



UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Triennale in Informatica

TESI DI LAUREA

Estrazione di requisiti software da conversazioni tramite tecniche di NLP

RELATORE

Prof. Fabio Palomba

Dott. Francesco Casillo

Università degli Studi di Salerno

CANDIDATO

Giovanni Borrelli

Matricola: 0512110177

Anno Accademico 2023-2024

Questa tesi è stata realizzata nel



"La passione è l'energia che ti guida a fare le cose con gioia, entusiasmo e determinazione."

Oprah Winfrey

Abstract

I requisiti software sono specifiche documentate e dettagliate che definiscono le funzionalità, le prestazioni, le restrizioni e le altre caratteristiche di un sistema software. Questi requisiti forniscono una guida chiara e comprensibile per gli sviluppatori durante il processo di progettazione e sviluppo del software, garantendo che il prodotto finale soddisfi le aspettative degli utenti e gli obiettivi del progetto. I requisiti software rappresentano una base fondamentale per la pianificazione e la realizzazione di un'applicazione software.

L'obiettivo di questa tesi è quello di automatizzare il processo di identificazione delle parti del discorso che indicano la presenza di requisiti software, in modo da semplificare il lavoro agli ingegneri del software.

Allo stato dell'arte esiste il tool *REConSum*, che attraverso tecniche di **NLP**, ovvero *natural language processing*, è capace di riconoscere i requisiti software. Tuttavia, presenta delle limitazioni importanti. A partire da *REConSum*, è stato sviluppato un tool più versatile ed efficiente.

Indice

Elenco delle Figure	iv
Elenco delle Tabelle	vi
1 Introduzione	1
1.1 Contesto Applicativo	1
1.2 Motivazioni e Obiettivi	2
1.3 Risultati Ottenuti	3
1.4 Struttura della tesi	3
2 Background e Stato dell'arte	4
2.1 Background	4
2.1.1 Requisiti Software	4
2.1.2 Identificazione dei Requisiti Software	6
2.2 Stato dell'arte	7
2.2.1 REConSum	7
3 Metodologia	9
3.1 Creazione del Dataset	9
3.2 Scelta dei Dataset da analizzare	10
3.2.1 ELITR Minuting Corpus	10

3.2.2	Slack Chats	10
3.2.3	REConSum Conversation	11
3.3	Scelta dei modelli di Machine Learning	11
3.3.1	SVC	12
3.3.2	MLPClassifier	14
3.3.3	Logistic Regression	16
3.4	Altri modelli di Machine Learning	18
3.4.1	Perceptron	18
3.4.2	PassiveAggressiveClassifier	19
3.4.3	HistGradientBoostingClassifier	20
3.4.4	RidgeClassifierCV	22
3.4.5	NuSVC	23
3.4.6	CalibratedClassifierCV	24
3.4.7	GaussianProcessClassifier	26
3.5	Scelta delle tecniche di Word Embedding	27
3.5.1	FastText	27
3.5.2	Word2Vec	29
3.5.3	GloVe	31
3.5.4	BERT	32
3.6	Tecniche di Validazione	35
3.7	Metriche di Validazione	36
4	Risultati	39
4.1	Risultati delle tecniche di Word Embedding	39
4.1.1	fastText	40
4.1.2	word2vec	40
4.1.3	GloVe	41
4.1.4	BERT	42
4.2	Risultati del testing sui Dataset	42
4.2.1	Slack Conversations	43
4.2.2	REConSum Conversation	43
4.2.3	ELITR Minuting Corpus	43

4.3	Analisi dei limiti tramite ChatGPT	44
4.3.1	Filosofia dei Requisiti Software	44
4.3.2	Ambiguità dei Requisiti Software	45
4.4	Risultati e Confronto con REConSum	46
5	Conclusioni	48
	Bibliografia	50

Elenco delle figure

3.1	Illustrazione schematica dell'algoritmo SVC. È evidenziato il <i>decision boundary</i>	13
3.2	Struttura di una rete neurale artificiale. A partire da sinistra, è presente uno strato di <i>Input</i> , al centro uno strato nascosto, e a destra uno strato di <i>Output</i>	15
3.3	Proprio come un neurone biologico ha <i>dendriti</i> per ricevere i segnali, un <i>corpo cellulare</i> per elaborarli e un <i>assone</i> per inviare segnali ad altri neuroni, il neurone artificiale ha diversi <i>canali di input</i> , una <i>fase di elaborazione</i> e un' <i>uscita</i> che può diramarsi verso diversi altri neuroni artificiali.	19
3.4	Rappresentazione schematica dell'organizzazione dei dati in <i>istogrammi</i>	21
3.5	La <i>linea verde</i> rappresenta un overfitted model , mentre la <i>linea nera</i> rappresenta un regularized model . Sebbene la linea verde segua meglio i dati di addestramento, essa è troppo dipendente da tali dati ed è probabile che abbia un tasso di errore più elevato su nuovi dati non visti, rispetto alla linea nera.	23
3.6	<i>Reliability Diagram</i> che mostra sia le <i>probabilità calibrate</i> (in arancione) che le <i>probabilità non calibrate</i> (in blu) sull'algoritmo SVM.	25

