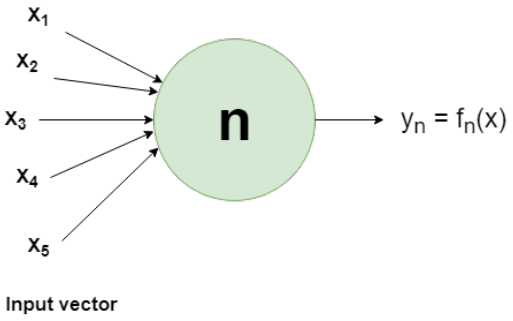


Figura 2.9: Struttura di un neurone.

2.3 Natural Language Processing

La disciplina che si occupa dei problemi riguardanti dati di tipo testuale, espressi in forma naturale, è detta **natural language processing**.

Le criticità da gestire in questi casi sono molte, derivate dalla complessità dei linguaggi:

- Ogni lingua ha regole strutturali differenti, che devono essere gestite e rispettate;
- Esistono innumerevoli ambiguità, ad esempio una singola parola in base al contesto:
  - Può avere funzioni grammaticali e sintattiche;
  - Può avere più significati differenti.
- Presenza di errori ortografici, di punteggiatura o semantici;
- Utilizzo di abbreviazioni;
- Difficoltà a gestire elementi complessi come sarcasmo, ironia e metafore.

Per questi motivi, il testo viene generalmente **pre-processato** prima di essere utilizzato per l'addestramento del modello.

Esistono svariate tecniche volte a sfruttare i dati testuali al massimo del loro potenziale, tra le tante, quelle più utilizzate sono:

Segmentazione Il testo viene scomposto in elementi più piccoli (frasi o parole), per poterlo elaborare in modo più dettagliato;

Part of speech Viene abbinata a ogni parola la propria classe grammaticale, così da aiutare il modello a estrarne il significato all'interno del contesto;

Case folding Convertire tutto in maiuscolo o minuscolo, poiché anche solo una lettera maiuscola differente può rendere diverse agli occhi del modello due parole identiche;

Stopword Rimozione di congiunzioni e articoli che non influiscono sull'argomento o sul contesto in cui si trovano;

Lemmatizzazione Processo utile a normalizzare tutte le forme di verbi e nomi, portandole alla *forma base* (ad esempio convertire tutte le forme verbali all'infinito per unire termini simili o con lo stesso significato);

Stemming Processo parallelo alla lemmatizzazione, che però ricava la *radice morfologica* al fine di unificare significati simili. Funziona in maniera più approssimativa e veloce della lemmatizzazione.

Uno degli ostacoli maggiori riscontrabili durante la risoluzione dei problemi di *natural language processing* rimane però la **codifica delle parole**. Generalmente i modelli che vengono utilizzati nel machine learning non gestiscono direttamente parole e frasi, ma vengono di solito trasformate in un qualche identificativo numerico, ad esempio:

Automobile $\longrightarrow$ *ID**456***
Autobus $\longrightarrow$ *ID**164***

Questa pratica però porta con se svariate problematiche:

1. La codifica è totalmente arbitraria e non fornisce in alcun modo nessun tipo di legame tra gli identificatori utilizzati.

2. Vengono rimosse tutte le relazioni che potrebbero esistere tra i singoli elementi.

Tutto ciò comporta che le correlazioni esistenti tra più elementi vengano ignorate dal modello, impossibilitando l'utilizzo delle informazioni condivise (nel nostro esempio, *che entrambi siano mezzi di trasporto o che siano dotati di ruote e motore*).

Questo implica che per ottenere risultati abbastanza soddisfacenti debbano essere presenti grandi quantità di dati, così da riuscire ad addestrare in maniera adeguata un modello, anche se questo comporta un aumento di difficoltà nel gestire i dati (e conseguentemente anche tempi più lunghi per il training) anche a causa di una maggiore sparsità.

Per poter affrontare questo tipo di problemi sono state recentemente sviluppate diverse tecniche, raggruppate sotto il nome di **word embedding**, che si pongono l'obiettivo di mantenere le informazioni semantiche e sintattiche delle parole.

2.3.1 Word2Vec

Uno dei più famosi algoritmi che permette di preservare informazioni riguardanti la semantica e la sintassi, è **word2vec** [2], uno strumento molto potente sviluppato per mappare *parole* o *frasi* in vettori, composti da numeri reali.

Tramite l'insieme dei vettori che rappresentano gli elementi di partenza, è possibile sfruttare lo spazio vettoriale costruito su quest'ultimi per svariate operazioni:

- Trovare affinità tra i singoli elementi: più i vettori sono vicini e più potranno essere considerati simili;

- Trovare relazioni semantiche tra coppie di parole in maniera geometrica (come in figura 2.10, dove la relazione tra *Spain* e *Madrid* è la stessa che intercorre tra *Italy* e *Rome* perché il vettore differenza è lo stesso).

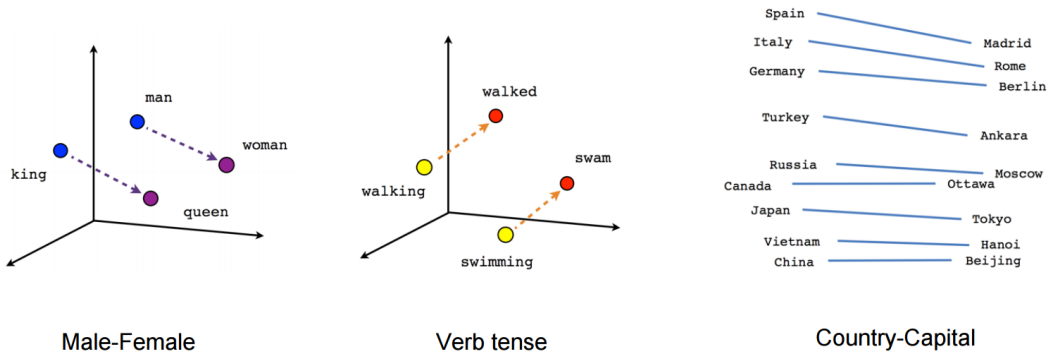


Figura 2.10: Esempi di similarità estraibili con word2vec.
Da https://www.tensorflow.org/tutorials/representation/word2vec

Queste relazioni comportano la possibilità di utilizzare i vettori anche per fini semantici sorprendentemente utili. Come è già stato detto, ogni parola non è altro che un vettore di numeri reali, sul quale è possibile eseguire tutti i tipi di operazioni conosciute.

Ad esempio è possibile **usare gli operatori matematici** come se si stessero trattando numeri reali (anche perché *lo sono* in fin dei conti) e sfruttarne le proprietà. In figura 2.11 ad esempio, è possibile vedere come viene calcolata la parola corrispettiva di *king*, se si prende come metro di misura la correlazione tra *man* e *woman*:

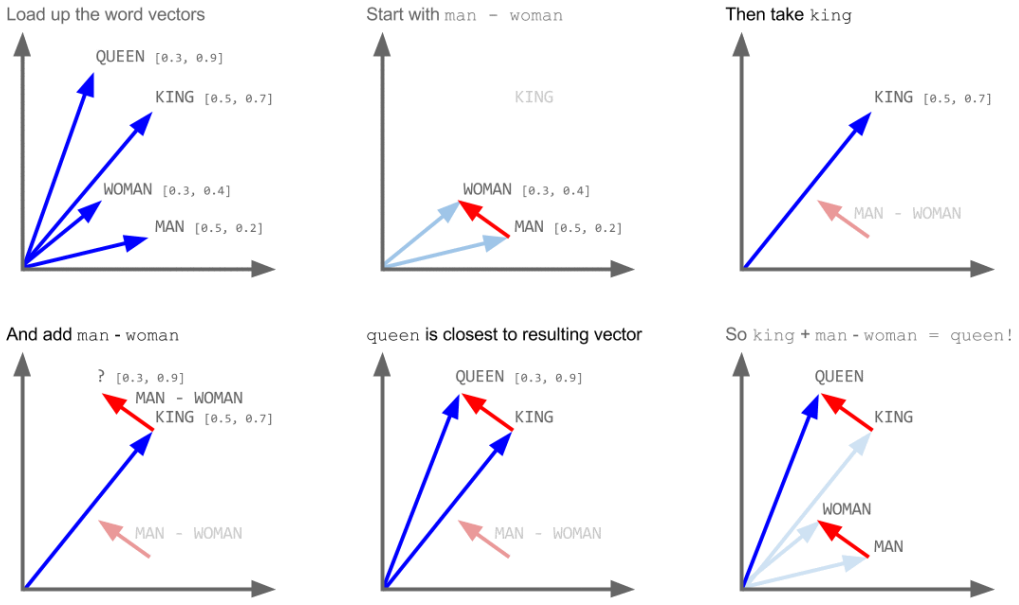


Figura 2.11: Esempio di operazioni sui vettori di word2vec.
Fonte: https://multithreaded.stitchfix.com/blog/2015/03/11/word-is-worth-a-thousand-vectors/

O ancora, è possibile controllare quanto due elementi siano simili, **trovare *l'elemento estraneo* in una lista di parole** o calcolare l'insieme di quelle semanticamente più simili a un elemento, come riportato nel listato 2.1:

```
>>> model.most_similar('vacation')
[('trip', 0.7234684228897095), ('honeymoon', 0.6447688341140747),
 ('beach', 0.6249285936355591), ('vacations', 0.5868890285491943),
 ('wedding', 0.5541957020759583), ('resort', 0.5231006145477295),
 ('traveling', 0.5194448232650757), ('vacation', 0.5068142414093018),
 ('vacationing', 0.5013546943664551)]

>>> model.doesnt_match("france england germany berlin".split())
'berlin'
```

Listato 2.1: Uso del modello word2vec.

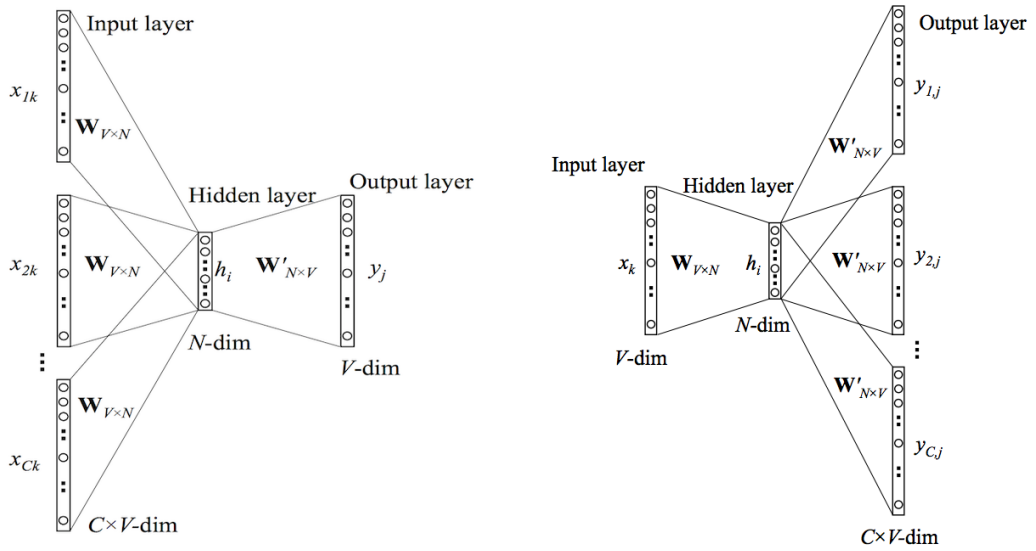
La particolarità di questo algoritmo è che appartiene alla famiglia dei modelli di machine learning **non supervisionati**, cioè non necessita che qualcuno etichetti i dati manualmente per poter addestrare il modello.

L'unica preoccupazione che si deve adottare per eseguire il training di word2vec, è che la quantità di frasi (semanticamente e sintatticamente corrette) sia molto elevata, anche nell'ordine **di qualche GB**, e lasciare che l'algoritmo analizzi parola per parola all'interno delle frasi. Riesce comunque a funzionare molto bene anche con dataset più piccoli, a patto di tollerare qualche imprecisione con casistiche particolarmente complesse.

Esistono due versioni differenti dell'algoritmo:

CBOW abbreviazione di *Continuous-bag-of-words*, predice una parola dato il contesto di n parole attorno;

Skip-Gram predice le parole che formano il contesto, data la parola centrale.



(a) CBOW (b) Skip-gram

Figura 2.12: Strutture delle reti neurali word2vec a confronto.

Oltre a differire per il diverso uso che se ne può fare, le modalità d'utilizzo si differenziano anche per **velocità** (la versione CBOW lo è di più) e **precisione** (Skip-gram predice meglio parole poco frequenti).

2.3.2 Doc2Vec

Nato poco dopo [4], questo algoritmo altro non è che un adattamento del precedente, in grado di lavorare non con le singole parole, bensì con elementi complessi composti anche da più frasi assieme.

Generalmente doc2vec è definito come **algoritmo per la rappresentazione di documenti** all'interno di uno spazio vettoriale complesso come quello di word2vec. Diventa quindi importante comprendere cosa si intenda con *documento*.

Non ne esiste una descrizione precisa ed esatta in realtà: potrebbe essere una sola frase, un paragrafo più o meno corto ma anche un intero articolo di giornale. La collezione di questi documenti è detta **corpus**, ed è ciò su cui si istruisce il modello.

