ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Dipartimento di Informatica - Scienza e Ingegneria
Corso di Laurea in Ingegneria e Scienze Informatiche

USER INTERNET PROTECTION MEDIANTE CLASSIFICAZIONE DI PAGINE WEB CON TEXT MINING

Elaborato in
Programmazione Di Applicazioni Data Intensive

Relatore
Prof. Gianluca Moro

Presentata da
Luca Bazzocchi

Seconda Sessione di Laurea
Anno Accademico 2019 – 2020

PAROLE CHIAVE

Text Mining

Artificial Intelligence

Machine Learning

Artificial Neural Networks

Natural Language Processing

Alla mia famiglia,
e ai miei amici che mi sono sempre stati vicino.

Introduzione

Abbiamo il futuro già nelle nostre mani. In un'epoca di così grandi cambiamenti, l'essere umano può solo che rimanere stupito davanti ad una tecnologia che ogni anno avanza superando quella precedente. La nostra vita sta migliorando in aree sempre più vaste portandoci a scoprire lati di noi stessi che non conoscevamo.

Grazie alla diffusione dell'istruzione in una platea sempre più ampia di famiglie e territori l'umanità, negli ultimi cento anni, ha compiuto grandi passi in avanti in tutti gli ambiti scientifici, soprattutto nel campo della neurobiologia, anche se molto rimane ancora da scoprire sul funzionamento del cervello umano.

> *"The conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it."*

Il cosiddetto ***Dartmouth proposal*** [1] è considerato come l'evento d'inizio dello studio dell'**intelligenza artificiale**, in tale proposta si afferma che il cervello umano, se costruito in modo preciso nei suoi principi di funzionamento, sia replicabile da una macchina, ma che l'uomo non sia ancora in grado di scrivere tali programmi intelligenti.

Dal 1956 questo ambito è andato incontro a grandi cambiamenti e progressi grazie a figure di riferimento come Frank Rosenblatt, l'inventore dell'algoritmo *Perceptron*, o Yann LeCun, il padre delle *reti neurali convoluzionali* (CNN), o anche Geoffrey Everest Hinton, figura di spicco nell'ambito del *deep learning*.

Gli algoritmi di intelligenza artificiale al giorno d'oggi presentano dei meccanismi che emulano l'intelligenza umana, non la imitano semplicemente fanno come noi, analizzano dati, apprendono, predicono, sbagliano e apprendono ancora, in un ciclo ininterrotto finché ci saranno dati a disposizione. Ciò che è chiamata intelligenza artificiale non è vera intelligenza e nemmeno un'imitazione, è pura emulazione.

Ma cosa accadrà quando verrà creata la prima vera intelligenza artificiale?

"La prima macchina ultra-intelligente sarà l'ultima invenzione che l'uomo avrà la necessità di fare" detto dal matematico I. J. Good.

Le macchine attuali sono utilizzate per interagire e aiutare le persone nel mondo del lavoro e nella vita di tutti i giorni, l'intelligenza artificiale è presente sul web, in ufficio, ma anche nei nostri cellulari ed elettrodomestici.

Negli ultimi anni sta acquisendo sempre più importanza un sottoinsieme dell'artificial intelligence: il **machine learning**, che include tecniche di *apprendimento automatico* basate sulla statistica, al fine di creare algoritmi che imparano attraverso l'esperienza.

Il cambiamento portato da questi modelli rispetto alla normale programmazione è la possibilità di risolvere problemi senza la scrittura specifica di un codice, ma solo di alcuni elementi e parametri di base che vengono poi manipolati e migliorati durante l'addestramento al fine di ridurre il valore della funzione d'errore.

Il machine learning ha reso possibile un efficace ed efficiente sviluppo del *data mining*, del *sentiment analysis* e dei *sistemi di raccomandazione*, ma il vero cambiamento, nell'apprendimento automatico, è dato dal **deep learning** un sottoinsieme del machine learning composto da algoritmi che utilizzano reti neurali multistrato. Esponendo la rete ad una grandissima quantità di dati i modelli generati sono in grado di ottenere ottimi risultati nel riconoscimento delle immagini e del linguaggio.

Il **natural language processing** (NLP) si occupa proprio nell'ottenimento di conoscenza dai testi in linguaggio naturale.

Negli ultimi 30 anni la quantità di dati presenti sulla rete è cresciuta esponenzialmente e, tramite le tecniche del NLP, siamo in grado di scoprire collegamenti tra i dati fino ad ora non visibili per all'uomo. Questo progresso ha portato sia ricchezza in termini economici, che aiuto in ambiti come la medicina e l'ingegneria.

La classificazione di pagine web, ovvero il raggruppamento delle pagine in categorie sulla base del contenuto, come testo in linguaggio naturale, link, immagini e video sta diventando un problema sempre più importante da affrontare ed è proprio attraverso la conoscenza, ottenuta dai sorgenti html, che si cerca di raggiungere tale classificazione.

Le pagine web e quindi i loro sorgenti html, non sono presenti né in schemi né in modelli composti da regole tali da definire la forma e tipologia dei dati contenuti, questo porta ad una loro difficile comprensione.

Il **Text Mining** è il processo di trasformazione di testi non strutturati. Nel nostro caso, cioè la conversione di testo in linguaggio naturale presente sul web in dati strutturati, organizzati in schemi e tabelle, è utilizzato con lo scopo di estrarre conoscenza per la classificazione e il raggruppamento in base ai contenuti del testo di input. Attraverso il Text Mining riusciamo a utilizzare tecniche e metodi per la classificazione di pagine web con lo scopo di offrire, all'utente, una navigazione sicura sul web.

La *User Internet Protection* è così attuabile in quanto, avendo ottenuto una classificazione delle pagine web, l'utente potrà decidere le tipologie di siti da oscurare in modo tale da evitare i siti indesiderati con la presenza di malware e virus o siti per adulti e così via. Per ottenere una *Web Page Classification*, verranno impiegate tecniche e algoritmi per l'estrazione di pattern e conoscenza dal linguaggio scritto e l'addestramento di modelli tramite machine learning, deep learning e natural language processing.

La tesi è suddivisa nei seguenti capitoli:

- **Capitolo 1**: Discussione del caso di studio;
- **Capitolo 2**: Sguardo generale sulle tecnologie presenti in letteratura;
- **Capitolo 3**: Analisi dei dati in possesso e scelta delle fasi riguardanti lo sviluppo;
- **Capitolo 4**: Tutto ciò che ha riguardato lo sviluppo, discutendo i procedimenti scelti e futuri;
- **Capitolo 5**: Discussione dei risultati ottenuti.

Indice

Elenco delle figure

Capitolo 1

Caso di studio

In questo capitolo vengono analizzati i dati e gli obiettivi da conseguire, fornendo specifiche sulle tecnologie e sui metodi da utilizzare. Viene preso in esame il dataset completo con i dati in forma grezza, che verrà poi approfondito nei capitoli successivi dopo il suo trattamento.

1.1 Contesto applicativo

L'azienda Flashstart si occupa di fornire ai propri clienti una navigazione sicura in rete filtrando i siti che vengono cercati tramite la barra di ricerca, un utente può decidere se negare o permettere la navigazione scegliendo un dominio, un'area geografica e/o una determinata categoria, in modo che siano oscurati i siti che il cliente non vuole vedere (ad esempio siti di giochi o malware per i computer di una scuola).

Sulla base dei domini precedentemente classificati, Flashstart ha concesso l'utilizzo di alcuni suoi dataset per l'elaborazione di questa tesi basata sulla **Web Page Classification**, ovvero la *classificazione* di pagine web attraverso il contenuto del sorgente html, nel nostro caso il testo in linguaggio naturale, applicando tecniche di Text Mining.

La classificazione nel machine learning è una tecnica di apprendimento supervisionato per l'individuazione di una funzione, l'iperpiano di separazione, che massimizzi la suddivisione delle varie classi e, nel nostro caso, le categorie di siti.

Sono stati forniti in totale 22 dataset, in formato *csv*, contenenti ognuno una tipologia di siti differente e di varia lunghezza in quanto alcune categorie sono state identificate in minor numero rispetto alle altre.

Ogni elemento classificato presenta il sorgente della home-page, la data di classificazione, la categoria e il dominio di appartenenza. Essendo il sorgente un tipo di dato tipo testuale e oggetto da cui ricavare maggior informazioni,

lo scopo principale di questa tesi sarà proprio l'estrazione di nozioni da questi html, tramite tecniche appropriate e la successiva predizione di siti mai visitati.

È rilevante notare la natura inconsistente dei testi presenti nel web: blog, social networks, siti di eCommerce e a seguire, sono molto diversi gli uni dagli altri, come diverso è il lessico e la struttura delle loro frasi il cui sorgente, ottenuto dai datase, dovrà essere convertito applicando un apposito *noise removal* e normalizzato.

L'esperienza sarà ottenuta tramite tecniche di **natural language processing** e il successivo addestramento di modelli di **machine learning** e **deep learning**.

1.2 Dataset in Analisi

Come accennato nel paragrafo precedente, sono stati forniti 22 dataset, uno per ogni categoria e in formato csv, in totale sono presenti quattro feature:

- **domain**: il nome del dominio;
- **text**: il sorgente html;
- **time**: la data di classificazione;
- **intent**: la categoria del sito.

	domain	text	time	intent
2563	vacanzalmare.com	<!DOCTYPE html> <html lang="en"> <he...	2020-07-02 10:57:42	vacation

Figura 1.1: Esempio di riga del dataset vacation.

I dataset contengono siti web da tutto il mondo e ciò li porta ad includere oltre 30 lingue diverse, delle quali, però solo una decina è presente in oltre il 90% dei testi, ciò è dovuto alla maggior presenza di utenti e quindi di classificazioni nelle lingue più conosciute.

Alla consegna dei dataset l'azienda ha menzionato la presenza di possibili duplicati e html errati (ad esempio "404 not found") o comunque di scarsa indicazione della categoria stessa. L'addestramento di algoritmi con questo tipo di dati porterebbe a previsioni poco accurate o addirittura totalmente errate, è perciò necessario un buon pre-processing che verrà trattato nel dettaglio nei capitoli successivi.

Al momento, con questa conoscenza di base e senza ancora un'analisi approfondita sui dati, sono state identificate le seguenti fasi di sviluppo del

progetto: **pre-processing**, **addestramento** e **predizione**, le stesse di un normale problema di classificazione di natural language processing.

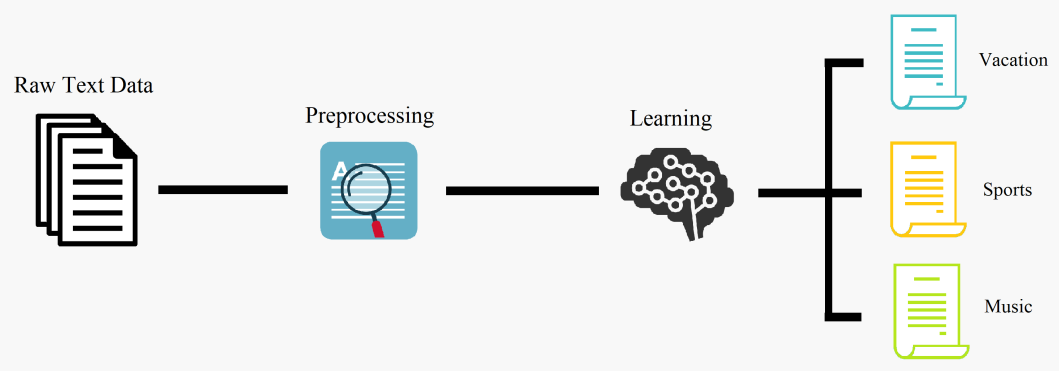

Figura 1.2: Dal testo grezzo alla classificazione.
Fonte: icone da https://icon-icons.com/it/

Capitolo 2

La continua evoluzione

Ogni anno, ogni mese, ogni giorno assistiamo alla nascita di nuovi algoritmi, di nuove tecnologie sempre più avanzate e migliori di quelle precedenti.

Ci stiamo muovendo verso un mondo in cui sistemi elettronici, informatici, computerizzati sono sempre più presenti nella vita quotidiana delle persone; abbiamo cellulari, smarth-watch, computer ma anche assistenti vocali, auto a guida autonoma e robot.

Grazie alla curiosità, all'intraprendenza dell'uomo, alla sua capacità di fare nuove scoperte ed il desiderio di migliorare la propria vita, la tecnologia si sta perfezionando sempre di più e ci sta portando verso una esistenza sempre più automatizzata.

Il periodo nel quale viviamo è stato definito *Industria 4.0*, termine attribuito a Henning Kagermann, Wolf-Dieter Lukas e Wolfgang Wahlster e usato per la prima volta nel 2011 nell'annuncio del *Zukunftsprojekt Industrie 4.0* [2], per descrivere il progetto di rilancio dell'industria tedesca. L'Industria 4.0 comprende sistemi *cyber-physical*, *internet of things (IOT)* e *cloud computing* e, mentre internet inizia ad essere sempre più presente negli oggetti di uso comune, aumentano le preoccupazioni degli effetti negativi dell'automazione e della digitalizzazione.

Questo importante aspetto è stato analizzato dagli economisti Erik Brynjolfsson e Andrew McAfee [3]:

> *"Come mettiamo in luce, non vi è mai stato un tempo migliore per essere lavoratori dotati di competenze elevate ed adeguate ai mutamenti in atto, poiché questi lavoratori potranno sfruttare le opportunità insite nelle nuove tecnologie. Tuttavia, non vi è mai stato un tempo peggiore per chi è dotato di competenze tradizionali poiché computer, robot ed altre tecnologie digitali stanno acquisendo queste competenze con una rapidità straordinaria..."*

I due economisti discutono proprio dell'automazione, del machine learning, dell'intelligenza artificiale e di come questi aspetti stiano cambiando la sfera lavorativa. Con l'avvento di Internet è stato possibile creare, inviare e immagazzinare grandissime quantità di dati, i quali aumentano in modo esponenziale di anno in anno e, tramite questi, vengono addestrati modelli di conoscenza per fare analisi e predizioni su problemi.