3.7 Visualizzazioni dei <i>word embedding</i> : mostrano relazioni geometriche che catturano legami semantici, come la relazione tra un paese e la sua capitale.	28
3.8 Rappresentazione visiva e intuitiva della differenza nell'approccio utilizzato da CBOW e da Skip-gram	29
3.9 L'architettura del Transformer segue una struttura <i>encoder-decoder</i> . Il compito dell' <i>encoder</i> , nella metà a sinistra, è quello di mappare una sequenza di input in una sequenza di rappresentazioni continue, che viene poi alimentata al decoder. Il <i>decoder</i> , nella metà a destra, riceve l'output dell'encoder insieme all'output del decoder al passo temporale precedente per generare una sequenza di output.	34
3.10 Esempio di una 5-fold Cross Validation. In azzurro è raffigurato il <i>training data</i> , mentre in giallo è mostrato il <i>validation data</i>	36
3.11 Matrice di Confusione per un dataset di classificazione binaria.	37

Elenco delle tabelle

3.1	Struttura del dataset creato	10
3.2	Lista in forma tabellare dei classificatori utilizzati da <i>LazyClassifier</i>	12
4.1	Risultati dei migliori classificatori con fastText	40
4.2	Risultati dei migliori classificatori con word2vec	40
4.3	Risultati dei migliori classificatori con GloVe	41
4.4	Risultati dei migliori classificatori con BERT	42

CAPITOLO 1

Introduzione

1.1 Contesto Applicativo

L'*Ingegneria dei Requisiti* è un'importante fase nel processo di sviluppo del software che si concentra sulla definizione, documentazione e gestione dei requisiti del sistema. Questo processo coinvolge l'identificazione, l'analisi, la specifica, la validazione e la gestione dei requisiti che un sistema software deve soddisfare. Gli obiettivi principali dell'ingegneria dei requisiti sono garantire che le esigenze degli stakeholder siano comprese e tradotte in requisiti ben definiti e comprensibili per gli sviluppatori del software.

I *Requisiti Software* sono dichiarazioni formali delle funzionalità, delle prestazioni e delle caratteristiche del sistema software che devono essere soddisfatte per incontrare le esigenze degli utenti e degli altri stakeholder. Esistono diversi tipi di requisiti software, tra cui:

- **Requisiti Funzionali:** definiscono le funzioni specifiche del sistema, descrivendo cosa il sistema dovrebbe fare in risposta a determinate situazioni. I requisiti funzionali sono spesso descritti in termini di casi d'uso e comportamenti del sistema. Ad esempio, un requisito funzionale potrebbe essere: "*Il sistema deve*

consentire agli utenti di effettuare l'accesso al proprio account utilizzando un nome utente e una password.”

- **Requisiti Non Funzionali:** definiscono le qualità del sistema, come le prestazioni, l'affidabilità, la sicurezza e l'usabilità. Sono aspetti del sistema che non riguardano direttamente le funzioni, ma piuttosto come queste funzioni vengono svolte. Ad esempio, un requisito non funzionale potrebbe essere: *“Il sistema deve essere in grado di gestire almeno 1000 utenti contemporaneamente senza subire rallentamenti nelle prestazioni.”*

In sintesi, l'ingegneria dei requisiti si occupa di definire e gestire in modo sistematico i requisiti software al fine di creare un prodotto software che soddisfi le esigenze degli utenti e degli altri interessati.

1.2 Motivazioni e Obiettivi

Le conversazioni riguardanti i requisiti, come interviste e workshop, sono un'attività chiave nell'**elicitazione dei requisiti** e giocano un ruolo significativo nella creazione delle specifiche dei requisiti. Nonostante queste conversazioni contengano una grande quantità di conoscenza, gli ingegneri dei requisiti le utilizzano principalmente prendendo appunti durante la conversazione e richiamando le informazioni dalla loro memoria. C'è il potenziale per supportare i professionisti recuperando informazioni importanti dalle registrazioni di queste conversazioni. Anche se le trascrizioni possono essere generate automaticamente con una buona precisione, spesso contengono testo eccessivo per essere utilizzato efficientemente durante le sessioni di elicitazione dei requisiti. Pertanto, abbiamo osservato la necessità di trasformare questi dati in un formato utile per consentire agli ingegneri dei requisiti di analizzarli. Per questo motivo il nostro obiettivo è sviluppare un tool che utilizza l'*elaborazione del linguaggio naturale* per riassumere le conversazioni sui requisiti. Il tool prende come input una conversazione trascritta e individua le frasi che contengono informazioni utili riguardo i requisiti software. Esiste già un tool di partenza chiamato *REConSum*, e il nostro scopo è quello di migliorarlo rendendolo più efficiente.

1.3 Risultati Ottenuti

È stata fatta un’analisi sia di discussioni nelle fasi iniziali dello sviluppo del software (*ChatGPT discussions* e la conversazione fornita dalla *repository GitHub* di *REConSum*), sia di discussioni quotidiane durante lo sviluppo del software (*Slack discussions*), sia di meeting tramite *ELITR*. Come tecniche di *word embedding* sono state utilizzate: **word2vec**, **fastText**, **GloVe** e **BERT**. La migliore è risultata essere **BERT**. Come algoritmi di *machine learning*, i più performanti sono stati **SVC**, **MLPClassifier** e **Logistic Regression**. Il migliore in assoluto è stato **SVC**. Dopo un’analisi del tool sui dataset e il suo confronto con *REConsum*, il nostro tool ha mostrato prestazioni migliori rispetto a *REConSum*.

1.4 Struttura della tesi

Nel **capitolo 2**, si trova una introduzione sull’ingegneria dei requisiti, vengono trattate tematiche relative ai requisiti funzionali, e infine viene illustrato, nella sezione relativa allo stato dell’arte, il tool *REConSum*, insieme alle sue capacità e limitazioni.