Può essere addestrato, al pari di word2vec, per identificare quali documenti siano più simili a quello preso in analisi, ma **anche per fini di classificazione** (come ad esempio identificare la categoria di una notizia, o il genere cinematografico dei film partendo dalla trama).

```
Test Document (6): «senior members of the saudi royal family paid at least million
to osama bin laden terror group and the taliban for an agreement his forces would n
ot attack targets in saudi arabia according to court documents the papers filed in
us billion billion lawsuit in the us allege the deal was made after two secret meet
ings between saudi royals and leaders of al qa ida including bin laden the money en
abled al qa ida to fund training camps in afghanistan later attended by the septemb
er hijackers the disclosures will increase tensions between the us and saudi arabi
a»

SIMILAR/DISSIMILAR DOCS PER MODEL Doc2Vec(dm/m,d50,n5,w5,mc2,s0.001,t3):

MOST (261, 0.6407690048217773): «afghan opposition leaders meeting in germany have
reached an agreement after seven days of talks on the structure of an interim post
taliban government for afghanistan the agreement calls for the immediate assembly o
f temporary group of multi national peacekeepers in kabul and possibly other areas
the four afghan factions have approved plan for member ruling council composed of c
hairman five deputy chairmen and other members the council would govern afghanistan
for six months at which time traditional afghan assembly called loya jirga would be
convened to decide on more permanent structure the agreement calls for elections wi
thin two years»
```

Figura 2.13: Esempio di utilizzo di doc2vec.
da https://github.com/RaRe-Technologies/gensim/blob/develop/docs/notebooks/doc2vec-lee.ipynb

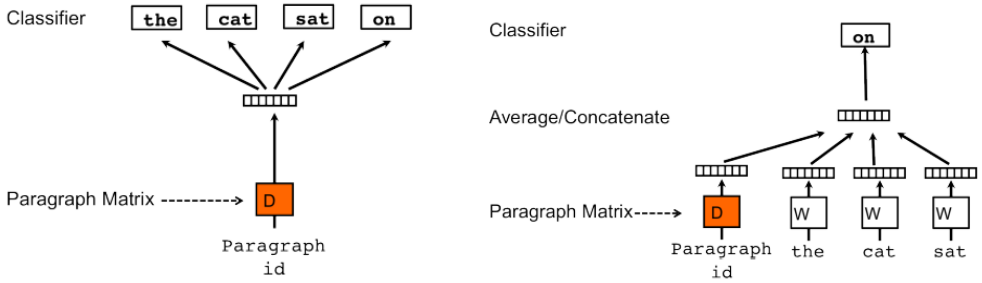
Come per word2vec, anche qui è possibile trovare due diverse versioni, usate per finalità simili a quelle viste in precedenza:

DBOW abbreviazione di *Distributed Bag of Words*, funziona in maniera analoga a quella skip-gram.

DM abbreviazione di *Distributed Memory*, è l'equivalente di CBOW.

Come è facile intuire, esiste un'associazione **inversa** tra i modelli di word2vec e doc2vec, almeno nel nome. Proprio per questo motivo la versione che ha un'accuratezza maggiore è quella *distributed memory*, mentre quella *DBOW* è più veloce e consuma meno memoria.

Anche per quanto riguarda il lato strutturale, le due versioni funzionano in maniera equivalente a quella vista con word2vec: DM ricalca in pieno CBOW, aggiungendo però l'identificatore del documento, mentre DBOW si occupa di predire quali parole siano più probabili dato l'identificatore del paragrafo.



(a) DBOW (b) DM

Figura 2.14: Strutture delle reti neurali doc2vec a confronto.
Fonte: https://medium.com/@michaelstecklein/hq-trivia-predictor-2c8583aa4adb [6]

Capitolo 3

Modellazione del progetto

Dopo aver descritto l'insieme delle tecnologie disponibili, si procede affrontando il modo in cui poterle utilizzare nella problematica in questione.

3.1 Analisi dei dataset

Come specificato nella sezione 1.4, l'obiettivo di questo lavoro di tesi è l'estrapolazione di informazioni riguardanti la similarità tra due elementi. Nello specifico, gli elementi di cui si sta parlando sono stringhe di testo molto corte (attorno ai 50-100 caratteri nella maggior parte dei casi).

Dopo una veloce analisi è possibile comprendere che non potranno essere usati tutti i campi aggiuntivi che sono stati inseriti all'interno del dataset, poiché sono i campi sono **fortemente sparsi**: ciò significa che non tutte le righe della tabella posseggono gli stessi attributi.

Inserire informazioni sparse, mancanti o incomplete rischia di essere un processo controproducente, in particolar modo nel machine learning dove è fondamentale struttura dei dati in ingresso.

CODICE	DESCRIZIONE	UM	DESCR_UM	REF_PRO	PIVA_PRO	CF_PRV	REF_FOR	PIVA_FOR	CF_FOR	AIC_PARAF	ATC	CODICE_EA	GTIN	CND	CODICE_CIVAB	TIPO_DM	CODICI	CI	CE
120003	ZOLETIL 50+50 MG/ML F. U/V	H87	PEZZO					1358640181		101580025									901009
120004	TERRAMICINA SPRAY ML.150 FLAC	H87	PEZZO					1358640181		100156013	8AA1								901009
120020	STRESNIL ML.100 MG.40 FLAC U	H87	PEZZO					1358640181		101294015	8AA1								901009
120021	DOMOSEDAN ML.5 FLA U/V	H87	PEZZO					1,0805E+10	983430281	100102019	8AA1								901009
120026	DEXADRESON FORTE SOSPEN.ML.	H87	PEZZO					1358640181		101867012	8AA1								901009
120027	COVINAN INIETT.ML.20 FLAC	H87	PEZZO					1,0805E+10	983430281	101904023	8AA1								901009
120037	BAYTRIL SOLUZ.2,5% ML.50 FLAC I	H87	PEZZO					1,0805E+10	983430281	100155213	8AA1								901009
120040	TANAX ML.50 FLAC U/VET	H87	PEZZO					1358640181		101583014	8AA1								901009
120050	DOMITOR ML.10 X 1 FLA U/V	H87	PEZZO					1358640181		100103011	8AA1								901009
120051	ANTISEDAN ML.10 X 1 FLA U/V	H87	PEZZO					1358640181		100104013	8AA1								901009
120061	ROMPUN SOL.INIET.2% ML25 FLAI	H87	PEZZO					1358640181		100390018	8AA1								901009
120063	IVOMEC SOLUZ.INIET.ML.50 FLAC	H87	PEZZO					1358640181		100197019	8AA1								901009
120098	DEXADRESON 2 MG/ML ML.50 FLA	H87	PEZZO					1358640181		101866010	8AA1								901009
120100	PREQUILLAN ML.50 FLE U/VE	H87	PEZZO					1358640181		101743033	8AA1								901009
120125	KLINIDRAPE TUBE-HOLDER STER	H87	PEZZO				708131	2426070120	1,2301E+10					T020199		1	174662		901085
120131	AGO SPINALE_SPROTTE_G24 MM	H87	PEZZO				021151-30A	1,1576E+10	1,1576E+10					A01030101		1	437733		901085
120235	GUIDA ANG_RADIOFOCUS GUIDE	H87	PEZZO				GA-35263M	7279701002	7279701002					C0499		1	176524		901085
120335	GUANTO STERILE BIOGEL REGENT	H87	PEZZO				82275-01	2426070120	1,2301E+10					T01010102		1	1031022		901085
120336	GUANTO STERILE BIOGEL REGENT	H87	PEZZO				82270-01	2426070120	1,2301E+10					T01010102		1	1031021		901085
120370	MASCHERA ANEST.MONOPAZ. NEI	H87	PEZZO				8805	9238800156	9238800156					R03010101		1	175119		901085
120371	STERI-DRAPE AD ANELLO 91X91 D	H87	PEZZO				1074	1,2972E+10	100190610					T020102		1	19602		901085
120381	SUTUR.ROTICULATOR 30-3.5 01	H87	PEZZO				17615	9238800156	9238800156					H020201					901085
120406	CIRCUITO X PENTAMIDINA 300/	H87	PEZZO				300/6367	9238800156	9238800156					R020199		1	1033030		901085
120413	CATETERE MOUNT X INTUBAZ. 33	H87	PEZZO				331/5112	9238800156	9238800156					R020201		1	173630		901085
120423	SUTUR.ROTICULATOR 30-V.3 01	H87	PEZZO				17619	9238800156	9238800156					H020201					901085
120448	RETE MARLEX MESH CM.26X36 PZ.	H87	PEZZO				0112660	1911071007	7931650589					P900202		1	24196		901063
120467	TUBO A SPIRALE__CM 180 22F/22	H87	PEZZO				285/5064	9238800156	9238800156					R020199		1	697363		901085
120507	TUBO REG.BOT.2000-4000ML 8(	H87	PEZZO				8884-729500	9238800156	9238800156					A06010102		1	233483		901085
120540	DILATAN DILAT.AN. MIS.VARIE A	H87	PEZZO				A.1003	926020066	926020066					L040902		1	48000		901011
120562	ECOJEKT G 21 X 150 MM. E211	H87	PEZZO				E2115	1296201005	4742650585					A01020102		1	28385		901085
120564	CATETERE MOUNT X INTUBAZ. 25	H87	PEZZO				331/5345	9238800156	9238800156					R020201		1	173637		901085
120567	AGO BIOPSIA MIELO-CAN G 15 22	H87	PEZZO				MCN22	3597020373	3597020373					A01020202		1	69478		901085
120613	CAT.URETERALE 9 FORI CH 6 (CUB	H87	PEZZO				195101H	2804530968	6324460150					U040301		1	21348		901085

Figura 3.1: Sparsità dei dati all'interno del dataset.

Un altro problema da gestire nel caso in cui si vogliano utilizzare gli attributi aggiuntivi risiede nel modo in cui questi sono stati memorizzati.

Il tipo di file `csv` non memorizza nessuna informazione accessoria riguardo a dati contenuti: il tipo del dato (stringa, numero, data) **viene dedotto durante il caricamento** dello stesso dataset. In questo caso, però, alcune colonne contengono attributi di tipo differente.

Questa situazione provoca confusione al modulo incaricato di leggere il file che, per poter concludere il suo lavoro, uniforma tutto a semplice *"oggetto"*.

Quindi, visto stanno trattando attributi non in grado di fornire alcun tipo di aiuto ai fini che si stanno perseguendo, semplicemente non verranno utilizzati o caricati durante il lavoro.

Nello stesso modo, anche tutte le colonne con dati mancanti o attributi non valorizzati, non sono stati utilizzati ne presi in considerazione.

3.1.1 Unione dei dati

Per poter lavorare sui due dataset è consono unire entrambe le tabelle in una unica, così da avere tutti i dati necessari in un unico luogo.

Sfortunatamente non è presente la struttura di un database relazionale, dove tipicamente è sufficiente utilizzare un identificatore comune ad entrambe le tabelle per poter unire i dati presenti.

Questo obbliga a cercare un metodo alternativo per la fusione, che viene generalmente effettuata sfruttando la similarità tra i valori degli attributi in comune.

3.1.2 Gestione dei duplicati

Come specificato dal fornitore dei dataset, è possibile che alcune righe possano risultare duplicate in maggior parte. Questa situazione è causata dalla presenza di prodotti identici ma assegnati ad aziende sanitarie ed ospedaliere differenti.

Infatti, l'assegnazione dello stesso prodotto a n aziende destinazione differenti genera n righe praticamente identiche, se non per i codici identificatori associati, che risultano differire.

AZIENDA	CODICE	DESCRIZIONE
080106	28.036	ABILIFY*28CPR 10MG
080109	10F002798	ABILIFY*28CPR 10MG
080109	10F002714	ABILIFY*28CPR 15MG
080105	104010761	ABILIFY*28CPR 5MG
080106	41.226	ABILIFY*28CPR 5MG
080109	10F002857	ABILIFY*28CPR 5MG
080106	50.613	ABILIFY*28CPR ORODISP 10MG
080109	10F005200	ABILIFY*28CPR ORODISP 10MG
080106	51.406	ABILIFY*28CPR ORODISP 15MG
080109	10F005201	ABILIFY*28CPR ORODISP 15MG
080106	46.930	ABILIFY*IM FL 1,3ML 7,5MG/ML
080105	104037996	ABILIFY*IM FL 1,3ML 7,5MG/ML
080109	10F006287	ABILIFY*IM FL 1,3ML 7,5MG/ML
080105	104011587	ABILIFY*OS FL 150ML 1MG/ML
080106	50.473	ABILIFY*OS FL 150ML 1MG/ML
080109	10F005198	ABILIFY*OS FL 150ML 1MG/ML

Figura 3.2: Esempio di righe duplicate.

Tutto ciò comporta l'esistenza di prodotti non strettamente necessari ai fini della tesi: i duplicati con il solo codice differente. Perché gestire come separati due prodotti identici e **senza alcuna informazione utile** che li differenzia?

Inoltre questo convalida l'idea di non utilizzare i campi aggiuntivi disponibili, limitando l'utilizzo a:

- **Descrizione** - La stringa identificativa del prodotto;
- **Codice regionale** - Il codice che verrà utilizzato per identificare le coppie all'interno dell'insieme dei prodotti.

3.1.3 Qualità dei testi elaborati

Il testo dei prodotti fornito è di qualità **decisamente scadente**:

- Spesso sono incluse nella descrizione alcune **indicazioni o informazioni** che nulla hanno a che fare con il prodotto;
- Utilizzo sconsiderato e non uniforme di **abbreviazioni**;
- **Caratteri speciali** utilizzati in abbondanza e in maniera del tutto arbitraria;
- Presenza di codici **alfanumerici** in mezzo alle stringhe.

DESCRIZIONE	codice_regionale
SACCA ILEO_SENSURA_15/33 MM_15500_F/A CONVEX 340ML TNT_	904977283
(FUORI PT) KESTINE*OS 30DOSI LIOF 10MG COMPRESSE SUBLINGUAL	34930141
(a esaurim.)VINCRISTINA SOLUZ. MG 1/ML EV (VCR)	26709016
(carenza in Italia)PENTOTHAL SODIUM FL.MG 500 EV	2347019
(revocato)TETABULIN 250 U.I. FL IM aic022601088	22601088
(rmp)ARIMIDEX MG 1 CP aic031809015	31809015
(rmp)ENANTONE DEPOT 3.75MG FL aic027066024	27066024
(rmp)HYALOFILL-F (Hyaff) cm5x5 cod.274410 CND: M040413 NrREP: 6	905975090
(rmp)LEUPRORELINA ACETATO 3,75MG IM-SC aic 027066125	27066125
(rmp)SMOFLIPID 200MG/ML emuls.infusionale 250 ML	37135050
*ACQUA PI*1FL 500ML	29824101
*ANTICOAG.ACD FORM. A 500 ML (C.T.)NOSCO	30701054
*CELSIOR/VIASPAN1LCONSERVAZIONE ORG.	40594018
*CLINIMIX N12G20E 2L CON ELET.NO SCORTA	32167280
*CLINIMIX N17G35E 2L CON ELET.	32167316
*ELETTR.MANT.CON GLUC.5% 500 ML (ESAU)	29836119

Figura 3.3: Esempi di descrizioni qualitativamente scadenti.