Di seguito sono descritte le principali tecniche e tecnologie utilizzate nel lavoro di questa tesi.

2.1 Machine Learning

Il termine Machine Learning (ML) è stato coniato nel 1959 da Arthur Lee Samuel, scienziato americano pioniere nel campo dell'Intelligenza Artificiale; il suo programma di dama [4] è stato tra le prime applicazioni al mondo ad utilizzare con successo l'apprendimento automatico.

La definizione più accreditata del termine machine learning è quella dell'informatico americano Tom Michael Mitchell [5]:

> *"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"*

Semplificando, un programma apprende se le sue prestazioni per un determinato compito accrescono con l'aumentare dell'esperienza. Per le macchine, questa esperienza è rappresentata dai dati che gli vengono forniti e non da come sono programmate.

Nella programmazione vengono scritti i comandi, il codice da fornire alla macchina per l'esecuzione di determinate azioni mentre l'output restituito sarà lo stesso a parità di input, diversamente nei modelli di Machine Learning vengono date determinate regole di base e, attraverso l'apprendimento, il modello impara a svolgere il compito richiesto.

Figura 2.1: Differenza tra la programmazione tradizionale e il machine learning.
Fonte: https://futurice.com/blog/differences-between-machine-learning-and-software-engineering

Lo sviluppo di un'applicazione di machine learning è un processo più esplorativo dell'ingegneria del software: l'apprendimento automatico si applica a problemi troppo complessi per la mente umana pertanto, un data scientist opterà per un'attitudine sperimentale e dovrà testare vari approcci prima di ottenere un modello soddisfacente.

Come si può vedere in figura 2.1, nel caso del ML, il computer trova un programma che si adatta ai dati, ciò che viene calcolata è una funzione y=f(x) che associa ad ogni dato in input, una classe / valore numerico. Prendendo ad esempio la regressione lineare, abbiamo:

$$\hat{y} = \alpha X + b$$

Figura 2.2: Funzione utilizzata nella regressione lineare.

Dove:

- $\hat{y}$ indica una stima sul valore effettivo y;
- α è il coefficiente angolare e indica quanto varia $\hat{y}$ al variare di x;
- b è l'intercetta e determina il valore "base" di $\hat{y}$ quando x è nullo.

Il programmatore, in base ai dati a sua disposizione, sceglierà gli iperparametri ottimali affinché le $\hat{y}$ siano le più vicine possibili alle y reali.

Il Machine Learning è utilizzabile per la risoluzione di un qualsiasi problema per il quale esistano dati sufficienti.

2.1.1 Tipi di apprendimento

L'algoritmo utilizzato per permettere l'apprendimento del modello, ossia le modalità in cui la macchina impara ed accumula informazioni, è possibile implementarlo sulla base di tre tipi di apprendimento: **supervisionato**, **non supervisionato** e **per rinforzo**.

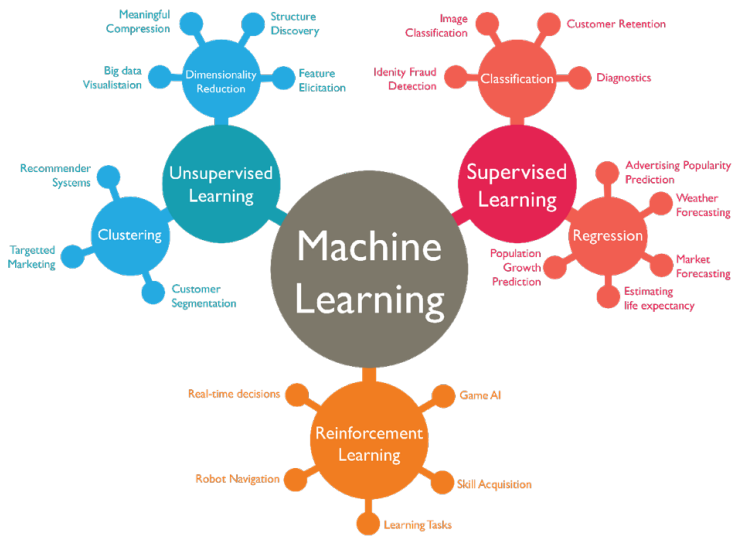

Figura 2.3: Tipi di apprendimento con relative sottoclassi.
Fonte: https://www.nextpng.com/en/transparent-png-rfopq

Apprendimento supervisionato In questa categoria disponiamo di dati già classificati / etichettati e vogliamo in uscita gli stessi valori già in nostro possesso, in modo tale da valutare i risultati e l'accuratezza. Per esempio, abbiamo una serie di figure ognuna già etichettata con 'triangolo', 'cerchio' o 'quadrato' e queste classificazioni dovranno essere predette dall'algoritmo.

L'apprendimento supervisionato viene utilizzato per risolvere problemi di:

1. **Regressione**: trovare il valore numerico dato da y = f(x);
2. **Classificazione**: predire la classe di appartenenza di un determinato oggetto.

Apprendimento non supervisionato In questo caso ai dati in ingresso non è data alcuna indicazione sui risultati desiderati, è l'algoritmo stesso che dovrà individuare, partendo dai dati, le associazioni, i pattern, le caratteristiche che non erano note a priori.

Questa acquisizione di conoscenza è utilizzata negli algoritmi di clustering, che cercano similarità tra gruppi di dati in base alla loro distanza reciproca.

Apprendimento per rinforzo È un tipo di apprendimento complesso in quanto richiede l'analisi dell'ambiente in cui è immerso. L'ambiente è dinamico ed è richiesto il raggiungimento di un obiettivo, l'apprendimento

è basato su premi o punizioni a seconda del risultato ottenuto ad ogni ciclo.

Molto utilizzato per le auto a guida autonoma o nell'acquisizione di regole e comportamento all'interno di un gioco.

2.1.2 Processo di estrazione della conoscenza

In questo paragrafo vediamo, per il learning supervisionato, tutti gli stadi che ci permettono di passare dai dati grezzi al modello che attua predizioni:

1. Raccolta dei dati con successiva analisi esplorativa e della loro qualità.
2. Rimozione dei valori mancanti e normalizzazione o standardizzazione.
3. Scelta in base agli obiettivi delle feature più rilevanti.
4. Divisione dei dati in training set, validation set e test set.
5. Interpretazione dei modelli di learning scelti.
6. Implementazione in applicazioni della conoscenza acquisita.

L'oggetto preso in esame in questa tesi utilizza il machine learning di tipo supervisionato, attenendosi alle fasi specificate sopra senza però alcun deployment.

2.1.3 Discesa del gradiente

Il metodo della *discesa del gradiente* [6] è la base del machine learning e ciò che permette al modello di migliorarsi iterazione dopo iterazione.

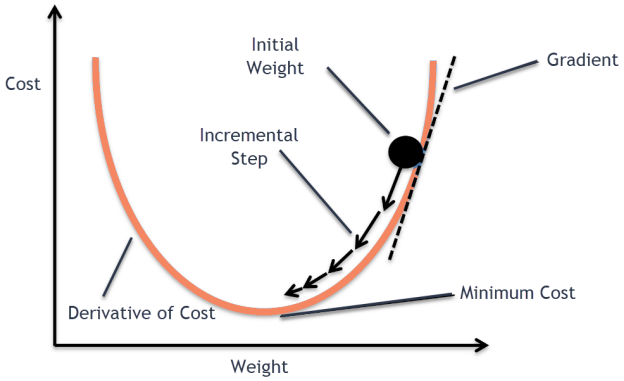

Figura 2.4: Discesa stocastica del gradiente fino ad un punto di minimo locale.
Fonte: https://medium.com/analytics-vidhya/stochastic-gradient-descent-1ab661fabf89

Il raggruppamento, in un vettore delle derivate parziali di ciascuna variabile di una funzione f, indica il gradiente ∇f della funzione stessa.

Graficamente il gradiente in un punto x, indica l'inclinazione della curva nel punto stesso.

I passi che vengono eseguiti per la discesa del gradiente sono i seguenti:

1. Viene preso un punto e si calcola il suo gradiente;
2. Al punto scelto è sottratto un vettore proporzionale al gradiente per ottenere un nuovo punto con valore nella funzione d'errore inferiore al precedente;
3. Si iterano i primi due passaggi finché non si ottiene una convergenza.

Ad ogni passo l'algoritmo conosce solo due valori: il gradiente e il tasso di apprendimento (la lunghezza del passo di discesa).

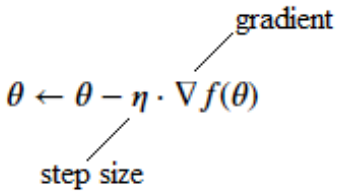

Figura 2.5: Passo i dell'algoritmo di discesa del gradiente.

In generale soluzioni ottime non possono essere garantite con il metodo di discesa del gradiente ma tale metodo è adatto alla risoluzione di problemi non convessi inoltre incrementale e parallelizzabile.

2.2 Reti neurali artificiali

Una *rete neurale artificiale* è un'architettura computazionale composta da neuroni artificiali, che prendono ispirazione dalle reti neurali biologiche.

Il sistema nervoso centrale dell'essere umano è composto da particolari cellule nervose chiamate neuroni che, interconnessi tra loro, formano la rete neurale biologica.

Anche se ogni cellula esegue una elaborazione molto semplice, l'invio di segnali elettrici nell'insieme di neuroni e connessioni porta ad un comportamento complesso.

2.2.1 Storia

Il primo modello di neurone artificiale è stato proposto nel 1943 dal matematico Walter Pitts e dal neurofisiologo McCulloc nel documento "*A Logical Calculus of the Ideas Immanent in Nervous Activity*" [7], lo scritto modella un prototipo di neurone artificiale denominato "**combinatore lineare a soglia**" in grado di prendere dati binari in input e di avere un solo dato binario come output. Componendo tra loro sufficienti elementi il combinatore lineare è in grado di svolgere delle elementari funzioni booleane.

Una notevole miglioria in questo ambito è avvenuta nel 1959 con il computer scientist Frank Rosenblatt che, nella rivista "*Psychological Review*", propose la prima rete neurale chiamata **Perceptron** [8]. Questo schema presenta uno strato d'ingresso e uno in uscita ed è in grado di riconoscere e classificare forme attraverso dei pesi sinaptici.

Nel 1974 venne presentata una rete **Multi-Layers Perceptron** con l'aggiunta di un hidden-layer all'algoritmo originale, ma il vero cambiamento arrivò nel 1986 con l'algoritmo di retropropagazione dell'errore (error back-propagation) proposto da Rumelhart, Hinton e Williams in grado di perfezionarsi confrontando l'output della rete con quello desiderato, utilizzando il metodo della discesa del gradiente.

Da questo momento vi è stato l'effettivo decollo di popolarità e prestazioni delle reti neurali artificiali.

2.2.2 Funzionamento

Entrando nello specifico, le reti neurali artificiali sono dei paradigmi computazionali caratterizzati da tre elementi:

- *Architettura* con il neurone come unità computazionale;
- *Funzionalità associata a tale architettura* che dipende dalla funzionalità messa a punto dal neurone e, quindi, dalle interconnessioni neurone-neurone;
- *Procedura di apprendimento* il cui obiettivo è quello di regolare i parametri associati alla funzionalità dei neuroni e quindi dell'architettura, per risolvere un problema applicativo specifico.

La funzionalità svolta da un singolo neurone può essere vista come un albero, dove il neurone (l'unità computazionale) riceve degli ingressi con associati dei pesi e ciò che viene valutato è una misura di affinità tra il vettore degli ingressi e il vettore dei parametri neurali.

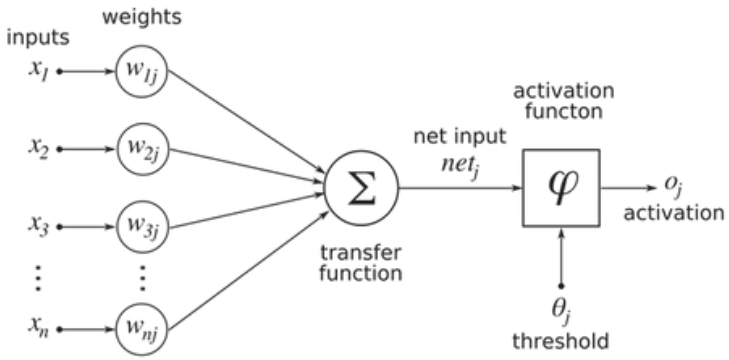

Figura 2.6: Esempio di neurone artificiale.
Fonte: http://www.pensierocritico.eu/intelligenza-artificiale.html

Valutata questa affinità, il valore di attivazione viene elaborato da una funzione detta funzione, di attivazione fornita in uscita neurale; l'unione di più neuroni, all'interno dell'architettura neurale, porta a una rete neurale multistrato, all'interno della quale si possono inserire *N hidden-layer* degli strati intermedi per una elaborazione più specifica e ottimale.

In una rete artificiale la procedura di apprendimento dovrà identificare la migliore configurazione dei parametri che consenta di risolvere il problema applicativo.

2.2.3 Deep Learning

Quando il numero di strati neurali all'interno della rete è profondo, ovvero più di uno, si parla di apprendimento profondo e quindi di *deep learning* [9].