Nel **capitolo 3**, vengono descritte la creazione del dataset ad hoc per l’addestramento, l’analisi del dataset scelti, la scelta dei modelli di *machine learning* e delle tecniche di *word embedding*, e infine tutte le tecniche e le metriche di validazione.

Nel **capitolo 4** sono presenti le informazioni relative ai risultati ottenuti dai modelli. Saranno descritte e motivate le metriche utilizzate e saranno confrontati tutti i risultati ottenuti.

Nel **capitolo 5**, è presente una descrizione di ciò che è stato raggiunto e/o migliorato rispetto ad altri tool, e considerazioni in merito a dei possibili sviluppi nel futuro.

CAPITOLO 2

Background e Stato dell'arte

2.1 Background

2.1.1 Requisiti Software

I requisiti software sono le specifiche funzionali e di prestazione che un'applicazione software deve soddisfare per incontrare le aspettative degli utenti e degli stakeholders. In altre parole, sono le caratteristiche e le capacità che il software deve avere per essere considerato efficace, utile e di alta qualità. Questi requisiti sono essenziali per comprendere cosa deve fare il software e come deve farlo.

I requisiti software possono essere suddivisi in Requisiti Funzionali, che descrivono le funzioni specifiche che il software deve eseguire, e Requisiti Non Funzionali, che non riguardano le funzioni specifiche del software, ma piuttosto le qualità generali del sistema.

I requisiti software sono fondamentali nel processo di ingegneria e sviluppo del software per diverse ragioni cruciali:

- **Comprendere le Aspettative degli Utenti:**

I requisiti software aiutano a definire e comprendere le aspettative degli utenti. Conoscere cosa si aspettano gli utenti dal software è fondamentale per soddisfa-

re le loro esigenze e creare un prodotto che venga adottato e apprezzato dagli utenti finali.

- **Guida per lo Sviluppo:**

I requisiti forniscono una guida chiara agli sviluppatori su cosa creare. Senza requisiti ben definiti, il rischio di sviluppare un prodotto che non soddisfi le aspettative degli utenti è molto alto. Definendo in modo dettagliato cosa deve fare il software, gli sviluppatori hanno un quadro chiaro da seguire.

- **Comunicazione Chiara:**

I requisiti servono come mezzo di comunicazione tra gli stakeholder, gli utenti finali e i membri del team di sviluppo. Forniscono una base comune per la comprensione di ciò che il software deve fare. Una comprensione chiara dei requisiti da parte di tutti gli interessati riduce al minimo i malintesi e i conflitti durante lo sviluppo.

- **Controllo e Gestione del Progetto:**

I requisiti consentono di definire il campo di gioco del progetto. Aiutano nella pianificazione, nell'allocazione delle risorse, nel monitoraggio e nel controllo del progresso del progetto. Definendo cosa deve essere raggiunto, i requisiti permettono di valutare se il progetto sta procedendo nella giusta direzione.

- **Facilitare i Cambiamenti:**

Avendo requisiti chiaramente definiti, è più facile valutare l'impatto dei cambiamenti. Nel corso dello sviluppo del software, potrebbero emergere nuove esigenze o requisiti precedentemente non considerati. Con requisiti ben definiti, è possibile valutare come tali cambiamenti influenzino il progetto esistente.

- **Ridurre i Rischi:**

Una comprensione dettagliata dei requisiti permette di identificare i rischi in modo più accurato. Gli sviluppatori possono anticipare le sfide che potrebbero emergere durante lo sviluppo e pianificare di conseguenza.

- **Assicurare la Qualità del Prodotto:**

I requisiti definiscono gli standard di prestazione e di funzionalità del software.

Questo aiuta a garantire che il prodotto finale soddisfi gli standard di qualità desiderati.

Quindi, i requisiti software sono essenziali perché creano un fondamento solido per il successo del progetto. Un’analisi dettagliata e una definizione chiara dei requisiti sono cruciali per evitare costosi errori di progettazione e sviluppo e per garantire che il prodotto software soddisfi le esigenze degli utenti in modo *efficace* ed *efficiente*.

2.1.2 Identificazione dei Requisiti Software

L’estrazione dei Requisiti Software è essenziale per lo sviluppo di un qualsiasi progetto software. Le conversazioni che riguardano i requisiti, come ad esempio le interviste, contengono una grande quantità di informazioni. Tuttavia, gli ingegneri dei requisiti prendono appunti durante la conversazione, oppure richiamano le informazioni dalla loro memoria. In questo modo non viene sfruttato a pieno il potenziale che queste conversazioni hanno. Se questi dati venissero trasformati in un formato idoneo, sarebbe possibile effettuare l’estrazione automatica dei Requisiti Software. Infatti, potrebbe essere utilizzato un tool che riconosce ed estrae i Requisiti Software da una conversazione. Diversi metodi sono stati utilizzati per l’*estrazione automatica dei requisiti software* [1].

Per quanto riguarda le reti neurali, sono state utilizzate in diversi studi sull’estrazione dei Requisiti Software. *J. Winkler e A. Vogelsang* [2] hanno utilizzato reti neurali convoluzionali (**CNN**). *R. Chatterjee et al* [3] hanno adottato reti neurali di tipo *long short-term memory* (**LSTM**). Infine, *A. Dekhtyar e V. Fong* [4] hanno implementato reti neurali superficiali (**SNN**). Le reti neurali artificiali sono note per essere utili nel gestire il rumore nei dati, il che può essere correlato all’importante processo di preelaborazione dei set di dati che richiedono NLP.