Questa situazione richiederà una profonda analisi e conseguente preprocessamento dei dati, prima ancora di implementare qualunque tipo di sistema (di machine learning o deep learning).

All'interno di questa fase dovranno essere il più possibile *pulite* le descrizioni testuali, cercando di non rimuovere eccessivamente informazioni che potrebbero risultare utili.

3.2 Scelta delle tecnologie adatte

Come indicato nel capitolo 2, è possibile affrontare il problema con 3 approcci differenti:

- Addestramento supervisionato;
- Addestramento non supervisionato;
- Addestramento per rinforzo.

Per gli obiettivi posti all'interno di questo lavoro di tesi, è possibile però implementare soltanto i primi due.

3.2.1 Addestramento non supervisionato

Il dataset contenente la maggior parte dei dati non contiene le relative *etichette*, cioè non contiene niente che permetta di stabilire quale valore in output ci si **aspetta di ottenere dall'elaborazione**.

Un primo approccio da poter attuare quindi, è quello di utilizzare un sistema non supervisionato per verificare se, **in assenza di input particolari**, questo riesca effettivamente a trovare similarità ed equivalenze tra tutti i prodotti presenti, senza addestrarlo a riconoscerle.

Tra i più famosi algoritmi disponibili nel campo della **semantic similarity search** è possibile trovare word2vec [2], GloVe [9] e doc2vec [4]. Soltanto quest'ultimo però, permette di non operare a livello di singole parole, e al contrario degli altri due gestisce documenti complessi. Per questo motivo sarà l'algoritmo utilizzato per l'approccio non supervisionato.

Per poter definire soddisfacenti i risultati ottenuti da questo algoritmo, è necessario utilizzare anche il dataset contenente gli accoppiamenti tramite codice regionale: se, dopo l'addestramento il modello fornisce una lista di elementi per la maggior parte equivalenti indicandoli come simili, allora doc2vec risulta essere un approccio molto valido.

3.2.2 Addestramento supervisionato

Il secondo approccio che è possibile utilizzare, è quello di fornire a un sistema supervisionato, i dati di input **associati all'etichetta corrispondente** al valore richiesto in uscita.

Questo approccio richiede, contrariamente al primo, un maggior lavoro di preprocessamento dei dati, in quanto questi non vengono forniti già accoppiati e etichettati come *equivalenti* o *non equivalenti*.

AZIENDA	CODICE	DESCRIZIONE	codice_regionale
080106	41.520	EPHYNAL*300MG 30CPS MOLLI	53037
AVEN	141828	PILOCARPINA 1% LUX COLL FLAC	248031
080106	374	PILOCARPINA 1%*COLL. 10 ML	248031
080114	00015940	PILOCARPINA CLOR COLL 10ML 1%	248031
080902	41279	PILOCARPINA LUX 1% CO	248031
080109	10F001448	PILOCARPINA LUX*COLL.1% 10ML(AFM)	248031
080114	00015950	PILOCARPINA 2% COLLIRIO 10 ML -	248056
AVEN	141829	PILOCARPINA 2% LUX COLL FLAC	248056
080909	000000000010000327	PILOCARPINA 2%/DROPILTON COLLIRIO	248056
080106	375	PILOCARPINA CLOR*COLL 10ML 2%	248056
080109	10F000717	PILOCARPINA LUX*COLL 10ML 2%	248056
080109	10F003306	PILOCARPINA LUX*COLL 10ML 2%_#AFM(AFM)	248056
080902	41283	PILOCARPINA-LUX 2% COLLIRIO	248056
AVEN	141824	ATROPINA 0,5% COLL. ML.10 FLAC	307037
080909	000000000010000328	ATROPINA 0,5% COLL.@	307037
080114	000137558	ATROPINA LUX COLL 10ML 5MG/ML	307037

Figura 3.4: Esempio di dati accoppiati tramite codice regionale.

Infatti, come è possibile vedere in figura 3.4, l'accoppiamento si basa semplicemente su un'associazione indiretta delle descrizioni a un codice definito *regionale*: quando due (**o più**) prodotti hanno lo stesso codice regionale, allora è possibile definirli **equivalenti**.

Questo insieme di informazioni è ricavabile confrontando ogni elemento con l'insieme dei restanti prodotti, e deve essere generato prima di poter sviluppare il modello.

Dovendo selezionare la tecnologia da impiegare, la scelta migliore ricade sul tipo di rete neurale (appartenente quindi ai sistemi di Deep Learning) detta **CharCNN**.

Le caratteristiche principali di questa rete sono:

- Gestione del testo a livello di caratteri;
- Rete **convoluzionale** con vari strati nascosti;
- Permette di eseguire la categorizzazione di un elemento.

La gestione a livello di caratteri è una caratteristica totalmente opposta a quella presente in Doc2Vec, che sfrutta le parole intere per calcolare le similarità, e fornirà un banco di prova interessante tramite il quale poter capire l'approccio che funzionerà meglio con il tipo di dato a disposizione.

Una rete **convoluzionale** lavora gli input nella loro interezza, senza dividerli in unità elementari, mantenendo memoria della vicinanza e della struttura.

Per fare un paragone, se il tipo di dato gestito fosse un'immagine, una rete convoluzionale al posto di elaborare pixel per pixel in maniera indipendente uno dall'altro, gestisse una matrice di pixel vicini contemporaneamente.

Questo approccio è molto importante sia nel trattamento di testi in linguaggio naturale, sia di immagini, in quanto è possibile produrre risultati più precisi e accurati.

L'ultimo aspetto da chiarire è come **categorizzare gli elementi** possa essere d'aiuto a calcolare la similarità: basterà creare una stringa contenente due prodotti, divisi da un separatore arbitrario, associato a un valore **1** o **0** per indicarne la similarità o la non equivalenza.

ASPIRINA 10CPRMAST500MG @@@ ASPIRINA 10 CPR MAST 500 MG

In questo modo la rete dovrà *predire* un valore tra i due disponibili.

Capitolo 4

Sviluppo

Si procede con la descrizione del lavoro svolto sui dataset, e lo sviluppo dei modelli descritti nei capitoli precedenti.

4.1 Preprocessamento

Dopo aver caricato tutte le librerie di utility, il primo passo da eseguire è *pulire* le stringhe in modo da addestrare il più efficacemente possibile entrambi i sistemi.

Dopo aver caricato da entrambi i dataset le sole colonne interessate (`Descrizione`, `codice_regionale, Azienda e Codice`) è possibile procedere con **l'unione** delle informazioni disponibili.

In questo modo c'è un solo dataframe contenente le informazioni di tutti gli elementi molto più facile da gestire, che contiene:

- Sia tutte le 300.000 descrizioni totali;
- Sia i codici regionali per poter accoppiare i prodotti;

Si procede successivamente con la rimozione di tutte le descrizioni *duplicate*, cioè le stringhe che compaiono più volte all'interno del dataset. Questo perché, come è stato detto, molti prodotti forniti a più aziende contemporaneamente compaiono altrettante volte nel dataset, identificatori numerici differenti.

Poiché verranno fornite ai modelli soltanto le descrizioni da analizzare e nient'altro, non c'è pericolo di perdere informazioni o accuratezza.

Si rimuovono inoltre le descrizioni *vuote* non valorizzate.

Lo step successivo prevede la rimozione di determinati caratteri dall'inizio delle descrizioni testuali, che potrebbero dare problemi in fase di accoppiamento. Per svolgere questo compito è stata scritta una funzione adibita proprio a questo lavoro, usata in ultima battuta per rimuovere gli spazi dall'inizio delle stringhe.

Molti prodotti contengono all'interno della propria descrizione:

- *In esaurimento*;
- *Sostituito da.*

Questo per indicare l'effettivo l'esaurimento di un prodotto non più disponibile: non potendo aggiungere un campo adibito a tal compito, è stato indicato direttamente nella descrizione. Lo stesso intervento è stato eseguito per indicare la sostituzione di un vecchio prodotto con uno nuovo.

Tutto questo è stato fatto per mantenere inalterato l'insieme dei prodotti disponibili, dando però una direttiva chiara a chiunque stesse cercarlo prodotti teoricamente non più presenti.

Sono state rimosse le descrizioni contenenti entrambe le stringhe, o abbreviazioni equivalenti (*esaurim.* o *sost.* ad esempio), sviluppando una funzione per visualizzarne gli elementi in base al pattern di ricerca, prima di effettuare la cancellazione.

Dopo l'insieme di rimozioni svolte, viene successivamente reimpostato l'indice del dataframe, così da non avere buchi nella sequenza di indici (ad esempio: $0 \rightarrow 1 \rightarrow \mathbf{3}$).

Questa operazione sarà molto utile nello sviluppo del modello Doc2Vec, in quanto ci permetterà di accedere molto più facilmente ai dati elaborati.

Di seguito il codice utilizzato per il preprocessamento dei dati.

4.2 Sviluppo modello Doc2Vec

Dopo aver preprocessato tutti i dati presenti nel dataset, si passa all'addestramento della rete Doc2Vec basata sulla libreria Gensim.

Dopo aver importato tutto il necessario, si procede con la creazione di 3 dizionari e una lista a fini di utility:

- `cod2count` - per assegnare a ogni codice regionale, il numero di volte che compare nel dataset;
- `cod2list` - per assegnare a ogni codice regionale la lista di identificatori nel dataset associati a quel codice regionale;
- `countGrouped` - viene utilizzato per controllare quanti gruppi di elementi nel dataset condividano il numero di simili. Ad esempio, se nel dizionario compare $\mathbf{\{3 \rightarrow 300\}}$, questo sta ad indicare che ci sono 300 codici regionali diversi associati a 3 identificatori ognuno;

- `listGrouped` - la lista dei codici regionali associati a un solo elemento, con i quali non è possibile trovare coppie equivalenti.

Queste strutture dati saranno utili in un secondo momento.

Come primo passo per sviluppare la rete Doc2Vec, è necessario segmentare i documenti (cioè le descrizioni dei prodotti) in liste di elementi singoli.

Per eseguire test maggiormente accurati e ampi, sono state eseguite 2 diverse segmentazioni:

1. La prima tokenizzando le descrizioni sulla singola parola, dividendo ad esempio `"Aspirina 500"` in `"Aspirina" "500"`;

2. La seconda scomponendo il testo in trigram, generando quindi delle sottostringhe di 3 caratteri per ogni parola, ad esempio elaborando `"Aspirina 500"` in `"Asp spi pir iri rin ina 500"`.

Viene successivamente creato il *corpus*, cioè l'insieme dei testi associati all'identificatore che utilizza Doc2Vec internamente per identificare ogni documento. Doc2Vec non necessita di altro, e si potrebbe procedere già all'addestramento della rete.

Ma visto che verranno sviluppati diversi modelli, ognuno con caratteristiche e impostazioni differenti, per controllare se esista un parametro in grado di migliorare notevolmente la precisione.

Si definiscono inoltre le funzioni che verranno utilizzate per controllare gli errori commessi dai modelli, in quanto essendo Doc2Vec non supervisionato, non possiede metodi per l'autovalutazione.

Le metriche utilizzate per il calcolo dell'accuratezza di Doc2Vec sono le seguenti:

Similarità Coseno - usata per controllare quanto due documenti siano simili, e si misura su una scala da zero a uno. Più sono simili, e più il valore sarà tendente a uno;

mAP - acronimo di mean Average Precision, indica la precisione di un sistema nel recuperare l'insieme di documenti rilevanti, dando molta importanza alla posizione in cui sono stati ottenuti. Si misura nella stessa scala della similarità coseno, da zero a uno;

r Precision - anch'essa serve a calcolare la precisione nella ricerca di documenti rilevanti, ma pone maggiore importanza sulla quantità, e non sulla posizione.

4.3 Sviluppo modello CharCNN

Per poter procedere con l'addestramento della rete neurale, è necessario generare le coppie di prodotti partendo dai codici regionali.

È buona prassi **generare un dataset** per il training adeguatamente bilanciato, senza esagerare con esempi negativi o positivi. Per questo motivo, le coppie contenute all'interno di questo insieme devono risultare, nel 50% dei casi, *prodotti equivalenti*, e nel restante 50% *coppie di prodotti diversi* (che verranno chiamate per semplicità *coppie negative*).

In base ai dati forniti, è possibile produrre al massimo 86574 coppie positive, che verranno selezionate tutte.

Così come avvenuto con Doc2Vec, anche in questo caso vengono intrapresi due percorsi differenti:

- Nel primo caso le coppie *negative* sono composte per il 25% da elementi molto simili e difficili da predire come diversi, mentre per il restante 75% le coppie sono generate in maniera casuale;

- Nel secondo caso, la **media delle similarità** delle coppie negative è circa la stessa di quella delle coppie *positive*.

Studiando due approcci diversi, è possibile constatare quale dei due ottenga i **risultati migliori**. Appartenendo alla famiglia dei approcci supervisionati, inoltre, la rete riesce a valutare in autonomia la propria precisione nei risultati conseguiti.

Per calcolare la similarità tra le coppie, è stata sviluppata la **matrice della similarità coseno**, cioè una matrice $n \times n$ contente la similarità di ogni prodotto con i restanti $n - 1$.

Per poter essere calcolata, ogni prodotto deve essere raffigurato come un vettore di elementi.

Il testo dei prodotti è stato dapprima scomposto in trigram, secondo l'elaborazione vista in precedenza, e successivamente è state generata la **Bag of Words** di questi elementi. Questa matrice è uno degli strumenti più utilizzati nel machine learning tradizionale per rappresentare oggetti e documenti all'interno di un piano dimensionale.