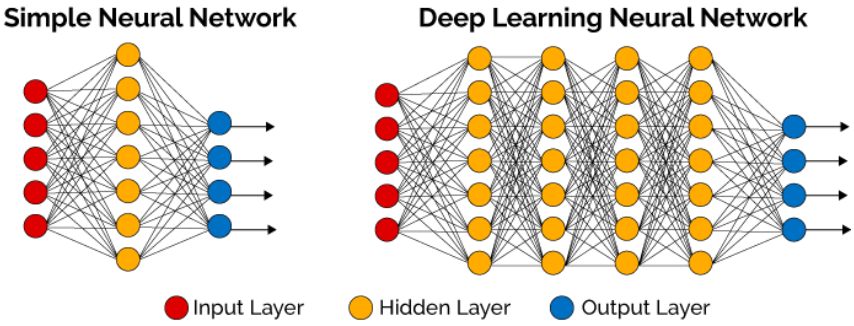

Figura 2.7: Differenza tra rete neurale semplice e profonda.
Fonte: https://towardsdatascience.com/a-conversation-about-deep-learning-9a915983107

Una delle caratteristiche fondamentali che ha reso il deep learning particolarmente interessante è che è un approssimatore universale di funzioni, nel senso che può emulare qualsiasi funzione.

Questo tipo di reti vengono utilizzate per risolvere problemi di:

- *Natura predittiva*;
- *Classificazione*, quando si vuole classificare, ad esempio, delle lettere di posta elettronica;
- *Approssimazione di funzioni* dove l'obiettivo è quello di ricostruire una funzione partendo da un certo numero di punti e informazioni date.

Le reti neurali artificiali, in particolare quelle con architetture deep learning, costituiscono oggigiorno l'elemento di spicco dell'intelligenza artificiale.

2.3 Natural Language Processing

Un testo in linguaggio naturale non è altro che un insieme ordinato di caratteri e numeri che si susseguono fino a formare parole e frasi di senso compiuto. La conoscenza ricavabile da questa tipologia di dati è molto alta vista l'incredibile mole di dati presenti sul web ma al tempo stesso molto complicata da ottenere.

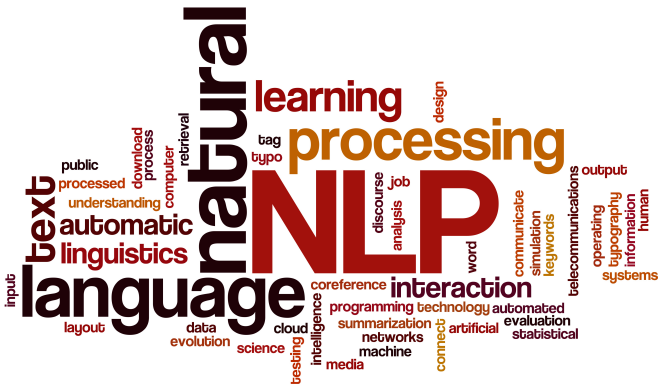

Figura 2.8: Natural Language Processing.
Fonte: https://www.blumeglobal.com/learning/natural-language-processing/

Ma quanti sono questi dati?

La cifra generalmente accettata è l'80%, come spiega Grimes Seth, nel suo documento *"Unstructured Data and the 80 Percent Rule"*, ragion per cui nelle aziende generalmente meno del 20% dei dati che vengono prodotti sono strutturati e quindi facilmente utilizzabili dai calcolatori, come quelli presenti nei database; la maggior parte dei dati è di tipo testuale, e sono prodotti dagli utilizzatori per la propria attività. Per una azienda l'acquisizione di conoscenza

riguardo questo tipo di "testo non strutturato" porta sia benefici per i propri lavoratori che vantaggi in termini economici e di efficienza.

Il natural language processing [10] si occupa proprio di analizzare documenti in linguaggio naturale per capirne e comprendere la semantica, i collegamenti nei i vari contesti ed estraendo così conoscenza.

2.3.1 Turing Test

Alan Turing nel suo famoso articolo "Computing Machinery and Intelligence" [11] si domanda: *"Can machines think?"*

Per rispondere a tale domanda, Turing propone un gioco da lui chiamato "The imitation game" in cui sono presenti tre entità: un uomo (A), una donna (B) e un interrogatore (C). Quest'ultimo sarà in una stanza differente e farà domande alle altre due persone le cui risposte non saranno pronunciate ma scritte.

Lo scopo di C è indovinare qual è il sesso di A e B basandosi sulle loro risposte e sapendo che una delle due entità può mentire.

Turing a questo punto pone un'altra domanda *"Che cosa accadrà se una macchina prende il posto del mentitore?"*. Questo è il punto focale del test: se la macchina è in grado di generare risposte simili a quelle umane, si avranno effettivamente gli stessi risultati di quando tutte le entità erano persone? Oppure la conversazione in linguaggio naturale tra umano e computer darà risultati differenti?

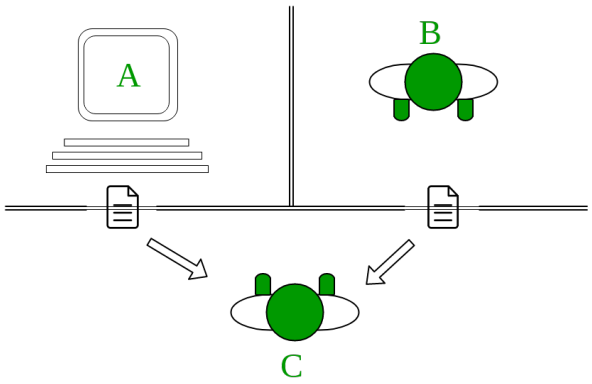

Figura 2.9: "The imitation game" tra un umano e una macchina.
Fonte: https://www.geeksforgeeks.org/turing-test-artificial-intelligence/

L'interrogazione iniziale viene perciò sostituita da un'altra più semplice: *"Are there imaginable digital computers which would do well in the imitation game?"* a cui può essere data risposta.

Questo test può essere definito come criterio di intelligenza per valutare l'abilità di un computer nel capire e scrivere linguaggio naturale.

2.3.2 Pre-processing di un testo

Il linguaggio naturale è spesso ambiguo e indipendente dal contesto; l'interpretazione di determinate parole o frasi può cambiare all'interno di due documenti all'apparenza simili.

Le sequenze di caratteri in forma non strutturata hanno bisogno di essere convertite o eliminate per ottenere un testo facilmente interpretabile dal calcolatore. Ad esempio, coniugazioni diverse dello stesso verbo dovranno far riferimento sempre alla stessa parola di base senza che esse vengano trattate come caratteri differenti, nel documento possono essere presenti termini poco indicativi del senso di una determinata frase o testo, come gli articoli e i segni di punteggiatura che dovranno essere eliminati per ottenere elementi strutturati e significativi.

Le principali tecniche utilizzate nel pre-processing sono le seguenti:

Segmentazione Lo scopo è di suddividere il testo nelle sue singole parole denominate *token*, in base a dei delimitatori quali lo spazio, con documenti lunghi si può suddividere il testo in frasi e poi nelle sue parole;

Part of speech (POS) Consiste nel trovare la classe grammaticale (verbo, aggettivo, nome...) di ogni parola. Nella *part of speech tagging* ogni parola è associata al proprio POS, facendo attenzione al contesto di ogni elemento all'interno di una frase, visto che ogni parola può avere diverse classi;

Case folding Tecnica con cui si converte il testo in minuscolo;

Stopwords Non tutti gli elementi all'interno di un testo sono significativi per capire l'argomento trattato, nella fase di stopwords viene rimossa una lista di queste parole detta **stoplist**;

Lemmatizzazione Le parole all'interno dei documenti si trovano nella loro versione flessa, ossia coniugata, declinata. La lemmatizzazione converte ogni parola nel suo *lemma*, ovvero la sua forma base definita nel dizionario (es. changing ⇒ change);

Stemming Procedimento simile alla lemmatizzazione ma in contrapposizione ad essa la lemmatizzazione. In questo caso lo stemming estrae la *radice morfologia* di ogni parola eliminando il suffisso (es. changing ⇒ chang), spesso la radice è una parola priva di senso.

Ottenuta la combinazione di queste tecniche il testo diviene composto da soli elementi significativi che dovranno essere trasformati in una struttura utile all'addestramento della macchina.

2.3.3 Modelli di trasformazione dei dati

Gli algoritmi di apprendimento nell'ambito del natural language processing hanno bisogno di grandi quantità di dati per essere addestrati in modo efficace ed efficiente. Questi dati per essere utilizzati devono presentare una struttura specifica che, solitamente, si presenta sotto forma di una matrice di numeri.

Di seguito verranno illustrati i principali modelli utilizzati in letteratura che permettono di trasformare dati di tipo testuale in forma numerica:

Bag of words (BOW) È una rappresentazione vettoriale che identifica i termini di un documento testuale.

In base ai contenuti, il Bag of words, si compone di due parti:

1. Un dizionario delle parole condivise.
2. Il numero di occorrenze di ogni termine nel documento.

Il nome Bow deriva proprio dalla mancanza di ordinamento e di struttura delle parole all'interno del vettore.

Se si vogliono rappresentare più documenti nel medesimo spazio, si può utilizzare il *Vector Space Model* che presenta le stesse funzionalità del Bow ma applicato a più testi.

	and	at	beautiful	delicious	is	look	looks	orange	sunset	that
look at that beautiful sunset	0	1	1	0	0	1	0	0	1	1
sunset is orange and sunset is beautiful	1	0	1	0	2	0	0	1	2	0
that orange looks delicious	0	0	0	1	0	0	1	1	0	1

Figura 2.10: Vector space model di tre documenti.

Questi vettori numerici saranno poi passati agli algoritmi di addestramento automatico.

TF-IDF Questo modello, come quello precedente, è utilizzato per enfatizzare l'importanza di determinati termini rispetto ad altri. Mentre in Bow, l'importanza era data solo dal numero di occorrenze della parola in quello specifico documento, senza considerare tutti gli altri, nella rappresentazione Tf-Idf invece, ogni valore è utilizzato per valutare la rilevanza delle parole.

- **TF (term frequency)** è l'importanza locale e indica quante volte un termine appare all'interno del documento;

- **IDF (inverse document frequency)** è l'importanza globale ed è più grande per le parole che appaiono meno frequentemente in tutti i testi.

Sulla base di questi due valori, il peso è dato dal loro prodotto:

$$W_{i,j} = tf_{i,j} * \log(\frac{N}{df_i})$$

$tf_{i,j} = number\ of\ occurences\ of\ term\ i\ in\ document\ j \quad df_i = number\ of\ documents\ containing\ term\ i$

$N = total\ number\ of\ documents$

Figura 2.11: Formula per il calcolo del Tf-Idf.
Fonte: https://towardsdatascience.com/word2vec-for-phrases-learning-embeddings-for-more-than-one-word-727b6cf723cf

Vista la possibile differenza tra i documenti in ingresso, ciascun vettore viene normalizzato utilizzando la norma euclidea pari a 1, ricavando così valori compresi tra 0 e 1.

Word2Vec Uno dei problemi maggiori nel NLP è sempre stato quello di trovare un modo per far capire al computer le varie associazioni che sono presenti tra le parole (boy ⇒ girl, mom ⇒ son), in assenza delle quali non è possibile eseguire operazioni matematiche di alcun tipo.

Nel 2013 è stato ideato l'algoritmo word2vec [12] che si serve di un modello di rete neurale per associare ad ogni parola il proprio **word embedding**. L'idea di base sta nel convertire la parola in un vettore composto da N feature numeriche (il word embedding), in modo tale da permettere calcoli matematici tra parole.

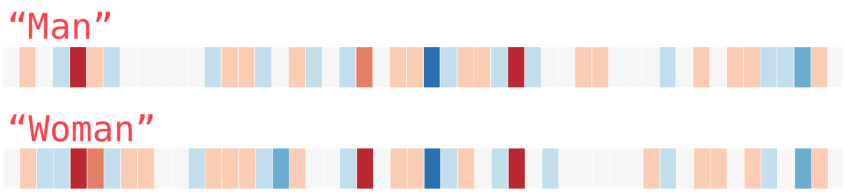

Figura 2.12: Word embeddings delle parole “man” e “woman”.
Fonte: http://jalammar.github.io/illustrated-word2vec/

Nella figura 2.12 a ogni colore è associato un numero positivo o negativo per ciascuna delle feature selezionate. Mettendo vicini i due vettori di uomo e donna rispettivamente si osserva una notevole somiglianza, infatti, come si può vedere nella figura 2.13, la proiezione di vettori analoghi e

quindi di parole analoghe, in uno spazio ad N dimensioni, dove N è il numero di feature, è più attigua rispetto a word embeddings poco simili.

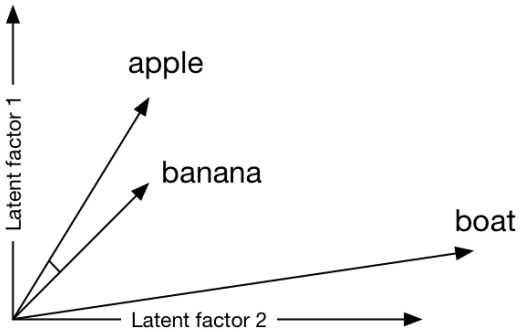

Figura 2.13: Esempio di word embeddings in uno spazio a due dimensioni.
Fonte: https://erikbern.com/2015/09/24/nearest-neighbor-methods-vector-models-part-1.html

La similarità tra due vettori è calcolata tramite la **similarità coseno**, data dal prodotto scalare dei due vettori diviso il prodotto delle norme dei vettori stessi.

