



UNIVERSITÀ DEGLI STUDI DI ROMA TOR VERGATA

Macroarea di Scienze Matematiche, Fisiche e Naturali

Corso di Laurea in Informatica

A.A. 2020/2021

Tesi di Laurea Triennale

UN FRAMEWORK PER IL NATURAL LANGUAGE PROCESSING:
ANALISI PRESTAZIONALE PER LA RISOLUZIONE DI TASK DI
TEXT CLASSIFICATION E NAMED ENTITY RECOGNITION IN
AMBIENTE DISTRIBUITO

Relatore

Prof. Roberto Basili

Co-Relatore

Prof. Danilo Croce

Candidato

Manuel Di Lullo

Vorrei dedicare qualche riga a tutte le persone che mi hanno permesso di arrivare fin qui e di portare a termine questo percorso.

Vorrei innanzitutto ringraziare il mio relatore Basili Roberto, il mio correlatore Croce Danilo ed il loro collega Antonio Scaiella che mi hanno seguito, passo dopo passo, in questo percorso, grazie ai quali ho acquisito un metodo di lavoro che sicuramente replicherò in futuro.

Ringrazio infinitamente i miei genitori ed i miei fratelli, che hanno reso possibile tutto questo. Grazie per avermi trasmesso tutti i principi di cui vado fiero, per poi lasciare che seguissi la mia strada alla ricerca di ciò che mi rende felice.

Ringrazio i miei parenti, che, nonostante non mi senta costantemente con tutti loro, mi hanno sempre trasmesso immenso affetto.

Ringrazio le piccole Martina e Chiara per il loro amore incondizionato che mi ha dimostrato quanto l'amore sia qualcosa di spontaneo che non segue regole.

Grazie a tutti i miei colleghi dell'università, ed in particolar modo Andrea, Bogdan, Davide, Gianluca, Marco e Mauro, che hanno condiviso con me gioie e dolori, i deliri prima di un'esame, le feste dopo averlo passato ed anche la rabbia per non esserci riusciti.

Grazie a Davide, Matteo e Francesco che, nonostante le settimane che passano prima di risentirci, sono sempre lì per dimostrarti che le vere amicizie non finiscono mai.

Ringrazio la Vertigo Crew, che mi ha visto crescere e mi ha insegnato a dare il meglio in tutto ciò che faccio. Grazie perché, da compagni, siete diventati per me una seconda famiglia.

Un immenso grazie a Rossi, che sono il nido nel quale so di potermi rifugiare nei momenti no. Grazie per aver ascoltato i miei sfoghi e per tutti i momenti di spensieratezza che mi avete donato. Grazie per tutti i vostri "Io ci sono" mai detti ma che si fanno sempre sentire.

Un altrettanto grande grazie ai miei amici di Mbrasa per essere da sempre al mio fianco. Grazie per tutte le notti passate insieme, soprattutto quando avevamo la sveglia a distanza di due ore. Grazie per le ore passate a giocare insieme, per gli infiniti consigli dispensati e per essere i primi a corrermi incontro nei momenti di difficoltà.

Non posso non ringraziare la persona che, da quasi 7 anni, sopporta le mie ansie e supporta il mio lavoro, la mia fidanzata Francesca. Grazie perché mi dai la forza per inseguire i miei sogni e mi fai sentire una persona migliore di quella che sono. Grazie per essere sempre il mio posto felice.

Concludo ringraziando me stesso. Mi ringrazio per non aver ceduto quando pensavo di non farcela e per essermi circondato di tutte queste magnifiche persone.

Indice

Abstract	i
1 Introduzione	1
1.1 Abstract	1
1.2 Struttura della tesi	4
2 Il processo di elaborazione linguistica	6
2.1 Natural Language Processing	6
2.1.1 Il Linguaggio Naturale	6
2.1.2 Il processo NLP	7
2.1.3 Ambiguità	12
2.2 Campi d'applicazione	13
2.3 Reti neurali per problemi linguistici	15
2.3.1 Primi approcci	15
2.3.2 Percettrone	17
2.3.3 Deep Learning	19
2.4 Language Modeling	20
2.4.1 Word Embedding	21
2.4.2 Embedding statico: GloVe	22

2.4.3	Embedding contestualizzati: BERT	23
2.4.4	Embedding contestualizzati: Universal Sentence Encoder	25
3	Una soluzione distribuita	28
3.1	Calcolo Distribuito	28
3.2	Hadoop	30
3.2.1	Cosa è Hadoop	30
3.2.2	Hadoop Distributed File System	31
3.2.3	MapReduce	33
3.2.4	YARN	35
3.3	Apache Spark	38
3.3.1	Cosa è Spark	38
3.3.2	Architettura	39
3.3.3	Spark RDD e DataFrame	40
3.3.4	Directed Acyclic Graphs in Spark	42
3.3.5	Spark Core	44
3.4	Spark NLP	46
3.4.1	Cosa è Spark NLP	46
3.4.2	Come funziona Spark NLP	47
3.4.3	Perchè usare Spark NLP	48
3.5	Configurazione dell'architettura utilizzata	50
3.5.1	Hardware	50
3.5.2	Software	51
4	Sperimentazione	54
4.1	Text Classification	54
4.1.1	Descrizione del task	54

4.1.2	Il dataset	56
4.1.3	Spark NLP per Text Classification	57
4.2	Named Entity Recognition	59
4.2.1	Descrizione del task	59
4.2.2	Dataset in lingua inglese	62
4.2.3	Dataset in lingua italiana	63
4.2.4	Spark NLP per Named Entity Recognition	64
4.2.5	Tabelle riassuntive dataset	66
4.3	Analisi Prestazionale	67
4.3.1	Metriche	67
4.3.2	Performance	69
4.3.3	Tempi di esecuzione	73
4.3.4	Valutazioni	73
Conclusioni		76

Elenco delle figure

2.1	Analisi Lessicale	8
2.2	Analisi Sintattica	9
2.3	Analisi Semantica: esempio di Named Entity Recognition	10
2.4	Il processo NLP	11
2.5	Livelli di ambiguità	13
2.6	Percettrone	17
2.7	Esempio di una rete neurale artificiale	20
2.8	BERT Base e BERT Large	24
3.1	Architettura HDFS	32
3.2	Algoritmo MapReduce	34
3.3	Architettura YARN	37
3.4	Architettura Spark	40
3.5	Operazioni su RDD	41
3.6	Esempio di struttura di un DataFrame	42
3.7	Esempio di Spark DAG	43
3.8	Spark core e librerie	45
3.9	Architettura del cluster	52

3.10	Richiesta di esecuzione dell'applicazione	53
3.11	Esecuzione dei compiti: interazione tra esecutori e driver	53
4.1	Text Classification: topic detection	56
4.2	Porzione del dataset per Text Classification	57
4.3	NER: annotazione semantica	62
4.4	Porzione del dataset per NER in lingua inglese	63
4.5	Porzione del dataset per NER in lingua italiana	64
4.6	Text Classification - USE: classification report	69
4.7	Text Classification - BERT: classification report	70
4.8	NER in Inglese - GloVe: classification report	71
4.9	NER in Inglese - BERT: classification report	71
4.10	NER in Italiano - GloVe: classification report	72
4.11	NER in Italiano - BERT: classification report	72
4.12	Named Entity Recognition per l'italiano - Errore 1	74
4.13	Named Entity Recognition per l'italiano - Errore 2	74

Elenco delle tabelle

4.1	Dataset per Text Classification	66
4.2	Dataset per NER	66
4.3	Risultati per Text Classification	69
4.4	Risultati per NER in Inglese	71
4.5	Risultati per NER in Inglese	72
4.6	Tempi di esecuzione	73

Abstract

Il nostro mondo è fatto di parole. L'uso della lingua è uno dei tratti principali che distingue l'homo sapiens dalle altre specie. I nostri antenati hanno inventato le lingue molte migliaia di anni fa conseguentemente alla necessità per la società umana di svilupparsi. Scimpanzé, delfini e altri animali hanno mostrato vocaboli di centinaia di segni ma solo gli esseri umani possono comunicare in modo affidabile un numero illimitato di messaggi, qualitativamente diversi, su qualsiasi argomento usando segni discreti. Il nostro vivere quotidiano si basa principalmente sul modo di comunicare con le altre persone: possiamo farlo oralmente, scrivendo lettere, pubblicando opere ma soprattutto, in questo periodo storico, lo si fa sfruttando la rete. Quindi a cosa serve che i nostri agenti informatici siano in grado di elaborare i linguaggi naturali? Principalmente per comunicare con gli esseri umani e per acquisire informazioni dal linguaggio scritto.

Natural Language Processing (NLP) è un campo dell'intelligenza artificiale (AI) che permette ai computer di analizzare e comprendere il linguaggio umano, sia scritto che parlato. L'elaborazione del linguaggio naturale si serve di algoritmi informatici e intelligenza artificiale per permettere ai computer di riconoscere e rispondere alla comunicazione umana. Ciò significa che ora possiamo automatizzare l'analisi e trovare informazioni che non sapevamo nemmeno di cercare.

Per rimanere al passo col crescere dei dati, soprattutto negli ultimi anni, è cresciuta anche la necessità di rendere sempre più performanti i sistemi che si occupano della

loro analisi. Non sempre basta un solo apparato, ma spesso si ricorre a quello che è chiamato *calcolo distribuito*

Lo sviluppo di questa tesi ha come obiettivo quello di sperimentare le prestazioni fornite dal framework Spark NLP quando esso viene impiegato per eseguire dei compiti di elaborazione del linguaggio naturale come *Text Classification* e *Named Entity Recognition*. Spark NLP è un progetto, dedicato appunto alla NLP, che basa le sue fondamenta su Apache Spark, un motore multilingue per l'esecuzione di ingegneria dei dati, scienza dei dati e apprendimento automatico su macchine a nodo singolo o cluster. Verranno analizzate in particolar modo la sua scalabilità, misurando e confrontando tra loro le prestazioni ottenute utilizzando cluster di diverse dimensioni.

Verrà posta l'attenzione sul concetto di elaborazione del linguaggio naturale, analizzando gli step che compongono questo processo, i suoi campi d'applicazione e i modelli al momento maggiormente utilizzati per la language modeling. Al fine di rendere chiaro il metodo con il quale è stata svolta la fase di sperimentazione, il progetto descriverà doverosamente le tecnologie che sono state utilizzate, entrando nel dettaglio del funzionamento di Apache Spark, Spark NLP e Hadoop. Infine, verrà proposta una panoramica sull'ambiente di lavoro utilizzato per poi focalizzarsi sui task coinvolti in questo progetto, esponendone lo scopo, i casi d'uso e discutendo i risultati ottenuti tramite l'esecuzione delle soluzioni software sui nostri sistemi.

Introduzione

1.1 Abstract

Il Natural Language Processing (NLP) è una branca dell'intelligenza artificiale in continuo sviluppo. I task che fanno parte di quest'area vengono affrontati tutti i giorni quando compiamo azioni che ormai ci sono naturali. Quando chiediamo qualcosa al nostro assistente digitale (es. Siri, Google Assistant, Alexa) oppure quando ci colleghiamo ad internet e utilizziamo un qualsiasi motore di ricerca per leggere le notizie della giornata, il sistema con il quale ci stiamo interfacciando ha il compito di interpretare la nostra richiesta ed elaborare un piano d'azione per produrre una risposta che possa essere per noi soddisfacente e altresì disponibile in tempi ragionevoli.

Il quantitativo di informazioni disponibili online è, però, infinitamente vasto e in continua crescita. Per citare dei numeri, soltanto nel 2021: sono state inviati circa 319.6 miliardi di email^[25], inviati 100 miliardi di messaggi tramite WhatsApp^[48], 1.8 miliardi di persone utilizzano Facebook e 1.3 miliardi accedono alla sua app di messaggistica istantanea Facebook Messenger^[8]. Inoltre, il *World Economic Forum*^[21] stima che entro il 2025, la quantità di dati generati ogni giorno raggiungerà 463 exabyte a livello globale. Le informazioni che si stanno raccogliendo a livello globale stanno crescendo esponenzialmente e mentre questo accade, il numero di analisti umani sta

crescendo solo linearmente - in altre parole, noi umani semplicemente non possiamo tenere il passo. Questo tesoro di dati non strutturati è così vasto che ormai non sappiamo nemmeno cosa non sappiamo.

Dall'esigenza di ottenere risposte a domande di varia natura in tempo utile, nasce l'idea di *calcolo distribuito* e di *sistema distribuito*. Si tratta di un concetto informatico, o meglio di una vera e propria branca dell'informatica, che studia i “sistemi distribuiti”, cioè “gruppi di computer” che collaborano tra loro per eseguire un determinato programma. Infatti, quasi la totalità dei software che lavorano coi Big Data, sono eseguiti in ambienti distribuiti di centinaia, se non migliaia di macchine connesse tra di loro. È da qui che nasce l'idea di base di questa Tesi, ovvero studiare uno degli strumenti maggiormente utilizzati per l'elaborazione del linguaggio naturale nel mondo dei Big Data: **Spark NLP**¹. Spark NLP, sviluppato dal John Snow Lab, è ad oggi il framework più utilizzato dalle aziende che trattano task di Natural Language Processing, con il 33% di utilizzatori sul mercato.

Il progetto è partito dallo studio di questo framework attraverso l'utilizzo del linguaggio Python e dell'interfaccia PySpark². Inizialmente sono state sviluppate soluzioni a problemi di natura linguistica di vario genere come *Tokenization*, *Lemmatization* o *PoS Tagging* e sono stati effettuati test su singole macchine con dataset di modeste dimensioni e l'uso di modelli pre-addestrati. Sono stati utilizzati componenti built-in di Spark NLP, come *Annotators* e *Transformers*, per testare le funzionalità di questo strumento.

Una volta presa familiarità con il framework, si è deciso di osservare nel dettaglio il comportamento di Spark NLP su due task specifici: *Text Classification* e *Named Entity Recognition (NER)*. Il primo è definito come il processo di categorizzazione del

¹Spark NLP: <https://nlp.johnsnowlabs.com/>

²Documentazione PySpark: <https://spark.apache.org/docs/latest/api/python/index.html>

testo in base al suo contenuto, compito presente in diversi campi come ad esempio la classificazione di email spam (classificazione binaria) o l'assegnazione di etichette a diversi articoli di notizie in base al loro contenuto (classificazione multi-classe). Il secondo è un sotto compito dell'estrazione di informazioni che cerca di localizzare e classificare elementi atomici nel testo in categorie predefinite come i nomi di persone, organizzazioni, luoghi, espressioni di tempi, quantità, ecc. Questi problemi sono approfonditi nel capitolo 4.

Per entrambi i task, sono state sviluppate soluzioni che comprendono addestramento di un modello, test e valutazione di quest'ultimo. A partire dai modelli prodotti, sono stati studiati i punteggi ottenuti in termini di precisione, recall, accuratezza e F1-score (descritti nella sezione 4.3.1).

Spark NLP, però, offre prestazioni migliori in ambiente distribuito. Pertanto l'esecuzione del software è stata migrata su di un cluster di 3 macchine. Per fare ciò sono stati installati e configurati su ognuna di esse:

- Ambiente Spark, sul quale si basa Spark NLP (approfondito nella sezione 3.3).
- File system distribuito HDFS, per memorizzare in ambiente distribuito i dataset utilizzati e i modelli addestrati (descritto nel dettaglio al paragrafo 3.2.2).
- Resource Manager YARN, per la gestione delle risorse durante l'esecuzione dell'applicazione (approfondito alla sezione 3.2.4).

Una volta pronto il sistema distribuito, si è passati alla fase di analisi delle soluzioni sviluppate. I modelli addestrati sono stati testati su dataset di grandi dimensioni (> 700.000 esempi) e sono stati registrati i tempi di esecuzione ottenuti scalando il sistema, inizialmente utilizzando un solo worker e successivamente parallelizzando l'esecuzione su due macchine distinte.

Una volta a disposizione un set di risultati in termini di accuratezza e velocità di esecuzione, sono stati confrontati con quelli che attualmente formano lo stato dell'arte, valutando la validità del framework Spark NLP rispetto alle prestazioni fornite da altre soluzioni disponibili sul mercato.

1.2 Struttura della tesi

Questa tesi di laurea si concentra sull'analisi di Spark NLP come soluzione distribuita per l'elaborazione del linguaggio naturale su vasti quantitativi di dati.

Il secondo capitolo discute il concetto di Natural Language Processing, a partire dallo scopo per il quale questa branca dall'intelligenza artificiale è nata, per poi passare alla descrizione degli step coinvolti nel processo NLP e le difficoltà che vengono incontrate durante lo stesso. Parte del capitolo pone l'attenzione sulle reti neurali utilizzate per problemi linguistici, percorrendo la storia dei primi approcci al problema fino ad arrivare ai perceptron e il Deep Learning. All'interno del medesimo capitolo viene trattato il concetto di Language Modeling, focalizzando l'attenzione in particolare modo sui Word Embedding e sui modelli utilizzati durante la fase sperimentazione.

A seguire, nel terzo capitolo, si analizzano le tecnologie coinvolte in questo progetto di tesi, iniziando con una panoramica sul calcolo distribuito per poi proseguire descrivendo nel dettaglio l'ecosistema Apache Hadoop e gli ambienti Apache Spark e Spark NLP. Al termine di ciò viene presentato il sistema sul quale è stato eseguito il codice prodotto presentandone le caratteristiche l'hardware e la configurazione dell'ambiente software.

Nel quarto capitolo si tratta la fase di sperimentazione. Sono introdotti i task di *Text Classification* di testi in lingua inglese e *Named Entity Recognition* per la lingua inglese e per l'italiano, descrivendone il quesito che essi pongono e le soluzioni

sviluppate per entrambi i problemi. Viene inoltre illustrata la struttura dei dataset su cui sono stati eseguiti i test e i risultati ottenuti elaborando questi ultimi.

Per concludere, nel capitolo finale, vengono espone le conclusioni tratte dagli esiti della sperimentazione, valutando le potenzialità delle tecnologie messe in campo.

Il processo di elaborazione linguistica

In questo capitolo verranno esposti i concetti fondamentali del Natural Language Processing (NLP), prestando una maggiore attenzione ai livelli di analisi linguistica e al language modeling.

2.1 Natural Language Processing

Il Natural Language Processing è un campo di ricerca interdisciplinare che abbraccia informatica, intelligenza artificiale e linguistica, il cui scopo è quello di sviluppare algoritmi in grado di analizzare, rappresentare e quindi “comprendere” il linguaggio naturale, scritto o parlato, in maniera simile o addirittura più performante rispetto agli esseri umani.

2.1.1 Il Linguaggio Naturale

Un linguaggio può essere definito come un insieme di stringhe fatte da simboli appartenenti ad un dato alfabeto. I linguaggi formali (come un linguaggio di programmazione) sono definiti con precisione: tutte le parole e il loro uso sono predefiniti nel sistema.

Il linguaggio naturale, d'altra parte, non è progettato; si evolve secondo la convenienza e l'apprendimento di un individuo. Inoltre, le macchine capiscono solo il

linguaggio dei numeri, pertanto per creare modelli linguistici, è necessario convertire tutte le parole in una sequenza di numeri.

Come si determina però la “comprensione” di un linguaggio? Quando si usa questo termine si intende il capire, e quindi essere poi in grado di usare, il linguaggio a varie granularità, a partire dalle parole, in relazione al loro significato e alla appropriatezza d’uso rispetto a un contesto, fino alla grammatica e alle regole di strutturazione, sia delle frasi a partire dalle parole, sia dei paragrafi e delle pagine a partire dalle frasi. La comprensione del linguaggio naturale però non è un compito facile, principalmente per due motivi:

1. Esso deve sottostare a specifiche regole sintattiche e semantiche ma viene spesso affiancato da forme idiomatiche e convenzioni che fanno sì che le frasi possano assumere un significato diverso in base al contesto nelle quali vengono utilizzate;
2. Un’altra cosa da notare riguardo al linguaggio umano è che si tratta di simboli. Secondo Chris Manning, professore di Machine Learning a Stanford, esso è un sistema di segnalazione discreto, simbolico e categorico. Questo significa che possiamo trasmettere lo stesso significato in modi diversi (cioè, discorso, gesto, segni, ecc.). La codifica da parte del cervello umano è un modello continuo di attivazione con cui i simboli sono trasmessi attraverso segnali continui di suono e visione.