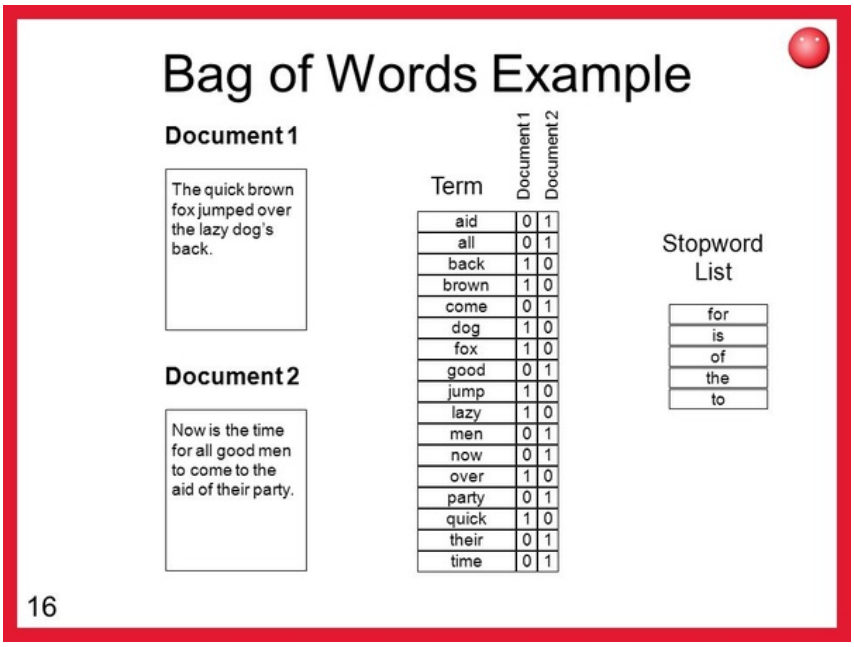


Figura 4.1: Esempio di Bag Of Words
Fonte: https://www.quora.com/What-is-the-bag-of-words-algorithm

Viene sviluppata associando a ogni prodotto il vettore contenente le occorrenze dei trigram che lo compongono.

Da un lato contare le occorrenze è un metodo estremamente efficace quando si tratta di natural language processing, ma risulta problematico quando l'ordine delle occorrenze è fondamentale, poiché la Bag of Words non è in grado di gestire questo tipo di informazione.

Moltiplicando la matrice per la sua **trasposta** e dividendo per le **norme** di ognuna, si ottiene la matrice quadrata con l'insieme delle similarità coseno dei prodotti.

La matrice ottenuta è poi utilizzata per generare la lista delle coppie negative, secondo le modalità illustrate in precedenza.

Nel primo caso si selezionano tutte le coppie **negative** con una similarità coseno maggiore di una determinata soglia. Le coppie mancanti per arrivare a 86574 elementi vengono generate in maniera casuale, senza preoccuparsi della similarità coseno.

Nel secondo caso invece, viene prima calcolata la distribuzione delle similarità coseno delle coppie *positive*, e solo successivamente vengono selezionate le coppie *negative* in base alla distribuzione calcolata. Questo secondo approccio permette di ottenere un insieme di coppie uniformemente distribuite tra negative e positive, ma anche uniforme sotto il profilo della similarità coseno.

4.4 Risultati

4.4.1 Doc2Vec

I risultati prodotti da Doc2Vec (raffigurati nella tabella della pagina successiva) non sono per niente soddisfacenti: benché la similarità coseno media sia a volte estremamente elevata, gli altri due indicatori (mAP e rPrec) sottolineano una **precisione estremamente bassa** nella ricerca dei prodotti simili.

Da sottolineare come l'impiego dei **trigram** abbia aiutato notevolmente l'aumento della similarità coseno, ma che non abbia sortito gli stessi risultati nella precisione di ricerca dei documenti equivalenti.

Questo è indice del fatto che Doc2Vec non riesca a distinguere oggetti non equivalenti tra loro, aggregando come *simili* oggetti che di simile non hanno nulla.

	AZIENDA	CODICE	DESCRIZIONE	codice_regionale	Char_Count
0	AVEN	120003	ZOLETIL 50+50 MG/ML F U/VET	101580025	27
544	AVEN	136462	FENISTIL GOCCE ML 20 FLAC	020124020	25
30895	080105	104007457	CETAFIL DET FLUIDO 470ML	NaN	24
87352	80109	10P004611	RILASTIL AQUA LEGERE CREMA ML 50 (AFM)	NaN	38
97833	80909	10063107	PROT BIL WALLFLEX BILIARY RX PC MM10X60	NaN	40
171171	80904	120003	ZOLETIL 50+50 MG/ML F U/VET - ZOLAZEPAM + TILE...	101580025	52
175535	80101	335375	VITE CEFALICA DOPPIO FIL D 5 MM 45COD DT750450	NaN	46
231806	80103	736482	VITE DA CORTICALE Ø3 5X40 TOT FIL AUTOFIL TI 3...	NaN	52
231807	80103	736483	VITE DA CORTICALE Ø3 5X30 TOT FIL AUTOFIL TI 3...	NaN	52
241185	80114	00081049	ZOLETIL 100 FLAC 5ML	NaN	20

Figura 4.2: Tabella raffigurante i 10 prodotti più simili per Doc2Vec a "Zoletil".

Effettuando una ricerca tramite i modelli addestrati sui trigram, è facile notare similarità derivanti da *sillabe*, o **sigle** condivise (nell'esempio sono tutti accomunati dalla presenza della sillaba *TIL*).

vector_size	min_count	window_size	epoch	is_trigram	med_cosine_dist	cosine_dev_std	med_mAP	mAP_std	rPrec_mean	rPrec_dev_std
50	0	5	100	no	0.7144	0.1559	0.0366	0.1744	0.0191	0.1013
50	0	5	100	yes	0.9302	0.0472	0.1579	0.3354	0.0909	0.2148
50	0	5	300	no	0.7434	0.1312	0.0495	0.2025	0.0256	0.1182
50	0	5	300	yes	0.8610	0.0695	0.1965	0.3657	0.1132	0.2358
50	5	5	100	no	0.7252	0.1465	0.0504	0.2029	0.0254	0.1155
50	5	5	100	yes	0.9279	0.0481	0.1601	0.3376	0.0910	0.2123
50	5	5	300	no	0.7340	0.1239	0.0723	0.2392	0.0353	0.1317
50	5	5	300	yes	0.8546	0.0716	0.2001	0.3670	0.1157	0.2371
50	10	5	100	no	0.7255	0.1441	0.0603	0.2204	0.0296	0.1230
50	10	5	100	yes	0.9250	0.0488	0.1635	0.3402	0.0923	0.2138
50	10	5	300	no	0.7319	0.1244	0.0804	0.2505	0.0393	0.1376
50	10	5	300	yes	0.8509	0.0721	0.2081	0.3714	0.1189	0.2362
100	0	5	100	no	0.7033	0.1490	0.0394	0.1813	0.0212	0.1087
100	0	5	100	yes	0.9131	0.0625	0.1604	0.3412	0.0878	0.2088
100	0	5	300	no	0.7322	0.1270	0.0504	0.2024	0.0264	0.1189
100	0	5	300	yes	0.8574	0.0770	0.1831	0.3591	0.1020	0.2251
100	5	5	100	no	0.7163	0.1421	0.0538	0.2109	0.0267	0.1194
100	5	5	100	yes	0.9107	0.0631	0.1605	0.3394	0.0909	0.2156
100	5	5	300	no	0.7313	0.1232	0.0688	0.2347	0.0341	0.1317
100	5	5	300	yes	0.8529	0.0774	0.1900	0.3660	0.1043	0.2272
100	10	5	100	no	0.7172	0.1413	0.0565	0.2130	0.0282	0.1203
100	10	5	100	yes	0.9086	0.0634	0.1655	0.3443	0.0948	0.2209
100	10	5	300	no	0.7274	0.1239	0.0794	0.2512	0.0376	0.1349
100	10	5	300	yes	0.8489	0.0779	0.1922	0.3671	0.1067	0.2304
50	0	15	100	yes	0.9433	0.0406	0.1721	0.3502	0.0964	0.2193
50	0	20	100	yes	0.9340	0.0513	0.1657	0.3435	0.0939	0.2174
50	5	15	100	yes	0.9404	0.0414	0.1657	0.3441	0.0937	0.2171
50	5	20	100	yes	0.9322	0.0505	0.1654	0.3436	0.0936	0.2162
50	10	15	100	yes	0.9377	0.0424	0.1648	0.3396	0.0937	0.2154
50	10	20	100	yes	0.9302	0.0491	0.1670	0.3440	0.0949	0.2176
100	0	15	100	yes	0.9320	0.0502	0.1803	0.3583	0.1054	0.2363
100	0	20	100	yes	0.9223	0.0599	0.1735	0.3531	0.1007	0.2304
100	5	15	100	yes	0.9310	0.0495	0.1827	0.3581	0.1068	0.2334
100	5	20	100	yes	0.9214	0.0594	0.1720	0.3511	0.0987	0.2262
100	10	15	100	yes	0.9293	0.0509	0.1810	0.3590	0.1034	0.2305
100	10	20	100	yes	0.9204	0.0587	0.1720	0.3524	0.0967	0.2236

4.4.2 CharCNN

Al contrario dei risultati ottenuti con Doc2Vec, questo approccio fornisce dei modelli con prestazioni maggiori.

Il primo modello, (addestrato su coppie negative *molto simili* al 25% mentre il restante 75% generato casualmente) ha ottenuto dopo solo 6 cicli di addestramento i seguenti risultati:

- **Training Accuracy**: 94.30%
- **Training Loss**: 0.1434
- **Test Accuracy**: 95.55%
- **Test Loss**: 0.1139

Mentre il secondo modello (addestrato su coppie negative scelte tramite la distribuzione della similarità coseno delle coppie positive) dopo 24 cicli di addestramento ha raggiunto:

- **Training Accuracy**: 93.13%
- **Training Loss**: 0.1643
- **Test Accuracy**: 93.20%
- **Test Loss**: 0.1660

Risultati ottimi in entrambi i casi. Tutto questo dimostra che una rete neurale addestrata in maniera supervisionata possa riuscire ad ottenere risultati di gran lunga superiori a una non supervisionata, anche se di contro ha il grande svantaggio di non poter essere utilizzata *direttamente* per ottenere la lista dei prodotti più simili a quello dato in input.

Capitolo 5

Il codice prodotto

Segue codice latex generato da jupyter ...

5.1 Preprocessamento dati

5.1.1 Caricamento dati e librerie

In [1]:
```
import numpy as np
import pandas as pd
import scipy as sc
from tqdm import tqdm_notebook
from time import sleep
```

In [2]:
```
# GLOBAL SETTINGS
CHECK_ALL_CODES_IN_COUPLES = False
```

In [ ]:
```
all_data = pd.read_csv(
    "Totale.csv",
    sep=";",
    usecols=["DESCRIZIONE", "CODICE", "AZIENDA"])
accop = pd.read_csv(
    "Appaiamenti al 07_09_2018.csv",
    sep=";",
    usecols=[
        "DESCRIZIONE", "CODICE", "AZIENDA",
        "codice_regionale"
    ])
all_data.head()
```

5.1.2 Unione dei dataframe

```
In [4]: all_data = pd.merge(
            all_data,
            accop,
            how='outer',
            on=["AZIENDA", "CODICE", "DESCRIZIONE"])
        assert all_data.codice_regionale.count(
        ) == accop.shape[0]
```

```
In [5]: original_data = all_data.copy()
```

5.1.3 Controlli sui dati

Controllo che tutte le righe in `Appaiamenti` siano in `Totale`

```
In [6]: mismatched_couples = []
        if CHECK_ALL_CODES_IN_COUPLES:
            for x in tqdm_notebook(range(accop.shape[0])):
                if accop.loc[0][
                        "CODICE"] not in all_data.CODICE.values:
                    mismatched_couples.append(x)
```

```
In [7]: if CHECK_ALL_CODES_IN_COUPLES:
            print(mismatched_couples)
```

Controllo duplicati & aggiunta numero caratteri

```
In [8]: count_unique, count_all = all_data.DESCRIZIONE.unique(
        ).shape[0], all_data.shape[0]
        print(
            "Descrizioni univoche: {}\nDescrizioni totali: {}\n(CioÃ¨ {} duplicat
            .format(count_unique, count_all,
                    count_all - count_unique))
```

```
Descrizioni univoche: 309960
Descrizioni totali: 350716
(CioÃ¨ 40756 duplicati)
```

```
In [9]: def updateCharCount():
            all_data[
                "Char_Count"] = all_data.DESCRIZIONE.str.len(
                )
```

```
In [10]: updateCharCount()
         all_data.sort_values(
             by="Char_Count", ascending=False).head()
```

Out[10]:

	AZIENDA	CODICE	DESCRIZ
167443	80904	00300211	SOSTITUITO DAL - --00300214----------
171915	80904	04301955	KIT CONSUMABILE BIOMATCH PER ZEBRA - GK420T C
167640	80904	00370302	SOSTITUITO DAL - --80007552-DI-FARMACIA------
173825	80904	643478	AGO IPO PEN_INSUPEN_31G X 8MM_ARTSA_ - _____C
165678	80904	001624485	PROT BIL/PANCR_HOT AXIOS__5355__ - 15X10MM -

	codice_regionale	Char_Count
167443	NaN	202.0
171915	NaN	199.0
167640	NaN	187.0
173825	NaN	182.0
165678	NaN	181.0

```
In [12]: all_data.sort_values(
             by="Char_Count", ascending=False).tail(10)
```

Out[12]:

	AZIENDA	CODICE	DESCRIZIONE	codice_regionale	Char_Count
118032	80960	211375	PET	NaN	3.0
78423	80908	99557188	ALK	NaN	3.0
179892	80101	340738	HGH	NaN	3.0
118406	80960	211780	RMN	NaN	3.0
43434	80105	20Z1151471	6S	NaN	2.0
43411	80105	20Z112364	4C	NaN	2.0
43455	80105	20Z1223123	9L	NaN	2.0
124711	80960	39120	m	NaN	1.0
167432	80904	00211294	NaN	NaN	NaN
168985	80904	01170159	NaN	NaN	NaN

5.1.4 Rimozione elementi inutili

Rimozione righe NaN

```
In [13]: all_data.dropna(
             subset=["DESCRIZIONE"], inplace=True)
         original_data.dropna(
             subset=["DESCRIZIONE"], inplace=True)

         all_data.sort_values(by="Char_Count").head()
```

```
Out[13]:        AZIENDA      CODICE DESCRIZIONE codice_regionale  Char_Count
         124711   80960       39120           m               NaN         1.0
         43411    80105   20Z112364          4C               NaN         2.0
         43434    80105  20Z1151471          6S               NaN         2.0
         43455    80105  20Z1223123          9L               NaN         2.0
         118406   80960      211780         RMN               NaN         3.0
```

```
In [14]: print(
             "Righe eliminate: {}\nIl nuovo numero di elementi Ã¨: {}"
             .format(count_all - all_data.shape[0],
                     all_data.shape[0]))
         count_all = all_data.shape[0]
```

```
Righe eliminate: 2
Il nuovo numero di elementi Ã¨: 350714
```