Doc2Vec Come comportarsi con grandi quantità di dati da collegare, associare e relazionare?

Uscito un anno dopo il word2vec, *doc2vec* [13] prende le basi del modello originale per creare una rappresentazione numerica di documenti, a prescindere dalla loro lunghezza (che sia una frase, un paragrafo, o più) piuttosto che le singole parole.

Di doc2vec sono state create due versioni: *PV-DM (Distributed Memory version of Paragraph Vector)* che si basa sulla versione *CBOW* di word2vec e *PV-DBOW (Distributed Bag of Words version of Paragraph Vector)* modellato sullo *Skip-Gram*.

Ai *word vectors* in *PV-DM*, rispetto all'algoritmo originale, è stato aggiunto un elemento denominato **paragraph ID**.

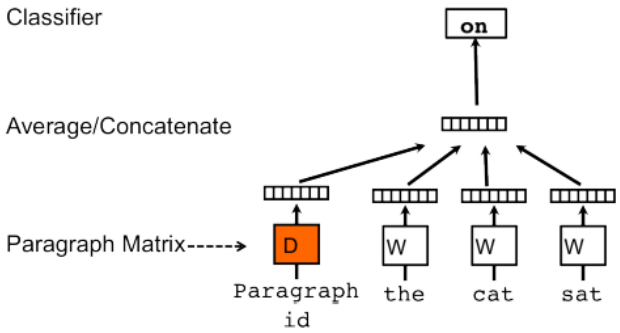

Figura 2.14: Addestramento del modello PV-DM.
Fonte: https://medium.com/wisio/a-gentle-introduction-to-doc2vec-db3e8c0cce5e

In questo modo, come illustrato in figura 2.14, i word vectors si addestrano, e contemporaneamente lo farà anche il paragraph ID, contenendo a fine addestramento la rappresentazione numerica del documento.

PV-DBOW è all'opposto di PV-DM: invece di predire una parola alla volta, usa il paragraph ID per predire tutte le parole circostanti (il contesto).

Normalmente si ottengono previsioni più accurate utilizzando una combinazione di entrambi i modelli.

2.4 Attention is all you need

Per capire il meccanismo dell'**attention** e in seguito parlare del concetto di **Transformer**, bisogna prima comprendere la conoscenza che l'ha preceduto.

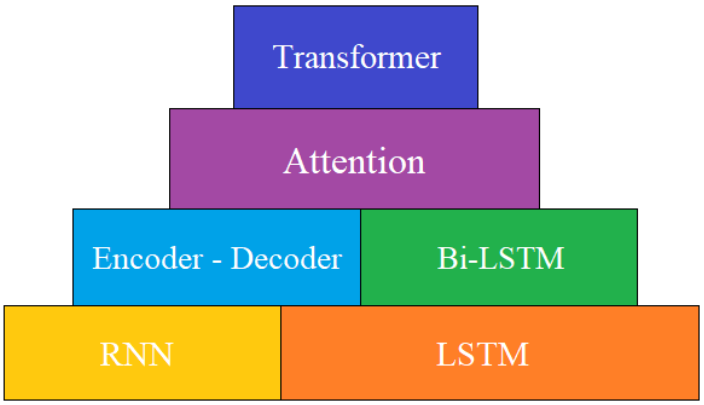

Figura 2.15: Conoscenza che ha portato al concetto di Transformer.

Fino all'uscita di tale documento, ci si basava su varie architetture quali:

Recurrent Neural Networks (RNN) Le reti ricorrenti [14] si basano sul concetto di reti neurali, con lo scopo di cogliere correlazioni tra input, lontani anche diversi istanti di tempo.

L'idea che sta alla base di tale architettura è di mandare in retroazione l'output che è calcolato ad ogni iterazione. L'output non è quantificato solamente dai pesi trovati in base all'input, ma anche dai pesi calcolati nell'*hidden layer* dell'input precedente. I parametri qui ottenuti vengono perciò passati di immissione in immissione e gli stessi dati d'ingresso produrranno valori differenti a seconda degli input precedenti.

Long Short Term Memory Le LSTM [15] sono reti che sono state progettate sulla base delle RNN, con lo scopo di risolvere il problema della dipendenza a lungo termine, infatti, le reti ricorrenti tendono a perdere informazioni col passare del tempo.

Ogni *hidden-unit* viene sostituita con una cella LSTM e a ogni cella viene aggiunta una nuova connessione chiamata *cell state*, al fine di mitigare il problema della scomparsa ed esplosione del gradiente, ogni cella mantiene un **cell state vector**, il quale può essere letto, scritto o resettato attraverso dei *gate*. Ogni unità presenta tre gate:

- **Input gate**: controlla se la cella è da aggiornare;
- **Forget gate**: verifica quali valori mantenere;
- **Output gate**: decide l'output da produrre.

È nell'ultimo gate che, attraverso la funzione di attivazione *tanh*, i gradienti sono distribuiti in un intervallo tra [-1, 1] prevenendo la scomparsa o esplosione del gradiente.

Encoder - Decoder È un'architettura formata da due entità: l'encoder e il decoder.

L'encoder riceve dati in input e produce una rappresentazione compressa di questi chiamata **context vector**, in seguito il decoder consuma il vettore producendo predizioni probabilistiche e selezionando, come output, gli elementi con probabilità maggiore.

Bidirectional LSTM È una variante dell'originale LSTM composta da due reti ricorrenti indipendenti invece che una. Solo questa struttura permette alla rete di immagazzinare informazioni sia iniziali che finali di una sequenza grazie alla bidirezionalità. La prima rete si muove in avanti

(forward), mentre la seconda all'indietro (backward) e l'output viene generato dalla combinazione degli hidden states delle due RNN.

Dopo questa breve spiegazione generale, passiamo a parlare di uno dei documenti più importanti dello scorso decennio.: l'*"Attention is all you need"* [16].

Fino alla pubblicazione dell'"Attention is all you need" per la risoluzione di modelli di machine translation in particolare dei *sequence-to-sequence* si utilizzavano le reti neurali di tipo RNN con la codifica dell'input in una rappresentazione intermedia (context vector) e la successiva predizione dell'output tramite l'architettura encoder-decoder.

Questi tipi di algoritmi risultano molto efficienti, ma solo per testi corti, ciò perchè l'approccio encoder-decoder, che richiede informazioni differenti in tempi differenti crea in alcuni casi, dipendenze a lungo termine. All'aumentare della grandezza delle sequenze in input il context vector agisce da collo di bottiglia, poiché è impossibile riassumere sequenze lunghe in un unico vettore. In questi casi il decoder non è in grado di estrarre in modo efficiente ed efficace tutte le informazioni.

Intuitivamente il meccanismo dell'attention permette al decoder di osservare l'intera sequenza per estrarre le informazioni necessarie alla decodifica.

Concretamente, l'attention fa scegliere al decoder, tra tutti gli stati nascosti (hidden states) prodotti dall'encoder, quelli da utilizzare e quelli da scartare perché di poca importanza. In seguito il decoder predirà la parola successiva basandosi proprio sulla somma pesata degli strati nascosti.

Riassumendo possiamo affermare che l'attention impiega tutte le informazioni, invece che una sintesi, consentendo così al decoder di scegliere quali informazioni utilizzare permettendogli così di focalizzarsi solamente sui dati necessari.

L'"attention is all you need" ha cambiato completamente le basi per la risoluzione di problemi di natural language processing, permettendo il passaggio dalle reti ricorrenti al meccanismo dell'attention diventando in tal modo una delle idee più influenti del deep learning e venendo utilizzato anche in altri ambiti, come ad esempio, la descrizione testuale di immagini.

2.4.1 Transformer

Nel documento "Attention is all you need" è stata presentata questa nuova architettura chiamata Transformer in sostituzione alle *"RNN with attention"*. Tale architettura proposta, come risolutrice di problemi sequence-to-sequence, è divenuto lo "state-of-the-art" nella traduzione inglese-francese e inglese-tedesco ed è basata sul meccanismo dell'attention, si guarda cioè la sequenza in input e ad ogni passo, si decide quali parti della sequenza sono importanti.

Internamente il Transformer presenta una struttura encoder-decoder composta da blocchi multipli.

Ogni encoder, come illustrato in figura 2.16 è composto principalmente da due livelli:

- **Self-Attention layer** che aiuta l'encoder a guardare le altre parole della sequenza in input mentre codifica una specifica parola;
- **Feed Forward neural network** dove le informazioni vengono processate.

Il decoder è formato da questi due strati con l'aggiunta, al centro, di un attention layer per focalizzarsi sulle parti importanti della sequenza in input.

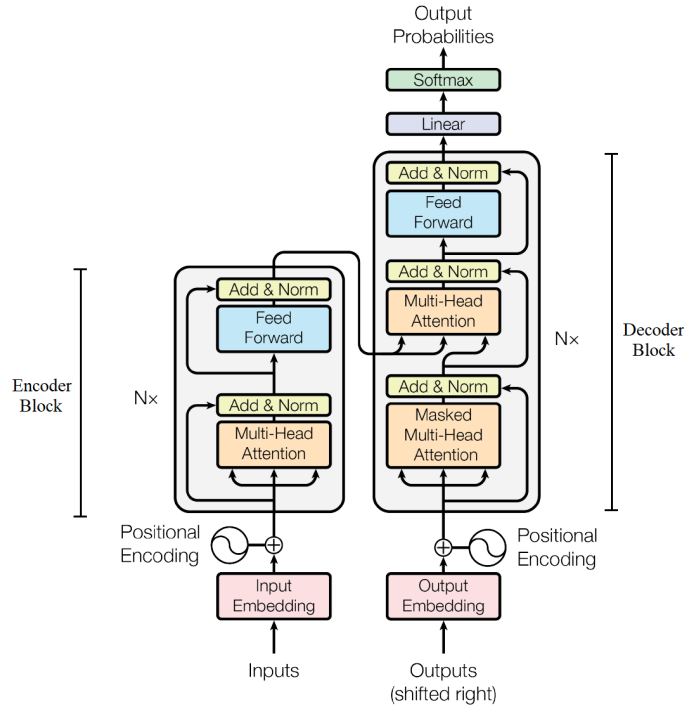

Figura 2.16: Struttura del Transformer.
Fonte: paper "Attention is all you need"

La maggior parte delle applicazioni dell'architettura del Transformer è utilizzato in ambito di NLP, quali speech recognition, machine translation e document summarization, ed è stato usato come base per la creazione di modelli pre-addestrati quali **BERT**, **XLNet**, **GPT-2**. Questi sono divenuti lo stato dell'arte della risoluzione di vari problemi e hanno dato la possibilità a studenti e ricercatori di utilizzare modelli avanzati senza l'effettiva necessità di strumenti moderni o immense quantità di dati.

Capitolo 3

Modellazione del problema

Analizzate le varie tecnologie in letteratura, verranno ora illustrati i dataset al fine di ottenere la conoscenza necessaria per la selezione delle fasi di sviluppo. Trattandosi di un problema di classificazione e di natural language processing, il punto focale sarà il testo.

3.1 Analisi dei dati

I 22 dataset forniti da Flashstart sono stati raggruppati in un unico dataset per rendere più semplice ed efficace l'analisi, al suo interno sono presenti 202.489 elementi e nessun valore nullo. Vista la presenza di valori duplicati all'interno del dataset si è deciso di rimuovere tali duplicati al fine di migliorare l'analisi.

Si sono così ottenuti 60.247 dati non duplicati, un numero notevolmente inferiore a quello di partenza, con le seguenti feature: dominio di appartenenza, sorgente della home-page, data di classificazione e categoria.

Nella prima parte di questa analisi è necessario controllare quali feature siano effettivamente utili per l'addestramento dei modelli:

- Il dominio è valutato come caratteristica di poca importanza in quanto il nome di un sito difficilmente sarà un indicatore della categoria cui appartiene; egual discorso per la data. Queste due feature perciò, non sono state prese in considerazione.

- Il sorgente e la categoria di appartenenza sono invece gli unici dati su cui ci si concentrerà nello sviluppo.

Scelti *text* e *intent*, nella figura 3.1 è possibile vedere tramite un grafico a torta ed uno a barre, la distribuzione dei dati nelle varie categorie:

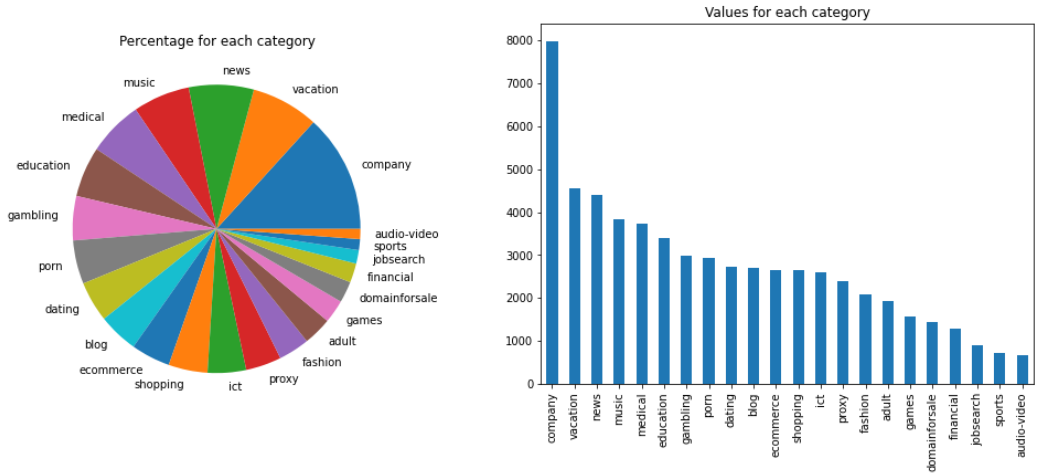

Figura 3.1: Grafico a torta e a barre sulla distribuzione dei dati.