Altri studi riguardano tecniche che invece non comprendono reti neurali. *Z. S. H. Abad et al* [5] hanno utilizzato l’algoritmo **Naive Bayes**. Il Naive Bayes è un classificatore semplice e veloce, e funziona bene anche con un dataset di dimensioni moderate. Quando si tratta di requisiti software, spesso ci sono molte parole chiave e

termini specifici del dominio. Il Naive Bayes può gestire bene queste caratteristiche, rendendolo una scelta ragionevole.

M. A. Haque [6] ha sfruttato l’algoritmo **SVM**. Quando i requisiti possono essere rappresentati in uno spazio multidimensionale, il *Support Vector Machine* può essere utile per trovare relazioni complesse tra i requisiti.

V. Silva-Rodríguez et al [7] hanno impiegato gli algoritmi **Random Forest** e **Logistic Regression**. Quando si ha un insieme di dati eterogeneo o composto da variabili di diversi tipi, *Random Forest* può essere una buona scelta. Questo spesso si verifica nei requisiti software dove si hanno diverse tipologie di informazioni da considerare. Quando si desidera avere una comprensione chiara di quali fattori stanno influenzando la classificazione dei requisiti e quando si dispone di un dataset relativamente piccolo, *Logistic Regression* può essere una buona scelta.

Infine, alcuni studi si sono concentrati sulla classificazione dei Requisiti Software in **FR** e **NFR** (che a loro volta si dividono in sottocategorie, quindi si parla di *classificazione multi-label*). Per questa tipologia l’algoritmo più utilizzato è stato Naive Bayes, come nello studio di *C. S. Rajender Kumar Surana et al* [8].

2.2 Stato dell’arte

2.2.1 REConSum

REConSum [9] (*Requirements Elicitation Conversations Summarizer*) è un tool basato su **NLP** che può assistere professionisti e ricercatori nell’elaborazione delle conversazioni di elicitation attraverso la sintesi delle trascrizioni e l’estrazione delle informazioni rilevanti per i requisiti software. *REConSum*¹ è in grado di identificare se in una conversazione sono presenti domande che contengono Requisiti Software, utilizzando un *TF-IDF* (composto da tutti gli articoli di Wikipedia in lingua inglese). Tuttavia, tale approccio presenta delle limitazioni. In primis, *REConSum* analizza solamente la rilevanza delle domande: ciò diminuisce notevolmente la possibilità di trovare tutti i requisiti software all’interno della conversazione. Infatti non vengono considerate le altre frasi della conversazione che non appartengono all’insieme delle

¹<https://github.com/RELabUU/REConSum>

domande, escludendo a priori la possibilità di riconoscerle. Una seconda importante limitazione è il fatto che il loro modello indica la conversazione (in questo caso intesa come "*insieme di frasi*") in cui è presente il requisito software, ma non la frase specifica. Per tale motivo, quando una conversazione viene etichettata come contenente un requisito software, bisogna rileggerla tutta per cercare di individuarlo.

CAPITOLO 3

Metodologia

3.1 Creazione del Dataset

Allo stato dell'arte non esistono dataset pubblici online in cui è presente una classificazione tra requisiti software e non-requisiti software. Per questo motivo, si è deciso di crearne uno *ad hoc*. Nel lavoro "*Extracting Software Requirements from Unstructured Documents*" [10] vengono estratti manualmente dei requisiti software dal dataset PURE¹ (PUblic REquirements), contenente dei documenti, ottenuti dal Web, al cui interno sono presenti dei requisiti software. Di conseguenza, abbiamo utilizzato PURE per la creazione del dataset. È stato necessario estrarre i requisiti software manualmente, poiché ogni documento ha una struttura diversa, e numerose tabelle che rendono l'estrazione automatica alquanto problematica. Per quanto riguarda la struttura, il dataset finale contiene 300 dati. I dati hanno 2 attributi:

- **Type:** Indica se un dato è un requisito software o se non lo è. Per semplicità può assumere valore 0 o 1, dove 1 indica che la frase è un requisito software.
- **Sentence:** È il contenuto. È una frase, che termina con un punto. Sono esclusi casi speciali in cui il punto è utilizzato in altri scopi, come ad esempio in '*e.g.*' o

¹<https://zenodo.org/record/1414117>

'Mr. Jhonsense'.

Type	Sentence
1	System should generate a confirmation email.
0	Users must follow posted rules.
1	Users must be able to reset their passwords.

Tabella 3.1: Struttura del dataset creato

3.2 Scelta dei Dataset da analizzare

3.2.1 ELITR Minuting Corpus

ELITR Minuting Corpus [11] è composto da trascrizioni di riunioni in lingua ceca e inglese. Le riunioni in lingua inglese riguardano esclusivamente il settore dell'informatica.

Tale corpus² contiene 59 trascrizioni di riunioni in lingua ceca e 120 trascrizioni di riunioni in lingua inglese, per un totale di 71097 e 87322 turni di dialogo rispettivamente.