Rimozione duplicati

Stando attenti a non rimuovere quelli che contengono il codice regionale.

```
In [15]: duplicated = all_data[all_data.duplicated(
             subset="DESCRIZIONE")]
         duplicated.head()
```

```
Out[15]:       AZIENDA     CODICE  \
         23124  080105   14013298
         23360  080105   14014046
         26479  080105   14021019
         30076  080105  103006565
         31991  080105  104013288

                                                                  DESCRIZI
         23124                        FORBICE CHIR. STILLE 150MM RET BC580R- AESCU
         23360                SCARPA TIPO TREKK.INV. N.44  mod. OW 31ex JOLLY 840
         26479  KIT SOMM IRRIG___1895522_TUBI IRRIGAZ_MANIPOLO STRAIGHTSHOT CONFX
         30076                                    ESTRAT PUNTI_PROXIMATE__PS
         31991                                          PROTESI RESINA FRATT

               codice_regionale  Char_Count
         23124              NaN        47.0
```

```
23360              NaN        54.0
26479              NaN        68.0
30076              NaN        29.0
31991              NaN        23.0
```

In [16]:
```
# conterrÃ  tutte le descrizioni duplicate CON codice regionale
cod_red_notna_desc = duplicated[
    duplicated.codice_regionale.
    notna()].DESCRIZIONE.values

print("Descrizioni duplicate da rimuovere : {} \n\
Descrizioni duplicate associate a un codice regionale: {}"
      .format(duplicated.shape[0],
              len(cod_red_notna_desc)))
```

```
Descrizioni duplicate da rimuovere : 40755
Descrizioni duplicate associate a un codice regionale: 27739
```

In [17]:
```
# ConterrÃ  TUTTI i valori duplicati, non solo quelli
all_duplicate = all_data.copy()

# Aggiunto l'indice originario per poter modificare il dataframe origino
all_duplicate.reset_index(inplace=True)
all_duplicate.rename(
    index=str,
    columns={'index': 'ORIG_INDEX'},
    inplace=True)

# Lascio le righe che hanno descrizione uguale a una di quelle duplicate
all_duplicate = all_duplicate[(
    all_duplicate.DESCRIZIONE.isin(
        cod_red_notna_desc))]

# Le ordino per essere sicuro successivamente di creare il dizionario pe
all_duplicate.sort_values(
    by="codice_regionale",
    ascending=False,
    inplace=True)

all_duplicate.shape[0] / len(cod_red_notna_desc)
```

Out[17]:
```
2.0541836403619453
```

In [18]:
```
all_duplicate.sort_values(
    by="DESCRIZIONE").head(2)
```

Out[18]:

	ORIG_INDEX	AZIENDA	CODICE	DESCRIZIONE	codice_regionale	Char_Count
341363	341365	080101	529552	BOMBOLA OSSIGENO*COMPR 200 BAR 1 LTCON VALVOLA RIDUTT SICO	039134111	59.0
189727	189729	80101	529552	BOMBOLA OSSIGENO*COMPR 200 BAR 1 LTCON VALVOLA RIDUTT SICO	NaN	59.0

In [19]:
```
# Dizionario che verrÃ  usato per associare DESCRIZIONE -> codice_regio
desc2cod = {}
nonSoddisfatti = 0

for i in tqdm_notebook(
        range(all_duplicate.shape[0])):
    row = all_duplicate.iloc[i]
    if (pd.isna(row.codice_regionale)):
        if (row.DESCRIZIONE in desc2cod):
            all_data.at[
                row.ORIG_INDEX,
                "codice_regionale"] = desc2cod[
                    row.DESCRIZIONE]
        else:
            nonSoddisfatti += 1
    elif row.DESCRIZIONE not in desc2cod:
        desc2cod[
            row.
            DESCRIZIONE] = row.codice_regionale
```

```
HBox(children=(IntProgress(value=0, max=56981), HTML(value='')))
```

In [20]:
```
nonSoddisfatti
```

```
        # Il numero di elementi ai quali non Ã¨
        # stata apportata modifica per inesistenza
        # di codici_regionali Ã¨:
```

```
Out[20]: 0
```

Per essere sicuri controlliamo che la coppia di prima abbia lo stesso codice nel dataframe originale. Parilamo della coppia:

```
In [21]: all_duplicate.sort_values(
             by="DESCRIZIONE").head(2)
```

```
Out[21]:         ORIG_INDEX AZIENDA  CODICE  \
         341363      341365  080101  529552
         189727      189729   80101  529552

                                                                DESCRIZIONE  \
         341363   BOMBOLA OSSIGENO*COMPR 200 BAR 1 LTCON VALVOLA RIDUTT SICO
         189727   BOMBOLA OSSIGENO*COMPR 200 BAR 1 LTCON VALVOLA RIDUTT SICO

                codice_regionale  Char_Count
         341363        039134111        59.0
         189727              NaN        59.0
```

```
In [22]: all_data[all_data.index.isin(
             all_duplicate.sort_values(by="DESCRIZIONE").
             head(2).ORIG_INDEX.values)]
```

```
Out[22]:        AZIENDA  CODICE  \
         189729   80101  529552
         341365  080101  529552

                                                                DESCRIZIONE  \
         189729   BOMBOLA OSSIGENO*COMPR 200 BAR 1 LTCON VALVOLA RIDUTT SICO
         341365   BOMBOLA OSSIGENO*COMPR 200 BAR 1 LTCON VALVOLA RIDUTT SICO

                codice_regionale  Char_Count
         189729        039134111        59.0
         341365        039134111        59.0
```

Possimo quindi procedere con la rimozione dei duplicati:

```
In [23]: all_data.drop_duplicates(
             subset="DESCRIZIONE", inplace=True)
         all_data[all_data.DESCRIZIONE == all_duplicate.
                  iloc[0]["DESCRIZIONE"]]
```

```
Out[23]:        AZIENDA  CODICE                                DESCRIZIONE  \
         310147   80106  20.461  MAGLIA TUBOLARE SOTTOGESSO CM  5,2      24003

                codice_regionale  Char_Count
         310147         908510516        45.0
```

```
In [24]: "Il nuovo numero di elementi Ã¨: {}".format(
             all_data.shape[0])
```

```
Out[24]: 'Il nuovo numero di elementi Ã¨: 309959'
```

5.1.5 Pulizia dei testi delle stringhe

Rimozione caratteri particolari all'inizio della stringa

```
In [25]: # Definizione di funzione per la rimozione di caratteri all'inizio delle
         def remove_from_start(special_char, dataframe,
                               col_name):
             toEdit = dataframe[dataframe[col_name].str.
                                startswith(special_char)]
             dataframe[col_name] = dataframe[
                 col_name].str.strip(special_char)
             print(
                 "Ho modificato {} elementi (carattere rimosso: '{}')."
                 .format(toEdit.shape[0], special_char))
```

```
In [26]: remove_from_start("#", all_data, "DESCRIZIONE")
```

```
Ho modificato 9 elementi (carattere rimosso: '#').
```

```
In [27]: remove_from_start("%", all_data, "DESCRIZIONE")
```

```
Ho modificato 7 elementi (carattere rimosso: '%').
```

```
In [28]: remove_from_start("$", all_data, "DESCRIZIONE")
```

```
Ho modificato 2 elementi (carattere rimosso: '$').


In [29]: remove_from_start("&", all_data, "DESCRIZIONE")

Ho modificato 19 elementi (carattere rimosso: '&').


In [30]: all_data.DESCRIZIONE.replace(
             '--+', '-', regex=True, inplace=True)
         all_data.DESCRIZIONE.replace(
             '(?<=\w)\.', ' ', regex=True, inplace=True)
         all_data.DESCRIZIONE.replace(
             '__+', '_', regex=True, inplace=True)
         all_data.DESCRIZIONE.replace(
             '_', ' ', regex=True, inplace=True)
         all_data.DESCRIZIONE.replace(
             '- -', '-', regex=True, inplace=True)
         all_data.DESCRIZIONE.replace(
             '\s+', ' ', regex=True, inplace=True)

         updateCharCount()
         all_data.sort_values(
             by="Char_Count", ascending=False).head(5)

Out[30]:        AZIENDA     CODICE  \
         171915   80904   04301955
         165678   80904  001624485
         164652   80904  001623378
         163378   80904  001621762
         157867   80904  001211433


         171915  KIT CONSUMABILE BIOMATCH PER ZEBRA - GK420T COD BM-CP74 COMPRENS
         165678                     PROT BIL/PANCR HOT AXIOS 5355 - 15X10MM - STE
         164652                     PROT BIL/PANCR HOT AXIOS 5354 - 10X10MM - STE
         163378                          SIST MONITOR CARD REVEAL LINQ LNQ11 - CO
         157867                           SACCA SANGUE QUADRUPLA CQ32250 - C/FILT

                codice_regionale  Char_Count
         171915              NaN         197
         165678              NaN         178
```

```
         164652              NaN         178
         163378              NaN         173
         157867              NaN         172
```

```
In [31]: remove_from_start(" ", all_data, "DESCRIZIONE")
```

```
Ho modificato 457 elementi (carattere rimosso: ' ').
```

Caratteri iniziali

```
In [32]: all_data.iloc[all_data.DESCRIZIONE.str.lower().
                       argsort()].head(5)
```

```
Out[32]:        AZIENDA      CODICE  \
         282684   80114   008619962
         261228   80114  0012044897
         250794   80114  0009003864
         202361   80902   101034721
         223540   80902  H051030906

                                                                    DESCRIZIO
         282684       "ARTROMOT" PER MOBILIZZAZIONE - CONTINUA PASSIVA CON CARREI
         261228  "BRACCIALE-CAMERA D'ARIA 5X26,5CM - 1 TUBO NEONATALE - 5001 01 1
         250794                   "MOSAIC GRANULOCITI 13" - 100 T - FA 1201-1010-
         202361                             ( estero ) CLORFENAMINA MG10 ML 1 FIA
         223540                            ( fibrosi cistica ) SIDERAL ORO 20 STI

                codice_regionale  Char_Count
         282684              NaN          61
         261228              NaN          66
         250794              NaN          49
         202361              NaN          39
         223540              NaN          40
```

```
In [33]: all_data.iloc[all_data.DESCRIZIONE.str.lower().
                       argsort()].tail(5)
```

```
Out[33]:        AZIENDA     CODICE                          DESCRIZIONE  \
         84358    80109  10F002262            ZYVOXID*10SACCHE INF 2MG/ML
         312158   80106     32.176               ZYVOXID*600MG 10 CPR RIV
         35375    80105  104025168  ZYVOXID*OS GRATxSOSP 100MG/5ML 240 ML
```

```
63021    80908   60004871            ÃŸ-ESTRADIOL 1 vl E1132-1VL
52071    80908   21200742   ÃŒ ANTIGEN ALEXA FLUOR 488 082IM99514

       codice_regionale  Char_Count
84358         035410048          27
312158        035410226          25
35375         035410075          37
63021               NaN          26
52071               NaN          36
```

Definizione funzioni di utilitÃ

```
In [34]: # Funzione per contare le occorrenze di una stringa
         from collections import Counter


         def search_sub(string, dataframe, col_name):
             string = string.lower()
             subcol = dataframe[
                 dataframe[col_name].str.contains(
                     string, case=False)].copy()
             subcol[col_name] = subcol[col_name].str.lower(
             )
             start_indx = [
                 s.find(string) for s in subcol[col_name]
             ]
             counter = [
                 subcol.iloc[i][col_name][idx:].partition(
                     ' ')[0]
                 for i, idx in enumerate(start_indx)
                 if start_indx[i] != -1
             ]
             return Counter(counter)
```

```
In [35]: # Funzione per rimuovere elementi che contentono una stringa
         def remove_sub(string, dataframe, col_name):
             string = string.lower()
             subcol = dataframe[
                 dataframe[col_name].str.contains(
                     string, case=False)].copy()
             toRemove = subcol.shape[0]
             dataframe.drop(
```

```
                index=subcol.index, inplace=True)
            print(
                "Ho rimosso {} elementi con il pattern '{}' (dopo la rimozione r
                .format(toRemove, string,
                        dataframe.shape[0]))
```

Gestione righe che contengono esaur*

```
In [36]: search_sub("esaur", all_data, "DESCRIZIONE")

Out[36]: Counter({'esaurim': 23,
                  'esaurimento': 113,
                  'esaur': 12,
                  'esaursavonge': 1,
                  'esaurimento)': 1,
                  'esaurito!!': 1,
                  'esaurimeto': 5,
                  'esauriemnto': 1,
                  'esaurinmento': 1,
                  'esauriti': 2,
                  'esaurito': 1,
                  'esaurite': 1})

In [37]: remove_sub("esaur", all_data, "DESCRIZIONE")

Ho rimosso 162 elementi con il pattern 'esaur' (dopo la rimozione ne rimangono 30
```

Gestione righe che contengono sostituit*

```
In [38]: search_sub("sost", all_data, "DESCRIZIONE")

Out[38]: Counter({'sost': 905,
                  'sostit': 14,
                  'sostituzione': 70,
                  'sostituz': 8,
                  'sostegno': 89,
                  'sostanze': 31,
                  'sostituisce': 28,
                  'sostitu': 1,
                  'sostituibilitÃ ': 4,
                  'sostituto': 38,
```

```
                    'sostanza': 4,
                    'sostituibile': 1,
                    'sostigmine': 3,
                    'sostru': 1,
                    'sostitutiva': 5,
                    "sostituira'": 3,
                    'sostanzepsicotrope': 1,
                    'sostar': 1,
                    'sost421822': 1,
                    'sostegnocod': 1,
                    'sostig)1mg/ml': 1,
                    'sostcortsurr': 1,
                    'sostituire': 2,
                    'sostitutivo': 4,
                    'soste': 2,
                    'sostiuito': 1,
                    'sostituito': 38,
                    'sostituta': 1,
                    'sosta': 2,
                    'sost83101101)': 1,
                    'sostanze/prep': 1,
                    'sosta"': 1,
                    'sostigmina': 3,
                    'sostuisce': 1,
                    'sostituzio': 1,
                    'sostitutivi': 1,
                    'sostegni': 1,
                    'sostiutito': 1,
                    'sostcompres': 1,
                    'sostruttore': 1})
```

```
In [39]: remove_sub("sost", all_data, "DESCRIZIONE")
```

```
Ho rimosso 1274 elementi con il pattern 'sost' (dopo la rimozione ne rimangono 30
```