Analizzando i dati notiamo un forte sbilanciamento tra le classi: i valori di 'company' sono 7.978 mentre quelli di 'audio-video' arrivano solamente a 683. Come già specificato nella sezione 1.1, il numero di elementi per ogni categoria è varia in quanto, per disponibilità, alcune categorie sono state identificate in minor quantità rispetto alle altre. Saranno certamente necessarie tecniche di *under sampling* e *over sampling* per ribilanciare i dati ed evitare la predizione delle sole classi maggioritarie.

La distribuzione ci può far capire, anche se non in modo accurato, la maggior presenza sul web, di determinati siti rispetto ad altri.

Un'altro aspetto importante da analizzare è la lunghezza, in caratteri, dei sorgenti:

Length

	type	value
0	count	60247.000000
1	mean	46967.934636
2	std	27455.422256
3	min	13.000000
4	25%	21785.000000
5	50%	54032.000000
6	75%	71797.000000
7	max	134595.000000

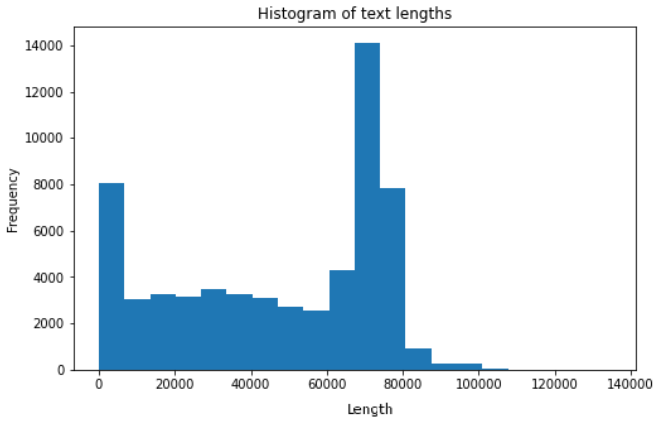

Figura 3.2: Statistiche e grafico sulla lunghezza dei sorgenti html.

La prima tabella di figura 3.2, contenente le statistiche di base, è stata ottenuta su python attraverso il metodo *describe* dei dataframe; mentre il grafico è un istogramma che mette in relazione la lunghezza dei testi con la loro frequenza all'interno del dataset.

I valori sono piuttosto eterogenei, cosa prevedibile, poiché non esiste uno standard per la lunghezza delle home-page html, è interessante notare la presenza di un sorgente con soli 13 caratteri, questo sarà certamente da rimuovere viste le limitate informazioni al suo interno.

Ulteriori valutazioni sono difficili da fare con i dati così grezzi, bisognerebbe quanto meno estrarre il testo dal codice html, determinati lavori come l'analisi sulle lingue, verranno quindi svolti dopo il preprocessing.

3.2 Selezione delle fasi di sviluppo

L'obiettivo è ottenere la predizione della categoria di un sito basandosi sul suo sorgente html (in formato testuale) e sulla classe precedentemente catalogata.

Queste informazioni ci fanno capire di essere all'interno di un problema di apprendimento supervisionato ma anche di un problema di classificazione nonché all'interno di un problema di natural language processing. Si adotteranno perciò le tecniche più opportune per il processamento e l'addestramento dei dati.

Dopo l'analisi dei dati e stabilito l'obiettivo, sono state individuate le seguenti fasi, che verranno descritte nel dettaglio nei capitoli successivi:

1. Preprocessing dei sorgenti al fine di ottenere testi significativi e senza codice di programmazione html;
2. Scelta dei modelli opportuni;
3. Conversione dei dati in strutture interpretabili dagli algoritmi e successivo loro addestramento;
4. Analisi finale sui risultati.

Capitolo 4

Sviluppo

In questo capitolo verranno illustrate, a fondo, tutte le tappe riguardanti lo sviluppo di un sistema di classificazione di siti web, partendo dal preprocessing dei testi html fino alla scelta dei modelli la cui analisi e relativa discussione dei risultati verrà affrontata nel capitolo successivo. Come linguaggio è stato utilizzato il **python**, sulla piattaforma Google Colab, vengono menzionate anche le difficoltà riscontrate e i possibili cambiamenti o migliorie da applicare in futuro.

4.1 Data Preprocessing

Il Data Preprocessing è il passo nel quale i dati vengono trasformati e codificati in modo da essere interpretati facilmente, ma soprattutto correttamente, dal calcolatore. È uno, se non il più importante, passo da effettuare nel Natural Language Processing: un text preprocessing poco o mal sviluppato porterà a modelli e previsioni scarsamente accurate.

Di base, i computer non sono in grado di addestrare se stessi tramite immagini, video e testo in linguaggio naturale ne tanto meno sono in grado di comprendere tali dati senza una trasformazione.

Charles Babbage nel suo scritto *"Passages from the Life of a Philosopher"* [17] riporta una domanda postagli in più discussioni:

> *"Pray, Mr. Babbage, if you put into the machine wrong figures, will the right answers come out?"*

La risposta è semplicemente "No" e la spiegazione è altrettanto semplice: una macchina potrà rispondere solamente in base a ciò che ha imparato in precedenza, con dati errati si otterranno risultati errati.

Di seguito sono riportatati in sequenza, i metodi di preprocessing utilizzati per ottenere un testo idoneo per i nostri modelli.

1. **Eliminazione delle righe nulle** A causa di una classificazione errata, in alcune righe, non è presente l'html. La mancata presenza di questo elemento rende inutile la riga che perciò viene rimossa.

2. **Rimozione dei duplicati** Testi duplicati portano a una sovrabbondante presenza di alcuni termini a discapito di altri, che potrebbero avere uguale o maggior importanza. Tutto ciò può portare a un modello poco generale con la possibilità che venga effettuato il test su valori utilizzati durante l'addestramento.

3. **Eliminazione dei testi che non presentano determinati match** Ogni file html senza errori, deve contenere almeno uno di questi tag:

   "<!DOCTYPE", "<!doctype", "<HMTL", "<html"

   In caso contrario si avrà un html mal formattato con ripercussioni nell'estrazione del testo e il successivo addestramento.

4. **Estrazione del testo dal html** In una web classification questa fase rappresenta la parte più importante di tutto il pre-processing: la distinzione del testo dalle parti di codice, in modo tale che queste ultime siano eliminate in modo corretto, è importantissimo per l'ottenimento di un buon modello.

```
<!DOCTYPE html>
<html lang="it">
<head>
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Toway</title>
</head>
<body>
    <div class="menu">
        <p>Sito di eventi, concerti e tanto altro!</p>
    </div><footer>
        <p>Progetto Tecnologie Web - A.A. 2019/2020</p>
        <p>Autore: Bazzocchi Luca</p>
    </footer>
</body>
</html>
```

Figura 4.1: Esempio di sito html di qualità.

Com'è ravvisabile in figura 4.1, un sito html di qualità deve essere ben formattato, presentando una opportuna codifica.

È significativo notare l'assenza di modifiche ai dati sino a questo momento, una manipolazione dell'html prima dell'estrazione del testo, come ad esempio la rimozione dei numeri o la modifica di determinate stringhe,

può produrre effetti indesiderati come la formattazione errata dell'html e, di conseguenza, un testo errato.

Sono state testate due diverse librerie per capire quale approccio risultasse migliore nell'estrazione del testo:

- **html2text**;
- **BeautifulSoup**.

Sono stati presi in analisi vari testi appartenenti a diverse categorie, in figura 4.2 sono riportati i risultati ottenuti utilizzando le due librerie, per un sito della categoria *'vacation'*:

Contact Us Cart: 0 Items * * Diving * Diving in Costa Rica * Best Dive Spots * Whe	html2text
Contact Us Cart: 0 Items Diving Diving in Costa Rica Best Dive Spots Where To Dive	beautifulSoup

Figura 4.2: Confronto tra le librerie html2text e BeautifulSoup.

I risultati sono pressoché identici, con la differenza che BeautifulSoup ha rimosso in automatico alcuni segni speciali come l'asterisco; in html2text è necessario aggiungere del codice per ottenere gli stessi risultati. Il punto di forza che ha fatto scegliere BeautifulSoup a discapito di html2text è stata la flessibilità, in quanto è possibile scegliere facilmente quali tag html rimuovere.

Le librerie non sono perfette e, se si ha del codice mal formattato, la conversione html $\Rightarrow$ testo potrebbe non essere ottimale. I risultati finora ottenuti sono più che soddisfacenti per l'addestramento dei modelli, in futuro si potrebbe fare un'analisi più approfondita per migliorare ancora di più la conversione.

Scelta la libreria bisogna approfondire i *tag* che possono essere ignorati e quindi eliminati. I tag principali sono i seguenti:

- ***style*** lo stile grafico dell'html;
- ***script*** codice per far funzionare dinamicamente il sito;
- ***title*** il titolo del sito, il nome del sito può riportare informazioni utili riguardanti la categoria stessa;
- ***meta*** definisce i metadati all'interno del documento html, ovvero le informazioni sui dati. Questo tag non compare spesso e può presentare parole concatenate e/o con abbreviazioni;

- ***head*** contenuto iniziale della pagina; difficilmente si troveranno informazioni adeguate ciò che ci interessa maggiormente è il *body*;
- ***[document]*** in che formato è salvata la pagina, come "html", "xml" etc..

In base al loro utilizzo, dall'html vengono rimossi i seguenti tag: ***style, script, meta, head, [document]*** e, nei rimanenti, viene estratto il testo con BeautifulSoup.

In figura 4.3 l'output ottenuto dando in input a BeautifulSoup il codice di figura 4.1.

```
Toway Sito di eventi, concerti e tanto altro! Progetto Tecnologie Web - A.A. 2019/2020 Autore: Bazzocchi Luca
```

Figura 4.3: Estrazione del testo dall'html di esempio.

5. **Eliminazione degli html che dentro i tag non presentano testo** Ci sono dei casi in cui i siti, oltre al codice, non presentano null'altro e, dall'estrazione del testo, si ottiene solamente una stringa vuota. In questo passaggio vengono rimossi questi elementi.

6. **Sostituzione delle mail e link** In questa fase vengono cercati link (http://...) e mail (mariorossi@...) nel testo che vengono sostituiti con stringhe vuote.

   Funzionamento:

   Esempio: *"Site http://www.worldbnk.org/ contact us wbnk@sm.com"*

   Risultato: *"Site contact us"*

   In futuro si potrebbe estrarre da ogni link il suo dominio senza alcun suffisso (http://www.worldbnk.org/ ⇒ worldbnk) per studiare la ricorrenza di determinati domini all'interno di una stessa categoria di siti. Percorrendo questa via è necessario fare attenzione ai link che puntano allo stesso sito prendendo in considerazione la possibilità di una loro rimozione.

7. **Noise Removal** Giunti a questa fase il testo in html è già stato sottoposto a notevoli cambiamenti rispetto alla sua forma iniziale sono però ancora presenti degli elementi che possono creare anomalie nell'addestramento, come numeri, segni di punteggiatura e caratteri speciali.

   Sul web testi con "rumore" sono facilmente trovabili in quanto solo una minima parte degli scritti sono trattati, elaborati o saggi. Basta considerare i post nei blog o su piattaforme come "Twitter"; nei quali i testi sono scritti velocemente, con abbreviazioni ed errori di ogni genere.

In circostanze come queste il noise removal è una fase importantissima in quanto cerca di riportare le parole alla loro forma originale.

Nel dettaglio, il testo viene frammentato nelle sue parole per l'effettiva rimozione del "rumore", ovvero cifre e caratteri non ascii, possibili rimanenze di markup html e spazi in eccesso. Nella seguente figura 4.4 sono esposti gli adempimenti del noise removal:

	raw word	cleaned word
0	..only..	only
1	(bibliography[54])	bibliography
2	fi----ne!	fi ne
3	<a>trouble	trouble

Figura 4.4: Noise removal in quattro parole differenti.

Un buon input è il primo passo per ottenere un buon output, segue la rappresentazione grafica del processo sull'html di partenza (figura 4.5):

Figura 4.5: Noise removal sull'html d'esempio.

È di fondamentale importanza che le fasi 6 e 7 vengano eseguite in tale ordine; in caso contrario si otterrebbero parole di scarso, se non nullo, valore come ad esempio *"httpwwwciaoit"*.

8. **Case-folding e Stopwords Removal** Per l'addestramento e fasi successive come la lemmatizzazione sono state impiegate le tecniche di:

   - **case-folding** conversione di tutte le lettere in minuscolo. Utile per identificare multiple istanze di una stessa parola come: "Roma", "roma", "ROMA";
   - **stopwords removal** rimozione delle parole dallo scarso significato. Uno dei problemi in una web classification è la presenza di parole

espresse in lingue differenti per ottimizzare questa fase è stata adoperata una stoplist che comprendesse alcune delle lingue più parlate al mondo.

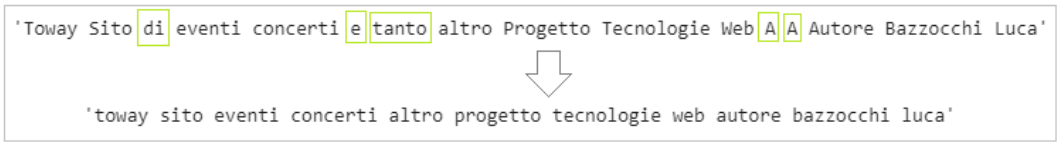

Figura 4.6: Case-folding e stopwords removal sull'html d'esempio.