3.2.2 Slack Chats

Questo dataset³ contiene conversazioni in stile Q&A su Slack relativi a codice informatico. Recentemente sono diventati disponibili numerosi canali di chat pubblici su Slack focalizzati su specifici argomenti di discussione legati all'ingegneria del software, ad esempio lo sviluppo in Python. Questi canali in cui i partecipanti fanno domande e rispondono sono chiamati *canali di Q&A di Slack*. Questi canali sono pubblicizzati sul Web e consentono a chiunque di unirsi, con un processo di iscrizione che richiede solo la creazione di un nome utente (qualsiasi stringa univoca) e una password. Una volta uniti, sui canali, i partecipanti possono fare domande o

²<https://lindat.mff.cuni.cz/repository/xmlui/handle/11234/1-4692>

³<https://github.com/preethac/Software-related-Slack-Chats-with-Disentangled-Conversations.git>

rispondere a qualsiasi domanda, purché riguardi l'argomento principale (ad esempio, la programmazione in Python).

3.2.3 REConSum Conversation

Gli sviluppatori di *REConSum* hanno reso pubblica una conversazione⁴ tra clienti e ingegneri del software. Sarà utile anche per fare un confronto diretto tra le prestazioni di *REConSum* e il modello da noi sviluppato.

3.3 Scelta dei modelli di Machine Learning

Per ricercare l'algoritmo di Machine Learning con le prestazioni migliori, è stata aggiunta anche una componente di sperimentazione empirica con **LazyClassifier**⁵: è uno strumento che semplifica il processo di valutazione delle prestazioni degli algoritmi di Machine Learning su un dataset specifico. Viene utilizzato per ottenere una visione generale delle prestazioni di vari algoritmi di classificazione su un dataset senza la necessità di configurare ciascun modello individualmente. Lazyclassifier mostra le prestazioni dei seguenti 30 classifier :

⁴https://github.com/RELabUU/REConSum/blob/main/data/example_conversation.txt

⁵<https://lazypredict.readthedocs.io/en/latest/usage.html>

Algoritmi di Classificazione di LazyClassifier		
LinearSVC	RandomForestClassifier	BernoulliNB
SGDClassifier	GradientBoostingClassifier	LinearDiscriminantAnalysis
MLPClassifier	QuadraticDiscriminantAnalysis	GaussianNB
Perceptron	HistGradientBoostingClassifier	NuSVC
LogisticRegression	RidgeClassifier	DecisionTreeClassifier
LogisticRegressionCV	RidgeClassifierCV	NearestCentroid
CalibratedClassifierCV	AdaBoostClassifier	ExtraTreeClassifier
PassiveAggressiveClassifier	ExtraTreesClassifier	CheckingClassifier
LabelPropagation	KNeighborsClassifier	DummyClassifier
LabelSpreading	BaggingClassifier	SVC

Tabella 3.2: Lista in forma tabellare dei classificatori utilizzati da *LazyClassifier*

Tra tutti i 30 classificatori, i 3 che hanno mostrato le prestazioni migliori sono *SVC*, *MLPClassifier* e *Logistic Regression*. Per questo motivo saranno trattati in maniera più approfondita.

3.3.1 SVC

L'SVM, che sta per *Support Vector Machine* (Macchina a Vettori di Supporto), è un algoritmo di apprendimento supervisionato utilizzato per problemi di classificazione e regressione. L'obiettivo della classificazione è suddividere un insieme di dati in diverse categorie, mentre l'obiettivo della regressione è predire un valore numerico basandosi sui dati in ingresso.

L'SVC è un tipo di *Support Vector Machine* che si concentra sulla classificazione. L'idea principale di questo algoritmo è trovare un *iperpiano ottimale* [12] che meglio separa le diverse classi nei dati di addestramento. Un iperpiano è un sottospazio di dimensioni n-1 in un contesto n-dimensionale. Ad esempio, in uno spazio bidimensionale (2D), un iperpiano è una linea retta, mentre in uno spazio tridimensionale (3D), è un piano.

L'SVC trova questo iperpiano ottimale nel modo seguente:

1. **Scelta dell'iperspazio:** L'algoritmo seleziona l'iperspazio che meglio separa le classi nel dataset. Questo iperpiano è scelto in modo tale che la distanza tra l'iperspazio e i punti più vicini di ciascuna classe (chiamati *vettori di supporto*) sia massimizzata. Questa distanza è nota come *margine*.
2. **Ottimizzazione del margine:** L'obiettivo è massimizzare il margine tra le classi. Questo è importante perché un margine più grande generalizza meglio il modello, il che significa che funzionerà meglio con nuovi dati non visti.
3. **Gestione dei dati non separabili linearmente:** Spesso, i dati del mondo reale non possono essere separati perfettamente da un iperpiano. L'SVC utilizza una tecnica chiamata *soft margin* [13], che permette alcuni errori di classificazione sul training set. Questo viene gestito mediante l'introduzione di una penalità per gli errori di classificazione.
4. **Utilizzo di una funzione di kernel (Kernel Trick):** L'SVC può separare le classi in spazi ad alta dimensione usando funzioni kernel. Un kernel è una funzione matematica che trasforma i dati in un formato appropriato per l'analisi. Questo consente all'SVC di gestire casi in cui i dati non possono essere separati linearmente nello spazio di origine (di default viene usato *RBF*) [14].