5.1.6 Ri-indicizzazione delle righe

Per evitare di avere situazioni del tipo:

-	val
0	a
1	b
3	d

Dove l'id 2 non esiste perchÃ¨ rimosso in una operazione precedente.

```
In [40]: all_data.reset_index(drop=True, inplace=True)
         assert all_data.shape[0] - 1 == all_data.index[-1]
```

Cancello il non necessario:

```
In [41]: del duplicated
         del cod_red_notna_desc
         del all_duplicate
         del desc2cod
```

5.1.7 Salvo i dati elaborati:

```
In [42]: all_data.to_pickle("edited.pkl")
```

5.2 Creazione modelli Doc2Vec tramite Gensim

5.2.1 Analisi della distribuzione dei dati

```
In [43]: from gensim.models.doc2vec import TaggedDocument, Doc2Vec
         from gensim.test.utils import datapath
         from itertools import combinations, product, permutations
         from sklearn.metrics.pairwise import cosine_similarity
         import nltk
```

```
C:\ProgramData\Anaconda3\lib\site-packages\gensim\utils.py:1209: UserWarning: det
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

Creo dizionari e liste che serviranno per le prossime elaborazioni:

```
In [44]: cod2count = {}
         cod2list = {}

         # aggrego per numero di codici regionali
```

```
        for i in tqdm_notebook(range(all_data.shape[0])):
            key = all_data.at[i, "codice_regionale"]
            if not pd.isnull(key):
                # Compilo il contatore codice -> elementi
                if key in cod2count:
                    cod2count[key] += 1
                else:
                    cod2count[key] = 1

                # Compilo il dizionario codice -> lista di indici degli element
                if key in cod2list:
                    cod2list[key].append(i)
                else:
                    cod2list[key] = [i]
```

```
HBox(children=(IntProgress(value=0, max=308523), HTML(value='')))
```

```
In [45]: countGruped = {
         }  # Dizionario "numero di elementi con quel codice regionale" -> "numer
         listGrouped = []

         for key in cod2count.keys():
             count = cod2count[key]
             if count == 1:
                 listGrouped.append(key)
             if count in countGruped:
                 countGruped[count] += 1
             else:
                 countGruped[count] = 1
```

Fatto ciÃ², possiamo farci un'idea di come siano organizzati i codici regionali:

```
In [46]: countGruped

Out[46]: {6: 499,
          2: 1637,
          1: 3610,
          3: 1262,
```

```
4: 911,
7: 351,
8: 223,
5: 683,
9: 59,
10: 12,
13: 4,
15: 1,
19: 1,
12: 4,
11: 2,
29: 1,
17: 1,
25: 1}
```

abbiamo svariate coppie di codici che non sono coppie, poichÃ¨ composte solo da 1 elemento.

Infatti, se ordinate per codice ci si accorge che anche nel file originale non preprocessato, sono tutte singole:

```
In [47]: accop[accop.codice_regionale.isin(
             listGrouped)].sort_values(
                 by="codice_regionale").head()
```

```
Out[47]:    AZIENDA     CODICE  \
         0   080106     41.520
         21  080101     330631
         36    AVEN     139354
         37  080105  104032999
         45  080904  0001GLUA1

                                                                DESCRIZIONE  \
         0                                          EPHYNAL*300MG 30CPS MOLLI
         21                                  RINAZINA*AD GTT 10ML 10MG 0,1%
         36                                EURAX CREMA G.20           TUBI
         37                                   ADISTEROLO*IM OS 2F 1ML
         45  GLUCANTIME 5ML G 1,5 FL (FRANCESE) V - ANTIMONIATO DI MEGLUMINA

            codice_regionale
         0         000053037
         21        000590012
         36        001578018
```

```
          37          001738020
          45          001801012

In [48]: del accop
```

5.2.2 Tokenizzazione

CioÃ¨ creare il testo da poter utilizzare con i modelli Doc2Vec. Creo anche quello strutturato a trigram:

```
In [49]: def word2trigram(word):
             if len(word) < 3:
                 return [word]
             return [
                 word[i:i + 3]
                 for i in range(len(word) - 2)
             ]


         def make_trigram(sentence):
             tr = []
             for w in sentence.split():
                 tr.extend(word2trigram(w))
             return (" ".join(tr))

In [50]: [x for x in all_data.DESCRIZIONE[:10]]

Out[50]: ['ZOLETIL 50+50 MG/ML F U/VET',
          'TERRAMICINA SPRAY ML 150 FLAC U/VET',
          'STRESNIL ML 100 MG 40 FLAC U/VET',
          'DOMOSEDAN ML 5 FLA U/VET',
          'DEXADRESON FORTE SOSPEN ML 50 U/VET',
          'COVINAN INIETT ML 20 FLAC',
          'BAYTRIL SOLUZ 2,5% ML 50 FLAC U/VET',
          'TANAX ML 50 FLAC U/VET',
          'DOMITOR ML 10 X 1 FLA U/VET',
          'ANTISEDAN ML 10 X 1 FLA U/VET']

In [51]: train_corpus = []

         for i in tqdm_notebook(range(all_data.shape[0])):
             desc = all_data.at[i, "DESCRIZIONE"]
```

```
            train_corpus.append(
                TaggedDocument(
                    nltk.tokenize.word_tokenize(
                        desc.lower()), [i]))
```

```
HBox(children=(IntProgress(value=0, max=308523), HTML(value='')))
```

```
In [52]: train_corpus_trigram = []

         for i in tqdm_notebook(range(all_data.shape[0])):
             desc = make_trigram(
                 all_data.at[i, "DESCRIZIONE"])
             train_corpus_trigram.append(
                 TaggedDocument(
                     nltk.tokenize.word_tokenize(
                         desc.lower()), [i]))
```

```
HBox(children=(IntProgress(value=0, max=308523), HTML(value='')))
```

```
In [53]: train_corpus[:10]
```

```
Out[53]: [TaggedDocument(words=['zoletil', '50+50', 'mg/ml', 'f', 'u/vet'], tags=
          TaggedDocument(words=['terramicina', 'spray', 'ml', '150', 'flac', 'u/v
          TaggedDocument(words=['stresnil', 'ml', '100', 'mg', '40', 'flac', 'u/v
          TaggedDocument(words=['domosedan', 'ml', '5', 'fla', 'u/vet'], tags=[3]
          TaggedDocument(words=['dexadreson', 'forte', 'sospen', 'ml', '50', 'u/v
          TaggedDocument(words=['covinan', 'iniett', 'ml', '20', 'flac'], tags=[5
          TaggedDocument(words=['baytril', 'soluz', '2,5', '%', 'ml', '50', 'flac
          TaggedDocument(words=['tanax', 'ml', '50', 'flac', 'u/vet'], tags=[7]),
          TaggedDocument(words=['domitor', 'ml', '10', 'x', '1', 'fla', 'u/vet'],
          TaggedDocument(words=['antisedan', 'ml', '10', 'x', '1', 'fla', 'u/vet'
```

```
In [54]: train_corpus_trigram[:10]
```

```
Out[54]: [TaggedDocument(words=['zol', 'ole', 'let', 'eti', 'til', '50+', '0+5',
          TaggedDocument(words=['ter', 'err', 'rra', 'ram', 'ami', 'mic', 'ici',
          TaggedDocument(words=['str', 'tre', 'res', 'esn', 'sni', 'nil', 'ml',
          TaggedDocument(words=['dom', 'omo', 'mos', 'ose', 'sed', 'eda', 'dan',
          TaggedDocument(words=['dex', 'exa', 'xad', 'adr', 'dre', 'res', 'eso',
          TaggedDocument(words=['cov', 'ovi', 'vin', 'ina', 'nan', 'ini', 'nie',
          TaggedDocument(words=['bay', 'ayt', 'ytr', 'tri', 'ril', 'sol', 'olu',
          TaggedDocument(words=['tan', 'ana', 'nax', 'ml', '50', 'fla', 'lac', 'u
          TaggedDocument(words=['dom', 'omi', 'mit', 'ito', 'tor', 'ml', '10', 'x
          TaggedDocument(words=['ant', 'nti', 'tis', 'ise', 'sed', 'eda', 'dan',
```

5.2.3 Definizione funzioni per calcolo degli errori

```
In [55]: def cos_array(index, model):
             return np.array(model.docvecs[index]).reshape(
                 1, -1)


         def calc_cosine(model):
             cos_sim = []
             for cod in cod2list.keys():
                 listOfIndex = cod2list[cod]
                 count = len(listOfIndex)
                 if count >= 2:
                     num_comb = combinations(
                         listOfIndex, 2)
                     for com_tuple in num_comb:
                         x = cos_array(com_tuple[0], model)
                         y = cos_array(com_tuple[1], model)
                         cos_sim.append(
                             (1 + cosine_similarity(x, y))
                             / 2)
             return np.array(
                 cos_sim)  # Per normalizzare in 0-1


         def calc_position(model):
             list_sequences = [
                 seq_list
                 for seq_list in cod2list.values()
                 if len(seq_list) >= 2
```

```
            ]
            mAP_array = []
            rPrecision_array = []
            for i in tqdm_notebook(
                    range(len(list_sequences))):
                seq = list_sequences[i]
                target = seq[0]
                sims = model.docvecs.most_similar(
                    [model.docvecs[target]],
                    topn=len(seq))
                df = pd.DataFrame(
                    sims, columns=['id', 'val'])
                df.set_index(keys="id", inplace=True)
                mAP = 0
                finded = 0
                for i in range(1, len(seq)):
                    if df.index[i] in seq:
                        finded += 1
                        mAP += (finded / i)
                mAP_array.append(
                    mAP / (finded if finded > 0 else 1))
                rPrecision_array.append(
                    finded / (len(seq) - 1))
            return np.array(mAP_array), np.array(
                rPrecision_array)
```

5.2.4 Creazione modelli

Definiamo e creiamo tutti i modelli che dovranno essere testati, e il dataframe che dovrÃ contenerli:

```
In [56]: results = pd.DataFrame(columns=[
             'vector_size', 'min_count', 'window_size',
             'epoch', 'is_trigram', 'med_cosine_dist',
             'cosine_dev_std', 'med_mAP', 'mAP_std',
             'rPrec_mean', 'rPrec_dev_std'
         ])
         results

Out[56]: Empty DataFrame
         Columns: [vector_size, min_count, window_size, epoch, is_trigram, med_co
         Index: []
```

```
In [57]: def train_model(vector_size, min_count, epoch,
                         window_size):
             model_name = "vs-{}_mc-{}_ep-{}_ws-{}_trigram.doc2vec".format(
                 vector_size, min_count, epoch,
                 window_size)
             model = Doc2Vec(
                 vector_size=vector_size,
                 min_count=min_count,
                 epochs=epoch,
                 window=window_size)
             model.build_vocab(train_corpus_trigram)
             for i in tqdm_notebook(range(epoch)):
                 model.train(
                     train_corpus_trigram,
                     total_examples=model.corpus_count,
                     epochs=1)
             model.save(model_name)
             cos = calc_cosine(model)
             pos, rPrec = calc_position(model)
             return {
                 'vector_size': [vector_size],
                 'min_count': [min_count],
                 'window_size': [window_size],
                 'epoch': [epoch],
                 'med_cosine_dist': [cos.mean()],
                 'cosine_dev_std': [cos.std(ddof=1)],
                 'med_pos': [pos.mean()],
                 'dev_std': [pos.std(ddof=1)],
                 'r_precision_mean': [rPrec.mean()]
             }

In [58]: def load_model(vector_size, min_count, epoch,
                        trigram, window_size, model_name):
             model = Doc2Vec.load(model_name)
             cos = calc_cosine(model)
             mAP, rPrec = calc_position(model)
             return {
                 'vector_size': [vector_size],
                 'min_count': [min_count],
                 'window_size': [window_size],
                 'epoch': [epoch],
                 'is_trigram':
```

```
                ("no" if trigram == '' else "yes"),
                'med_cosine_dist': [cos.mean()],
                'cosine_dev_std': [cos.std(ddof=1)],
                'med_mAP': [mAP.mean()],
                'mAP_std': [mAP.std(ddof=1)],
                'rPrec_mean': [rPrec.mean()],
                'rPrec_dev_std': [rPrec.std(ddof=1)]
            }
```

Eseguo tutte le misurazioni tra i vari modelli e le salvo in un file csv per la comparazione:

```
In [ ]: vector_size = [50, 100]
        min_count = [0, 5, 10]
        epoch = [100, 300]
        trigram = ['', '_trigram']

        listOfModel = [
            mod for mod in product(vector_size, min_count,
                                   epoch, trigram)
        ]

        for i in tqdm_notebook(range(len(listOfModel))):
            setting = listOfModel[i]
            model_name = "vs-{}_mc-{}_ep-{}{}.doc2vec".format(
                setting[0], setting[1], setting[2],
                setting[3])
            toAdd = load_model(setting[0], setting[1],
                               setting[2], setting[3], 5,
                               model_name)
            results = results.append(pd.DataFrame(toAdd))

        epoch = [100]
        window_size = [15, 20]

        listOfModel = [
            mod for mod in product(vector_size, min_count,
                                   epoch, window_size)
        ]

        for i in tqdm_notebook(range(len(listOfModel))):
            setting = listOfModel[i]
```

```
            model_name = "vs-{}_mc-{}_ep-{}_ws-{}_trigram.doc2vec".format(
                setting[0], setting[1], setting[2],
                setting[3])
            toAdd = load_model(setting[0], setting[1],
                               setting[2], 'trigram',
                               setting[3], model_name)
            results = results.append(pd.DataFrame(toAdd))

        results.to_csv("results.csv")
```

5.3 Generazione delle coppie

```
In [37]: import numpy as np
         import pandas as pd
         from tqdm import tqdm_notebook
         from scipy.spatial.distance import cosine
         from scipy import sparse
         import scipy as sp
         from sklearn.metrics.pairwise import cosine_similarity
         from sklearn.feature_extraction.text import CountVectorizer
         from sklearn.model_selection import train_test_split

         import gc
         from secrets import randbelow

         COS_SIM_THRESHOLD = 0.83
         NUM_VERY_NEG = 22000

         all_data = pd.read_pickle("edited.pkl")
         all_data.DESCRIZIONE = all_data.DESCRIZIONE.str.lower(
         )
         assert all_data.shape[0] - 1 == all_data.tail(
             1).index[0]
```