9. Pulizia del testo Vengono eliminate le parole formate da un numero di caratteri inferiori a N; nel nostro caso $N = 2$.

10. Multi-Language Lemmatizzation Come spiegato nel capitolo 2, la *lemmatizzazione* è la tecnica tramite la quale si sostituiscono le parole con il proprio *lemma*, ovvero la forma base presente nel dizionario.

Quest'ultima fase tiene conto delle diverse lingue presenti nei testi html ed è a sua volta suddivisa in due stadi: *associazione testo-lingua* e utilizzo di un'*opportuna lemmatizzazione*.

In primis, si è considerata la possibilità di determinare la lingua di un testo tramite lo stesso html, facendo uso dell'attributo *"lang"*. Questo approccio presenta due problemi:

1. La mancanza, molto diffusa, di questo attributo nei siti;
2. Può capitare che, in un sito, siano presenti lingue differenti da quella identificata nel sorgente, ad esempio un sito italiano può contenere testo o frasi in inglese. Ci si pone il problema se sia opportuno utilizzare un dizionario italiano per effettuare la lemmatizzazione.

Come soluzione, è stata preferita una rete neurale pre-addestrata per individuare con una certa precisione (maggiore del 75%) la lingua. Trattandosi di un modello probabilistico viene stata stabilita una soglia minima di 20 parole per effettuare la predizione. I testi troppo corti sono stati classificati come "sht" (short) e quelli con bassa affidabilità con "unk" (unknown), tutti gli altri con la loro rispettiva lingua ("it" per italiano, "en" per inglese, "es" per spagnolo e così di seguito).

	text	intent
45708	obscured text	medical
8199	obscured text	blog
29146	obscured text	company
40395	obscured text	financial
27737	obscured text	fashion

	text	intent	language
45708	obscured text	medical	unk
8199	obscured text	blog	la
29146	obscured text	company	en
40395	obscured text	financial	en
27737	obscured text	fashion	pl

Figura 4.7: Determinazione della lingua di un sito.

Prima di passare alla lemmatizzazione, si prendono in esame i testi "sht", quelli succinti, per capirne il contenuto. Da questa breve analisi risulta che, nella maggior parte dei casi, il loro contenuto era simile a *"403 Forbidden"*, presentavano errori o era presente solo il nome del sito.

Si decide quindi la loro rimozione volta anche ad evitare possibili errori nell'addestramento o nei test dei modelli.

Nel secondo stadio è stato usato *spaCy* e tredici dei suoi modelli di lemmatizzazione per comprendere il maggior numero di lingue possibili. La funzione sviluppata controlla per ogni testo se la lingua predetta è presente tra i modelli e, in tal caso, questo viene lemmatizzato.

Figura 4.8: Lemmatizzazione sull'html d'esempio.

4.2 Analisi dei testi

Dopo il data preprocessing, è necessaria una breve analisi sui testi così ottenuti.

Si è partiti con 60.247 sorgenti fino a rimanere con 50.793 elementi significativi. La maggior parte dei testi sono stati rimossi perché troppo corti (minori di 20 parole) e quindi di poca importanza o devianti per la loro categoria.

Con la rimozione del codice anche la lunghezza media dei testi è diminuita: passando da 46.968 a 3.802 caratteri.

Come menzionato nel paragrafo 4.1, è stata predetta con una certa probabilità, la lingua per ogni riga del dataset completo. In presenza di probabilità inferiore al 75%, il suo valore è stato posto a "unk" per non effettuare alcuna lemmatizzazione.

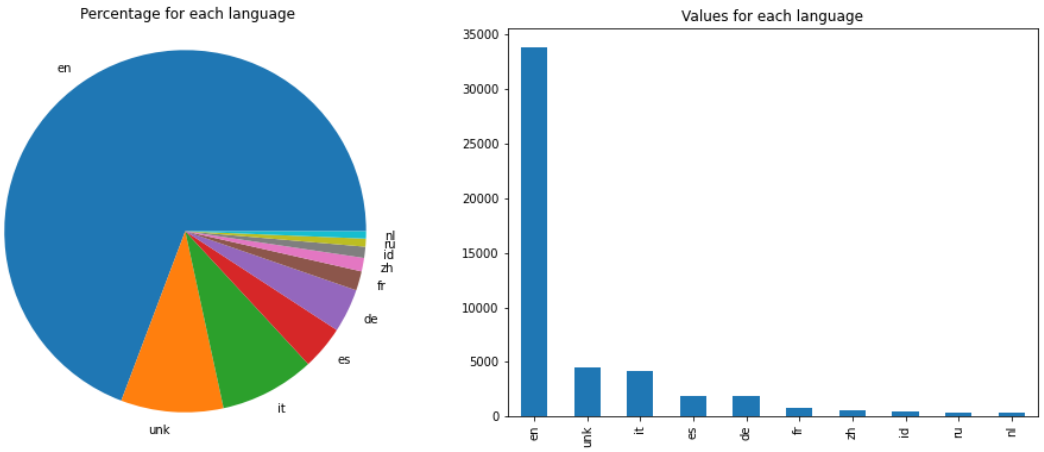

Figura 4.9: Grafico a torta e a barre sulla distribuzione delle prime dieci lingue.

In figura 4.9, notiamo come ogni lingua sia rappresentata mediante la propria sigla (italiano ⇒ it) e la netta prevalenza dell'inglese rispetto a tutte le altre.

Per avere ben presente la distribuzione delle parole più importanti in ogni categoria, è stata utilizzata la libreria *wordcloud*. Questa permette appunto, di creare delle wordclouds, delle "nuvole di parole", le funzioni prendono in input del testo per rilasciare in output le parole; tanto più grandi sarà la rappresentazione grafica della parola e tanto maggiore sarà la loro presenza nel testo.

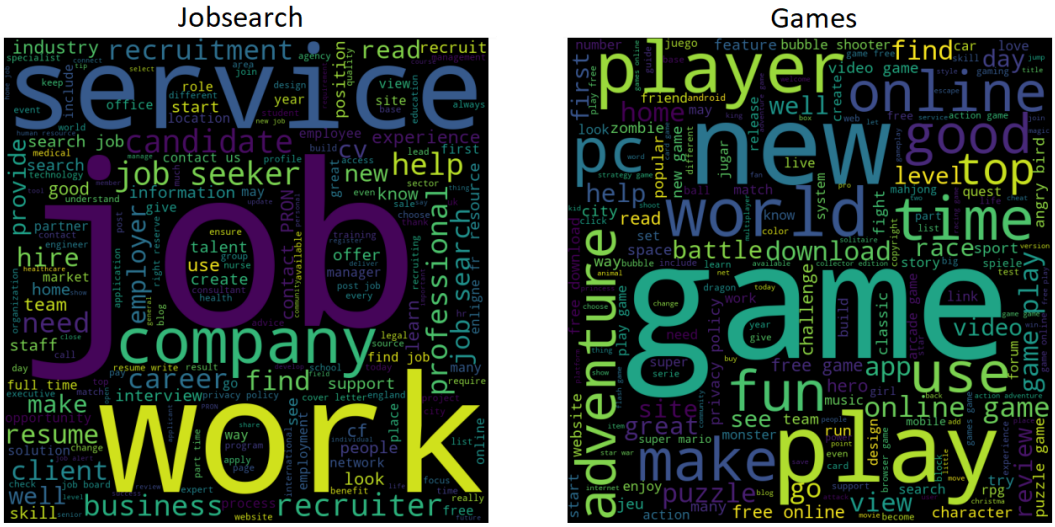

Figura 4.10: Wordclouds delle categorie jobsearch e games.

Dalle due immagini così prodotte (fig. 4.10), si osserva come le parole siano effettivamente rappresentative della propria categoria. Inoltre, in quasi tutti i wordclouds generati la maggior parte delle parole è in lingua inglese, ciò è dovuto alla numerosa presenza di questi elementi, rispetto a parole espresse in altre lingue come si è potuto notare dai grafici a barre e a torta, precedentemente illustrati.

4.3 Trasformazione dei dati

Un algoritmo è addestrato tramite vettori o matrici di numeri ed è, generalmente, specializzato per la risoluzione di determinati problemi; algoritmi diversi possono richiedere input in formati diversi.

Prima dell'inizio delle trasformazioni dei dati, sono stati analizzati dei modelli da impiegare nella classificazione di testi:

- ***Logistic Regression***;
- ***Random Forest***;
- ***Support Vector Machine***;
- ***XGBoost***;
- ***MBERT***;
- ***XLNet***.

Sulla base dei modelli elencati, vengono scelti opportuni algoritmi di conversione dei dati, al fine di ottenere strutture che siano facilmente interpretabili dagli stessi modelli che desideriamo utilizzare.

Possiamo ora analizzare brevemente gli algoritmi di conversione dei dati:

Vector Space Model È una rappresentazione vettoriale composta da N Bag of Words pari al numero dei documenti passati in input, in cui ogni vettore è composto dal numero di occorrenze di ogni termine.

Per i BoW sono stati scelti i seguenti parametri:

- $min\_df = 2$ (ignora i termini con frequenza minore);
- $max\_df = 0.9$ (ignora i termini con frequenza maggiore);
- $ngram\_range = (1, 2)$ (limite inferiore e superiore per gli n-gram estratti).

TF-IDF È una funzione di peso simile a BoW, dove il valore calcolato per ogni termine è definito dall'occorrenza sia all'interno dello stesso documento, che in tutti gli altri (combinazione tra rilevanza locale e globale).

- $min\_df = 2$;
- $max\_df = 0.8$;
- $ngram\_range = (1, 2)$.

Word2Vec Associa a ogni parola il proprio word embedding, ovvero un vettore composto da N feature numeriche in modo tale da permettere calcoli matematici.

Invece che addestrare l'algoritmo con i dati a disposizione si è deciso di utilizzare un modello preaddestrato: il "GoogleNews-vectors-negative300", ciò perché i dati di partenza non sono sufficienti a creare un modello generale e buono di word2vec. In questo modo si possono ottenere in modo veloce, efficiente ed efficace vettori dalle parole dal testo di input.

Doc2Vec Prende le basi dal modello word2vec per creare una rappresentazione numerica di documenti a prescindere dalla loro lunghezza, piuttosto che utilizzare le singole parole. La versione scelta è una combinazione di **Distribuited Memory** (DM) e **Distribuited Bag of Words** (DBOW) al fine di migliorarne l'accuratezza.

Per doc2vec sono stati scelti i seguenti parametri:

- $vector\_size = 300$ (dimensione del vettore delle feature ottenuto);

- *window* = 10 (distanza massima di predizione dalla parola corrente);
- *dm_mean* = 1 (viene utilizzata la media invece che la somma dei vettori);
- *alpha* = 0.065 (il learning rate iniziale);
- *negative* = 5 (impiego del campionamento negativo per specificare quante "noise words" usare);

BertTokenizer Diversa è invece la trasformazione dei dati per MBERT e XLNet, i quali richiedono un proprio tokenizer.

In MBERT, i dati vengono sottoposti ad una ulteriore fase di preprocessing: ogni testo è suddiviso in sequenze da 256 token, ognuno aggiungendo opportuni caratteri riconosciuti da BERT. *"[CLS]"* indica che si tratta di un problema di classificazione e *"[SEP]"* la fine di un testo.

MBERT sta per *Multilingual BERT*, questo, rispetto al BERT originale, presenta un vocabolario con oltre cento lingue diverse. Con il suo Tokenizer, ogni parola viene convertita in un numero in base al vocabolario. Si utilizza questa nuova versione proprio perché il dataset presenta siti in lingue diverse.

XLNetTokenizer Come per BERT, XLNet presenta un proprio tokenizer. Questo viene utilizzato per dividere il testo in token che abbiano una corrispondenza nel vocabolario di XLNet necessaria per farli riconoscere al modello stesso.

L'addestramento avverrà con sequenze tra 128 e 192 parole, abbiamo usato questo range poiché il 192 rappresenta il limite massimo che il nostro hardware può reggere.

4.4 Modelli di learning

Dopo il preprocessing, i dati sono stati divisi in training set e test set con una proporzione 70:30. Sui dati di training sono state utilizzate le diverse *encoding technique* descritte nel paragrafo 4.2 con la successiva trasformazione anche del test set.

In questa sezione verranno analizzati i modelli di learning usati per l'estrazione di conoscenza, per ogni algoritmo verranno illustrati i parametri testati e i risultati ottenuti, con quale accuratezza, precision, recall e f1-score.

Nei primi quattro modelli, il test set non è stato a sua volta suddiviso in validation e test set perchè si è utilizzata una *Grid Search* con *Stratified K Fold*. In Bert e XLNet invece, l'ultimo set è stato frazionato in due parti per consentire il fine tuning dei parametri, ottenendo così la seguente proporzione: 72:18:10.

Ogni modello è stato addestrato con tutte e 22 le classi bilanciate effettuando un'under-sampling, portando così a 373 elementi per categoria. Non sono stati applicati ulteriori algoritmi per la risoluzione delle classi sbilanciate, quali i metodi di over-sampling come SMOTE. Questi erano stati presi in considerazione ma a causa delle limitazioni hardware, software e temporali, non è stato possibile effettuare prove su tutti gli algoritmi selezionati.

Lo scopo è trovare il modello con l'F1-score migliore e che quindi sia rappresentativo di tutte le categorie senza trascurare le classi peggiori.

Logistic Regression

La regressione logistica è una tecnica utilizzata nei problemi di classificazione che calcola una funzione sigmoidale al fine di ottenere l'iperpiano di separazione tra le classi.