2.1.2 Il processo NLP

Il processo di language processing deve, utilizzando una decomposizione analoga, generare una progressiva rimozione degli elementi di ambiguità partendo da un testo. Il suo obiettivo, pertanto, è quello di prendere un testo e attraverso una serie di trattamenti, in una cascata deterministica che decompone le fasi di analisi, produrre un’interpretazione del testo e rispondere al comando ricevuto con la pianificazione

di un'azione da compiere. La decomposizione si suddivide nelle fasi mostrate nella Figura 2.4 a pagina 11):

- **Analisi lessicale:** Il testo viene sottoposto a una scansione per trovare le parole chiave o comunque i termini principali del discorso. Questa operazione avviene tramite un algoritmo detto **analizzatore lessicale**. Il testo viene suddiviso in frasi e parole prendendo come riferimento dei caratteri separatori, in genere lo spazio *blank* e i segni di punteggiatura. Una volta ottenuto l'insieme dei termini (*token*) del testo, si analizzano per estrarre tutte le proprietà (*feature*) che il linguaggio assegna a quella parola (la sua categoria grammaticale, se essa è nome, se è un numero, ecc.).

Un testo senza *stop-word* può essere ulteriormente normalizzato tramite un algoritmo di **stemming** che riduce le parole alla loro forma flessa (*radice*). Si riduce così il numero delle varianti nel documento.

Testo	Estrazione dei Token	Lemmatizzazione
Fuori piove e Marco ha deciso di rimanere a casa	Fuori piove e Marco ha deciso di rimanere a casa .	Fuori piovere e Marco decidere di rimanere a casa

Figura 2.1: Analisi Lessicale

- **Analisi sintattica:** applica alle sequenze di token, ottenute tramite l'analisi lessicale, i principi della grammatica che genera il linguaggio nel quale è stato

scritto il testo. Generalmente produce una struttura ad albero dove ad ogni nodo corrisponde o una parola o un costituente linguistico (frammento di frase), al quale corrisponderà una collocazione in una struttura gerarchica che nella sua radice copre l'intera frase. Si può dire che l'albero è la descrizione di tutte le relazioni grammaticali vigenti nella sequenza dei token in ingresso. L'insieme delle regole della sintassi prende nome di **regole di produzione** ed è strettamente collegato alla lingua.

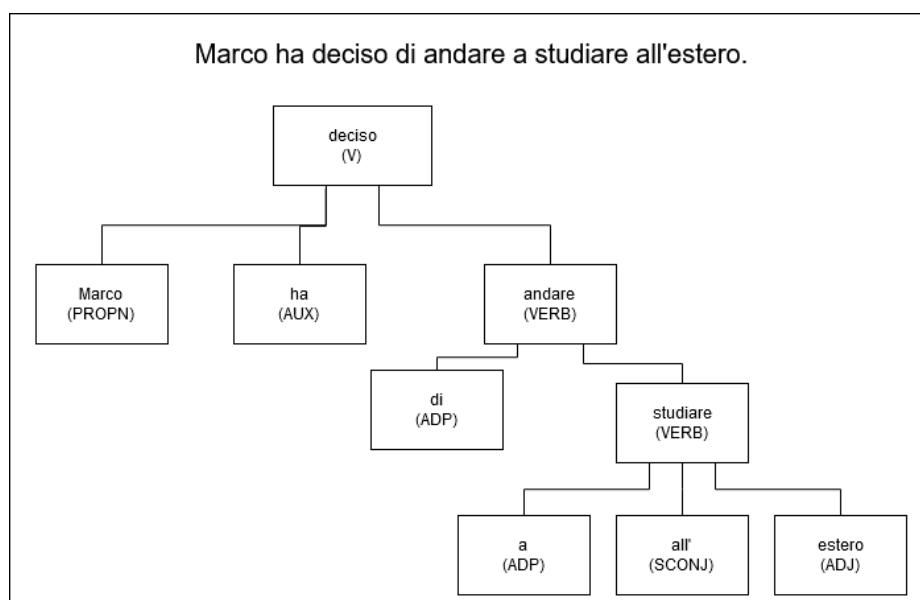


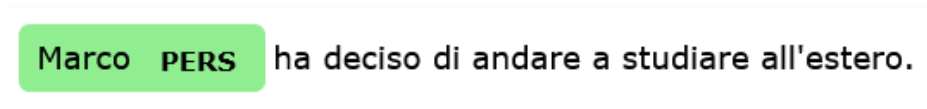
Figura 2.2: Analisi Sintattica

- **Analisi semantica:** si occupa di dare una spiegazione che descriva la relazione tra la frase e il mondo di riferimento (il "world model" nella figura 2.4). L'analisi semantica è, pertanto, la ricerca del significato di un termine o di una frase. Non è sempre facile trovare il senso giusto delle parole. Spesso i termini hanno più significati (*polisemici*) e occorre trovare quello giusto. Questo processo di selezione delle accezioni è detto **disambiguazione**. Le relazioni tra i concetti e i *synset* consentono di costruire una struttura a rete detta **ontologia**. In particolare si lavora su due livelli semantici differenti:

Semantica Lessicale: studia il significato delle singole parole o di più termini, ovvero catene di parole il cui significato è diverso dalla somma dei significati delle singole parole (e.s. *uscita d'emergenza*).

Semantica Frasale: studia il significato di una frase, ponendo l'attenzione sulle interazioni tra i significati a livello delle parole. Prendiamo queste due frasi: *Maria ha paura* vs. *Maria sta giocando*. Maria ha lo stesso ruolo sintattico nei due casi (quello di soggetto) ma ha un diverso 'ruolo semantico': ha un ruolo attivo nell'attività di giocare, ma non nel caso di avere paura, che identifica uno stato emotivo che Maria vive involontariamente.

A seconda del tipo di informazioni che si vogliono ottenere dai dati, è possibile usare una delle due tecniche di analisi semantica: *text classification* o *text extraction*. In particolare, in questa tesi sono stati trattati due task appartenenti alle queste due tecniche: **Topic Classification** per la Text Classification ed **Named Entity Recognition** per la Entity Extraction.



Marco PERS ha deciso di andare a studiare all'estero.

Figura 2.3: Analisi Semantica: esempio di Named Entity Recognition

- **Analisi pragmatica:** La dimensione pragmatica dell'analisi linguistica, secondo il filosofo americano Charles W. Morris, riguarda quegli aspetti che concernono l'azione indotta dall'uso del linguaggio. Studia il parlare in quanto forma di agire linguistico all'interno di una data situazione comunicativa. Molti enunciati, ad esempio, non veicolano informazioni, ma equivalgono ad azioni: "Scusami", "Prometto", "Sì, lo voglio", ecc.

Nella forma logica, il significato delle parole viene messo in corrispondenza con il livello pragmatico (ovvero "cosa si aspetta da me la persona?", "qual è il suo

scopo?") e quindi, da un punto di vista applicativo, il sistema reagisce. Le parole della forma logica rappresentano degli scopi e, dal punto di vista del modello, interpretare la forma logica e trasformarla in un'azione significa interpretare l'ultimo livello dell'analisi che è quello degli scopi che per l'utente ha quel testo.

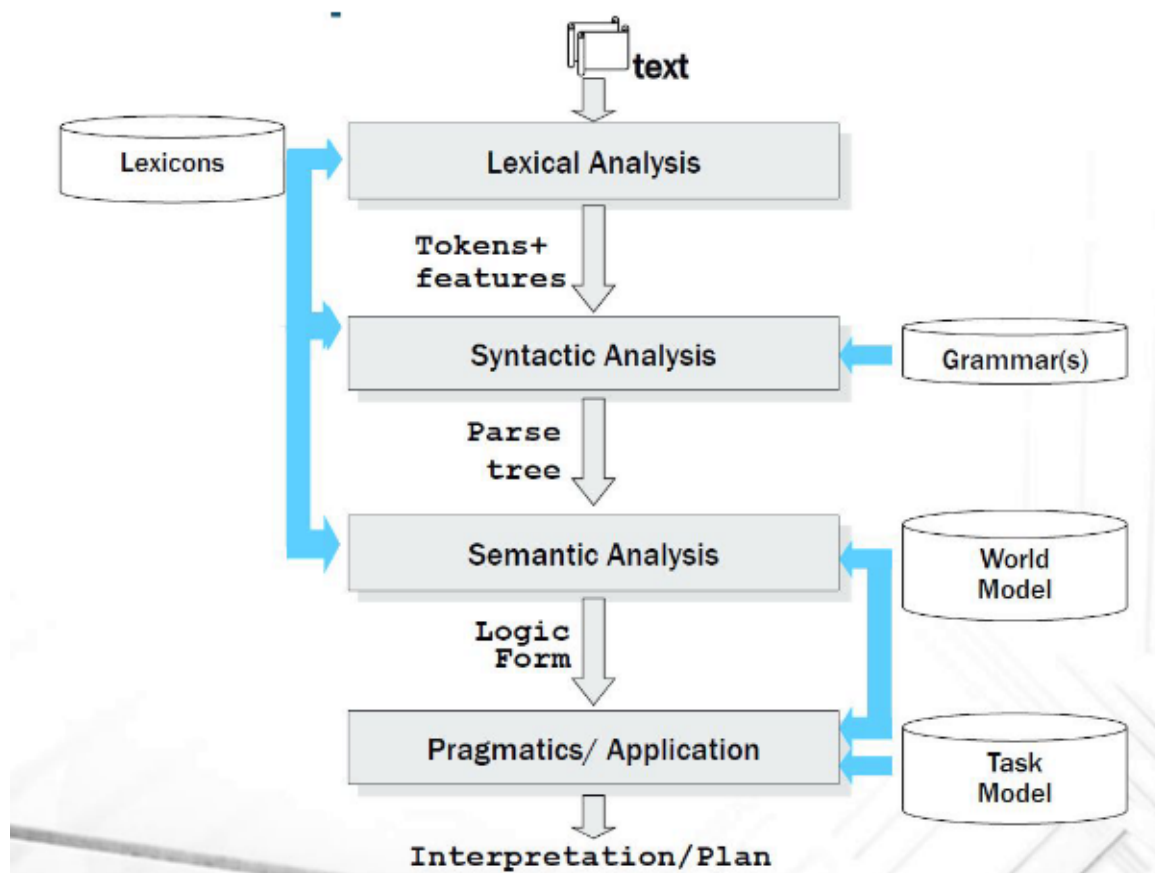


Figura 2.4: Il processo NLP

2.1.3 Ambiguità

L'interpretazione delle parole contestualizzate in una frase è un processo molto articolato e complesso. Ancora oggi si riscontrano ambiguità nel parlato. L'aspetto della generazione del linguaggio viene risolto invece con le regole sintattiche e sintagmatiche della lingua. L'elaborazione di testi corrisponde ad esempio a comprendere molteplici aspetti relativi al suo significato.

Un'accuratezza linguistica (uso del linguaggio) viene paragonato al grado di approssimazione della performance dei parlanti nativi. Tanto più questa accuratezza viene riscontrata tanto più verrà usata ad esempio da un programma di sintesi vocale. Molto spesso però la natura stessa del linguaggio nasconde delle ambiguità. Possono esserci quattro tipi di ambiguità (Figura 2.5 a pagina 13):

1. **Fonologica:** l'utilizzo variabile degli accenti. Parole scritte nello stesso modo hanno spesso diverse fonologie (es. déi, dèi);
2. **Morfologica:** derivante dalla struttura grammaticale delle parole. Stabilisce la classificazione delle parole e l'appartenenza a determinate categorie come il nome, il pronome, il verbo, l'aggettivo;
3. **Grammaticale:** l'associazione delle parole al contesto. Analisi della struttura grammaticale dell'espressione linguistica;
4. **Semantica:** il significato intrinseco stesso della parola. Tolle le ambiguità strutturali, ricerca l'insieme dei significati dell'espressione derivata dagli step precedenti

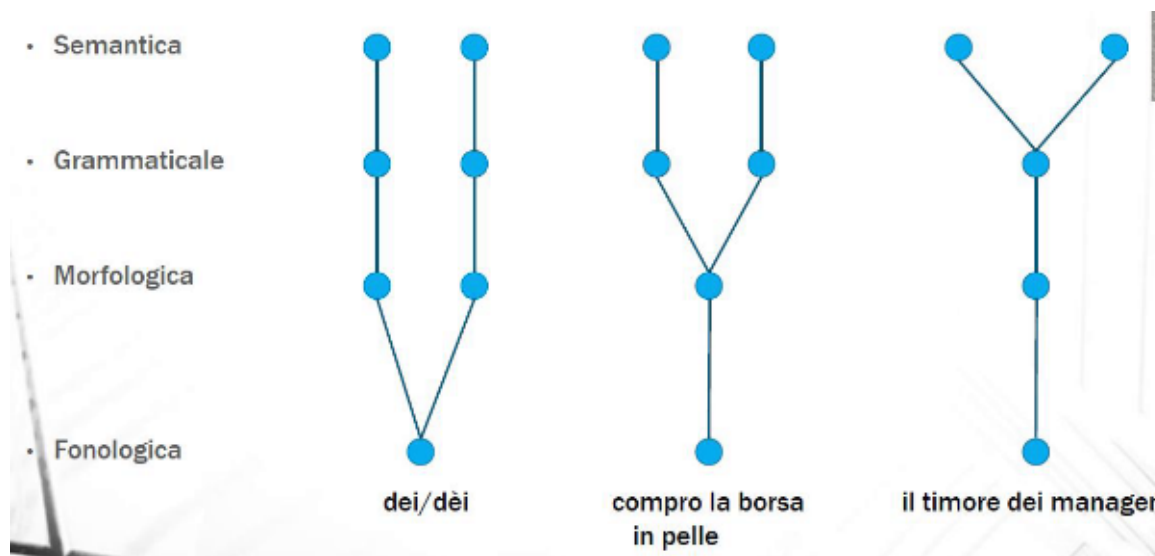


Figura 2.5: Livelli di ambiguità

2.2 Campi d'applicazione

Nella fase di sperimentazione riportata in questa tesi, come già citato, sono state proposte strategie risolutive ai problemi di *Classificazione del testo* e *Riconoscimento delle entità nominate*. L'attenzione è stata focalizzata su questi problemi poiché su di essi si fondano numerevoli sistemi software che, ormai, vengono utilizzati da quasi la totalità delle persone nella vita quotidiana e spesso sono impiegati anche a livello aziendale da diverse imprese. Alcuni esempi sono:

Chatbots

I chatbots sono una forma d'intelligenza artificiale programmata per interagire con gli esseri umani tanto da simulare loro stessi gli esseri umani. Capiscono la complessità del linguaggio naturale e trovano il significato reale della frase e imparano anche dalle loro conversazioni con gli umani migliorandosi con il tempo.

Completamento automatico nei motori di ricerca

I motori di ricerca usano i loro enormi set di dati per analizzare ciò che i loro clienti stanno probabilmente digitando quando inseriscono determinate parole e suggeriscono le possibilità più comuni.

Assistenti vocali

Usano una complessa combinazione di riconoscimento vocale, comprensione del linguaggio naturale ed elaborazione del linguaggio naturale per capire ciò che gli esseri umani dicono e poi agire di conseguenza.

Traduttore di lingua

Questi strumenti di traduzione utilizzano anche la modellazione sequence to sequence che è una tecnica di elaborazione del linguaggio naturale. In precedenza, i traduttori di lingue usavano la traduzione automatica statistica (SMT) che significava analizzare milioni di documenti già tradotti da una lingua all'altra (dall'inglese all'hindi in questo caso) e poi cercare i modelli comuni e il vocabolario di base della lingua. Tuttavia, questo metodo non era così accurato rispetto alla modellazione Sequence to sequence.

Analisi del sentimento

Le aziende possono usare la sentiment analysis per capire come un particolare tipo di utente reagisce a un particolare argomento, prodotto, ecc. Possono usare l'elaborazione del linguaggio naturale, la linguistica computazionale, l'analisi del testo, ecc. per capire il sentimento generale degli utenti per i loro prodotti e servizi e scoprire se il sentimento è buono, cattivo o neutrale.

Controlli di grammatica

Non solo possono correggere la grammatica e controllare l'ortografia, ma anche suggerire sinonimi migliori e migliorare la leggibilità complessiva del contenuto.

Classificazione e filtraggio delle e-mail

I servizi di posta elettronica utilizzano l'elaborazione del linguaggio naturale per identificare il contenuto di ogni e-mail con la classificazione del testo in modo che possa essere messo nella sezione corretta. In casi più avanzati, alcune aziende utilizzano anche software antivirus speciali con elaborazione del linguaggio naturale per scansionare le e-mail e vedere se ci sono modelli e frasi che possono indicare un tentativo di phishing sui dipendenti.

2.3 Reti neurali per problemi linguistici

In questa sezione viene trattata la storia dello sviluppo delle tecniche per il Natural Language Processing, per poi approfondire i concetti di perceptrone, rete neurale e Deep Learning.

2.3.1 Primi approcci

La storia della NLP viene fatta spesso partire dagli anni '50 del XX secolo, quando, nel famoso articolo *Computing Machinery and Intelligence*^[45], Alan Turing propose il suo famoso test per valutare l'abilità di un computer di impersonare un umano durante una conversazione scritta in tempo reale. In realtà, quasi un decennio prima dell'uscita del suddetto articolo, si parlava già del concetto di *Machine Translation*, un sottocampo della linguistica computazionale che studia l'uso di software per tradurre testi o discorsi da una lingua all'altra. Ma ciò è stato solo l'inizio della lunga storia dell'elaborazione del linguaggio naturale. Difatti, con il passare degli anni, si sono

succeduti diversi approcci alla NLP.

Nel 1957, il linguista americano Noam Chomsky, pubblicò *Syntactic Structures*^[7], opera che offrì un contributo fondamentale al problema introducendo la *grammatica generativa*, insieme di regole che specificano in modo formale e ricorsivo le strutture sintattiche di un linguaggio. Ad esso, fino agli anni '80, anche a causa di un congelamento dei fondi USA destinati alla ricerca, seguirono sistemi NLP costituiti da strutture estremamente complesse di regole procedurali ed euristiche, che però a causa della loro rigidità risultarono incapaci di gestire l'estrema variabilità ed ambiguità del linguaggio naturale.

A cavallo tra gli anni '80 e '90, c'è stata una vera e propria rivoluzione dettata dall'introduzione degli algoritmi di machine learning per l'elaborazione del linguaggio. Si passò da sistemi *rule-based* codificati manualmente a sistemi *corpus-based* nei quali l'intervento umano diretto veniva limitato grazie all'apprendimento automatico a partire da uno o più corpus di riferimento. La maggior parte di questi approcci, utilizzati anche attualmente, forniscono un modo più avanzato per interpretare l'ambiguità e fornire ulteriori prove per la valutazione di una decisione. Algoritmi come gli alberi decisionali usano regole *if-then* per ottenere il risultato ottimale e algoritmi probabilistici che sostengono la decisione presa dalla macchina fornendo una buona accuratezza.

Nel 2011, per la prima volta un algoritmo basato sul deep learning è stato applicato a differenti problemi di NLP, tra cui l'identificazione di entità e l'assegnazione di categorie morfologiche a parole, mostrando prestazioni sensibilmente migliori rispetto ad altri approcci rappresentativi dello stato dell'arte. Da allora, sono stati realizzati algoritmi sempre più complessi basati sul deep learning per affrontare problemi di NLP ancora non risolti o trattati in passato ma con risultati non soddisfacenti.

Il Deep Learning è un ramo del Machine Learning, nel quale viene utilizzata una

rete neurale con tre o più strati. Queste reti neurali tentano di simulare il comportamento del cervello umano permettendogli di "imparare" da grandi quantità di dati. Mentre una rete neurale con un solo strato può ancora fare previsioni approssimative, ulteriori strati nascosti possono aiutare a ottimizzare e raffinare la precisione.

2.3.2 Percettrone

Per comprendere il deep learning occorre prima chiarire il concetto relativo alle reti neurali artificiali (ANN). Il prototipo delle ANN sono le corrispettive biologiche: le reti neurali del cervello umano sono la sede della nostra capacità di comprendere l'ambiente e i suoi mutamenti e di fornire quindi risposte adattive calibrate sulle esigenze che si presentano. Un singolo neurone può ricevere simultaneamente segnali da diverse sinapsi e misurando il potenziale elettrico di tali segnali, stabilisce se è stata raggiunta la soglia di attivazione per generare a sua volta un impulso nervoso. Tale proprietà è implementata anche nelle reti artificiali.