```
In [2]: all_data.head()
```

```
Out[2]:   AZIENDA  CODICE                          DESCRIZIONE codice_regionale
        0    AVEN  120003          zoletil 50+50 mg/ml f u/vet        101580025
        1    AVEN  120004  terramicina spray ml 150 flac u/vet        100156013
```

```
2    AVEN  120020     stresnil ml 100 mg 40 flac u/vet        101294015
3    AVEN  120021              domosedan ml 5 fla u/vet        100102019
4    AVEN  120026  dexadreson forte sospen ml 50 u/vet        101867012

   Char_Count
0          27
1          35
2          32
3          24
4          35
```

Ricopio alcuni strumenti utili:

Creo dizionari e liste che serviranno per le prossime elaborazioni:

```
In [3]: cod2count = {}
        cod2list = {}

        # aggrego per numero di codici regionali
        for i in tqdm_notebook(range(all_data.shape[0])):
            key = all_data.at[i, "codice_regionale"]
            if not pd.isnull(key):
                # Compilo il contatore codice -> elementi
                if key in cod2count:
                    cod2count[key] += 1
                else:
                    cod2count[key] = 1

                # Compilo il dizionario codice -> lista di indici degli elementi
                if key in cod2list:
                    cod2list[key].append(i)
                else:
                    cod2list[key] = [i]

HBox(children=(IntProgress(value=0, max=308523), HTML(value='')))
```

```
In [4]: countGruped = {
        }  # Dizionario "numero di elementi con quel codice regionale" -> "numero
```

```
listGrouped = []

for key in cod2count.keys():
    count = cod2count[key]
    if count == 1:
        listGrouped.append(key)
    if count in countGruped:
        countGruped[count] += 1
    else:
        countGruped[count] = 1
```

Fatto ciÃ², possiamo farci un'idea di come siano organizzati i codici regionali:

```
In [5]: countGruped

Out[5]: {6: 499,
         2: 1637,
         1: 3610,
         3: 1262,
         4: 911,
         7: 351,
         8: 223,
         5: 683,
         9: 59,
         10: 12,
         13: 4,
         15: 1,
         19: 1,
         12: 4,
         11: 2,
         29: 1,
         17: 1,
         25: 1}

In [6]: listGrouped[:10]

Out[6]: ['100102019',
         '101867012',
         '101904023',
         '101743033',
         '904359989',
         '900185075',
```

```
'900185048',
'903009633',
'905022240',
'900161403']
```

5.4 Creazione dei dati accoppiati

Valori molto simili ma diversi

```
In [7]: vect = CountVectorizer(min_df=5)
```

```
In [8]: def word2trigram(word):
            if len(word) < 3:
                return [word]
            return [
                word[i:i + 3]
                for i in range(len(word) - 2)
            ]


        def make_trigram(sentence):
            tr = []
            for w in sentence.split():
                tr.extend(word2trigram(w))
            return (" ".join(tr))
```

```
In [9]: all_data["trigram"] = all_data.DESCRIZIONE.apply(
            make_trigram)
        all_data.head()
```

```
Out[9]:   AZIENDA  CODICE                          DESCRIZIONE codice_regionale
        0    AVEN  120003          zoletil 50+50 mg/ml f u/vet        101580025
        1    AVEN  120004  terramicina spray ml 150 flac u/vet        100156013
        2    AVEN  120020     stresnil ml 100 mg 40 flac u/vet        101294015
        3    AVEN  120021              domosedan ml 5 fla u/vet        100102019
        4    AVEN  120026  dexadreson forte sospen ml 50 u/vet        101867012

           Char_Count                                              trigram
        0          27  zol ole let eti til 50+ 0+5 +50 mg/ g/m /ml f ...
        1          35  ter err rra ram ami mic ici cin ina spr pra ra...
```

```
        2          32  str tre res esn sni nil ml 100 mg 40 fla lac u...
        3          24   dom omo mos ose sed eda dan ml 5 fla u/v /ve vet
        4          35  dex exa xad adr dre res eso son for ort rte so...

In [10]: base = all_data[all_data.codice_regionale.notna()]

In [11]: dtm = vect.fit_transform(base.trigram)

In [12]: rr = dtm.astype(np.int8)
         arraydtm = rr.toarray()
         arraydtm[:10]

Out[12]: array([[0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0],
                ...,
                [0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0]], dtype=int8)

In [13]: arraydtm.shape

Out[13]: (25842, 5346)

In [14]: m = sparse.csr_matrix(arraydtm[:NUM_VERY_NEG])
         t = m.T

In [15]: m = m.dot(t)

In [16]: norm = sp.sqrt(m.multiply(m).sum(axis=0))

Wall time: 3.93 s


In [17]: toDivide = sp.sqrt(t.multiply(t).sum(axis=0))

In [18]: target = m.multiply(1 / toDivide).multiply(
             1 / toDivide.T)

In [19]: data = target.data
         col = target.col
         row = target.row
```

```
In [38]: trueNeg = []

         for i in tqdm_notebook(range(len(col))):
             if (col[i] != row[i]) & (data[i] >=
                                      COS_SIM_THRESHOLD):
                 trueNeg.append((base.iloc[col[i]].name,
                                 base.iloc[row[i]].name))
```

```
HBox(children=(IntProgress(value=0, max=175850194), HTML(value='')))
```

```
In [39]: len(trueNeg)
```

```
Out[39]: 30108
```

```
In [40]: veryNegative = []
         for i in tqdm_notebook(range(len(trueNeg))):
             tup = trueNeg[i]
             if base.at[tup[0],
                        "codice_regionale"] != base.at[
                            tup[1], "codice_regionale"]:
                 veryNegative.append(tup)
```

```
HBox(children=(IntProgress(value=0, max=30108), HTML(value='')))
```

```
In [41]: len(veryNegative)
```

```
Out[41]: 20572
```

Valori Positivi

```
In [42]: from math import factorial


         def comb(r, n):
             return factorial(n) / (
                 factorial(r) * factorial(n - r))


         counter = 0

         for length in countGruped.keys():
             if length > 1:
                 a = comb(2, length)
                 counter += (a * countGruped[length]) * 2

         print(
             'Possibili combinazioni positive (anche invertite): {}'
             .format(counter))
```

```
Possibili combinazioni positive (anche invertite): 86574.0
```

```
In [43]: from itertools import permutations

         positive = []

         for cod_reg in cod2list.keys():
             if cod2count[cod_reg] > 1:
                 positive.extend([
                     comb for comb in permutations(
                         cod2list[cod_reg], 2)
                 ])

         positive[:10]
```

```
Out[43]: [(0, 84119),
          (0, 85031),
          (0, 171171),
          (0, 239742),
          (0, 298458),
```

```
          (84119, 0),
          (84119, 85031),
          (84119, 171171),
          (84119, 239742),
          (84119, 298458)]
```

```
In [44]: len(positive)
```

```
Out[44]: 86574
```

Valori negativi (generazione randomica)

```
In [45]: negative = []

         while len(negative) < (86574 - len(veryNegative)):
             a = randbelow(base.shape[0])
             b = randbelow(base.shape[0])

             a_txt = base.iloc[a]
             b_txt = base.iloc[b]
             x = (a_txt.name, b_txt.name)
             if (a_txt["codice_regionale"] !=
                     b_txt["codice_regionale"]) & (
                         x not in negative) & (
                             x not in veryNegative):
                 negative.append(x)
                 if not (len(negative) % 5000):
                     print("{} num".format(len(negative)))

         print('Done. {} pairs created.'.format(
             len(negative)))
```

```
5000 num
10000 num
15000 num
20000 num
25000 num
30000 num
35000 num
40000 num
```

```
45000 num
50000 num
55000 num
60000 num
65000 num
Done. 66002 pairs created.
Wall time: 4min
```

Creazione dataset

```
In [46]: def id2str(id):
             return all_data.iloc[id].DESCRIZIONE


         def appendTuple(t, val, datas):
             datas.append([
                 id2str(t[0]) + " @@@ " + id2str(t[1]), val
             ])


         pos = []
         neg = []
         ver = []

         # Positivi
         for i in tqdm_notebook(range(len(positive))):
             appendTuple(positive[i], 1, pos)

         # Negativi
         for i in tqdm_notebook(range(len(negative))):
             appendTuple(negative[i], 0, neg)

         ## Alta similaritÃ  coseno
         for i in tqdm_notebook(range(len(veryNegative))):
             appendTuple(veryNegative[i], 0, ver)

HBox(children=(IntProgress(value=0, max=86574), HTML(value='')))
```

```
HBox(children=(IntProgress(value=0, max=66002), HTML(value='')))
```

```
HBox(children=(IntProgress(value=0, max=20572), HTML(value='')))
```

```
In [47]: df_pos = pd.DataFrame(
             pos, columns=['string', 'equal'])
         df_neg = pd.DataFrame(
             neg, columns=['string', 'equal'])
         df_ver = pd.DataFrame(
             ver, columns=['string', 'equal'])
```

```
In [52]: df_pos.to_pickle("df_hard_pos.pkl")
         df_neg.to_pickle("df_hard_neg.pkl")
         df_ver.to_pickle("df_hard_ver.pkl")
```

```
In [48]: pd.set_option('display.max_colwidth', 150)

         df_pos.head()
```

Out[48]:

	string	equal
0	zoletil 50+50 mg/ml f u/vet @@@ zoletil*100 1f...	1
1	zoletil 50+50 mg/ml f u/vet @@@ zoletil 100*1f...	1
2	zoletil 50+50 mg/ml f u/vet @@@ zoletil 50+50 ...	1
3	zoletil 50+50 mg/ml f u/vet @@@ zoletil 50+50m...	1
4	zoletil 50+50 mg/ml f u/vet @@@ veter zoletil*...	1

```
In [49]: df_neg.head()
```

Out[49]:

	string	equal
0	oxis turbohaler 4,5*polv 60d(new) @@@ humalog ...	0
1	montegen 4mg bs @@@ en*iniet 3f 1ml 0,5mg/ml	0
2	enbrel*bb sc 4fl10mg+4fl1ml+8t(new) @@@ polioi...	0
3	ossigeno*compr 200bar 7lt acc valv rid @@@ nad...	0
4	minocin 100 mg x8cps ptr @@@ sonda nelaton ch1...	0

```
In [51]: df_ver.head()
```

```
Out[51]:
         0                          baytril sol 5% ml 50 flac u/vet @@@ baytril sol
         1              sonda ureterostomia cut ch10 ac68 @@@ sonda ureterostomia
         2  sonda ureterostomia cut ch14 ac68 ars chirur @@@ sonda ureterostomia
         3              sonda ureterostomia cut ch12 ac68 ars chirur @@@ sonda ure
         4              sonda ureterostomia cut ch14 ac68 ars chirur @@@ sonda ure

            equal
         0      0
         1      0
         2      0
         3      0
         4      0
```

5.4.1 Generazione Negative con distribuzione = positive

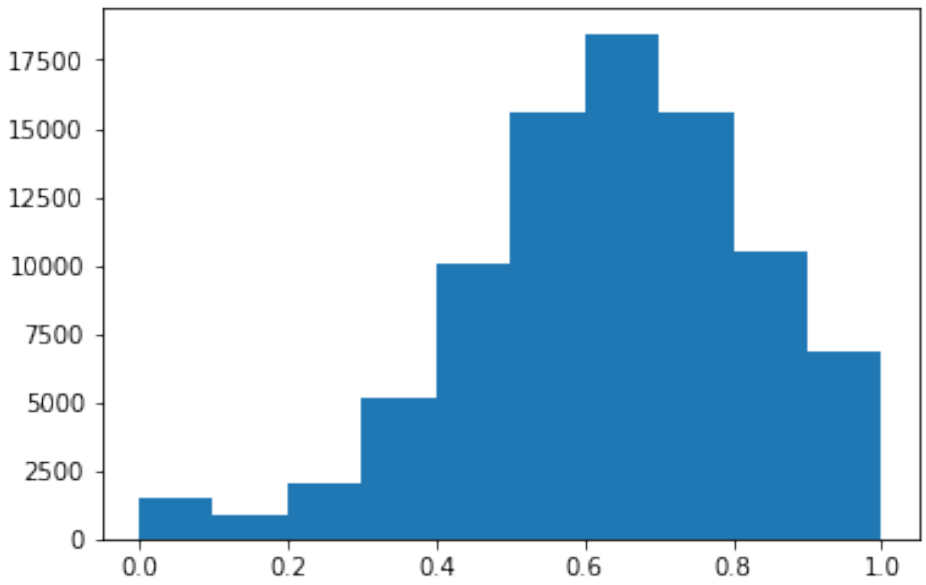
Calcolo la media della similaritÃ coseno delle positive:

```
In [24]: np.array(positive_sim).mean()
```

```
Out[24]: 0.631067966719712
```

```
In [25]: hist(positive_sim)
```

```
Out[25]: (array([ 1506.,   848.,  2006.,  5138., 10072., 15602., 18458., 15538.,
                 10528.,  6878.]),
          array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ]),
          <a list of 10 Patch objects>)
```



Valori molto simili ma diversi

Utilizziamo la distribuzione vista nel grafico precedente per selezionare coppie negative.

Avremo quindi una quantitÃ di coppie prevalentemente nell'intervallo 0.6-0.7.