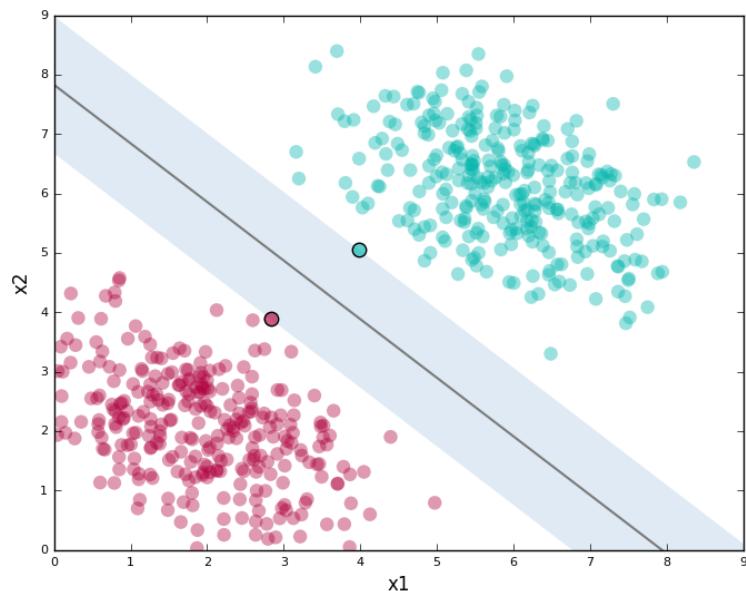


Figura 3.1: Illustrazione schematica dell'algoritmo SVC. È evidenziato il *decision boundary*.

In breve, l'SVC è un algoritmo di apprendimento supervisionato che trova un iperpiano ottimale per separare le classi nei dati di addestramento. Utilizza il concetto di margine per massimizzare la separazione tra le classi e può gestire dati non separabili linearmente utilizzando funzioni kernel. L'SVC è generalmente indicato per dataset di piccole dimensioni, e quindi si adatta bene al nostro dataset utilizzato per l'addestramento.

3.3.2 MLPClassifier

MLPClassifier sta per *Multi-Layer Perceptron Classifier*, ed è un algoritmo di apprendimento supervisionato per la classificazione basato su reti neurali artificiali. Questo tipo di rete neurale è chiamato *multi-layer* perché consiste in almeno tre strati di nodi: uno strato di input, uno o più strati nascosti e uno strato di output.

Per una visione più dettagliata:

1. **Strato di Input:** Nel contesto di *MLPClassifier*, il primo strato è chiamato strato di input. Ogni nodo in questo strato rappresenta una caratteristica del dato di input. Ad esempio, se stiamo classificando immagini in bianco e nero di dimensione 28x28 pixel, avremo 784 nodi nel nostro strato di input, ciascuno rappresentando il valore di un pixel nell'immagine.
2. **Strati Nascosti:** Dopo lo strato di input, ci sono uno o più strati nascosti. Questi strati contengono nodi che eseguono calcoli basati sui valori delle caratteristiche di input. Ogni nodo in uno strato nascosto applica una trasformazione ai dati che riceve come input, trasmettendo poi il risultato agli strati successivi. Questi strati nascosti sono ciò che rende la rete neurale "profonda" e permette di catturare complessi modelli nei dati.
3. **Funzioni di Attivazione:** Ogni nodo, sia negli strati nascosti che in quello di output, utilizza una funzione di attivazione per introdurre non linearità nel modello. Questo è importante perché permette alla rete neurale di apprendere modelli complessi nei dati. Alcune funzioni di attivazione comuni includono la funzione sigmoide, la funzione tangente iperbolica (\tanh) e la funzione di attivazione rettificata (ReLU).

4. **Strato di Output:** Lo strato di output produce i risultati della rete neurale. Il numero di nodi in questo strato dipende dal numero di classi nel problema di classificazione. Ad esempio, se stiamo classificando immagini in 10 diverse categorie, ci saranno 10 nodi nello strato di output, ognuno rappresentante una classe possibile.
5. **Apprendimento e Ottimizzazione:** La rete neurale apprende dai dati di addestramento attraverso un processo chiamato *backpropagation* [15]. Questo processo implica la minimizzazione di una funzione di perdita, che misura la discrepanza tra le previsioni della rete e i veri valori delle classi nel set di addestramento. L'algoritmo di ottimizzazione, come l'*algoritmo di discesa del gradiente*, viene utilizzato per regolare i pesi della rete in modo da ridurre questa perdita.

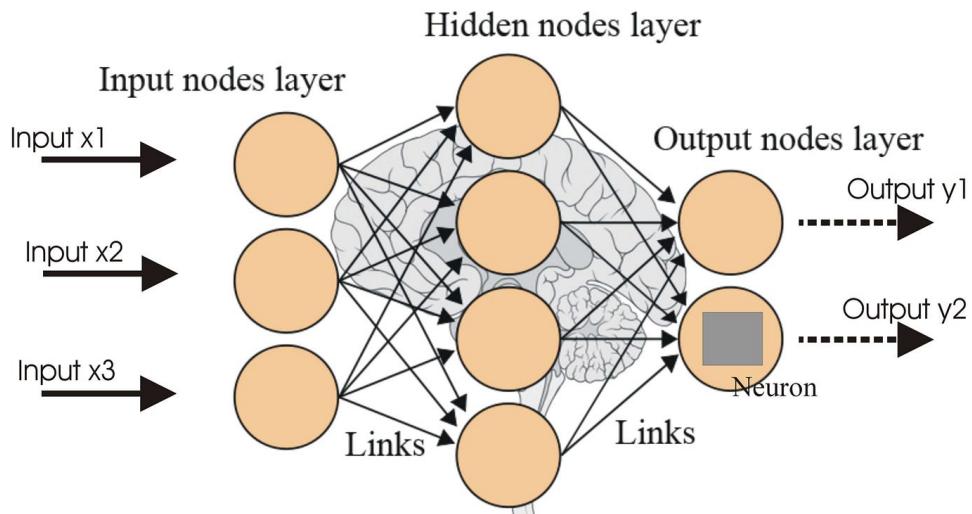


Figura 3.2: Struttura di una rete neurale artificiale. A partire da sinistra, è presente uno strato di *Input*, al centro uno strato nascosto, e a destra uno strato di *Output*.