Nel 1958 viene proposta da Rosenblatt la prima rete neurale **Perceptron**, fatta di un singolo strato con la seguente forma:

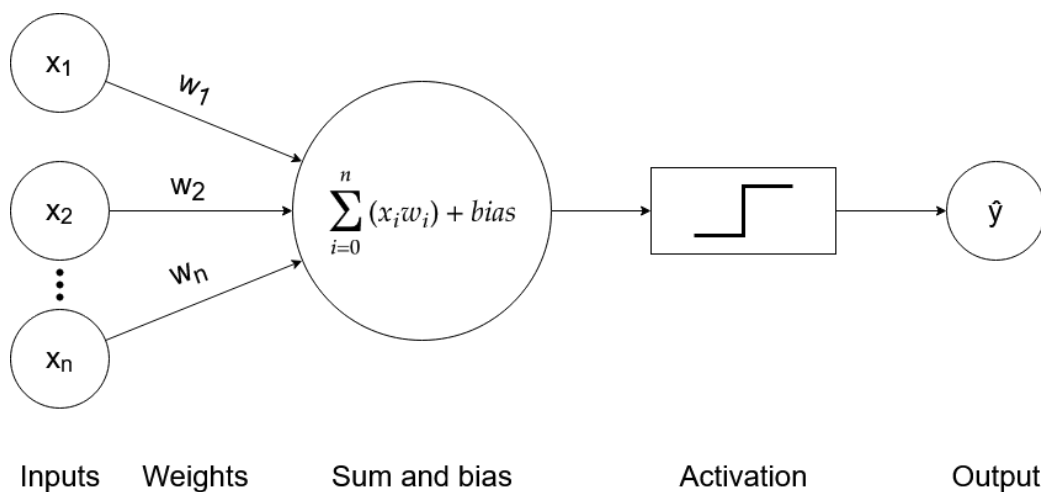


Figura 2.6: Percettrone

L'idea di base è di usare diversi pesi per rappresentare l'importanza di ogni input. Se la somma di questi valori è maggiore di un certo valore di soglia, verrà presa una decisione come vero o falso (0 o 1). Nel dettaglio, un percettrone accetta gli input (x_1, x_2, \dots, x_n) , li modera con dei valori di peso (w_1, w_2, \dots, w_n) ed applica loro una funzione di attivazione per produrre il risultato finale. La funzione di attivazione, solitamente, è definita come segue:

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases} \quad (2.3.1)$$

Dove:

- **x**: è il vettore degli input
- **w**: è il vettore dei pesi
- **b**: è il bias, una costante che non dipende dai valori in input. Può essere pensato come un livello base di attivazione per l'output.

Una volta che abbiamo a disposizione un percettrone, si può cercare di istruirlo in modo che, dato un input x , l'output $f(x)$ sia quanto più vicino possibile a un dato valore $g(x)$ scelto a priori. Guardando la funzione di attivazione, notiamo che, essendo il vettore degli input non prevedibile ed il bias una costante, l'unico valore che è possibile modulare per far sì che il percettrone restituisca il risultato desiderato è il vettore dei pesi $w = (w_1, w_2, \dots, w_n)$. Infatti, nel cosiddetto *apprendimento supervisionato*, man mano che la macchina elabora output, si procede a correggerla per migliorarne le risposte variando i pesi.

Le capacità computazionali di un singolo percettrone, però, sono limitate e le prestazioni che è possibile ottenere dipendono fortemente sia dalla scelta degli input, che dalla scelta della funzione che si desidera implementare.

2.3.3 Deep Learning

I modelli di Deep Learning vengono progettati per analizzare continuamente i dati con una struttura logica simile a quella utilizzata dagli esseri umani per trarre conclusioni. Per raggiungere questo obiettivo, le applicazioni di deep learning si avvalgono di una rete neurale artificiale a più strati, solitamente composte da: un livello di input, uno o più livelli nascosti (*Hidden Layers*) e un livello di output. Ciascun nodo, o neurone artificiale, si connette ad un altro e ha un peso e una soglia associati. Se l'output di qualsiasi singolo nodo è al di sopra del valore di soglia specificato, tale nodo viene attivato, inviando i dati al successivo livello della rete. In caso contrario, non viene passato alcun dato al livello successivo della rete.

I perceptron multistrato vengono addestrati su coppie input-output e imparano a modellare le dipendenze tra ingresso e uscita. L'addestramento comporta la regolazione dei pesi o dei bias ad ogni strato della rete mediante l'utilizzo del backpropagation, un algoritmo di apprendimento supervisionato. La retropropagazione dell'errore cerca il valore minimo della funzione di errore nello spazio dei pesi usando la tecnica del *gradient descent*. I pesi che minimizzano la funzione di errore sono poi considerati una soluzione al problema di apprendimento.

Una prima caratteristica fondamentale di tali reti è che sono in grado di apprendere, in maniera autonoma e contestuale, le modalità con le quali combinare al meglio le informazioni ricevute per la risoluzione di un compito specifico.

Una seconda caratteristica è che, in maniera simile al cervello umano, sono in grado di imparare dalle loro esperienze, ossia di migliorare le proprie prestazioni nella risoluzione di un problema complesso in funzione della quantità di esempi con cui sono addestrati.

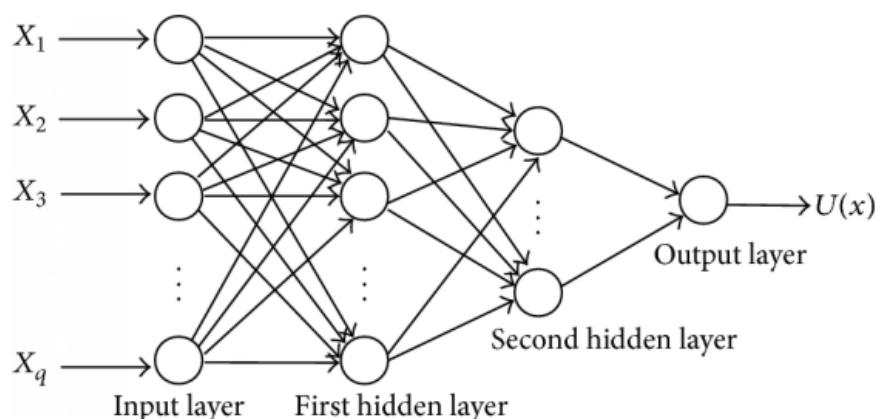


Figura 2.7: Esempio di una rete neurale artificiale

Tali reti sono in grado di elaborare, però, come input, solo dati numerici e non stringhe testuali. Questa è una delle motivazioni per le quali le prime applicazioni di successo del deep learning hanno riguardato il trattamento di immagini o segnali. Ad oggi sono state sviluppate diverse tecniche per la mappatura di contenuti testuali in vettori di numeri reali che permettono alle reti neurali di elaborare anche informazioni derivanti da corpus linguistici e pertanto essere impiegate per la risoluzione di task di Natural Language Processing.

Nei capitoli successivi tratteremo il concetto di **Word Embedding**, l'insieme delle tecniche il cui scopo è proprio quello di eseguire questa mappatura.

2.4 Language Modeling

Quando si lavora con un enorme corpus di dati testuali, è utile conoscere la probabilità con cui una sequenza di parole si succederà e quali particolari caratteristiche sono necessarie per capire questa dipendenza.

Il Language Modeling è il compito di capire questa distribuzione di probabilità su una sequenza di parole. Ciò aiuta a creare caratteristiche che possono distinguere tra frasi e frasi, secondo il contesto in cui appaiono. I modelli linguistici interpretano

questi dati alimentandoli attraverso un algoritmo che stabilisce regole per il contesto nel linguaggio naturale. A questo punto, il modello applica queste regole in compiti linguistici per prevedere o produrre nuove frasi. Il modello essenzialmente impara le caratteristiche del linguaggio di base e usa queste caratteristiche per capire nuove proposizioni.

Esistono diversi approcci probabilistici al language modeling, che variano a seconda dello scopo del modello linguistico. In questa tesi, è stato implementato un approccio al language modeling atto a modellare il testo in modo tale che questo possa essere convertito in un input numerico per una rete neurale artificiale. Nelle sezioni successive verrà approfondito proprio questo concetto, chiamato *Word Embedding*, per poi discutere in particolar modo dei modelli *GloVe*^[35], *BERT*^[10] e *USE*^[6], utilizzati durante la fase di sperimentazione descritta nei capitoli successivi.

2.4.1 Word Embedding

Word embedding è il nome collettivo per definire un insieme di tecniche di language modeling e di apprendimento delle caratteristiche nell'elaborazione del linguaggio naturale in cui le parole o le frasi del vocabolario sono mappate in vettori di numeri reali. La necessità di questa pratica è nata con lo sviluppo delle reti neurali artificiali dato che, come accennato nella sezione riguardante il Deep Learning, esse possono elaborare soltanto input di natura numerica.

L'embedding si basa sul fatto che, in genere, le parole con un significato simile avranno rappresentazioni vettoriali che sono vicine tra loro nello spazio di incorporazione (anche se questo non è sempre stato il caso). Quando si codificano le parole, tipicamente l'obiettivo è quello di catturare qualche tipo di relazione in quello spazio, che sia il significato, la morfologia, il contesto o qualche altro tipo di connessione.

Molti word embedding sono creati sulla base della nozione introdotta dall'*ipotesi*

distributiva di Zellig Harris^[49], che si riduce a una semplice idea che le parole che sono usate vicine l'una all'altra hanno tipicamente lo stesso significato.

Ciò diventa particolarmente utile quando i set di dati diventano sempre più grandi, perché con l'aumentare delle dimensioni spesso aumenta anche il numero di parole uniche. La presenza di molte parole usate raramente può causare problemi per un modello lineare; questo perché la quantità di possibili sequenze di parole aumenta, e i modelli che informano i risultati diventano più deboli. Ponderando le parole in modo non lineare e distribuito, questo modello può "imparare" ad approssimare le parole e quindi non essere fuorviato da eventuali valori sconosciuti. La sua "comprensione" di una data parola non è così strettamente legata alle parole immediatamente circostanti.

2.4.2 Embedding statico: GloVe

GlobalVectors (GloVe) è un modello assai noto che apprende i vettori o le parole dalle informazioni di co-occorrenza, ovvero la frequenza con cui compaiono insieme in grandi corpora di testo. GloVe è basato sul conteggio. In linea generale i modelli basati sul conteggio apprendono i vettori, operando una riduzione della dimensionalità sulla matrice di conteggio delle co-occorrenze.

Per prima cosa, si costruisce una grande **matrice di co-occorrenza** (parole x colonne), che contiene le informazioni sulla frequenza con cui ogni parola viene usata in un contesto. Il numero di contesti deve essere grande, poiché è essenzialmente di dimensioni combinatorie.

In seguito, tale matrice viene riscritta in forma algebrica e fattorizzata per ottenerne una dimensionalmente più piccola. Il risultato di questa operazione è una rappresentazione vettoriale per ogni parola.

L'addestramento può poi essere eseguito in due modi diversi: utilizzando il con-

testo per predire una parola target (utilizzando metodi noti come il BoW^[50] o il CBoW^[31]) oppure usando una parola per predire il contesto target (Skip-Gram^[30]).

2.4.3 Embedding contestualizzati: BERT

I modelli statici come GloVe presentano diverse limitazioni:

- L'uso di modelli linguistici molto superficiali. Questo significa che c'è un limite alla quantità di informazioni che possono catturare.
- Un'altra limitazione chiave è che questi modelli non prendono in considerazione il contesto della parola: questi modelli producono un solo embedding per ogni parola, combinando tutti i diversi sensi della parola in un unico vettore.

Alla fine del 2018 i ricercatori di Google AI Language hanno reso open-source una nuova tecnica per l'elaborazione del linguaggio naturale (NLP) chiamata **BERT (Bi-directional Encoder Representations from Transformers)** - una grande svolta che ha preso d'assalto la comunità del Deep Learning per le sue incredibili prestazioni. Il team di ricerca che ha lavorato dietro BERT lo descrive così:

"BERT sta per Bidirectional Encoder Representations from Transformers. È progettato per pre-addestrare rappresentazioni bidirezionali profonde da testi non etichettati, condizionando congiuntamente il contesto sinistro e destro. Come risultato, il modello BERT pre-addestrato può essere messo a punto con un solo strato di output aggiuntivo per creare modelli all'avanguardia per una vasta gamma di compiti NLP."

In primo luogo, BERT è basato sull'architettura **Transformer**, un modello proposto nel paper *Attention is All You Need*^[47] che usa l'*attenzione* (successore del modello sequence-to-sequence) per accelerare il processo di addestramento. In secondo luogo,

BERT è pre-addestrato su un grande corpus di testo non etichettato che include l'intera **Wikipedia** (2.500 milioni di parole) e il Book Corpus (800 milioni di parole). In terzo luogo, BERT è un modello *"profondamente bidirezionale"*. Bidirezionale significa che BERT apprende informazioni sia dal lato sinistro che da quello destro del contesto di un token durante la fase di formazione.

Il paper pubblicato dai creatori di Bert, presenta due modelli che si differenziano per le loro dimensioni:

- **BERT Base**: 12 strati (blocchi di trasformatori), 12 teste di attenzione e 110 milioni di parametri
- **BERT Large**: 24 strati (blocchi di trasformatori), 16 teste di attenzione e 340 milioni di parametri

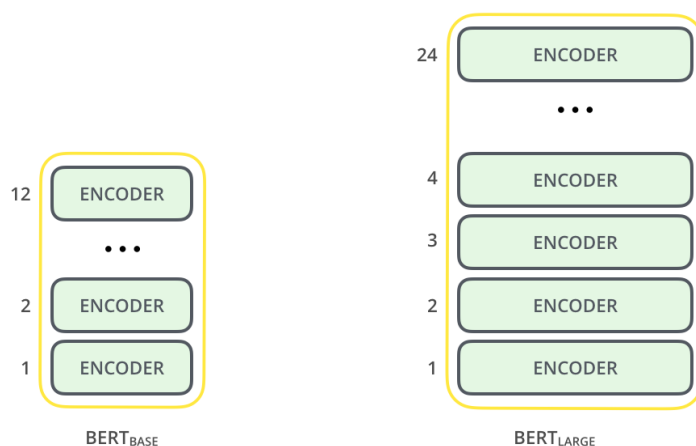


Figura 2.8: BERT Base e BERT Large

BERT è pre-addestrato su due compiti NLP:

- **Masked Language Model:** Viene utilizzato per addestrare la capacità di BERT di catturare informazioni in maniera bidirezionale. Data una frase, il 15% di essa viene *mascherato* usando token speciali (ad esempio "[MASK]") e viene chiesto di prevedere la parola corretta che dovrebbe trovarsi in quella posizione.

Durante l'addestramento di BERT viene utilizzata la seguente tecnica:

1. L'80% delle volte le parole sono state sostituite con il token mascherato [MASK].
 2. Il 10% delle volte le parole sono state sostituite con parole casuali.
 3. Il 10% delle volte le parole sono rimaste invariate.
- **Next Sentence Prediction:** Questo è un task di classificazione binaria che viene utilizzato per addestrare BERT, in modo tale che quest'ultimo possa essere utilizzato in compiti nei quali è necessario sapere comprendere le relazioni che intercorrono tra le frasi. Il quesito posto è: date due frasi A e B prese da un corpus linguistico, determinare se la frase B segue la frase A nel testo oppure no.
1. Per il 50% delle coppie, la seconda frase sarebbe in realtà la frase successiva alla prima.
 2. Per il restante 50% delle coppie, la seconda frase sarebbe una frase casuale dal corpus.

2.4.4 **Embedding contestualizzati: Universal Sentence Encoder**

BERT è uno strumento molto potente ma, proprio per questo motivo non è la soluzione migliore quando si tratta di utilizzarlo su dispositivi con memoria o potenza di calcolo

limitata. È da questa idea che nasce il progetto proposto da Cer et al. [6].

L'**Universal Sentence Encoder (USE)** codifica il testo in vettori ad alta dimensione che possono essere utilizzati per la classificazione del testo, la similarità semantica, il clustering e altri compiti del Natural Language Processing. L'idea è quella di progettare un codificatore che riassume qualsiasi frase data in un'embedding a 512 dimensioni. Si usa questo stesso embedding per risolvere compiti multipli e in base agli errori che vengono fatti su di essi, aggiorniamo la codifica della frase. Poiché essa deve lavorare su più compiti generici, catturerà solo le caratteristiche più informative e scarnerà il rumore. L'intuizione è che, così facendo, il risultato possa essere universalmente compatibile per essere incorporato a diversi task anche molto distanti tra loro.

L'Universal Sentence Encoder pre-addestrato è disponibile pubblicamente in **Tensorflow-hub**. Viene fornito in due varianti:

1. **Transformer Encoder:** In questa variante, usiamo la parte encoder dell'architettura originale del transformer. L'architettura consiste di 6 strati di trasformatori impilati. Ogni strato ha un modulo di self-attention seguito da una rete feed-forward.
2. **Deep Averaging Network:** Word embeddings e i bi-grammi presenti in una frase sono mediate insieme. Poi attraversano una DNN profonda a 4 strati feed-forward per ottenere in uscita un'embedding per l'intera frase a 512 dimensioni. Le embeddings per le parole e i bi-grammi sono apprese durante l'addestramento.

I due modelli hanno un compromesso di accuratezza e richiesta di risorse computazionali. Mentre quello con un codificatore Transformer ha una maggiore precisione ma è computazionalmente più intenso, quello con codifica DAN usa meno memoria con un leggero compromesso a livello di precisione.

Il modello di codifica è progettato per essere il più generale possibile. Ciò viene realizzato utilizzando l'apprendimento multi-task in cui un singolo modello di codifica viene addestrato per risolvere più compiti diversi. I tasks supportati includono: un compito simile a Skip-Thought (Kiros et al. [26]) per l'apprendimento non supervisionato da un testo corrente arbitrario; un task di input-risposta conversazionale (Henderson et al. [16]); e compiti di classificazione per l'addestramento su dati supervisionati. Il compito Skip-Thought sostituisce il LSTM (Hochreiter and Schmidhuber [17]) usato nella formulazione originale con un modello un modello basato sull'architettura Transformer.

Una soluzione distribuita

In questo capitolo verranno presentate le tecnologie coinvolte nello sviluppo della soluzione utilizzata durante la fase di sperimentazione, focalizzandosi particolarmente sui concetti di calcolo distribuito e Transfer Learning.

3.1 Calcolo Distribuito

Questa tesi nasce dall'idea di sperimentare ed analizzare le prestazioni offerte dal framework Spark NLP eseguendolo in ambiente distribuito. Ma cosa è Spark NLP? Su cosa si basa? E perché usare un ambiente distribuito? Per rispondere a queste domande bisogna partire descrivendo cosa è il calcolo distribuito.

Wikipedia definisce il calcolo distribuito come *«un campo dell'informatica che studia i sistemi distribuiti, ovvero sistemi che consistono in numerosi computer che interagiscono tra loro attraverso una rete al fine di raggiungere un obiettivo comune»*. In altre parole, i sistemi distribuiti sono una collezione di componenti indipendenti situati su diverse macchine che, messi in comunicazione tra loro, interagiscono al fine di raggiungere un obiettivo comune fornendo importanti vantaggi a chi li implementa, come: aumento delle prestazioni, tolleranza agli errori e diminuzione del carico. Un sistema distribuito può consistere in qualsiasi numero di possibili configurazioni, come

mainframe, personal computer, workstation, minicomputer e così via. L'obiettivo è quello di far funzionare tale rete come un singolo computer. I motivi principali per cui utilizzare un sistema distribuito sono:

- **Affidabilità:** il sistema generalmente non subisce interruzioni se una singola macchina si guasta.
- **Scalabilità:** è facile e generalmente poco costoso aggiungere altri nodi e funzionalità se necessario.
- **Performance:** sono estremamente efficienti perché i carichi di lavoro possono essere suddivisi e inviati a più macchine.