```
In [26]: neg_dict = dict()
         neg_count = dict()
         valueCount = [
             1506, 848, 2006, 5138, 10072, 15602, 18458,
             15538, 10528, 6878
         ]
         baseVal = 0

         first5 = 0
         last5 = 0

         for v in valueCount:
             c = round(baseVal, 1)
```

```
            neg_dict[c] = v
            neg_count[c] = 0
            baseVal += 1
            if c < 5:
                first5 += v
            else:
                last5 += v
```

```
In [27]: neg_dict
```

```
Out[27]: {0: 1506,
          1: 848,
          2: 2006,
          3: 5138,
          4: 10072,
          5: 15602,
          6: 18458,
          7: 15538,
          8: 10528,
          9: 6878}
```

```
In [28]: neg_count
```

```
Out[28]: {0: 0, 1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 0}
```

```
In [29]: print(
             'Nell\'intervallo [0.0-0.49] ricadono {} numeri, da [0.5 - 0.99] inv
             .format(first5, last5))
```

```
Nell'intervallo [0.0-0.49] ricadono 19570 numeri, da [0.5 - 0.99] invece 67004
```

```
In [ ]: negative = []

        #Riempio la lista con i valori da 0.0 a 0.5 [impiega circa 42 secondi, fe

        for i in tqdm_notebook(range(len(col))):
            if (len(negative) >= first5):
                break
            if (col[i] != row[i]):
                sc = int(str(data[i] * 10)[0])
                if sc < 5:
```

```
                    if neg_count[sc] < neg_dict[sc]:
                        idx1 = base.iloc[col[i]].name
                        idx2 = base.iloc[row[i]].name
                        if base.at[
                                idx1,
                                "codice_regionale"] != base.at[
                                    idx2,
                                    "codice_regionale"]:
                            negative.append((idx1, idx2))
                            neg_count[sc] += 1
```

```
In [31]: len(negative)
```

```
Out[31]: 19570
```

```
In [32]: neg_count
```

```
Out[32]: {0: 1506, 1: 848, 2: 2006, 3: 5138, 4: 10072, 5: 0, 6: 0, 7: 0, 8: 0, 9:
```

```
In [33]: neg_dict
```

```
Out[33]: {0: 1506,
          1: 848,
          2: 2006,
          3: 5138,
          4: 10072,
          5: 15602,
          6: 18458,
          7: 15538,
          8: 10528,
          9: 6878}
```

```
In [34]: # Aggiungo i rimanenti, potendo scremare la maggior parte dei valori e v
         #Si ferma a circa al 94%, 165283760]

         for i in tqdm_notebook(range(len(col))):
             if (len(negative) >= len(positive)):
                 break
             if (col[i] != row[i]) & (data[i] >= 0.5):
                 if (data[i] != 1.0):
                     sc = int(str(data[i] * 10)[0])
                     if (neg_count[sc] < neg_dict[sc]):
```

```
                            idx1 = base.iloc[col[i]].name
                            idx2 = base.iloc[row[i]].name
                            if base.at[
                                    idx1,
                                    "codice_regionale"] != base.at[
                                        idx2,
                                        "codice_regionale"]:
                                negative.append((idx1, idx2))
                                neg_count[sc] += 1
```

```
HBox(children=(IntProgress(value=0, max=175850194), HTML(value='')))
```

```
In [35]: len(negative)

Out[35]: 86574

In [36]: neg_count

Out[36]: {0: 1506,
          1: 848,
          2: 2006,
          3: 5138,
          4: 10072,
          5: 15602,
          6: 18458,
          7: 15538,
          8: 10528,
          9: 6878}

In [37]: neg_dict

Out[37]: {0: 1506,
          1: 848,
          2: 2006,
          3: 5138,
          4: 10072,
          5: 15602,
          6: 18458,
          7: 15538,
          8: 10528,
          9: 6878}
```

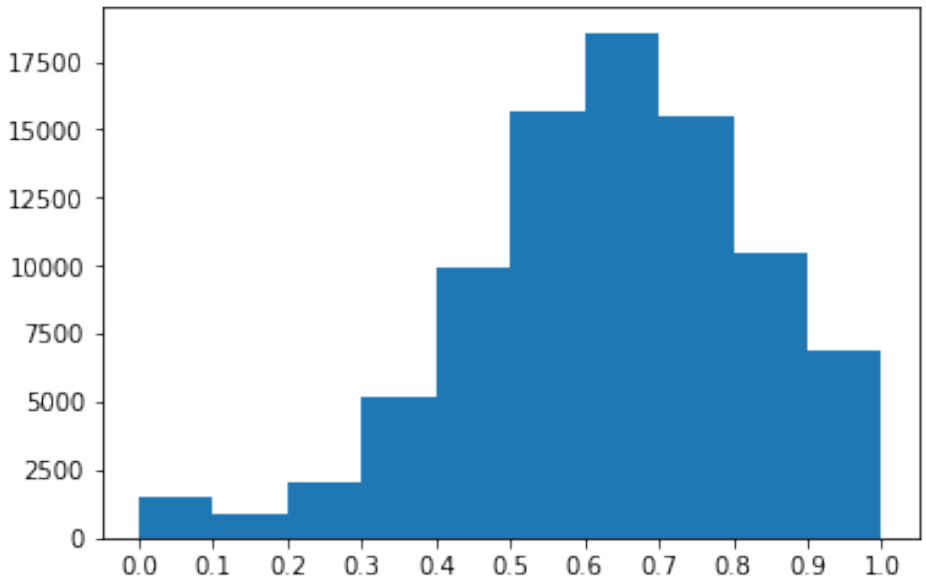
```
In [38]: negative_sim = []

         for i in tqdm_notebook(range(len(negative))):
             t = negative[i]
             negative_sim.append(1 - cosine(
                 arraydtm[base.at[t[0], "id2rowCount"], :],
                 arraydtm[base.at[t[1], "id2rowCount"], :])
                                 )
```

```
HBox(children=(IntProgress(value=0, max=86574), HTML(value='')))
```

```
In [39]: np.array(negative_sim).mean()
```

```
Out[39]: 0.6237155995150063
```

```
In [40]: plt.hist(negative_sim, np.arange(0.0, 1.1, 0.1))
         plt.xticks(np.arange(0.0, 1.1, 0.1))
         plt.show()
```



5.5 Rete neurale CharCNN

```
In [1]: import sys, os, re, csv, codecs, numpy as np, pandas as pd
        import matplotlib.pyplot as plt
        from keras.preprocessing.text import Tokenizer
        from keras.preprocessing.sequence import pad_sequences
        from keras.layers import Dense, Input, LSTM, Embedding, Dropout, Activati
        from keras.layers import Bidirectional, GlobalMaxPool1D, Bidirectional
        from keras.models import Model
        from keras import initializers, regularizers, constraints, optimizers, la
        from keras.callbacks import EarlyStopping, ModelCheckpoint
        import gc
        from sklearn.model_selection import train_test_split
        from sklearn.utils import shuffle
        from keras.models import load_model
```

```
Using TensorFlow backend.
```

```
In [2]: train_pos = pd.read_pickle('df_pos.pkl')
        train_pos.head()
```

Out[2]:

	string	equal
0	zoletil 50+50 mg/ml f u/vet @@@ zoletil*100 1f...	1
1	zoletil 50+50 mg/ml f u/vet @@@ zoletil 100*1f...	1
2	zoletil 50+50 mg/ml f u/vet @@@ zoletil 50+50 ...	1
3	zoletil 50+50 mg/ml f u/vet @@@ zoletil 50+50m...	1
4	zoletil 50+50 mg/ml f u/vet @@@ veter zoletil*...	1

```
In [3]: train_neg = pd.read_pickle('df_neg.pkl')
        train_neg.head()
```

Out[3]:

	string	equal
0	terramicina spray ml 150 flac u/vet @@@ zoleti...	0
1	stresnil ml 100 mg 40 flac u/vet @@@ zoletil 5...	0
2	domosedan ml 5 fla u/vet @@@ zoletil 50+50 mg/...	0
3	dexadreson forte sospen ml 50 u/vet @@@ zoleti...	0
4	covinan iniett ml 20 flac @@@ zoletil 50+50 mg...	0

5.5.1 Divisione in training e validation set:

```
In [4]: X_train_pos, X_test_pos, y_train_pos, y_test_pos = train_test_split(
            train_pos,
```

```
            train_pos[["equal"]],
            test_size=0.30,
            random_state=42)
        X_train_neg, X_test_neg, y_train_neg, y_test_neg = train_test_split(
            train_neg,
            train_neg[["equal"]],
            test_size=0.30,
            random_state=42)
```

```
In [5]: X_train = X_train_pos.append(X_train_neg)
        X_test = X_test_pos.append(X_test_neg)
        y_train = y_train_pos.append(y_train_neg)
        y_test = y_test_pos.append(y_test_neg)
```

```
In [6]: X_test.shape
```

```
Out[6]: (51946, 2)
```

Si separano tutte le descrizioni accoppiate dei prodotti:

```
In [7]: list_sentences_train = X_train["string"]
        list_sentences_test = X_test["string"]
```

Impostiamo un tokenizzatore a livello di carattere:

```
In [8]: max_features = 20000
        tokenizer = Tokenizer(
            num_words=max_features, char_level=True)
```

Gli diamo in input la lista dei prodotti di training:

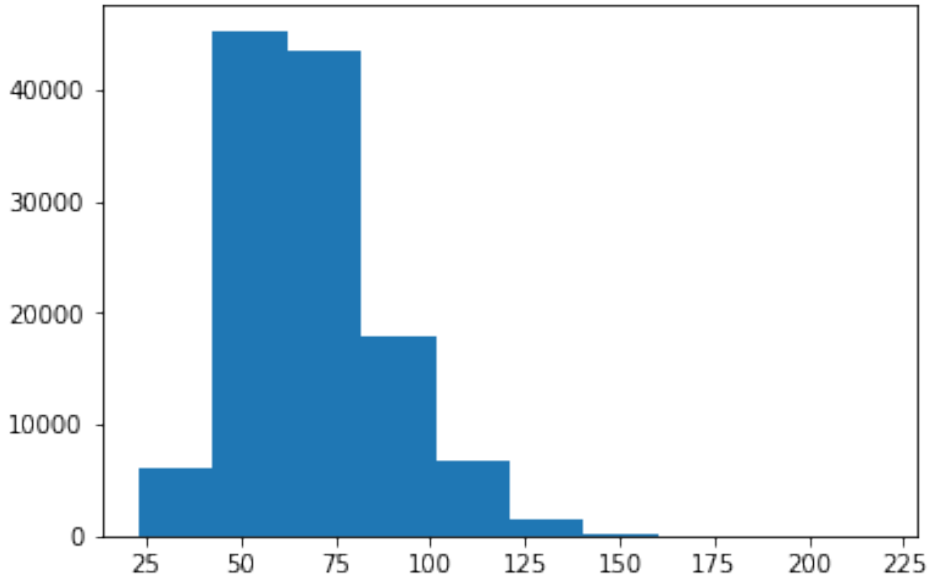
```
In [9]: tokenizer.fit_on_texts(list(list_sentences_train))
```

E poi tokenizziamo la lista di descrizioni di training e validation:

```
In [10]: list_tokenized_train = tokenizer.texts_to_sequences(
             list_sentences_train)
         list_sentences_test = tokenizer.texts_to_sequences(
             list_sentences_test)
```

Controlliamo la dimensione media della lista di caratteri per poterla utilizzare come livello iniziale in keras:

```
In [11]: totalNumWords = [
             len(one_comment)
             for one_comment in list_tokenized_train
         ]
         plt.hist(totalNumWords)
         plt.show()
```



La maggior parte rimane sotto i 100 caratteri, quindi useremo 100:

```
In [12]: maxlen = 100
         X_t = pad_sequences(
             list_tokenized_train, maxlen=maxlen)
         X_te = pad_sequences(
             list_sentences_test, maxlen=maxlen)
```

Conclusioni e sviluppi futuri

Il problema iniziale richiedeva l'utilizzo di un algoritmo abbastanza intelligente da poter accoppiare brevi testi etichettati come *simili*.

Dopo aver utilizzato e testato entrambe le tecnologie disponibili nel campo del deep learning (applicato al natural language processing) si è giunti alla conclusione che Doc2Vec, l'algoritmo non supervisionato, fa molta fatica quando i testi non sono semanticamente corretti e logici, e quanto sono presenti molte abbreviazioni.

Al contrario, una rete CharCNN permette di raggiungere livelli di accuratezza molto elevati, anche se necessita di una quantità consistente di esempi etichettati a mano per apprendere a distinguere descrizioni molto complesse.

Inoltre, Doc2Vec è fornito della funzionalità di ricerca degli elementi più vicini, che in una frazione di secondo permette di ottenere la lista dei K elementi più simili: la rete CharCNN non possiede questa funzionalità, potrebbe implementarla calcolando a priori tutte le possibili similarità con tutti i possibili prodotti, ma tempi e risorse computazionali sarebbero direttamente proporzionali al numero di prodotti.

Inoltre, la rete CharCNN addestrata è stata calibrata con una dimensione massima in entrata di 100 caratteri: non è possibile sottoporre stringhe più lunghe di quella dimensione alla rete.

Se si volesse quindi sfruttare per identificare similarità con altri tipi di stringhe, il modello potrebbe troncare le coppie troppo lunghe.

Uno sviluppo futuro di questo progetto potrebbe essere l'evoluzione della rete CharCNN in un autoencoder, associando alle istanze in input le istanze considerate identiche di output. Così facendo, dalla CharCNN addestrata si potrebbe estrarre la matrice dei pesi, da usare per mappare nello spazio ridotto le istanze di input, e successivamente implementare una versione basata su questa matrice di un algoritmo **K-nearest neighbors**.

Bibliografia

[1] Tom M. Mitchell, *Machine Learning*, McGraw Hill, 1997.

[2] Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean, *Efficient Estimation of Word Representations in Vector Space*, 2013.

[3] Xiang Zhang, Junbo Zhao, Yann LeCun. *Character-level Convolutional Networks for Text Classification*, In Proceedings of NIPS, Apr. 2016.

[4] Quoc Le, Tomas Mikolov. *Distributed Representations of Sentences and Documents*, 2014.

[5] Ryan Kiros, Yukun Zhu, Ruslan Salakhutdinov, Richard S. Zemel, Antonio Torralba, Raquel Urtasun, Sanja Fidler. *Skip-Thought Vectors*, In NIPS, 2015.

[6] Antonio Gulli, Sujit Pal. *Deep Learning with Keras*, Packt Publishing Ltd., 2017.

[7] Steven Bird, Ewan Klein, Edward Loper. *Natural Language Processing with Python*, O'Reilly Media Inc., 2009.

[8] Dan Cireşan, Ueli Meier, Juergen Schmidhuber. *Multi-column Deep Neural Networks for Image Classification*, Ciresan et al. CVPR, 2012.

[9] Jeffrey Pennington, Richard Socher, Christopher D. Manning. *GloVe: Global Vectors for Word Representation* 2014.