In breve, *MLPClassifier* è un algoritmo di classificazione basato su reti neurali artificiali con uno o più strati nascosti. Questa complessità permette di modellare relazioni intricate nei dati, rendendolo adatto per una vasta gamma di problemi di classificazione complessi. Le sue prestazioni generali lo rendono un algoritmo adatto

al nostro dataset, tuttavia un problema relativamente semplice da risolvere potrebbe avere maggiore compatibilità con un algoritmo non troppo complesso.

3.3.3 Logistic Regression

La *Logistic Regression*, nonostante il nome, è un algoritmo di classificazione e non di regressione. È utilizzato per problemi di classificazione binaria, dove l'obiettivo è prevedere a quale delle due classi appartiene un dato campione. L'algoritmo è chiamato "logistico" perché utilizza la funzione logistica (o sigmoide) per effettuare la classificazione. Logistic Regression è un modello di tipo *discriminativo* [16]. La differenza tra modello generativo e modello discriminativo è la seguente:

- **Modelli Generativi:** Questi modelli cercano di apprendere la distribuzione di probabilità dei dati di addestramento. In altre parole, comprendono come i dati sono generati, consentendo loro di generare nuovi dati simili.
- **Modelli Discriminativi:** Questi modelli si concentrano sulle differenze tra le classi nel dataset. L'obiettivo principale è trovare la *decision boundary* [17] che separa le classi nel tuo spazio delle feature. Questi modelli non cercano di comprendere come i dati sono generati, ma piuttosto come distinguere tra le classi esistenti.

L'algoritmo funziona nel modo seguente:

1. **Funzione Logistica (Sigmoide):** La regressione logistica utilizza la funzione logistica per trasformare una combinazione lineare delle caratteristiche di input in un valore compreso tra 0 e 1. La funzione logistica è definita come:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Dove z è la combinazione lineare delle caratteristiche di input e dei pesi associati:

$$z = w_0x_0 + w_1x_1 + w_2x_2 + \dots + w_nx_n$$

Dove w_0, w_1, \dots, w_n sono i pesi associati alle caratteristiche di input x_0, x_1, \dots, x_n rispettivamente.

La funzione logistica trasforma z in un valore compreso tra 0 e 1. Se il risultato della funzione logistica è vicino a 1, significa che l'algoritmo è molto sicuro che l'input appartiene alla classe positiva, mentre se è vicino a 0 si ha un'alta probabilità che l'input appartenga alla classe negativa.

2. **Addestramento:** Durante la fase di addestramento, l'algoritmo cerca di trovare i migliori pesi w_0, w_1, \dots, w_n che minimizzano l'errore tra le previsioni della regressione logistica e le etichette di classe effettive nel set di addestramento. Questo viene fatto utilizzando una funzione di costo come la *cross-entropy loss* [18], che misura la discrepanza tra le previsioni del modello e le etichette reali.
3. **Decision Boundary:** Una volta addestrato, il modello di regressione logistica utilizza il valore restituito dalla funzione logistica per classificare nuovi dati. Tipicamente, se il valore restituito è maggiore di una soglia (tipicamente 0.5), il modello classifica l'input come appartenente alla classe positiva; altrimenti, viene classificato come appartenente alla classe negativa.
4. **Estensione a Classificazione Multiclasse:** La regressione logistica è intrinsecamente una tecnica di classificazione binaria, ma può essere estesa per affrontare problemi di classificazione multiclasse. Un modo comune per farlo è utilizzare l'approccio *one-vs-all* (*one-vs-rest*) [19], in cui viene addestrato un classificatore binario per ogni classe, e il classificatore con la previsione più sicura è scelto come risultato finale.

In sintesi, la regressione logistica è un algoritmo di classificazione che utilizza la funzione logistica per produrre previsioni di probabilità per appartenenza a una classe. È un metodo semplice ed efficiente per problemi di classificazione binaria, è computazionalmente efficiente e può essere addestrata anche su piccoli dataset senza richiedere molte risorse computazionali.

3.4 Altri modelli di Machine Learning

Di seguito sono riportati gli algoritmi che hanno avuto le prestazioni migliori tra i 30 classificatori, oltre a SVC, MLPClassifier e Logistic Regression che sono stati analizzati in precedenza.

3.4.1 Perceptron

Il *Perceptron* è un algoritmo di apprendimento automatico supervisionato per la classificazione binaria. Creato dallo scienziato informatico Frank Rosenblatt nel 1957, il perceptron è considerato il più semplice tipo di rete neurale. L'obiettivo principale del *Perceptron* è separare due classi lineamente, cioè trovare una linea di separazione che possa distinguere tra le classi in uno spazio bidimensionale o iperpiano in uno spazio multidimensionale.