Parametri testati:

- *solver* (algoritmo usato per l'ottimizzazione del problema);
- *C* (parametro inverso alla regolarizzazione);
- *penality* (il tipo di norma usata nella penalizzazione);
- *l1_ratio* (indica la percentuale di penalità l1 rispetto a l2 nella elastic-net).

Risultati:

	Accuracy	F1-score	Precision	Recall
Bow	0.596	0.564	0.580	0.604
TfIdf	0.638	0.628	0.629	0.655
Word2Vec	0.645	0.619	0.646	0.607
Doc2Vec	0.573	0.560	0.563	0.570

Nella logistic regression, l'utilizzo di TfIdf porta a prestazioni migliori, trovando la seguente combinazione:

[solver = 'saga', C = 10.0, penality = 'l2']

I valori così ottenuti saranno poi confrontati con i migliori risultati degli altri modelli.

Random Forest

La foresta casuale [18] è un algoritmo di apprendimento supervisionato che crea alberi di decisione a partire da esempi casuali di input. Ogni albero esegue una predizione e la soluzione è data dalla media dei voti di tutti gli alberi di decisione.

Parametri testati:

- *n_estimators* (numero di alberi nella foresta);
- *max_depth* (la massima profondità degli alberi);
- *min_samples_leaf* (numero minimo di campioni richiesti per trovarsi in un nodo foglia);
- *min_samples_split* (numero minimo di campioni richiesti per dividere un nodo interno).

Risultati:

	Accuracy	F1-score	Precision	Recall
Bow	0.440	0.421	0.475	0.454
TfIdf	0.549	0.534	0.567	0.560
Word2Vec	0.569	0.515	0.731	0.474
Doc2Vec	0.465	0.368	0.537	0.348

Nella random forest l'utilizzo di TfIdf porta ancora alle prestazioni migliori, però i risultati ottenuti sono molto più bassi rispetto alla logistic regression. Si notano inoltre in word2vec e doc2vec valori molto più alti in precision rispetto al recall. Parameri ottimali:

[n_estimators = 300, max_depth = 10, min_samples_leaf = 2, min_samples_split = 2]

Support Vector Machine (SVM)

Support Vector Machine [19] è un algoritmo di apprendimento supervisionato che costruisce un iperpiano ottimale per la separazione delle classi. Si basa su due concetti fondamentali:

- **Support vector**: i punti più vicini all'iperpiano;

- **Margine**: distanza dall'iperpiano dei vettori di supporto di classi differenti.

Parametri testati:

- *kernel* (specifica il tipo di funzioni kernel usate dall'algoritmo);
- *C* (parametro inverso alla regolarizzazione);

Risultati:

	Accuracy	F1-score	Precision	Recall
Bow	0.582	0.581	0.592	0.605
TfIdf	0.628	0.624	0.638	0.641
Word2Vec	0.649	0.621	0.674	0.605
Doc2Vec	0.557	0.543	0.550	0.566

In SVM i risultati ottenuti con TfIdf e Word2Vec sono molto simili. Nel primo caso i risultati sono più distribuiti, mentre nel secondo è la precision che ottiene i valori migliori, proprio come nella Random Forest. Lo scopo è ottenere dai modelli valori che siano i più equilibrati possibili, perciò è stato scelto TfIdf. I suoi parametri sono i seguenti:

[kernel = 'linear', C = 1]

XGBoost

L'algoritmo XGBoost [20] deriva dall'acronimo eXtreme Gradient Boosting ponendo l'attenzione sull'importanza dell'ottimizzazione e i principi dei modelli **gradient boosting**. Questi si basano sull'idea che l'unione di tanti modelli semplici (**weak learner**) possano formarne uno molto più potente (**strong learner**). L'algoritmo ad ogni iterazione calcola il k-esimo albero migliore da aggiungere alla struttura e la sua efficacia è data principalmente da due espedienti:

1. calcolo dei gradienti del secondo ordine, ossia le secondarie derivate parziali della funzione loss;
2. la regolarizzazione L1, L2.

Parametri testati:

- *eta* (learning rate);
- *min_child_weight* (somma minima del peso di una istanza);

- *max_ depth* (profondità massima degli alberi);
- *n_ estimators* (numero di round di boost);
- *alpha* (parametro per la regolarizzazione L1).

Risultati:

	Accuracy	F1-score	Precision	Recall
Bow	0.618	0.601	0.592	0.635
TfIdf	0.619	0.602	0.593	0.634
Word2Vec	0.598	0.581	0.573	0.616
Doc2Vec	0.535	0.516	0.508	0.546

Con questi risultati si può osservare come, a volte, si riescano ad ottenere risultati migliori con le rappresentazioni più semplici; ciò che può aver influito su questi risultati è dato da testi non sempre grammaticalmente corretti e in particolare per doc2vec il numero di istanze usate per ogni classe. In un problema riguardante il natural language processing, un addestramento con circa 35000 elementi in totale (il 70% dei dati processati), è poco significativo.

Gli F1-score in Bow e TfIdf delle 22 categorie, prese singolarmente, sono molto simili ma leggermente migliori nel secondo caso e per questo è stato scelto tale modello. Questi specifici valori verranno poi discussi nel capitolo successivo.

[eta = 0.01, min_child_weight = 8, max_depth = 6, n_estimators = 300, alpha = 1e-04]

BERT

BERT (Bidirectional Encoder Representations from Transformers) [21] è un modello addestrato su miliardi di documenti con lo scopo di risolvere problemi di natural language processing.

La sua architettura è basata su quella del Transformer, il quale presenta un encoder per la lettura e codifica del testo in input ed un decoder per la predizione. Di questi due elementi, l'algoritmo utilizza solamente multipli encoder.

La chiave che differenzia BERT dai suoi predecessori è la bidirezionalità, che permette al modello di imparare il contesto di una parola basandosi su tutte le sue adiacenze. La scelta innovatrice è stata quella di addestrare l'architettura invece che su un modello di linguaggio, su un "modello di linguaggio mascherato" (**masked language model**) con in aggiunta la "previsione della frase successiva" (**next sentence prediction**).

Il masked language model chiede al modello di prevedere non la parola successiva, ma parole casuali all'interno della sequenza; le quali potrebbero o meno essere rimpiazzate da un token *'<MASK>'*.

Più specificatamente il modello utilizzato è stato MBERT; sempre basato sull'algoritmo originale ma pensato per avere la multilingua invece del solo inglese.

Parametri testati:

- *batch_size* (numero di campioni per l'aggiornamento del gradiente);
- *epochs* (numero di epoche per addestrare il modello);
- *epsilon* (piccola costante per la stabilità numerica);
- *learning_rate*.

Sia in BERT che XLNet, prima dell'utilizzo del proprio tokenizer, è stata effettuata un'ulteriore fase di preprocessing. Tramite TfIdf sono stati calcolati i pesi di tutte le parole suddividendo i testi per categoria e successivamente sono state rimosse le parole con valore inferiore a 0.04.

Con vari test si è visto che, tramite questa strategia, i due modelli riescono ad ottenere risultati nettamente migliori, riducendo anche il tempo di addestramento. Ciò avviene a causa della lunghezza fissa e non variabile delle sequenze in fase di training: la conoscenza in questo modo sarà racchiusa in testi più brevi, i quali dovranno poi essere ridotti o frazionati.

Visto l'utilizzo di Keras per la creazione di questo modello, oltre al 'BertModelLayer' sono stati creati altri cinque livelli per il fine-tuning. Di questi vi è un livello 'lambda', due di 'dropout' pari a 0.5%, in modo tale da far perdere parte della conoscenza acquisita e obbligare il modello a imparare più adeguatamente e due livelli 'densi' con le funzioni di attivazione *tanh* e *softmax*.

Ogni test è stato effettuato su cinque Stratified Fold, di seguito riportiamo il miglior valore ottenuto dal test set.

Risultati:

	Accuracy	F1-score	Precision	Recall
1° Test	0.582	0.555	0.564	0.589
2° Test	0.565	0.523	0.522	0.578

Pur essendo una rete neurale molto avanzata che ha conseguito lo stato dell'arte a fine 2018, BERT non è riuscito ad ottenere buoni risultati ciò è principalmente dovuto ad una distribuzione irregolare delle lingue e dalla scarsa quantità dei dati rispetto al numero di categorie presenti. I migliori parametri trovati sono i seguenti:

[batch_size = 16, epochs = 10, epsilon = 1e-07, learning_rate = 2e-05]

XLNet

XLNet [22] è un metodo di pre-addestramento autoregressivo generalizzato creato da Google basandosi su BERT. Lo scopo era di eliminare alcuni errori del suo predecessore.

XLNet nella modellazione del linguaggio introduce la permutazione. Qui a differenza dei modelli linguistici tradizionali, i token non sono predetti in modo sequenziale, ma in ordine casuale. Questa tecnica aiuta il modello a imparare relazioni bidirezionali e, quindi, a gestire e capire meglio la dipendenza tra le parole.

Brevemente, l'algoritmo usa un metodo detto AR (**autoregressive**) che utilizza una parola contestuale per prevedere quella successiva. Qui la parola contestuale è vincolata a due direzioni, avanti e indietro. In tutto ciò, viene inserita la permutazione per introdurre la bidirezionalità.

Parametri testati:

- *max_len* (massima lunghezza delle sequenze);
- *batch_size* (numero di campioni per l'aggiornamento del gradiente);
- *epochs* (numero di epoche per addestrare il modello);
- *epsilon* (piccola costante per la stabilità numerica);
- *learning_rate*.

Dei parametri sopra elencati sono state scelte tre combinazioni e per ognuna è stata fatta una StratifiedKFold a cinque fold. Il modello migliore per ogni test è stato poi validato tramite lo stesso test set. Inoltre i dati, prima dell'addestramento, hanno subito un'ulteriore fase di preprocessing come spiegato in BERT.

	max_len	batch_size	epochs	epsilon	learning_rate
1° Test	128	32	10	1e-06	2e-05
2° Test	128	48	10	1e-05	3e-05
3° Test	192	32	10	1e-06	3e-05

Risultati:

	Accuracy	F1-score	Precision	Recall
1° Test	0.623	0.619	0.620	0.642
2° Test	0.602	0.597	0.598	0.639
3° Test	0.637	0.635	0.634	0.663

Da questi risultati si intuisce l'importanza di lunghe sequenze in input rispetto a quelle corte per estrarre conoscenza. Infatti, la lunghezza media dei testi processati era intorno ai 3800 caratteri, molto di più di 128 in parole.

Rispetto agli altri modelli, con XLNet si sono ottenuti buoni risultati che verranno discussi nella sezione successiva.

[max_len = 192, batch_size = 32, epochs = 10, epsilon = 1e-06, learning_rate = 3e-05]

Capitolo 5

Discussione dei risultati

Nel capitolo precedente sono state analizzate tutte le fasi concernenti lo sviluppo: riassumendo, siamo partiti da una fase della disciplina del natural language processing denominata data preprocessing; qui, con l'uso del linguaggio di programmazione python, sono stati rimossi tutti i valori duplicati o mancanti, è stato estratto il testo tramite la libreria BeautifulSoup con successiva rimozione del "rumore" rimasto e delle stopwords e vi è stata apportata una lemmatizzazione multilingua. In seguito, trattandosi di un problema di classificazione volto quindi all'ottenimento di una funzione di separazione delle classi, sono stati scelti specifici modelli di learning e i rispettivi tokenizer. In questo modo sono state ottenute e poi addestrate strutture facilmente interpretabili dai modelli.

Durante tutte queste fasi di sviluppo, sono state portate alla luce alcune problematiche come la comprensione di quali testi fossero realmente significativi e quali mal procurati e la scarsità dei dati rispetto all'ampio numero delle categorie. L'eliminazione degli html errati è stata affrontata controllando la presenza di pattern specifici nei sorgenti e successivamente con la verifica del numero di caratteri e parole; la scarsità dei dati non è direttamente risolvibile tramite la manipolazione degli stessi ma solo parzialmente, utilizzando tecniche di over-sampling. Per limitazioni hardware e temporali, queste ultime procedure non sono state effettuate, ma sulla base dei risultati già prodotti sarà possibile rimuovere questa complicazione in futuro.

Ora che sono state denotate meglio le fasi precedenti, è il momento di analizzare e discutere i risultati prodotti dal modello con l'F1-score maggiore, ovvero quello con la media armonica tra precision e recall più elevata, portando in primo piano i valori ottenuti per ogni categoria al fine di capire quanto il modello sia rappresentativo delle classi addestrate.

I migliori risultati sono stati ottenuti con il modello XLNet e, come visto precedentemente, con i seguenti parametri:

[max_len = 192, batch_size = 32, epochs = 10, epsilon = 1e-06, learning_rate = 3e-05]

Nella tabella seguente sono riportati tutti i valori divisi per categoria:

Category	Precision	Recall	F1-score
adult	0.50	0.65	0.57
porn	0.74	0.73	0.73
games	0.68	0.75	0.72
blog	0.47	0.64	0.54
vacation	0.87	0.71	0.78
audio-video	0.34	0.44	0.39
company	0.70	0.44	0.54
domainforsale	0.31	0.85	0.46
ecommerce	0.40	0.46	0.43
fashion	0.29	0.53	0.38
ict	0.54	0.50	0.52
shopping	0.41	0.46	0.43
sports	0.51	0.66	0.57
music	0.78	0.70	0.74
news	0.55	0.57	0.56
proxy	0.67	0.58	0.62
financial	0.66	0.87	0.75
gambling	0.94	0.88	0.90
jobsearch	0.85	0.89	0.87
medical	0.88	0.83	0.85
dating	0.96	0.73	0.83
education	0.91	0.71	0.80

I valori su cui ci concentreremo sono quelli presenti nella colonna F1-score; la colonna precision, ossia il rapporto tra gli elementi predetti correttamente e quelli predetti della specifica classe e la colonna recall, che indica quanti degli elementi predetti sono stati predetti correttamente, sono presenti per dare informazioni aggiuntive alle medesime categorie.