La progettazione di questi sistemi è comunque un lavoro dispendioso. Nonostante i considerevoli benefici che essi portano, questi ultimi potrebbero non ripagare i costi di realizzazione. Le maggiori sfide che un sistema distribuito potrebbe incontrare si dividono in:

- **Scheduling:** decidere quali lavori devono essere eseguiti, quando devono essere eseguiti e dove devono essere eseguiti.
- **Latenza:** spesso potrebbero essere incontrati dei problemi a livello di latenza, che creano inconvenienti a livello di efficienza e consistenza dei dati. Questo motivo porta i team a fare compromessi tra disponibilità, coerenza e latenza.
- **Osservabilità:** raccogliere, elaborare, presentare e monitorare le metriche di utilizzo dell'hardware per grandi sistemi è una sfida significativa

Nello specifico, in questo progetto è stato utilizzato un tipo di sistema distribuito chiamato **Cluster**. Quando si parla di cluster si intende un sistema di computer connessi tramite una rete con lo scopo di distribuire una elaborazione (parallelizzabile) tra i computer che compongono il cluster.

3.2 Hadoop

I big data sono un'industria a sé ma sono anche al centro delle strategie di molte aziende che desiderano migliorare la gestione dei clienti, il marketing e lo sviluppo. È quando si considerano campi dove la quantità di informazioni che girano intorno ad essi è così estesa, che entrano in gioco i sistemi distribuiti e strumenti come Hadoop che sono essenziali per gestire un notevole afflusso di dati. Proprio per questo motivo, in questa tesi, alla base del sistema distribuito implementato all'interno del progetto, c'è proprio Apache Hadoop.

3.2.1 Cosa è Hadoop

Hadoop¹ è stato creato dalla Apache Software Foundation² ed è stato prodotto nei primi anni 2000 per rispondere alla crescita dei motori di ricerca come Yahoo e Google. Nato da Doug Cutting e Michael Cafarella, il progetto prese il nome dall'elefante giocattolo di uno degli sviluppatori. Hadoop è stato rilasciato come progetto open-source nel 2008 e poi nel 2012 dalla Apache Software Foundation. Oggi, Hadoop è composto da librerie open-source destinate ad elaborare grandi insiemi di dati su migliaia di computer in cluster.

Si tratta di un framework per l'esecuzione di applicazioni su grandi cluster costruiti con hardware di largo consumo a basso costo. Hadoop fornisce in modo trasparente applicazioni sia per l'affidabilità che per il trasferimento dei dati e consiste di quattro moduli principali:

- **Hadoop Common:** utilities che supportano gli altri moduli Hadoop.
- **Hadoop Yet Another Resource Negotiator (YARN):** Un framework per

¹Apache Hadoop: <https://hadoop.apache.org/>

²Apache Software Foundation <https://www.apache.org/>

lo scheduling dei job e la gestione delle risorse del cluster

- **Hadoop Distributed File System (HDFS)**: Un file system distribuito che fornisce un accesso ad alta velocità ai dati delle applicazioni.
- **Hadoop MapReduce**: Un sistema basato su YARN per l'elaborazione parallela di grandi insiemi di dati.

Nella fase di sperimentazione, Hadoop è stato utilizzato per implementare un file system distribuito attraverso il quale, le macchine appartenenti al cluster possono processare i dataset di addestramento e di test in maniera distribuita. Per chiarire questo concetto abbiamo però bisogno di introdurre il funzionamento di HDFS, MapReduce e YARN.

3.2.2 Hadoop Distributed File System

L'Hadoop Distributed File System (HDFS)³ è un file system gerarchico e distribuito progettato per funzionare su hardware di base. Ha molte somiglianze con i file system distribuiti esistenti. Tuttavia, le differenze con essi sono significative. HDFS è altamente tollerante agli errori ed è progettato per essere implementato su hardware a basso costo. Inoltre, fornisce un accesso ad alta velocità ai dati delle applicazioni ed è adatto ai software che utilizzano grandi set di dati.

L'architettura di HDFS è di tipo master/slave composta da un singolo master (**NameNode**) e un numero arbitrario di slaves/workers (**DataNode**), solitamente uno per ogni nodo del cluster. HDFS espone un namespace del file system e permette ai dati dell'utente di essere memorizzati in file, i quali sono divisi in uno o più blocchi che verranno memorizzati in un insieme di DataNodes.

³Documentazione HDFS: https://hadoop.apache.org/docs/r1.2.1/hdfs_user_guide.html

Il NameNode è un master server che possiede l'albero di tutte le directory del file-system e gestisce operazioni di apertura, rinominazione e chiusura dei file. Risponde alla richiesta del client restituendo una lista di server DataNode pertinenti dove risiedono i dati. Qualsiasi cambiamento al namespace del file system o alle sue proprietà viene registrato dal NameNode.

I DataNode contengono multipli blocchi di dati e la loro responsabilità è quella di eseguire operazioni di creazione, eliminazione e replicazione dei blocchi sotto istruzioni del NameNode. Una volta che il NameNode fornisce la posizione dei dati, le applicazioni client possono parlare direttamente con un DataNode mentre, replicando i dati, le istanze DataNode possono parlare tra loro.

Per garantire una buona tolleranza ai guasti, i blocchi di un file vengono replicati. La dimensione del blocco e il fattore di replica sono configurabili per ogni file e un'applicazione può specificare il numero di repliche al momento della sua creazione o cambiarlo in seguito.

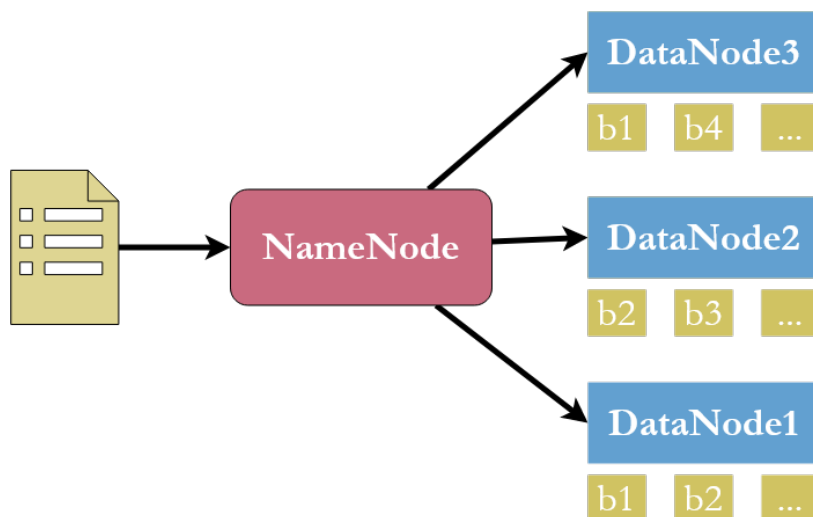


Figura 3.1: Architettura HDFS

3.2.3 MapReduce

MapReduce⁴ è un framework per la creazione di applicazioni in grado di elaborare grandi quantità di dati in parallelo, permettendo una scalabilità massiccia su centinaia o migliaia di server in un cluster Hadoop. È stato reso popolare come modello di programmazione nel 2004 da Jeffery Dean e Sanjay Ghemawat di Google. Nel loro articolo *MapReduce: simplified data processing on large clusters*^[22] hanno discusso l'approccio di Google alla raccolta e all'analisi dei dati dei siti web per l'ottimizzazione della ricerca. Successivamente implementato da Apache, è diventato uno dei componenti principali del progetto Hadoop, nel quale, come il componente HDFS è il responsabile della memorizzazione dei file, MapReduce si occupa di processare questi ultimi.

L'algoritmo implementato prende il nome dalle principali funzioni da cui è composto:

- **Map:** trasforma i dati ricevuti in input in tuple formate da coppie chiave/valore.
- **Reduce:** prende in input l'output generato da Map e combina le coppie chiave-valore in un insieme più piccolo di tuple.

Tra le funzioni Map e Reduce avvengono operazioni di *sorting* e *shuffling* che si occupano di ordinare e raggruppare per chiave l'output di Map e fornirlo come input a Reduce.

MapReduce lavora secondo il principio del *divide et impera*, suddividendo l'operazione di calcolo in diverse parti che verranno processate in modo autonomo per poi ricomporle (o ridurle) in un unico risultato finale. Queste parti sono dette *jobs* e sono formate da sorgente di input, destinazione dei dati e le funzioni Map e Reduce.

⁴Documentazione MapReduce https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html

Generalmente, l'algoritmo viene eseguito sugli stessi nodi su cui risiede HDFS, il che significa che ognuno di essi viene utilizzato sia per il calcolo che per lo storage. Il vantaggio di una tale configurazione è che i compiti possono essere programmati sui nodi dove risiedono i dati e quindi risulta un'elevata larghezza di banda aggregata in tutto il cluster.

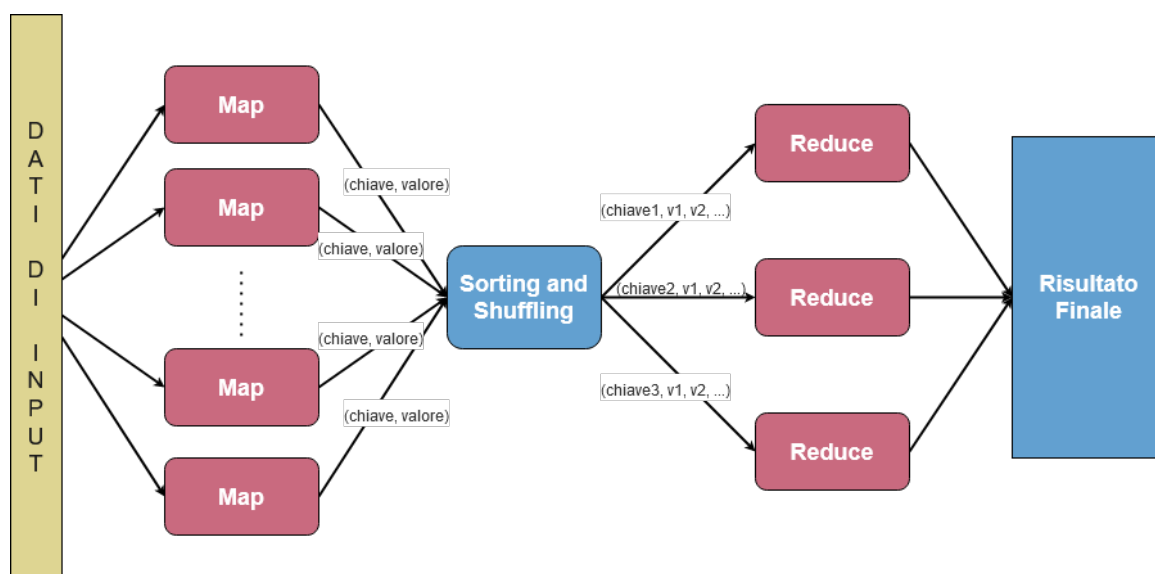


Figura 3.2: Algoritmo MapReduce

Come e da chi vengono gestite internamente queste operazioni? MapReduce, a livello architetturale, presenta due componenti:

- **Job Tracker:** si occupa della gestione delle risorse (CPU e memoria) e del ciclo di vita di un job MapReduce. Il JobTracker distribuisce il lavoro tra i nodi più vicini che contengono i dati da elaborare; nel caso in cui un nodo non possa ospitare il task, si fa poi carico della schedulazione del job nonché della ripetizione dell'esecuzione dei singoli task di MapReduce che si trovano in uno stato di errore.

- **Task Tracker:** Sono le componenti che girano sui singoli nodi e che eseguono effettivamente i task sotto la direzione del JobTracker.

Cosa rende questo framework così importante? I benefici che porta sono notevoli e, alla sua nascita, portò grande innovazione nel campo del calcolo distribuito. I due principali vantaggi che questa tecnologia offre sono:

1. **Elaborazione parallela:** l'intero lavoro è diviso in job che vengono elaborati in modo parallelo simultaneamente, riducendo drasticamente i tempi di esecuzione.
2. **Località dei dati:** invece di spostare tutti i dati per l'elaborazione, il processo completo viene spostato su ogni nodo. Col crescere del quantitativo di dati da processare, può diventare difficile spostarli da un posto all'altro e quindi questa tecnica è considerata un'alternativa assai vantaggiosa.

3.2.4 YARN

Quando i dati hanno cominciato a diventare sempre più grandi, Hadoop File System è stato in grado di immagazzinarli, ma MapReduce è diventato un collo di bottiglia nelle prestazioni. Questo perché, in Hadoop 1.x, il JobTracker si occupava sia della gestione delle risorse, sia dell'elaborazione dei dati. Per questo motivo, in Hadoop 2.0⁵ è stato introdotto *YARN*⁶ (*Yet Another Resources Navigator*) che separa il livello di gestione delle risorse dal livello di elaborazione. L'idea fondamentale di YARN è di dividere queste funzionalità in processi separati:

- **ResourceManager:** è il processo master di YARN ed è responsabile dell'assegnazione e della gestione delle risorse tra tutte le applicazioni. Ogni volta che

⁵Hadoop 2.0 changelog: <https://hadoop.apache.org/release/2.2.0.html>

⁶YARN docs: <https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html>

riceve una richiesta di elaborazione, la inoltra al gestore del nodo corrispondente (NodeManager) e alloca le risorse per il completamento della richiesta. Viene suddiviso in ulteriori due componenti:

- **Scheduler:** esegue la pianificazione in base all'applicazione assegnata e alle risorse disponibili. Non esegue altri compiti come il monitoraggio o il tracking e non garantisce un riavvio se un compito fallisce.
 - **ApplicationManager:** è responsabile di dell'accettazione dell'applicazione e della negoziazione del primo container⁷ dal gestore delle risorse. Riavvia anche il container ApplicationMaster se un job fallisce.
-
- **NodeManager:** è l'agente del framework per ogni macchina ed è responsabile del monitoraggio del loro utilizzo delle risorse (cpu, memoria, disco, rete) e della segnalazione al ResourceManager. Si registra con il ResourceManager e invia *heartbeat* (segnali) con lo stato di salute del nodo. Monitora l'uso delle risorse, esegue la gestione dei log e uccide anche un container in base alle indicazioni del gestore delle risorse.
 - **ApplicationMaster:** è una libreria specifica del framework e ha il compito di negoziare le risorse dal ResourceManager e lavorare con i NodeManager per eseguire e monitorare i job.

⁷Per container si intende un insieme di risorse fisiche come RAM, core di CPU e disco su un singolo nodo.

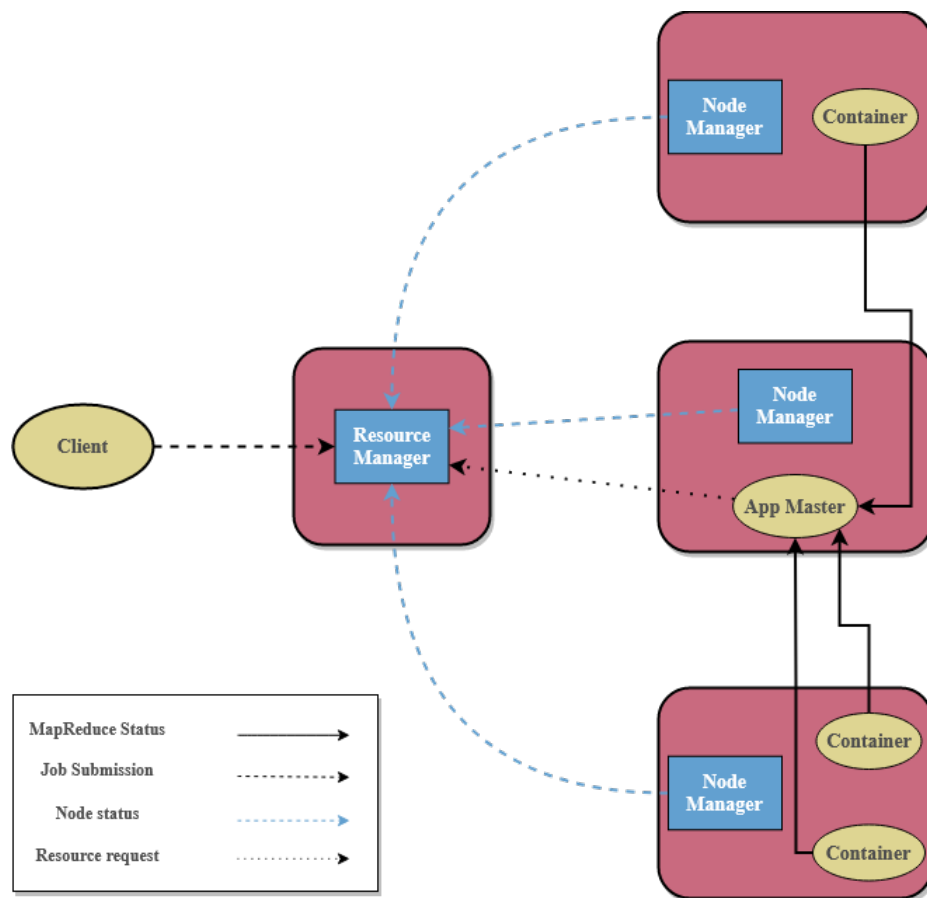


Figura 3.3: Architettura YARN

3.3 Apache Spark

Apache Spark⁸ nasce come un progetto di ricerca al UC Berkeley AMPLab nel 2009 per poi essere reso pubblico sotto forma di progetto open source nel 2010. Acquistato dall'Apache Software Foundation nel 2013, ad oggi è uno dei framework più utilizzati nel campo delle elaborazioni distribuite di grandi quantità di dati, con alle spalle una comunità di centinaia di sviluppatori e centinaia di organizzazioni.

3.3.1 Cosa è Spark

Apache Spark è un motore multilingue per la automazione di ingegneria dei dati, scienza dei dati e apprendimento automatico su macchine a nodo singolo o cluster. La peculiarità di Apache Spark è la sua capacità di elaborare set di dati di grandi dimensioni in maniera efficiente, distribuendo attività di elaborazione dati su più computer, anche integrando Hadoop YARN e HDFS. Questi attributi sono fondamentali per il mondo dei big data e del machine learning.

All'epoca della nascita di questo progetto, Hadoop MapReduce era il motore di programmazione parallela predominante per i cluster, essendo il primo sistema open source ad affrontare l'elaborazione dei dati su migliaia di nodi. L'AMPLab aveva lavorato con molti dei primi utenti di MapReduce per capire i benefici e gli svantaggi di questo nuovo modello di programmazione ed era quindi in grado di sintetizzare una lista di problemi attraverso diversi casi d'uso e iniziare a progettare piattaforme di calcolo più generali. Spark pertanto nasce prendendo i pregi di MapReduce ed introducendo nuove funzionalità che, ad oggi, lo hanno reso uno dei framework più solidi nell'ambito dei big data:

- Spark supporta l'elaborazione in memoria centrale per migliorare le prestazioni

⁸Apache Spark: <https://spark.apache.org>

delle applicazioni, a differenza di MapReduce che deve riportare i dati sul disco dopo ogni azione Map o Reduce.

- Il suo predecessore poteva essere implementato soltanto utilizzando il linguaggio Java. Spark, invece, fornisce connessioni native per i linguaggi di programmazione Java, Scala, Python e R e supporta operazioni SQL.
- Può elaborare grafi ed è anche dotato di una propria libreria di machine learning. Grazie alle sue alte prestazioni, è possibile utilizzarlo sia per l'elaborazione in batch sia per l'elaborazione simil real-time.
- Sfrutta il paradigma del Transfer Learning (il riutilizzo di un modello pre-addestrato su un nuovo problema).
- Permette di elaborare dei dati da vari repository come Hadoop Distributed File System (HDFS), database NoSQL e Apache Hive utilizzando le API di storage di dell'ecosistema Hadoop.
- In MapReduce, ogni operazione è indipendente dall'altra e Hadoop non ha idea di quale verrà dopo. Spark, invece, utilizza un modello di programmazione completamente innovativo basato su grafi diretti aciclici (DAGs).

3.3.2 Architettura

L'architettura di Apache Spark si ispira fortemente a quella di Hadoop. Difatti, presenta una struttura gerarchica **Master-Slaves** dove il nodo master che gestisce i nodi in esecuzione e controlla l'amministratore del cluster.

All'interno del nodo master viene istanziato un *driver*, processo che si occupa di distribuire il codice dell'utente tra i nodi e convertirlo in più attività (*jobs*). Il driver distribuisce questi compiti sui nodi slave (worker) e organizza la loro esecuzione. Le

applicazioni Spark vengono eseguite come serie indipendenti di processi in ambiente distribuito, coordinate dallo **SparkContext**, un processo che funziona come una porta d'accesso a tutte le funzionalità di Spark.

I **nodi worker** eseguono le attività assegnate dal driver su questi nodi. Eseguiti i jobs assegnati, restituiscono il risultato allo SparkContext. È però necessario implementare un gestore del cluster o **cluster manager** (Spark Standalone Cluster Manager, Hadoop Yarn, Apache Mesos, Kubernetes) per mediare tra i workers e driver.

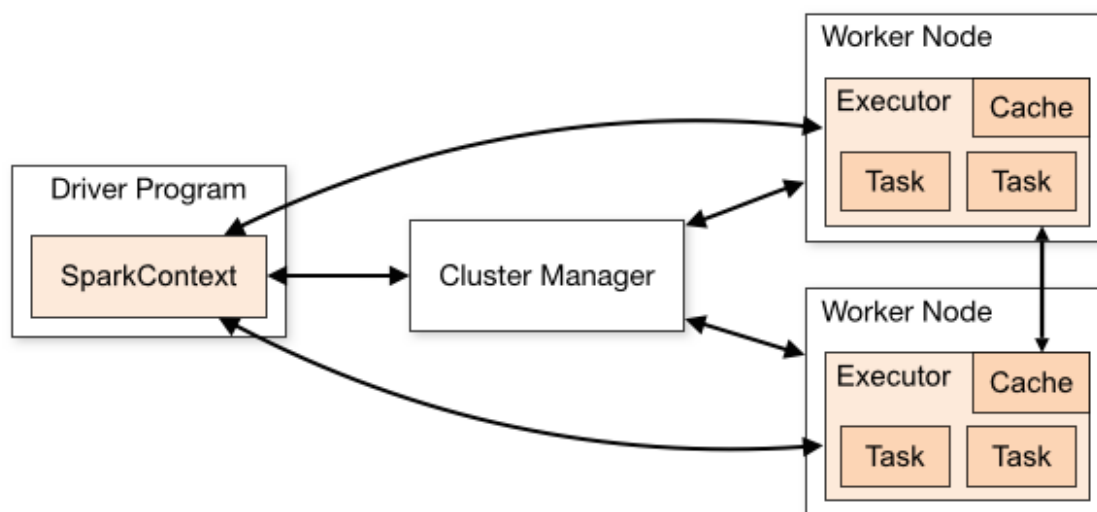


Figura 3.4: Architettura Spark

3.3.3 Spark RDD e DataFrame

Il motore Spark usa principalmente set di *Resilient Distributed Dataset (RDD)* come tipo di dati sottostante. Gli RDD sono raccolte di elementi a tolleranza di errore che possono essere distribuiti tra più nodi in cluster e lavorati in parallelo. Hanno una struttura progettata per nascondere la complessità computazionale agli utenti: non hanno bisogno di definire dove vengono inviati file specifici, quali risorse di calcolo

verranno utilizzate per archiviare o recuperare i file. Sono altamente resilienti, cioè sono in grado di superare rapidamente qualsiasi problema poiché gli stessi pezzi di dati sono replicati su più nodi esecutori, così, anche se un nodo fallisce, un altro elaborerà comunque i dati.

Ci sono due modi per creare RDD - parallelizzando una collezione esistente nel programma driver, o facendo riferimento a un set di dati in un sistema di storage esterno, come un file system condiviso, HDFS, HBase, ecc. Con gli RDD, si possono eseguire due tipi di operazioni:

- **Transformations:** prendono RDD come input e producono uno o più RDD come output. Ogni volta creano un nuovo RDD poiché essi sono immutabili.
- **Actions:** operazioni Spark RDD che danno valori non RDD. I valori dell'azione sono memorizzati nel driver o nel sistema di archiviazione esterno. Un'azione è uno dei modi per inviare dati da Executer al driver.

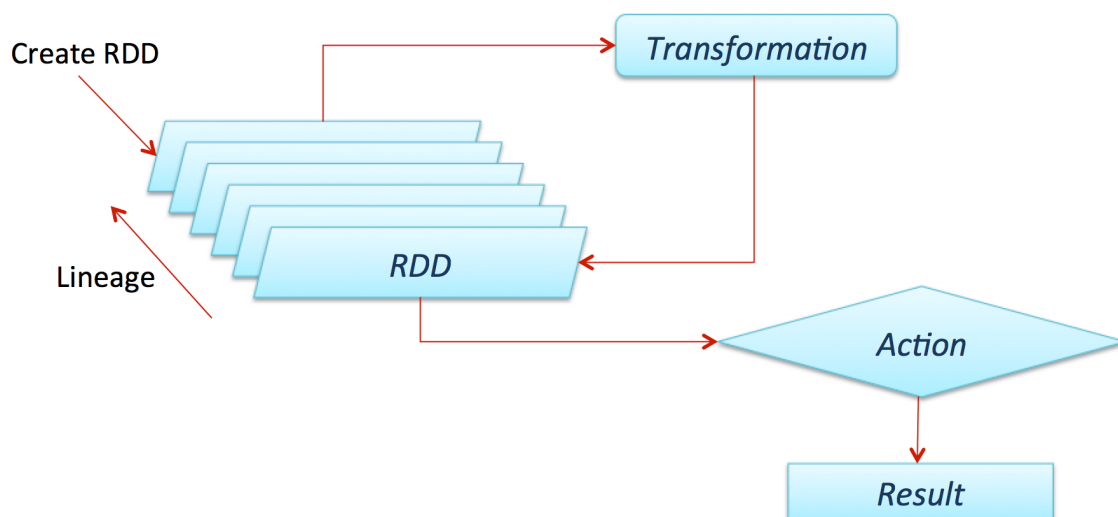


Figura 3.5: Operazioni su RDD

Nel progetto di Tesi è stato però utilizzato un altro tipo astrazione presente in Spark per memorizzare dati, i *DataFrame*. Sono implementati come degli RDD, per-

tanto sono anch'essi una collezione di dati distribuiti. La differenza sta nel fatto che sono organizzati in colonne nominate, come avviene nelle tabelle dei database relazionali. Un'altra caratteristica di DataFrame è che le operazioni sono ottimizzabili da Spark mentre le operazioni su RDD sono imperative e passano attraverso le trasformazioni e le azioni in ordine. Questo perfezionamento viene eseguito da un ottimizzatore di query, *Catalyst Optimizer*, che supporta sia l'ottimizzazione basata sulle regole che quella basata sui costi. Nell'ottimizzazione basata su regole, usa un insieme di regole per determinare come eseguire la query. L'ottimizzazione basata sul costo trova il modo più adatto per eseguire l'istruzione. Nell'ottimizzazione basata sui costi, vengono generati più piani utilizzando le regole e poi viene calcolato il loro costo.

Direction	Year	Date	Weekday	Country	Commodity	Transport_Mode	Measure	Value	Cumulative
Exports	2015	01/01/2015	Thursday	All	All	All	\$	104000000	104000000
Exports	2015	02/01/2015	Friday	All	All	All	\$	96000000	200000000
Exports	2015	03/01/2015	Saturday	All	All	All	\$	61000000	262000000
Exports	2015	04/01/2015	Sunday	All	All	All	\$	74000000	336000000
Exports	2015	05/01/2015	Monday	All	All	All	\$	105000000	442000000

only showing top 5 rows

Figura 3.6: Esempio di struttura di un DataFrame

Durante la fase di sperimentazione, sono stati utilizzati DataFrame per memorizzare i record presenti nei dataset ed effettuare operazioni su di essi. I DataFrame sono stati popolati utilizzando metodi implementati all'interno della struttura dati che permettono di estrarre informazioni da file di diversi formati. Nello specifico sono stati utilizzati per leggere dati da file in formato CSV e CoNLL '03.

3.3.4 Directed Acyclic Graphs in Spark

Con tutti i vari job, Spark crea un flusso logico di operazioni, che è noto come **Directed Acyclic Graph (DAG)**. In Spark, un DAG è un grafo diretto aciclico dove i

vertici rappresentano le strutture dati (RDD o DataFrame) e gli *archi* rappresentano l'operazione da applicare su di esse. Alla chiamata di un'azione, il DAG creato viene sottoposto al *DAGScheduler* che divide ulteriormente il grafo nelle fasi da svolgere per portare a termine il compito. Questo aiuta a: ridurre al minimo il rimescolamento dei dati, ridurre la durata dei calcoli, migliorare l'efficienza del processo nel tempo.

Inoltre, Spark sfrutta la strategia denominata **lazy evaluation**, ovvero posticipa la valutazione di un'espressione finché non è necessaria. Per le trasformazioni, Spark le aggiunge a un DAG e solo quando il driver richiede alcuni dati, questo DAG viene effettivamente eseguito. Ciò significa che, memorizzando ogni dettaglio delle operazioni eseguite su diverse partizioni di RDD, è possibile recuperare facilmente dati persi in caso di fallimento o di perdita di qualsiasi RDD.

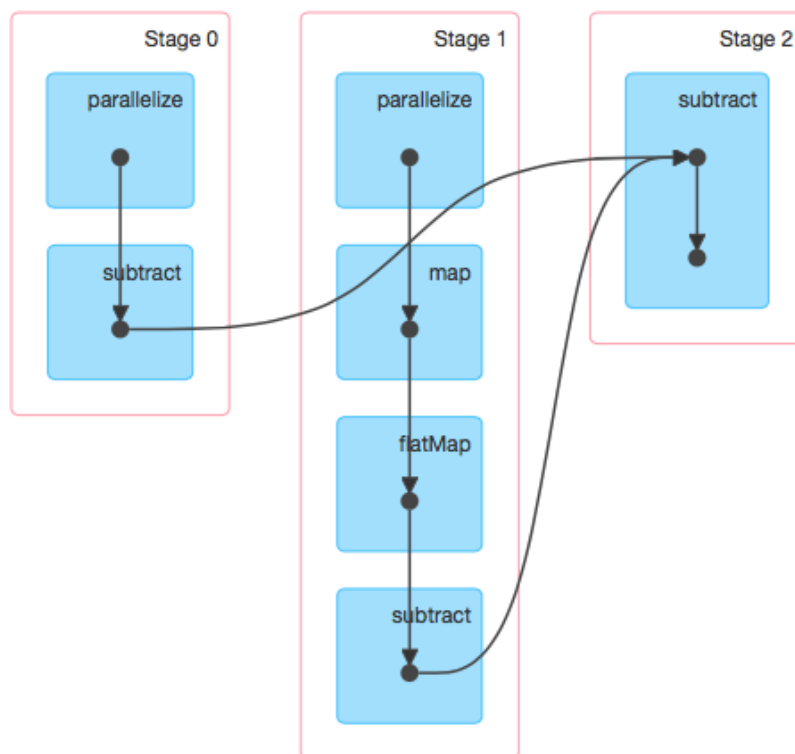


Figura 3.7: Esempio di Spark DAG

3.3.5 Spark Core

Spark Core è la base dell'intero progetto: fornisce il dispacciamento distribuito dei compiti, lo scheduling e le funzionalità I/O di base. Spark Core funziona in parte come un livello API. Oltre al motore Spark Core, l'ambiente API Apache Spark viene fornito insieme ad alcune librerie da utilizzare nelle applicazioni di analisi dei dati.

- **Spark SQL:** è un modulo Spark per l'elaborazione di dati strutturati. A differenza delle API di base di Spark RDD, le interfacce fornite da Spark SQL forniscono a Spark più informazioni sulla struttura dei dati e sul calcolo che viene eseguito. Internamente, Spark SQL utilizza queste informazioni extra per eseguire ottimizzazioni. Dato che quando si calcola un risultato, viene utilizzato lo stesso motore di esecuzione, indipendentemente da quale API/linguaggio si sta utilizzando, gli sviluppatori possono facilmente fare avanti e indietro tra diverse API a loro piacimento. `DataFrame` fa parte di questa libreria.
- **Spark Streaming:** una libreria per l'elaborazione scalabile, ad alta velocità e fault-tolerant di flussi di dati in tempo reale. I dati possono essere ottenuti da molte fonti come Kafka, Kinesis, o socket TCP, possono essere elaborati tramite algoritmi di Machine Learning ed elaborazione grafica ed infine possono essere inviati a file system, database e dashboard live.
- **MLlib:** una libreria di Machine Learning (ML) il cui scopo è di rendere l'apprendimento automatico facile e scalabile. Essa mette a disposizione algoritmi di Machine Learning, Pipelines e diversi altri strumenti per lo svolgimento di operazioni statistiche avanzate sui dati e per creare applicazioni attorno a queste analisi.
- **GraphX:** è un nuovo componente di Spark per i grafi e il calcolo parallelo su di

essi. Per supportare il calcolo dei grafi, GraphX mette a disposizione un insieme di operazioni fondamentali (ad esempio, *subgraph*, *joinVertices*, e *aggregateMessages*). Inoltre, include una crescente collezione di algoritmi e costruttori di grafi per semplificare i compiti di analisi.

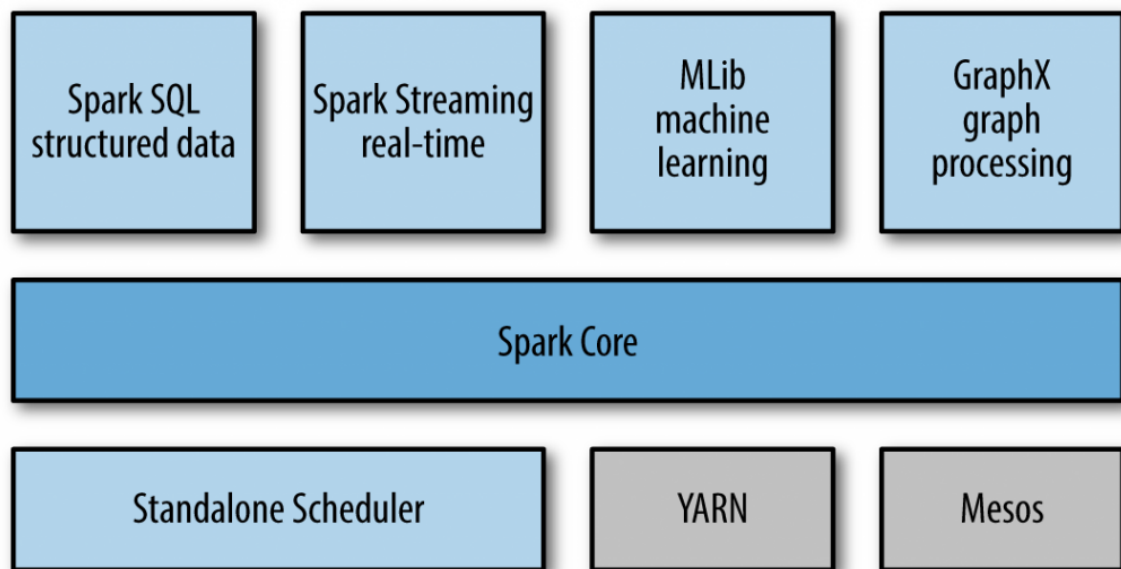


Figura 3.8: Spark core e librerie

3.4 Spark NLP

La popolarità del Natural Language Processing ha fatto sì che, negli ultimi anni, siano state sviluppate decine di librerie che risolvono numerosi problemi legati all'argomento (NLTK, SpaCy, Stanford Core NLP e molte altre). In particolare, negli ultimi anni ha attirato particolarmente l'attenzione Spark NLP⁹, libreria costruita su Apache Spark e Spark ML. Spark NLP, nata nell'ottobre del 2017 dal *John Snow Labs*, è al momento la soluzione adottata maggiormente dalle aziende, grazie soprattutto alle caratteristiche ereditate da Spark che fanno sì che essa ottenga i risultati migliori a livello di accuratezza e velocità di esecuzione su numerosissimi task NLP. Proprio per questo motivo, in questa Tesi, si sono volute testare in prima persona le potenzialità di questo framework.

3.4.1 Cosa è Spark NLP

Spark NLP è una libreria di elaborazione del linguaggio naturale open source, costruita su Apache Spark e Spark ML. Fornisce un'API semplice da integrare con ML Pipelines ed è commercialmente supportato da John Snow Labs. Gli annotatori di Spark NLP utilizzano algoritmi basati su regole, Machine Learning e Deep Learning. La libreria Spark NLP è scritta in Scala e include API Scala, Java e Python per l'uso da Spark. Copre quasi la totalità dei task NLP comuni, tra cui tokenizzazione, lemmatizzazione, part-of-speech tagging, analisi del sentimento, controllo ortografico e altro ancora attraverso numerosi componenti che raggiungono lo stato dell'arte nella maggior parte dei compiti citati. In particolare, come già citato, in questo progetto sono stati trattati, utilizzando questa libreria, Named Entity Recognition e Text Classification.

⁹Spark NLP: <https://nlp.johnsnowlabs.com/>

Spark NLP possiede oltre 70 pipelines ed oltre 90 modelli pre-addestrati¹⁰, sebbene servano da modo per avere un'idea del funzionamento della libreria e non per l'uso in produzione. Essendo un'estensione nativa di Spark ML API, la libreria offre la possibilità di allenare, personalizzare e salvare i modelli in modo che possano essere eseguiti su un cluster, su altre macchine o salvati per un secondo momento (transfer learning).

3.4.2 Come funziona Spark NLP

Spark ML fornisce un insieme di applicazioni di Machine Learning che possono essere costruite utilizzando due componenti principali: **Estimators** e **Transformers**. Gli Estimators hanno un metodo chiamato `fit(data)` che addestra un pezzo di dati a tale applicazione. Il Transformer¹¹ è generalmente il risultato di un processo di addestramento e applica le modifiche al set di dati di destinazione. Questi componenti sono stati incorporati per essere applicabili a Spark NLP. Per combinare più estimators e transformers in un unico flusso di lavoro viene utilizzato il meccanismo delle *Pipelines*. Esse permettono più trasformazioni concatenate lungo un task di Machine Learning restituendo come risultato un'**annotazione**.

Gli annotatori¹² sono la punta di diamante delle funzioni NLP in Spark NLP e sono disponibili in due forme:

- **Annotator Approaches:** sono quelli che rappresentano uno Spark ML Estimator e richiedono una fase di allenamento. Hanno una funzione chiamata `fit(data)` che allena un modello basato su alcuni dati. Producono il secondo tipo di annotatore che è un modello annotatore o trasformatore.

¹⁰Models Hub: <https://nlp.johnsnowlabs.com/models>

¹¹Elenco Transformers: <https://nlp.johnsnowlabs.com/docs/en/transformers>

¹²Elenco annotatori: <https://nlp.johnsnowlabs.com/docs/en/annotators>

- **Annotator Models:** sono modelli spark o trasformatori, cioè hanno una funzione *transform(data)*. Questa funzione prende come input un dataframe al quale aggiunge una nuova colonna contenente il risultato dell'annotazione corrente. Tutti i trasformatori sono additivi, il che significa che aggiungono ai dati correnti, senza mai sostituire o cancellare le informazioni precedenti

Entrambe le forme di annotatori possono essere incluse in una Pipeline. Tutti gli annotatori inclusi in una Pipeline saranno automaticamente eseguiti nell'ordine definito e trasformeranno i dati di conseguenza. Una Pipeline viene trasformata in un **PipelineModel** dopo la fase `fit(data)`. La Pipeline può essere salvata su disco e ricaricata in qualsiasi momento.

Nel capitolo 4 verrà illustrato nel dettaglio come questo framework è stato utilizzato durante la fase di sperimentazione.

3.4.3 Perché usare Spark NLP

Quali sono quindi i punti di forza di Spark NLP?

1. **Accuratezza:** La libreria Spark NLP 2.0 ha ottenuto i migliori risultati accademici peer-reviewed.
2. **Velocità:** Le ragioni della sua velocità sono il motore Tungsten di seconda generazione per i dati colonnari vettoriali in-memoria, nessuna copia del testo in memoria, ampia profilazione, configurazione e ottimizzazione del codice di Spark e TensorFlow, e ottimizzazione per l'addestramento e l'inferenza.
3. **Scalabilità:** Questa libreria è capace di scalare allenamento dei modelli, inferenza e pipelines da una macchina locale ad un cluster con piccoli, se non nessun, cambiamenti di codice.