Si può subito notare la varietà dei dati in scala, che variano tra 0.38 e 0.90, la prima cosa da fare è, quindi cercare di capire il motivo di valori così bassi o così alti.

Le categorie “fashion” e “audio-video” sono quelle che presentano valori minori, questa valutazione è data principalmente da due motivi:

1. le istanze non sono sufficientemente rappresentative della loro stessa categoria;

2. vi sono dipendenze tra categorie di ambito simile o con argomenti e parole presenti in altre classi.

Il primo problema è risolvibile aumentando il numero di elementi per classe; mentre per risolvere il secondo potrebbe essere necessaria una rivalutazione delle classi, con una possibile eliminazione di alcune di esse, oppure una rimozione totale o parziale di determinati vocaboli in comune, con il rischio di perdere dati significativi e, conseguentemente, una diminuzione nell’accuratezza della predizione.

Già solamente leggendo il nome della categoria “audio-video” sono capibili le dipendenze parziali che si avranno con i testi di “music”, “games” e altri generi; “fashion” invece con “shopping” ed “ecommerce”. Queste dipendenze portano il modello a confondere le istanze le une con le altre, producendo così predizioni errate anche se di ambito simile. In contrapposizione, le classi con lo score maggiore sono anche quelle che presentano degli ambiti più specifici rispetto a tutte le altre, avendo così molte parole proprie e uniche di quella determinata categoria.

Il secondo approccio, quello della rimozione delle parole più frequenti tra categorie simili tramite l’aggiunta di queste alle stopwords, è stato testato prima con una eliminazione parziale e poi totale. In entrambi i casi i risultati ottenuti non sono stati abbastanza soddisfacenti e per questo non sono stati trattati in questa tesi. Soprattutto la scelta della rimozione totale ha portato ad un abbassamento dell’accuratezza di circa due/tre punti percentuale; ciò è stato causato probabilmente dai testi più corti ove, tolte le parole più importanti, non sono state più riconoscibili nella loro categoria di appartenenza.

5.1 Confronto tra modelli

Vista la valutazione, si vogliono confrontare tra di loro tutti gli f1-score di ogni categoria e per ogni tipo di modello, cercando così possibili anomalie o cambiamenti in positivo o negativo tra tutti i valori e la successiva discussione se XLNet sia stata effettivamente l’architettura migliore tra tutte quelle selezionate.

category	log-reg	rand-for	svm	xgboost	bert	xlnet
adult	0.60	0.61	0.61	0.53	0.61	0.57
porn	0.66	0.66	0.64	0.65	0.64	0.73
games	0.74	0.64	0.75	0.71	0.74	0.72
blog	0.55	0.46	0.53	0.53	0.52	0.54
vacation	0.74	0.67	0.75	0.69	0.59	0.78
audio-video	0.27	0.13	0.26	0.20	0.00	0.39
company	0.55	0.33	0.55	0.51	0.52	0.54
domainforsale	0.49	0.47	0.47	0.44	0.39	0.46
ecommerce	0.40	0.33	0.40	0.39	0.00	0.43
fashion	0.39	0.24	0.35	0.43	0.35	0.38
ict	0.53	0.44	0.53	0.51	0.42	0.52
shopping	0.43	0.20	0.42	0.38	0.32	0.43
sports	0.45	0.39	0.46	0.44	0.49	0.57
music	0.74	0.71	0.73	0.73	0.72	0.74
news	0.59	0.52	0.59	0.61	0.54	0.56
proxy	0.63	0.60	0.59	0.62	0.60	0.62
financial	0.81	0.68	0.83	0.73	0.83	0.75
gambling	0.86	0.87	0.88	0.87	0.86	0.90
jobsearch	0.84	0.51	0.84	0.77	0.67	0.87
medical	0.83	0.74	0.84	0.84	0.83	0.85
dating	0.88	0.84	0.88	0.88	0.78	0.83
education	0.83	0.70	0.83	0.79	0.80	0.80

Nella tabella sono presenti due anomalie, entrambe date dal modello BERT e in corrispondenza di alcune delle categorie peggio classificate: sia "audio-video" che "ecommerce" sono state completamente ignorate dall'algoritmo portando un f1-score pari a 0. Pur trattandosi di un addestramento bilanciato, si è ottenuto lo stesso fenomeno dell'**accuracy paradox** dove, nei problemi con classi sbilanciate, l'algoritmo decide di ignorare la classe minore così da ottenere un'accuratezza migliore prevedendo solo e unicamente le classi con più istanze. BERT deve aver deciso che ignorando "audio-video" ed "ecommerce", avrebbe ottenuto statistiche migliori, cosa che noi non vogliamo assolutamente accada in un modello.

Escluse le due anomalie, i valori sono piuttosto stabili e proporzionali ai risultati generali degli stessi modelli in cui sono presenti senza grandi cambiamenti tra un modello e l'altro; inoltre le classi peggiori e migliori sono quasi sempre le stesse.

Dai risultati ottenuti si riconferma XLNet come modello migliore tra quelli scelti per questo specifico caso di studio, ovvero la classificazione di siti web tramite il codice sorgente della loro home-page.

Conclusioni e sviluppi futuri

Lo scopo di questa tesi è scoprire e capire i migliori metodi di Text Mining utili per ottenere un input facilmente interpretabile dai calcolatori, basandosi sui sorgenti html delle home-page dei siti web, e il successivo ottenimento di un algoritmo intelligente per classificare pagine web e che quindi sia in grado di distinguere la categoria di un testo html precedentemente processato.

Tutte le prove e gli studi hanno portato alle fasi di preprocessing presenti nel paragrafo 4.1 e alla scelta del modello XLNet con i parametri trovati nel paragrafo 4.4. Si è potuto anche osservare che, l'ausilio di strutture complesse come word2vec e doc2vec non sempre ottengono i risultati migliori rispetto alle più semplici come Bow e TfIdf, infatti entrambe presentano difficoltà con testi in multilingua o testi non grammaticalmente corretti.

In base alla quantità stessa dei dati a disposizione, i risultati sono stati ritenuti soddisfacenti anche se migliorabili tramite l'utilizzo di più test e l'addestramento anche su modelli non discussi come GPT-2 ed ERNIE 2.0.

A causa della natura stessa dei dati a disposizione, la parte che ha richiesto più lavoro è stata quella riguardante il processamento del testo; qui si sono ottenuti risultati dapprima sufficienti per poi passare a discreti fino ad essere ritenuti soddisfacenti per il quantitativo di dati a disposizione.

Lo sviluppo futuro però non si ferma alla semplice aggiunta di nuovi dati per migliorare l'accuratezza, ma potrebbe anche l'implementazione di nuove tecniche. Un'idea attuabile, una volta ottenuti dati sufficienti, sarebbe quella di applicare la tecnica "divide et impera" per suddividere il problema in uno più piccolo e semplice e, forse, anche più ottimizzabile:

1. Le 22 o più categorie iniziali verrebbero divise in 5/6 macrocategorie in modo tale da ridurre le classi totali ma sempre tenendo da conto la loro classe iniziale;

2. I testi sarebbero poi addestrati, producendo risultati generalmente superiori perché l'algoritmo si focalizzerà su un numero inferiore di categorie;

3. Creato il macro-modello, si addestreranno tanti modelli quante sono le macrocategorie scelte una per ognuna e queste, daranno l'effettiva previsione di ogni istanza.

Perciò il macro-modello predirà la macrocategoria di un testo e, successivamente, lo specifico modello di quella macrocategoria predirà la vera classe del testo. Questa tecnica richiede un grande lavoro per addestrare gli algoritmi, poiché avrà necessità di un numero di algoritmi, pari al numero di macrocategorie più uno, ma dovrebbe ottenere risultati migliori rispetto all'utilizzo di un unico modello e, nel caso di poche classi sbilanciate, il problema sarebbe posto in modelli con meno categorie rispetto a quello intero.

Questa tesi è solo il primo passo della *User Internet Protection mediante Classificazione di Pagine Web con Text Mining*, il passo successivo sarà l'aggiunta di dati ai modelli e la creazione di un'applicazione che sia in grado di utilizzare questi stessi metodi, apportando eventuali modifiche e migliorie in modo da fornire all'utente una navigazione sulla rete libera dall'invasione di malware, virus ed evitando l'apertura di siti sgraditi e poco appropriati.

Ringraziamenti

So di non essere molto bravo con le parole e perciò cercherò di esprimere al meglio la mia gratitudine per aver incontrato persone speciali nella mia vita, se dovessi citare nomi e virtù di tutte le persone importanti per me questa tesi non avrebbe mai fine spero quindi che nessuno ne abbia a male.

Un primo ringraziamento va a mia mamma che mi è sempre stata accanto in tutte le mie decisioni spronandomi ogni qualvolta mi sia capitato di sbagliare, la ringrazio di cuore di tutto. Ringrazio Davide che nei momenti di bisogno è sempre stato presente a dare una mano e che non ha mai smesso di appoggiarmi. Ringrazio mia sorella Rebecca con cui nonostante caratterialmente ci troviamo agli antipodi, ho potuto ridere e scherzare in tutti questi anni. Ringrazio mio babbo che è sempre stata una figura di riferimento per lo studio con tutte le sue nozioni di storia e geografia.

Non possono mancare i ringraziamenti ai miei nonni che sono sempre stati gentili e disponibili con me e che in tutti questi anni hanno sempre risposto anche alle mie più assurde richieste. Ringrazio i miei zii che mi hanno sempre voluto bene e supportato.

Un ringraziamento va anche al mio relatore e professore Gianluca Moro che mi ha fatto entusiasmare a tutti gli argomenti riguardanti l'intelligenza artificiale e machine learning e pur passando un momento di difficoltà, mi ha dato la possibilità di scrivere questa tesi.

Ringrazio anche Francesco Collini che, insieme alla sua azienda, mi ha sostenuto e aiutato negli ultimi progetti informatici che ho implementato.

Infine, un grande ringraziamento va a tutti i miei amici con cui ho potuto giocare, ridere, arrabbiare, scherzare in tutti questi anni ma anche girare insieme per l'Europa, creare e modificare giochi; per non parlare di Giacomo Casadei, amico e collega universitario con cui mi sono confrontato e divertito durante questi tre anni.

Grazie ancora a tutti per esserci stati e avermi accompagnato fino a questo momento importante della mia vita.

Bibliografia

[1] John McCarthy, Marvin Minsky, Nathaniel Rochester, and Claude E. Shannon. A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955. *AI Mag.*, 27(4):12–14, 2006.

[2] H. Kagermann, J. Helbig, A. Hellinger, and W. Wahlster. *Umsetzungsempfehlungen für das Zukunftsprojekt Industrie 4.0: Deutschlands Zukunft als Produktionsstandort sichern ; Abschlussbericht des Arbeitskreises Industrie 4.0.* Forschungsunion, 2013.

[3] Erik Brynjolfsson and Andrew McAfee. *The second machine age: work, progress, and prosperity in a time of brilliant technologies, 1st Edition.* Norton, 2014.

[4] Arthur L. Samuel. Some studies in machine learning using the game of checkers. *IBM J. Res. Dev.*, 3(3):210–229, 1959.

[5] Tom M. Mitchell. *Machine learning, International Edition.* McGraw-Hill Series in Computer Science. McGraw-Hill, 1997.

[6] Augustin Cauchy. Méthode générale pour la résolution des systemes d'équations simultanées. *Comp. Rend. Sci. Paris*, 25(1847):536–538, 1847.

[7] Warren S. McCulloch and Walter H. Pitts. A logical calculus of the ideas immanent in nervous activity. In Margaret A. Boden, editor, *The Philosophy of Artificial Intelligence*, Oxford readings in philosophy, pages 22–39. Oxford University Press, 1990.

[8] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65 6:386–408, 1958.

[9] Li Deng and Dong Yu. Deep learning: Methods and applications. *Foundations and Trends in Signal Processing*, 7(3-4):197–387, 2014.

[10] Christopher Manning and Hinrich Schutze. *Foundations of statistical natural language processing*. MIT press, 1999.

[11] Alan M. Turing. Computing machinery and intelligence. In Margaret A. Boden, editor, *The Philosophy of Artificial Intelligence*, Oxford readings in philosophy, pages 40–66. Oxford University Press, 1990.

[12] Yoshua Bengio and Yann LeCun, editors. *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, 2013.

[13] Quoc V. Le and Tomas Mikolov. Distributed representations of sentences and documents. *CoRR*, abs/1405.4053, 2014.

[14] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. *Learning Representations by Back-Propagating Errors*, page 696–699. MIT Press, Cambridge, MA, USA, 1988.

[15] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, 1997.

[16] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[17] Charles Babbage and Martin Campbell-Kelly. *Passages from the Life of a Philosopher: Passages from the Life of a Philosopher*. Rutgers University Press, USA, 1994.

[18] Tin Kam Ho. Random decision forests. In *Third International Conference on Document Analysis and Recognition, ICDAR 1995, August 14 - 15, 1995, Montreal, Canada. Volume I*, pages 278–282. IEEE Computer Society, 1995.

[19] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Mach. Learn.*, 20(3):273–297, 1995.

[20] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In Balaji Krishnapuram, Mohak Shah, Alexander J. Smola, Charu C. Aggarwal, Dou Shen, and Rajeev Rastogi, editors, *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 785–794. ACM, 2016.

[21] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[22] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. *CoRR*, abs/1906.08237, 2019.