4. **Performance:** Spark NLP include caratteristiche che forniscono API Java, Scala e Python complete, supporta la formazione su GPU, supporta reti di deep learning definite dall'utente, supporta Spark nativamente, supporta Hadoop (YARN e HDFS).
5. **API in Python, Java e Scala:** Una libreria che supporta più lingue non solo guadagna pubblico, ma permette anche di sfruttare i modelli implementati senza dover spostare i dati avanti e indietro tra gli ambienti di runtime.

3.5 Configurazione dell'architettura utilizzata

Il processo di sperimentazione è stato svolto grazie all'utilizzo di macchine messe a disposizione da *Reveal s.r.l.* Il codice prodotto è stato prima eseguito su di un PC desktop domestico, per poi essere testato su un piccolo cluster di 3 macchine. Su di esse sono stati configurati ed installati l'ambiente Apache Spark, Spark NLP e Hadoop. Per testare la scalabilità offerta dal framework del John Snow Labs, il software è stato eseguito prima utilizzando soltanto una macchina per poi incrementare la potenza di calcolo a disposizione migrando l'esecuzione su di un sistema distribuito.

3.5.1 Hardware

Le configurazioni delle macchine utilizzate sono le seguenti:

Configurazione macchina domestica:

- Sistema operativo: Windows 10 Home
- Numero di CPU: 16
- Dimensione RAM: 16GB

Configurazione cluster (1 master, 2 workers):

- Sistema operativo: CentOS 7
- Numero CPU per driver: 8
- Dimensione memoria per driver: 10 GB
- Numero di CPU per ogni worker: 4
- Dimensione memoria per ogni worker: 10 GB

3.5.2 Software

Come citato precedentemente, l'obiettivo è quello di testare soluzioni per la risoluzione di task di NLP in ambiente distribuito, confrontando i risultati ottenuti in termini di accuratezza e tempi di esecuzione. Per eseguire l'applicazione, all'interno del sistema sono stati configurati: l'ambiente Spark, un database distribuito HDFS ed il resource manager YARN.

Nella macchina master risiedono il driver di Spark, il **NameNode** (daemon per HDFS), responsabile della gestione del namespace del file system e il **Resource-Manager** (daemon per YARN), responsabile dell'assegnazione e della gestione delle risorse tra tutte le applicazioni. Quando Spark NLP richiede al sistema l'esecuzione di un job o di un DAG di jobs, questa richiesta viene inoltrata all'istanza di YARN in esecuzione sul nodo master che verifica se l'esecuzione è possibile e, in caso affermativo, registra l'applicazione assegnandogli un Job ID e aggiungendolo alla coda dei job da eseguire.

Ogni volta che un job viene estratto dalla coda ed è mandato in esecuzione, il ResourceManager seleziona casualmente un DataNode (processo di HDFS) e avvia su quello stesso nodo un processo ApplicationMaster (processo di YARN). Al passo successivo, l'ApplicationMaster comunicherà con il NameNode che a sua volta si occuperà di accertarsi della posizione in cui si trovano i file (blocchi) all'interno del cluster e di quante risorse ha bisogno il job.

Una volta svolte tutte le valutazioni, l'ApplicationMaster invia le informazioni sulla richiesta di risorse al ResourceManager che le esamina e inoltra una richiesta di allocazione di queste, sotto forma di container, ai nodi del cluster. Questi container sono detti *Esecutori*. I NodeManager di ogni singolo nodo worker si occuperanno quindi di allocare le risorse come da richiesta del ResourceManager.

Infine, gli esecutori inizieranno l'esecuzione dei job a loro assegnati e proseguiranno la comunicazione direttamente con il programma driver (SparkContext). L'output sarà direttamente restituito al client.

Sul cluster utilizzato in questo progetto, l'applicazione è stata eseguita nella cosiddetta *Client Mode*, ovvero, il driver Spark viene avviato sulla stessa macchina da cui viene inviata l'applicazione.

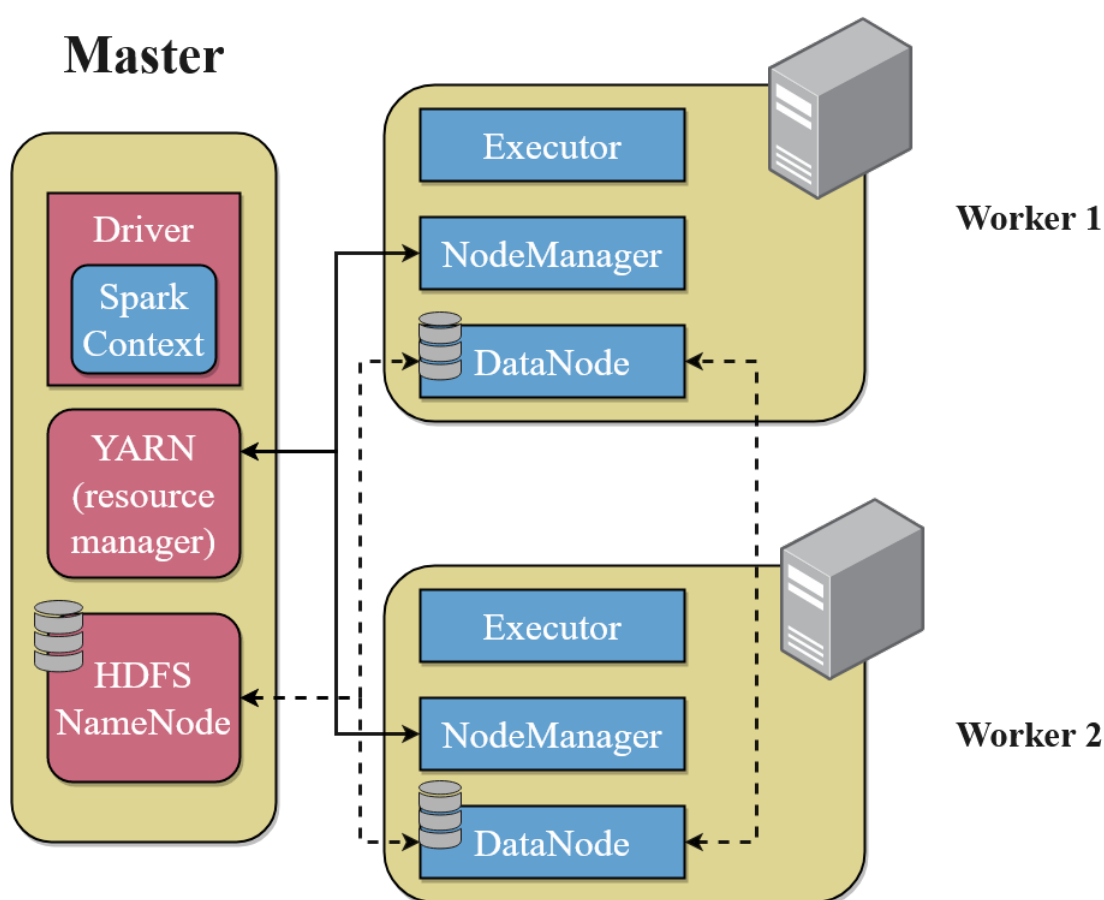


Figura 3.9: Architettura del cluster

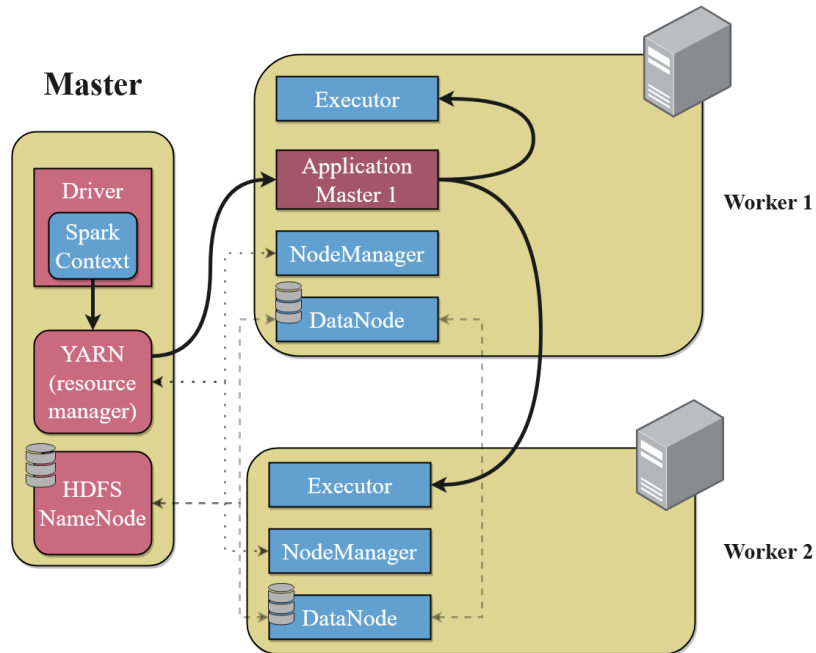


Figura 3.10: Richiesta di esecuzione dell'applicazione

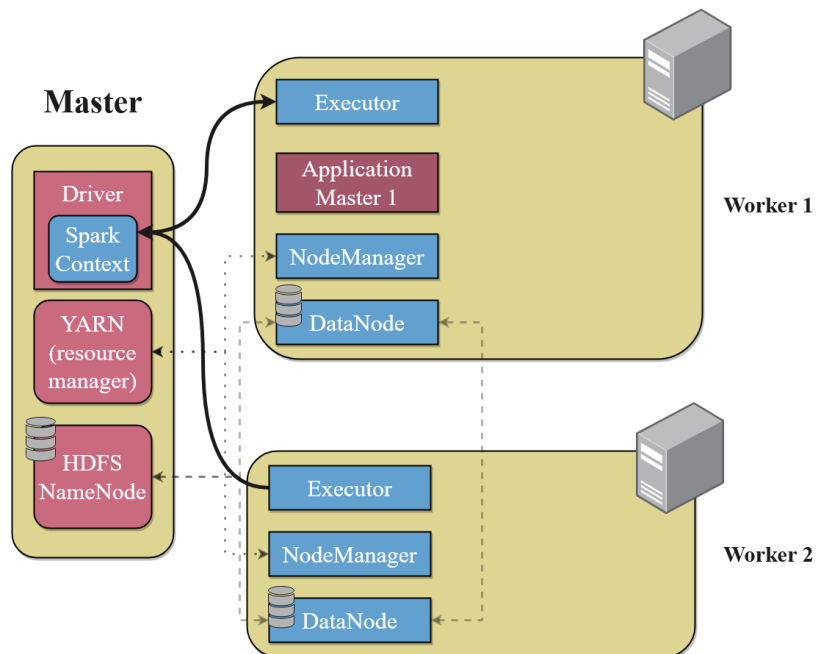


Figura 3.11: Esecuzione dei compiti: interazione tra esecutori e driver

Sperimentazione

In questo capitolo verrà trattata nel dettaglio la fase di sperimentazione dalla quale nasce questa Tesi. Verranno trattati i task che il software prodotto tenta di risolvere e come sono state introdotte ed utilizzate le tecnologie descritte nel capitolo 3. Infine seguirà un'analisi dei risultati ottenuti.

4.1 Text Classification

In questa sezione verrà trattato il task di *Text Classification* e verrà illustrato il dataset utilizzato.

4.1.1 Descrizione del task

La **classificazione del testo**, conosciuta anche come *tagging del testo* o *categorizzazione del testo*, è il processo di categorizzazione dei corpora in gruppi organizzati. Utilizzando il Natural Language Processing, i classificatori possono analizzare automaticamente i testi e poi assegnare loro un insieme di tag o categorie predefinite in base al suo contenuto. Manning et al. [29], ricercatori della Stanford University, definiscono il problema nella maniera seguente.

Nella classificazione del testo, ci viene data una descrizione $d \in \mathbb{X}$ di un documento,

dove \mathbb{X} è lo *spazio dei documenti*, e un insieme fissato di *classi* $\mathbb{C} = c_1, c_2, \dots, c_j$, chiamate anche *categorie* o *etichette*. Tipicamente, lo spazio dei documenti \mathbb{X} è un qualche tipo di spazio di grandi dimensioni (high-dimensional space), e le classi sono definite dall'uomo per la necessità di una specifica applicazione. Ci viene fornito anche un insieme per l'addestramento \mathbb{D} di documenti etichettati $\langle d, c \rangle \in \mathbb{X} \times \mathbb{C}$. Per esempio:

$$\langle d, c \rangle = \langle \text{Pechino entra nell'Organizzazione Mondiale del Commercio, China} \rangle$$

per il documento *Pechino entra nell'Organizzazione Mondiale del Commercio* e la classe *China*.

Utilizzando un *algoritmo di apprendimento*, vogliamo poi creare un classificatore o una *funzione di classificazione* γ che mappa documenti in classi:

$$\gamma : \mathbb{X} \rightarrow \mathbb{C}$$

Questo tipo di apprendimento è chiamato **apprendimento supervisionato** perché un supervisore (la persona che definisce le classi ed etichetta i documenti utilizzati per l'addestramento) serve come un *insegnante* che dirige il processo di apprendimento. Per il momento, consideriamo solo i problemi *one-of* in cui un documento è membro esattamente di una classe: il nostro obiettivo nella classificazione del testo è un'alta accuratezza sui dati di test.

La classificazione del testo sta diventando una parte sempre più importante nelle aziende, poiché permette di ottenere facilmente informazioni dai dati e automatizzare i processi aziendali. Alcuni degli esempi e dei casi d'uso più comuni per la classificazione automatica del testo sono i seguenti:

- **Sentiment Analysis:** il processo di capire se un dato testo sta parlando positivamente o negativamente di un dato argomento (ad esempio per scopi di monitoraggio del marchio).

- **Language Detection:** la procedura di rilevamento della lingua di un dato testo (ad esempio sapere se un ticket di supporto in arrivo è scritto in inglese o in spagnolo per indirizzarlo automaticamente al team appropriato).
- **Topic Detection:** il compito di identificare il tema o l'argomento di una porzione di testo (ad esempio, sapere se una recensione di un prodotto da parte di un cliente riguarda la facilità d'uso, il supporto clienti o il prezzo). Questo è il compito che è stato studiato durante la fase di analisi descritta in questo capitolo.

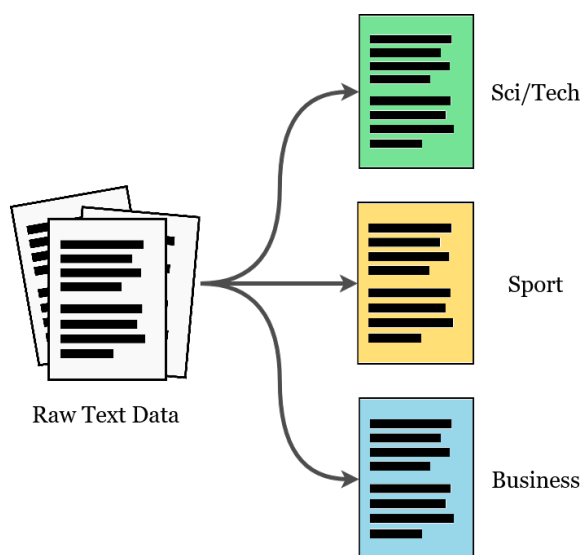


Figura 4.1: *Text Classification: topic detection*

4.1.2 Il dataset

Il dataset utilizzato è stato messo a disposizione dagli sviluppatori di SparkNLP (John Snow Lab) per un workshop da loro svolto e reso disponibile sul loro profilo Github¹.

Il dataset è composto da due file: *news_category_train.csv* con 120.000 esempi per l'addestramento e *news_category_test.csv* con 7.600 esempi per il testing. Ogni

¹Repository: <https://github.com/JohnSnowLabs/spark-nlp-workshop/>

riga di entrambi i file è formata dai campi **description** e **category**. Il primo contiene il testo da classificare, il secondo indica di quale categoria fa parte il testo. Ogni testo appartiene esattamente ad una sola classe. In totale, ci sono 4 categorie: *Business*, *Sci/Tech*, *World*, *Sports*.

category	description
Business	Unions representing workers at Turner Newall say they are 'disappointed' after talks with stricken parent firm Federal Mogul.
Sci/Tech	TORONTO, Canada A second team of rocketeers competing for the \$36.10 million Ansari X Prize, a contest for privately funded suborbital space flight, has officially announced the first launch date for its manned rocket.
Sci/Tech	A company founded by a chemistry researcher at the University of Louisville won a grant to develop a method of producing better peptides, which are short chains of amino acids, the building blocks of proteins.
Sci/Tech	It's barely dawn when Mike Fitzpatrick starts his shift with a blur of colorful maps, figures and endless charts, but already he knows what the day will bring. Lightning will strike in places he expects. Winds will pick up, moist places will dry and flames will roar.
Sci/Tech	Southern California's smog fighting agency went after emissions of the bovine variety Friday, adopting the nation's first rules to reduce air pollution from dairy cow manure.

Figura 4.2: Porzione del dataset per Text Classification

4.1.3 Spark NLP per Text Classification

Spark NLP dispone di un annotatore chiamato *ClassifierDL*, un classificatore multi-classe che utilizza una Deep Neural Network e supporta fino a 100 classi diverse. *ClassifierDL* richiede dei `SENTENCE_EMBEDDINGS` (ovvero embedding di intere frasi) come input e restituisce una predizione sulla categoria a cui appartiene il testo.

I dataset estratti dai file in formato CSV (descritti nel paragrafo 4.1.2) vengono importati in uno `Spark Dataframe` (argomento trattato nel paragrafo 3.3.3).

Per ottenere gli embeddings a partire dal testo, Spark NLP mette a disposizione diverse alternative; i due componenti utilizzati in questo progetto sono:

- **Universal Sentence Encoder**

- **Bert Sentence Embeddings**

Non è il nostro scopo studiare come avviene l'addestramento dei modelli utilizzati per la produzione degli embeddings, pertanto verranno utilizzati modelli pre-addestrati presenti nella sezione *NLP Models Hub*² della documentazione di Spark NLP. Per la precisione sono stati importati:

- `tfhub_use`³ per `UniversalSentenceEncoder`, addestrato con una Deep Averaging Network (DAN)
- `sent_bert_base_cased`⁴ per `BertSentenceEmbeddings` (24 Layers di dimensione 1024).

Le *pipelines* sono costruite seguendo il seguente ordine:

1. **Document Assembler**: prepara i dati in un formato processabile da Spark NLP.
2. **Encoder** (USE o Bert Sentence Embeddings): aggiunge una colonna al Dataframe dove, per ogni riga viene aggiunto il vettore ottenuto codificando il testo.
3. **ClassifierDL**: data in input la colonna degli embeddings ottenuta dallo step precedente, si occupa della classificazione ed aggiunge un'ulteriore colonna nella quale viene inserita la predizione della classe associata al testo.

²Models Hub: <https://nlp.johnsnowlabs.com/models>

³`tfhub_use`: https://nlp.johnsnowlabs.com/2020/04/17/tfhub_use.html

⁴`sent_bert_base_cased`: https://nlp.johnsnowlabs.com/2020/08/25/sent_bert_base_cased.html

4.2 Named Entity Recognition

In questa sezione verrà trattato il task di *Named Entity Recognition* e verranno illustrati i due dataset utilizzati, rispettivamente in lingua inglese e in lingua italiana.

4.2.1 Descrizione del task

Il **Named Entity Recognition (NER)** è uno dei compiti di pre-elaborazione dei dati. Comporta l'identificazione delle entità chiave nel testo e la classificazione di esse in un insieme di categorie predefinite. Un'entità è fondamentalmente la cosa di cui si parla o a cui ci si riferisce costantemente nel testo.

Per imparare cos'è un'entità, un modello NER deve essere in grado di rilevare una parola, o una stringa di parole che formano un'entità (ad esempio *New York City*), e sapere a quale categoria appartiene. Esistono diversi modi di approcciarsi a questo task:

- Un metodo è quello di addestrare il modello per la **classificazione multi classe** utilizzando diversi algoritmi di Machine Learning, ma è un compito molto impegnativo. Innanzitutto esso richiede un sacco di etichettatura ed inoltre il modello richiede anche una profonda comprensione del contesto per affrontare l'ambiguità delle frasi.
- Un altro modo è il **Conditional Random Field**^[38]. È un modello probabilistico che può essere usato per modellare dati sequenziali come le parole. Il CRF può catturare una profonda comprensione del contesto della frase.
- il **Deep Learning Based NER** è molto più accurato del metodo precedente, poiché è in grado di assemblare le parole. Questo è dovuto al fatto che usa il word embedding, che è in grado di capire la relazione semantica e sintattica tra

le varie parole. È anche in grado di apprendere l'analisi di parole specifiche per l'argomento e di alto livello automaticamente. Questo rende il deep learning NER applicabile per eseguire più compiti.

La tecnologia utilizzata in questo progetto per risolvere il task di Named Entity Recognition è basata proprio sul Deep Learning e sfrutta embedding generati dal testo grazie all'utilizzo di modelli preaddestrati presenti in Spark NLP.

Il Named Entity Recognition viene usato per elaborare grandi quantità di contenuti testuali. Un algoritmo per l'estrazione delle entità può trovare persone, organizzazioni, prodotti, luoghi e diverse altre entità all'interno di interi libri, documenti e articoli vari. Queste informazioni possono poi essere utilizzare per la **categorizzazione automatica** sulla base della quale i risultati di ricerca possono essere compilati in modo più preciso, i contenuti possono essere curati in **cluster tematici**, i post relativi ai contenuti possono essere mostrati all'utente o la **pubblicità mirata** può essere riprodotta. Oltre ad essere utilizzate nei portali di notizie, le caratteristiche di raccomandazione dei servizi di media si basano anche sul riconoscimento delle entità nominate. Un altro campo di applicazione oltre all'industria dei media sarebbe il servizio di **Google AdSense** o l'ordinamento delle richieste di supporto via e-mail o chat.

Non ci sono **standard di etichettatura**. Le etichette sono in gran parte orientate secondo la necessità dell'applicazione: generalmente si usano le classi di etichette radice *Persona*, *Organizzazione*, *Prodotto*, *Luoghi*, alle quali si aggiungono le etichette di durata e quantità (*tempo* e *quantità*). A queste entità radice viene poi aggiunto un secondo livello gerarchico: *Organizzazione Commerciale*, *Organizzazione Non profit* per esempio, permettono di affinare la descrizione delle entità.

In campagne recenti (Ester 2^[13] e Automatic Content Extraction (ACE)^[12]) ci sono 5-6 classi radice, e un totale di 40-50 classi con sottosezioni di etichettatura.

Alcuni sistemi di motori di domande e risposte (che usano entità nominate) possono usare diverse centinaia di classi.

Alcuni degli esempi e dei casi d'uso più comuni per il riconoscimento delle entità nominate sono i seguenti:

- **Categorizzazione dei ticket del supporto clienti:** i grandi marchi e le aziende devono esaminare regolarmente una migliaia di ticket di assistenza clienti. L'estrazione di entità nominate può aiutare a categorizzare questi ticket di assistenza clienti in base alla domanda per poi assegnarli al giusto responsabile dell'assistenza clienti.
- **Raccomandazione di contenuto:** molte applicazioni moderne e siti web di e-commerce si basano su sistemi di raccomandazione per migliorare l'esperienza complessiva dell'utente. Un grande esempio di questo è rappresentato dalle piattaforme di streaming video ampiamente utilizzate come Netflix e YouTube. Usano il named entity recognition per analizzare la cronologia di ricerca degli utenti e raccomandare suggerimenti basati su di essa.
- **Estrarre informazioni dal feedback dei clienti:** le recensioni online su varie piattaforme sono una grande fonte di feedback. Possono aiutare a identificare cosa piace e cosa no ai clienti. L'estrazione di entità nominate può aiutare a categorizzare le recensioni e ad identificare i problemi ricorrenti.
- **Processamento dei curricula:** trovare un candidato capace non è un compito facile, i reclutatori devono passare manualmente attraverso un sacco di curricula e analizzare le loro qualifiche, abilità, esperienza e altro. Questo può richiedere una notevole quantità di tempo, rendendolo un processo lungo e noioso. L'estrazione di entità nominate viene spesso utilizzata per analizzare il testo in tonnellate di curricula per trovare i candidati più idonei.

- **Annotazione semantica:** l'annotazione semantica può essere definita come il processo di combinare vari pezzi di informazione a concetti come persone, luoghi e cose. A differenza delle annotazioni tipiche, le annotazioni semantiche coinvolgono l'identificazione e l'analisi del testo, l'estrazione di concetti, l'estrazione di relazioni e l'indicizzazione.

Durante la fase di sperimentazione si è deciso di trattare nello specifico proprio questo compito.

In fact, the **Chinese** **NORP** market has the **three** **CARDINAL** most influential names of the retail and tech space – **Alibaba** **GPE**, **Baidu** **ORG**, and **Tencent** **PERSON** (collectively touted as **BAT** **ORG**), and is betting big in the global **AI** **GPE** in retail industry space. The **three** **CARDINAL** giants which are claimed to have a cut-throat competition with the **U.S.** **GPE** (in terms of resources and capital) are positioning themselves to become the 'future **AI** **PERSON** platforms'. The trio is also expanding in other **Asian** **NORP** countries and investing heavily in the **U.S.** **GPE** based **AI** **GPE** startups to leverage the power of **AI** **GPE**. Backed by such powerful initiatives and presence of these conglomerates, the market in APAC AI is forecast to be the fastest-growing **one** **CARDINAL**, with an anticipated **CAGR** **PERSON** of **45%** **PERCENT** over **2018 - 2024** **DATE**.

Figura 4.3: NER: annotazione semantica

4.2.2 Dataset in lingua inglese

Il dataset utilizzato è la versione in inglese del task CoNLL-2003, disponibile nella repository Github⁵ di Spark NLP.

Il dataset è composto da tre file: *eng.train* con 14.041 esempi per l'addestramento, *eng.test* con 6.603 esempi per il testing.

Il primo elemento su ogni riga è una parola, il secondo un tag **part-of-speech** (POS), il terzo un tag **chunk sintattico** e il quarto il tag **named entity**. I tag chunk e i tag named entity hanno il formato *I-TYPE* che significa che la parola è dentro una frase di tipo *TYPE*. Solo se due frasi dello stesso tipo si susseguono immediatamente,

⁵Dataset: <https://github.com/JohnSnowLabs/spark-nlp/tree/master/src/test/resources/conll2003>

la prima parola della seconda frase avrà il tag *B-TYPE* per mostrare che inizia una nuova frase. Una parola con il tag *O* non fa parte di una frase. Le entità presenti nel dataset sono: *persone*, *luoghi*, *organizzazioni* ed entità *varie* che non appartengono ai tre gruppi precedenti.

```
-DOCSTART- -X- -X- O

EU NNP B-NP B-ORG
rejects VBZ B-VP O
German JJ B-NP B-MISC
call NN I-NP O
to TO B-VP O
boycott VB I-VP O
British JJ B-NP B-MISC
lamb NN I-NP O
. . O O

Peter NNP B-NP B-PER
Blackburn NNP I-NP I-PER

BRUSSELS NNP B-NP B-LOC
1996-08-22 CD I-NP O
```

Figura 4.4: Porzione del dataset per NER in lingua inglese

4.2.3 Dataset in lingua italiana

Il dataset utilizzato per l'italiano è stato generato utilizzando training e test set di Named Entity Recognition di EVALITA 2009⁶. La valutazione è basata sull' *Italian Content Annotation Bank* (I-CAB) dove le entità nominate sono annotate nel formato *IOB* (dove "B-begin" e "I-inside" denotano i token appartenenti a Named Entities e "O-outside" è usato per tutti gli altri token). Per la precisione sono state rigenerate le quattro colonne presenti anche nel dataset in lingua inglese (vedi pagina 62):

1. La prima colonna contiene ogni **token** iniziale della frase.
2. La seconda il **POS tag**, secondo il tagset ISST-TANL⁷.

⁶Evalita 2009: <https://www.evalita.it/evalita-2009/tasks/entity-recognition/>

⁷ISST-TANL: <http://www.italianlp.it/docs/ISST-TANL-POStagset.pdf>

3. La terza colonna NON contiene il **chunk sintattico**, che non era previsto nella competizione. Per simmetria con il dataset inglese è stata aggiunta la colonna, ma ogni riga contiene solo il token " _".
4. La quarta colonna racchiude il tag della **named entity** che rispecchia la descrizione del sito, riportata anche sopra.

Il dataset è composto da due file: *I-CAB-evalita09-NER-train_utf8.tsv* con 11.227 esempi per l'addestramento e *I-CAB-evalita09-NER-test_utf8.tsv* con 4.136 esempi per il testing.

Le entità presenti nel dataset sono: *persone*, *luoghi*, *organizzazioni* ed entità *geopolitiche*.

```
-DOCSTART-X-X-O  
  
La RS _ O  
truffa SS _ O  
di E _ O  
Massimo SS _ B-PER  
Bassoli SP _ I-PER  
del ES _ O  
« XPO _ O  
Giornale SS _ B-ORG  
d' E _ I-ORG  
Italia SPN _ I-ORG  
» XPO _ O  
per E _ O  
ottenere VF _ O  
i RP _ O  
contributi SP _ O  
sull' ES _ O  
editoria SS _ O
```

Figura 4.5: Porzione del dataset per NER in lingua italiana

4.2.4 Spark NLP per Named Entity Recognition

SparkNLP dispone di un annotatore chiamato **NerDL** che permette di addestrare un modello NER generico basato su reti neurali e deep learning. I dati devono avere colonne di tipo `DOCUMENT`, `TOKEN`, `WORD_EMBEDDINGS` e un'ulteriore colonna etichetta

di tipo `NAMED_ENTITY` per l'addestramento supervisionato.

I dataset (descritti nelle sezioni 4.2.2, 4.2.3) sono stati importati attraverso il metodo `sparknlp.training.CoNLL.readDataset(spark, path)` che legge il set di dati (nel formato di CoNLL 2003) da una risorsa esterna e genera uno `Spark Dataframe`.

A differenza di quanto fatto per il task di Text Classification che si serviva di `SENTENCE_EMBEDDINGS`, per risolvere questo compito è stato necessario ottenere dei `WORD_EMBEDDINGS` dai singoli token. Per ottenere i vettori a partire dai token, Spark NLP presenta alcune alternative. I due componenti che sono stati scelti ed impiegati in questo progetto sono:

- **WordEmbeddings**, con il modello `glove_100d`⁸ pre-addestrato utilizzando i dataset Wikipedia 2014 e Gigaword 5 (6B token, 400K vocaboli, non classificati, vettori: 50d, 100d, 200d e 300d). I risultati sono vettori di dimensione 100.
- **BertEmbeddings**, con il modello pre-addestrato `bert_base_cased`⁹ per la lingua inglese e `bert_base_italian_cased`¹⁰ (12 Layers di dimensione 768).

Le *pipelines* sono costruite seguendo il seguente ordine:

1. **Encoder**: aggiunge una colonna al Dataframe dove, per ogni riga viene aggiunto il vettore ottenuto codificando il token.
2. **NerDLApproach**: data in input la colonna degli embeddings ottenuta dallo step precedente, si occupa del riconoscimento delle named entities ed aggiunge un'ulteriore colonna dove, per ogni riga viene inserita la predizione della classe associata all'entità.

⁸Modello `glove_100d`: https://nlp.johnsnowlabs.com/2020/01/22/glove_100d.html

⁹`bert_base_cased`: https://nlp.johnsnowlabs.com/2020/08/25/bert_base_cased.html

¹⁰`bert_base_italian_cased`: https://nlp.johnsnowlabs.com/2021/05/20/bert_base_italian_cased_it.html

4.2.5 Tabelle riassuntive dataset

Text Classification

Utilizzo	Formato	Numero di esempi
Addestramento	CSV	120.000
Test	CSV	7.600

Tabella 4.1: Dataset per Text Classification

Named Entity Recognition

Lingua	Utilizzo	Formato	Numero di esempi
Inglese	Addestramento	CoNLL '03	14.041
	Test	CoNLL '03	6.603
Italiano	Addestramento	CoNLL '03	11.227
	Test	CoNLL '03	4.136

Tabella 4.2: Dataset per NER

4.3 Analisi Prestazionale

Costruite le pipelines, a partire da ognuna di esse è stato addestrato un modello utilizzando il metodo `.fit()`:

```
pipelineModel = pipeline.fit(train_data)
```

Una volta pronto il modello, è stato testato utilizzando il metodo `.transform()`. Facendo ciò, viene generato un nuovo DataFrame aggiungendo a quello di partenza una colonna con le previsioni fatte dal modello:

```
predictions = pipelineModel.transform(test_data)
```

Utilizzando i modelli addestrati, sono state svolte due tipi di valutazioni. La prima tiene conto dell'accuratezza delle predizioni ottenute dai modelli utilizzando i dataset di test descritti nelle sezioni 4.1.2, 4.2.2 e 4.2.3. La seconda misura i tempi di esecuzione ottenuti parallelizzando il lavoro ed opera su di un dataset di 760.000 record.

4.3.1 Metriche

A partire dalle previsioni prodotte, è stato generato un *classification report*. Si tratta di un rapporto che descrive varie metriche relative a quanto bene ha funzionato un modello di machine learning e che si ottengono confrontando i risultati prodotti dal modello con quelli attesi. Il report mostra le principali metriche di classificazione:

- **Precisione (Precision):** la capacità di un classificatore di identificare solo le istanze corrette per ogni classe.
- **Richiamo (Recall):** la capacità di un classificatore di trovare tutte le istanze corrette per una classe.
- **F1-Score:** media armonica ponderata di precisione e richiamo normalizzata

tra 0 e 1. Un punteggio F di 1 indica un equilibrio perfetto, poiché precisione e richiamo sono inversamente correlati.

- **Accuratezza (Accuracy):** valore che indica quanto spesso ci si può aspettare che il modello di machine learning preveda correttamente un risultato sul numero totale di volte che ha fatto previsioni.
- **Support:** numero di occorrenze effettive di una classe nel dataset.

Per i due task sono state utilizzate due tipi di valutazione differenti:

- **Text Classification:** per questo task è stata scelta una valutazione rigida. Per ogni testo presente nel dataset, se la previsione del modello corrisponde al valore atteso, allora viene valutato come corretto, altrimenti come incorretto. Pertanto, il valore a cui si fa riferimento è l'accuratezza, che si calcola come il rapporto tra le predizioni corrette ed il numero di esempi totali.
- **Named Entity Recognition:** per il NER, si prende come riferimento la valutazione proposta nel task CoNLL-2003¹¹ e nello script *conlevall*¹². Le prestazioni dei sistemi sono misurate in termini di precisione, richiamo e f1-score, dove:
"La precisione è la percentuale di named entities trovate dal sistema di apprendimento che sono corrette. Recall è la percentuale di entità presenti nel corpus che vengono trovate dal sistema. Una named entity è corretta solo se c'è una corrispondenza esatta dell'entità corrispondente nel file di dati".
Pertanto, il calcolo non tiene in considerazione le previsioni del tag 'O' (ovvero, chunk che non è una named entity) che sono invece valutate dall'accuracy.

¹¹CoNLL-2003 Paper: <https://aclanthology.org/W03-0419.pdf>

¹²conllevall script in perl: <https://www.clips.uantwerpen.be/conll2000/chunking/conllevall.txt>

4.3.2 Performance

Text Classification - Inglese

Embedding	Accuracy
USE	89%
Sentence BERT (base_case)	89%

Tabella 4.3: Risultati per Text Classification

	precision	recall	f1-score	support
Business	0.84	0.85	0.84	7540
Sci/Tech	0.88	0.85	0.87	7812
Sports	0.98	0.95	0.96	7832
World	0.87	0.92	0.90	7216
accuracy			0.89	30400
macro avg	0.89	0.89	0.89	30400
weighted avg	0.89	0.89	0.89	30400

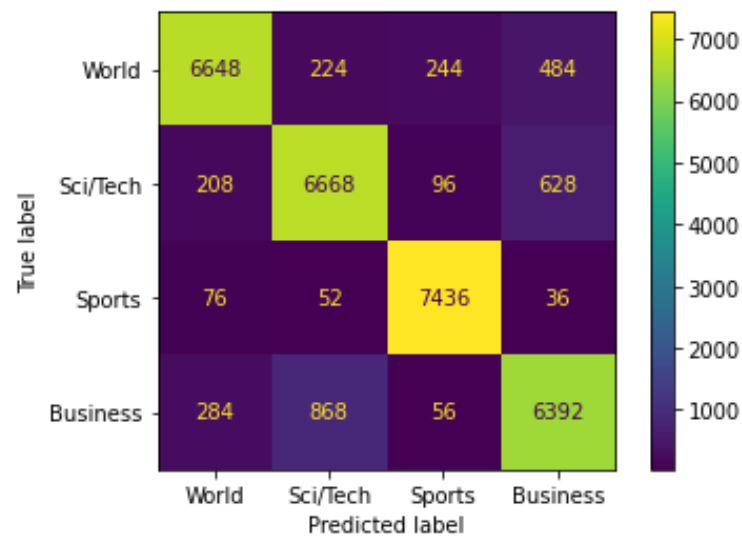


Figura 4.6: Text Classification - USE: classification report

	precision	recall	f1-score	support
Business	0.85	0.86	0.85	7544
Sci/Tech	0.88	0.85	0.87	7868
Sports	0.97	0.94	0.96	7844
World	0.87	0.92	0.90	7144
accuracy			0.89	30400
macro avg	0.89	0.89	0.89	30400
weighted avg	0.90	0.89	0.89	30400

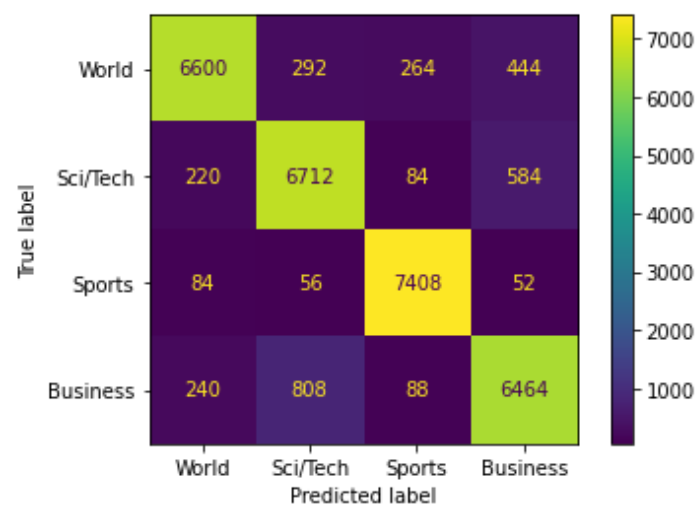


Figura 4.7: Text Classification - BERT: classification report

Named Entity Recognition - Inglese

Embedding	F1-Score
GloVe	85.8%
BERT (base_case)	86.5%

Tabella 4.4: Risultati per NER in Inglese

	precision	recall	f1-score	support
LOC	0.89	0.90	0.90	14020
MISC	0.82	0.71	0.76	6496
ORG	0.80	0.79	0.80	12008
PER	0.92	0.95	0.93	13836
micro avg	0.87	0.86	0.86	46360
macro avg	0.86	0.84	0.85	46360
weighted avg	0.87	0.86	0.86	46360

Figura 4.8: NER in Inglese - GloVe: classification report

	precision	recall	f1-score	support
LOC	0.81	0.93	0.87	14020
MISC	0.81	0.76	0.79	6496
ORG	0.87	0.75	0.80	12008
PER	0.94	0.95	0.95	13836
micro avg	0.86	0.87	0.87	46360
macro avg	0.86	0.85	0.85	46360
weighted avg	0.87	0.87	0.86	46360

Figura 4.9: NER in Inglese - BERT: classification report

Named Entity Recognition - Italiano

Embedding	F1-Score
GloVe	65%
BERT (base_case)	60.5%

Tabella 4.5: Risultati per NER in Inglese

	precision	recall	f1-score	support
GPE	0.65	0.69	0.67	6858
LOC	0.49	0.25	0.33	936
ORG	0.44	0.54	0.48	7734
PER	0.87	0.69	0.77	14268
micro avg	0.66	0.64	0.65	29796
macro avg	0.61	0.54	0.56	29796
weighted avg	0.69	0.64	0.66	29796

Figura 4.10: *NER in Italiano - GloVe: classification report*

	precision	recall	f1-score	support
GPE	0.62	0.59	0.60	6858
LOC	0.80	0.15	0.26	936
ORG	0.40	0.53	0.46	7734
PER	0.75	0.68	0.72	14268
micro avg	0.60	0.61	0.60	29796
macro avg	0.51	0.39	0.41	29796
weighted avg	0.63	0.61	0.61	29796

Figura 4.11: *NER in Italiano - BERT: classification report*

4.3.3 Tempi di esecuzione

Embedding	Esecutori	Tempo di esecuzione
USE	1	8min 05s
	2	4min 21s
BERT (base_case)	1	8h 6min 03s
	2	4h 6min 05s

Tabella 4.6: Tempi di esecuzione

4.3.4 Valutazioni

Nel caso del task di Named Entity Recognition, i risultati presentano una notevole differenza tra le performance ottenute con i dataset in lingua inglese e in lingua italiana. Infatti, si passa da un F1-score dell'87% con BERT Base nel primo caso, ad un 60% con il modello corrispettivo per la lingua italiana. Il risultato per l'inglese non raggiunge lo stato dell'arte¹³, ma non si distacca eccessivamente dalle altre soluzioni al momento disponibili. Drastico è il calo che si ha con la lingua italiana, dove il modello sbaglia eccessivamente. Da quanto si evince dei risultati mostrati sul sito di *evalita*¹⁴, le soluzioni per questo task raggiungono anche valori di F1-score pari all'82%, ovvero circa il 20% in più del modello studiato in questa tesi. I risultati peggiori, come si legge dal classification report, si incontrano nei casi dei tag che identificano *entità geo-politiche (GPE)* e *organizzazioni (ORG)*.

¹³Risultati CoNLL 2003 (English): <https://paperswithcode.com/sota/named-entity-recognition-ner-on-conll-2003>

¹⁴Risultati evalita 2009: <https://www.evalita.it/evalita-2009/results/>

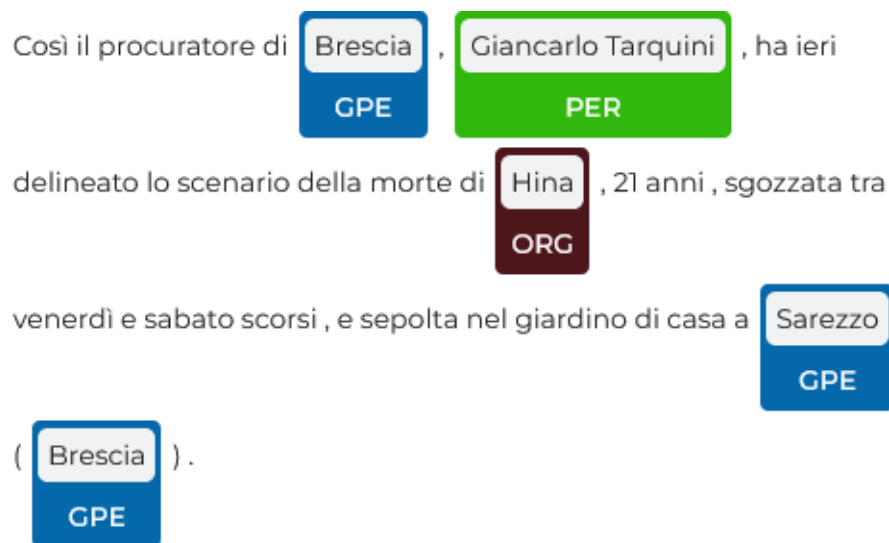


Figura 4.12: Named Entity Recognition per l'italiano - Errore 1

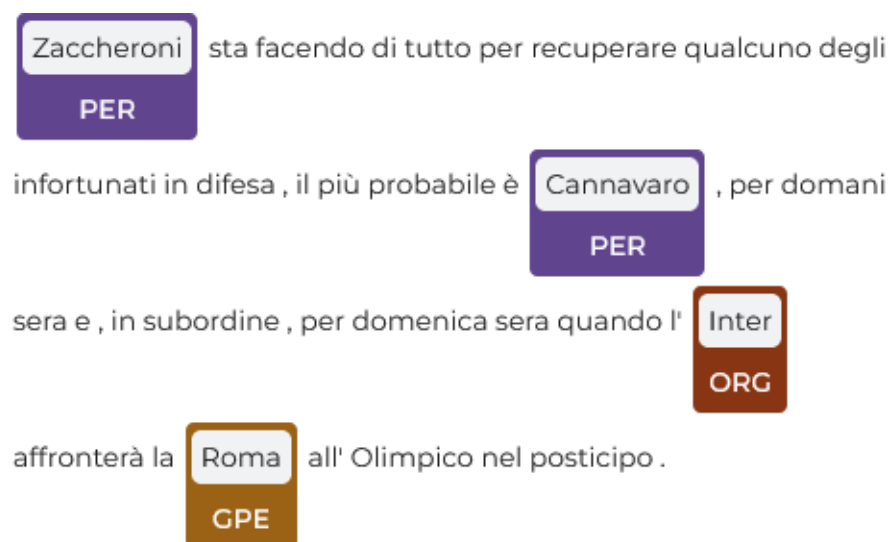


Figura 4.13: Named Entity Recognition per l'italiano - Errore 2

Osservando la figura 4.12, si nota come nomi poco frequenti in Italia (in questo caso *Hina*), non vengono interpretati come riferimenti a persone, ma ad altre entità (in questo caso *organizzazione* (*ORG*)). Inoltre, il modello fatica a distinguere le ambiguità che si presentano quando un'entità che solitamente è utilizzata con un significato, si presenta nella frase con un'altra accezione. Facendo riferimento all'immagine 4.13, il concetto di *Roma* come associazione sportiva, e quindi *organizzazione*, viene confuso

con il significato di *Roma* intesa come la città capitale d'Italia. La parola è stata pertanto classificata come entità *geo-politica*.

Un'ulteriore differenza tra i dati ottenuti dai due dataset, si ha con l'accuratezza raggiunta dai modelli. Anche se di poco, BERT performa meglio di GloVe nella lingua inglese (86,5% il primo, 85,8% il secondo), mentre per la lingua italiana, GloVe riconosce leggermente meglio le ambiguità rispetto alla controparte (65% per GloVe, 60,5% BERT).

Passando al problema di Text Classification, sia il componente UniversalSentenceEncoder sia il BertSentenceEmbedding, riportano ottimi risultati in termini di accuratezza, raggiungendo entrambi l'89%. La differenza sostanziale si nota nei tempi di esecuzione, nei quali l'UniversalSentenceEncoder risulta essere decine di volte più veloce. Pertanto, a parità di risultati, USE è l'alternativa che offre il miglior rapporto tra l'accuracy e i tempi di esecuzione sui dataset utilizzati.

È facilmente osservabile che, all'aumentare della dimensione del sistema distribuito, e quindi degli esecutori, Spark NLP migliora di gran lunga le sue prestazioni. La differenza si nota in particolar modo con BERT, dove non si ottiene ancora un risultato in tempi competitivi ma, con l'aggiunta di un nodo worker, il tempo di esecuzione si dimezza. Questo denota una buona distribuzione del carico di lavoro tra i nodi dovuta ad un'equa suddivisione dei jobs tra i vari esecutori.

In sistemi che utilizzano frequentemente centinaia se non migliaia di nodi, Spark NLP può offrire predizioni accurate in tempi contenuti.

Conclusioni

L'obiettivo di questo lavoro è quello di analizzare risultati a livello prestazionale ottenuti tramite l'utilizzo del framework Spark NLP per l'elaborazione del linguaggio naturale in ambiente distribuito; in particolar modo sui task di Text Classification e Named Entity Recognition.

Al fine di facilitare la comprensione del lavoro svolto, nel primo capitolo sono stati introdotti i temi trattati all'interno della Tesi ed è stata fatta una panoramica sugli scenari che richiedono una soluzione a problemi di Natural Language Processing.

All'interno del secondo capitolo, si è parlato proprio del concetto di NLP, entrando nel dettaglio delle fasi che questo processo attraversa e della complessità di alcuni problemi di elaborazione linguistica. Allacciandosi a questo discorso, si è proceduto a raccontare quali sono stati e quali sono ad oggi, gli approcci che gli scienziati e gli sviluppatori hanno nei confronti di questo problema, partendo dal famoso *Test di Turing* fino ad arrivare al moderno Deep Learning. Questo *excursus* è servito anche per trattare in maniera più dettagliata il concetto di Language Modeling e di Word Embedding, per poi porre l'attenzione alle tecnologie attualmente più implementate per la rappresentazione delle parole attraverso vettori numerici. Questo capitolo ci è stato utile per comprendere l'ambiente in cui si colloca il progetto e conoscere i problemi di natura linguistica a cui fa riferimento.

Dal terzo capitolo si è passati ad esporre le tecnologie utilizzate all'interno del progetto. Dopo una rapida digressione sui concetti di calcolo distribuito e sistemi distribuiti, si è passati all'analisi di *Apache Hadoop*, descrivendone il file system distribuito che esso offre (HDFS) e l'implementazione di MapReduce, per poi illustrare gli ambienti *Apache Spark* e *Spark NLP*, soggetti principali di questo lavoro. Nel dettaglio, degli ultimi due, sono state trattate le loro architetture, i vantaggi che offrono ed i problemi ai quali puntano a dare delle soluzioni. Una descrizione completa di questi strumenti è fondamentale per comprendere a pieno il perché si è deciso di studiarli e per facilitare la lettura del capitolo successivo.

Il capitolo si conclude con la descrizione dell'architettura distribuita sulla quale è stato eseguito il software sviluppato. È stata descritta la sua struttura fisica e logica, riportando dati riguardanti l'hardware utilizzato, il file system distribuito utilizzato e la configurazione dell'ambiente Spark.

Infine, nel capitolo quarto, è stata presentata la fase di sperimentazione, partendo dall'esposizione dei task di Text Classification e Named Entity Recognition e i quesiti che essi pongono, passando per la descrizione dei dataset utilizzati per l'addestramento dei modelli e per il testing di questi ultimi e concludendo con l'illustrare i componenti sui quali il software si basa. In particolar modo, riguardo a quest'ultimo punto, sono stati descritti i componenti di Spark NLP che sono stati utilizzati, i modelli che sono stati inclusi per la fase di embedding dei testi e come sono state costruite le pipeline che formano il processo di elaborazione dei record contenuti nei dataset.

Il capitolo si chiude con l'analisi dettagliata dei risultati ottenuti dalla soluzione sviluppata. Si è posta l'attenzione sui tempi di esecuzione ottenuti durante le fasi di addestramento e di testing, e sui valori di accuratezza raggiunti, approfondendo le

differenze ottenute tra l'accuratezza vera e propria e l'*F1-Score*, media armonica di precisione e recupero.

Questo lavoro è un punto di partenza per un proseguimento dello studio di questo framework. A partire dalla soluzione sviluppata è possibile divagare all'interno delle possibilità offerte da Spark NLP, magari aggiungendo una fase di preprocessing dei testi e andando a modulare più nel dettaglio le opzioni offerte da questo framework per un addestramento più efficace. Altre prospettive per il futuro sono quelle di risolvere i problemi riscontrati durante la parallelizzazione del lavoro e di testare prestazioni ottenute utilizzando altri modelli di embedding come RoBERTa¹⁵, ALBERT¹⁶ o DistilBERT¹⁷.

¹⁵RoBERTa: A Robustly Optimized BERT Pretraining Approach

¹⁶ALBERT: A Lite BERT for Self-supervised Learning of Language Representations

¹⁷DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter

Bibliografia

- [1] Distributed computing. URL https://en.wikipedia.org/wiki/Distributed_computing.
- [2] Amazon Web Services. What is Hadoop? URL <https://aws.amazon.com/it/emr/details/hadoop/what-is-hadoop/>.
- [3] Andrea Minimi. Analisi lessicale del testo . URL <https://www.andreaminini.com/semantica/analisi-lessicale/>.
- [4] Aravind CR. Word Embeddings in NLP | Word2Vec | GloVe | fastText, 2020. URL <https://medium.com/analytics-vidhya/word-embeddings-in-nlp-word2vec-glove-fasttext-24d4d4286a73>.
- [5] Ben Lutkevich. Language Modeling, 2020. URL <https://www.techtarget.com/searchenterpriseai/definition/language-modeling>.
- [6] D. Cer, Y. Yang, S. yi Kong, N. Hua, N. Limtiaco, R. S. John, N. Constant, M. Guajardo-Cespedes, S. Yuan, C. Tar, Y.-H. Sung, B. Strope, and R. Kurzweil. Universal sentence encoder. 2018.
- [7] N. Chomsky and D. Lightfoot. *Syntactic Structures*. De Gruyter Reference Global. Mouton de Gruyter, 2002. ISBN 9783110172799. URL <https://books>.

- [google.it/books?id=a6a_b-CXYAkC](https://books.google.it/books?id=a6a_b-CXYAkC).
- [8] Christina Newberry. 47 Facebook Stats That Matter to Marketers in 2021, 2021. URL <https://blog.hootsuite.com/facebook-statistics/>.
- [9] DataFlair. Directed Acyclic Graph DAG in Apache Spark, 2017. URL <https://data-flair.training/blogs/dag-in-apache-spark/>.
- [10] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. 2019.
- [11] Dmitriy Selivanov. GloVe Word Embeddings, 2020. URL <http://text2vec.org/glove.html>.
- [12] G. R. Doddington, A. Mitchell, M. A. Przybocki, L. A. Ramshaw, S. Strassel, and R. M. Weischedel. The automatic content extraction (ace) program - tasks, data, and evaluation. In *LREC*, 2004.
- [13] S. Galliano, G. Gravier, and L. Chaubard. The ester 2 evaluation campaign for the rich transcription of french radio broadcasts. In *In In: Proceedings of Interspeech, Brighton (United Kingdom)*, 2009.
- [14] GeeksforGeeks. Named Entity Recognition, 2021. URL <https://www.geeksforgeeks.org/named-entity-recognition/>.
- [15] Gianpietro Storari. Sintassi Semantica Pragmatica, 2020. URL <https://people.unica.it/gianpietrostorari/files/2020/04/sintassi-semantica-pragmatica-pdf-copia.pdf>.
- [16] M. Henderson, R. Al-Rfou, B. Strope, Y. hsuan Sung, L. Lukacs, R. Guo, S. Kumar, B. Miklos, and R. Kurzweil. Efficient natural language response suggestion for smart reply, 2017.

- [17] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 1997.
- [18] Ionos. Che cos'è l'elaborazione del linguaggio naturale?, 2019. URL <https://www.ionos.it/digitalguide/online-marketing/vendere-online/come-funziona-l-elaborazione-del-linguaggio-naturale/>.
- [19] Javatpoint. What is Hadoop . URL <https://www.javatpoint.com/what-is-hadoop>.
- [20] Jay Alammar. The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning), 2018. URL <https://jalammar.github.io/illustrated-bert/>.
- [21] Jeff Desjardins. How much data is generated each day?, 2019. URL <https://www.weforum.org/agenda/2019/04/how-much-data-is-generated-each-day-cf4bddf29f/>.
- [22] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI'04: Sixth Symposium on Operating System Design and Implementation*, pages 137–150, San Francisco, CA, 2004.
- [23] John Snow Labs. NER with BERT in Spark NLP, 2021. URL https://github.com/JohnSnowLabs/spark-nlp-workshop/blob/master/tutorials/blogposts/3.NER_with_BERT.ipynb.
- [24] John Snow Labs. Named Entity Recognition (NER) DL Training, 2021. URL https://github.com/JohnSnowLabs/spark-nlp-workshop/blob/master/tutorials/Certification_Trainings/Public/4.NERDL_Training.ipynb.
- [25] Joseph Johnson. Daily number of emails worldwide, 2021. URL <https://www>.

- statista.com/statistics/456500/daily-number-of-e-mails-worldwide/.
- [26] R. Kiros, Y. Zhu, R. R. Salakhutdinov, R. Zemel, R. Urtasun, A. Torralba, and S. Fidler. Skip-thought vectors. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL <https://proceedings.neurips.cc/paper/2015/file/f442d33fa06832082290ad8544a8da27-Paper.pdf>.
- [27] Koray Tuğberk GÜBÜR. Named Entity Recognition: Definition, Examples, and Guide, 2021. URL <https://www.bytesview.com/blog/named-entity-extraction-concept-tools-tutorials/>.
- [28] Mandar Deshpande. Language Modeling, 2020. URL <https://towardsdatascience.com/language-modeling-c1cf7b983685>.
- [29] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*, chapter 13.1. Cambridge University Press, 2008.
- [30] McCormick, C. Word2Vec Tutorial - The Skip-Gram Model, 2016. URL <http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/>.
- [31] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space, 2013.
- [32] MonkeyLearn. Named Entity Recognition, . URL <https://monkeylearn.com/blog/named-entity-recognition/>.
- [33] MonkeyLearn. What Is Text Classification?, . URL <https://monkeylearn.com/what-is-text-classification/>.
- [34] Niklas Donges. Introduction to NLP, 2019. URL <https://builtin.com/data-science/introduction-nlp>.

- [35] Pennington, Jeffrey and Socher, Richard and Manning, Christopher. GloVe: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, Oct. 2014. Association for Computational Linguistics. doi: {10.3115/v1/D14-1162}. URL <https://aclanthology.org/D14-1162>.
- [36] Primer.ai. What is NLP and Why Do You Need It?, 2021. URL <https://primer.ai/blog/what-is-nlp-and-why-do-you-need-it/>.
- [37] ProjectPro.io. Hadoop explained: How does hadoop work and how to use it? URL <https://www.projectpro.io/article/hadoop-explained-how-does-hadoop-work-and-how-to-use-it/237>.
- [38] Ravish Chawla. Overview of Conditional Random Fields, 2017. URL <https://medium.com/ml2vec/overview-of-conditional-random-fields-68a2a20fa541>.
- [39] Robert Gibb. What is a distributed system?, 2019. URL <https://blog.stackpath.com/distributed-system/>.
- [40] Sarfaraz Hussain. Understanding how Spark runs on YARN with HDFS, 2019. URL <https://blog.knoldus.com/understanding-how-spark-runs-on-yarn-with-hdfs/>.
- [41] SAS. Natural Language Processing (NLP). What it is and why it matters, 2021. URL https://www.sas.com/en_us/insights/analytics/what-is-natural-language-processing-nlp.html.
- [42] Shiksha Dahiya. Apache Spark History, 2021. URL <https://www.programsbuzz.com/article/apache-spark-history>.

- [43] Stuart J. Russell and Peter Norvig. *Artificial Intelligence, A Modern Approach 3e*. Pearson, 2009. ISBN 9780136042594.
- [44] Turgut K. Che cos'è Apache Spark? Dove si utilizza? URL <https://trgtkls.org/dal-web/che-cose-apache-spark-dove-si-utilizza/>.
- [45] A. M. TURING. I.—COMPUTING MACHINERY AND INTELLIGENCE. *Mind*, LIX(236):433–460, 10 1950. ISSN 0026-4423. doi: 10.1093/mind/LIX.236.433. URL <https://doi.org/10.1093/mind/LIX.236.433>.
- [46] UC Berkeley Extension. Hadoop: How It Is Used and Its Benefits to Business, 2021. URL <https://bootcamp.berkeley.edu/blog/hadoop-uses-and-benefits/>.
- [47] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need, 2017.
- [48] Ying Lin. 10 WhatsApp Statistics Every Marketer Should Know in 2021 [Infographic], 2021. URL <https://www.oberlo.com/blog/whatsapp-statistics>.
- [49] Zellig S. Harris. Distributional structure. *WORD*, 10(2-3):146–162, 1954. doi: 10.1080/00437956.1954.11659520. URL <https://doi.org/10.1080/00437956.1954.11659520>.
- [50] Y. Zhang, R. Jin, and Z.-H. Zhou. Understanding bag-of-words model: A statistical framework. *International Journal of Machine Learning and Cybernetics*, 1:43–52, 12 2010. doi: 10.1007/s13042-010-0001-0.