Il *Perceptron* si ispira all'elaborazione delle informazioni di una singola cellula neurale chiamata neurone. Un neurone accetta segnali in ingresso tramite i suoi dendriti, che trasmettono il segnale elettrico al corpo cellulare. In modo simile, il *Perceptron* riceve segnali di input dagli esempi di dati di allenamento, li pesa e li combina in un'equazione lineare chiamata attivazione. Essa viene quindi trasformata in un valore di output o previsione utilizzando una funzione di trasferimento [20], come la *step function*. In questo modo, il *Perceptron* è un algoritmo di classificazione per problemi con due classi (0 e 1) in cui è possibile utilizzare un'equazione lineare per separare le due classi. Il *Perceptron* prende in input un vettore di dati numerici e assegna loro pesi. Calcola la somma pesata degli input e, se questa somma supera una certa soglia, restituirà un output positivo (1), altrimenti restituirà un output negativo (0). La soglia e i pesi dei singoli input vengono regolati durante il processo di addestramento dell'algoritmo, che coinvolge la correzione degli errori commessi nelle previsioni.

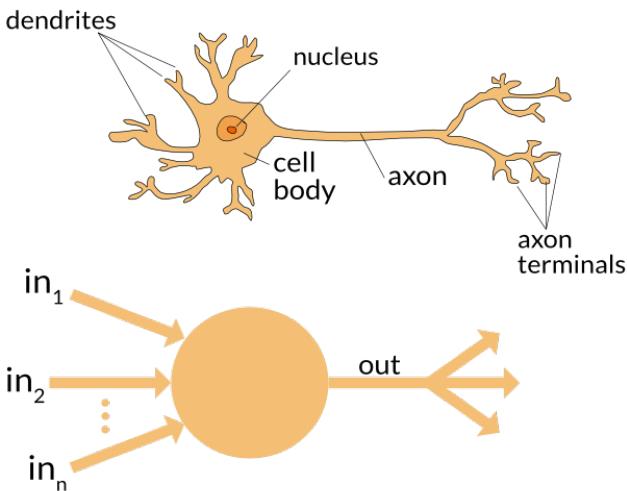


Figura 3.3: Proprio come un **neurone biologico** ha *dendriti* per ricevere i segnali, un *corpo cellulare* per elaborarli e un *assone* per inviare segnali ad altri neuroni, il **neurone artificiale** ha diversi *canali di input*, una *fase di elaborazione* e un'*uscita* che può diramarsi verso diversi altri neuroni artificiali.

Tuttavia, il *Perceptron* ha alcune limitazioni; ad esempio, non può risolvere problemi in cui le classi non sono linearmente separabili. Per superare questa limitazione, sono stati sviluppati algoritmi più complessi come reti neurali multistrato che utilizzano strati intermedi di neuroni (noti come strati nascosti) per affrontare problemi di classificazione più complessi e non linearmente separabili.

3.4.2 PassiveAggressiveClassifier

Il *PassiveAggressiveClassifier* è un algoritmo di classificazione utilizzato nell'apprendimento automatico, appartenente alla famiglia degli algoritmi di apprendimento online. Gli algoritmi di apprendimento online sono progettati per adattarsi e imparare da nuovi dati man mano che arrivano, senza necessità di rielaborare l'intero dataset di addestramento. Questo rende gli algoritmi di apprendimento online particolarmente utili per situazioni in cui i dati sono in arrivo costante o in tempo reale. Il nome "*Passive Aggressive*" deriva dalla natura dell'algoritmo: è passivo nel senso che cerca di minimizzare le perdite, ma diventa **aggressivo** quando c'è un errore di classificazione e deve adattarsi a nuovi dati in modo rapido ed efficace. In termini più tecnici, il *PassiveAggressiveClassifier* è un classificatore lineare. Questo significa

che cerca di trovare un iperpiano (una sorta di piano in spazi multidimensionali) che separi le classi nel miglior modo possibile. Quando arriva un nuovo esempio di addestramento, il classificatore verifica se è classificato correttamente. Se lo è, il modello resta passivo, cioè non viene aggiornato significativamente. Se viene classificato erroneamente, diventa **aggressivo** e si adatta ai nuovi dati in modo da correggere l'errore. L'algoritmo utilizza un parametro chiamato C , che rappresenta il grado di aggressività dell'algoritmo. Un valore di C più grande renderà l'algoritmo più aggressivo nell'aggiornamento del modello per correggere gli errori, mentre un valore più piccolo renderà l'algoritmo più passivo. Il *PassiveAggressiveClassifier* è utilizzato in situazioni in cui l'apprendimento deve essere eseguito in tempo reale e l'adattamento ai nuovi dati deve essere rapido. Tuttavia, è importante notare che questo algoritmo può essere sensibile ai parametri e può richiedere una buona taratura per ottenere prestazioni ottimali sui dati specifici.

3.4.3 HistGradientBoostingClassifier

L'*HistGradientBoostingClassifier* è un algoritmo di classificazione basato sul concetto di *boosting*, una tecnica di apprendimento automatico che combina diversi modelli deboli (generalmente alberi decisionali poco profondi) per creare un modello forte capace di compiere previsioni accurate. Questo algoritmo è particolarmente potente e efficiente per gestire grandi set di dati.

La particolarità del *HistGradientBoostingClassifier* risiede nella gestione efficiente dell'addestramento dei modelli e nella manipolazione di grandi quantità di dati attraverso l'uso di istogrammi. Ecco come funziona:

- 1. Campionamento con istogrammi:** Invece di considerare ogni singolo valore delle feature durante la fase di addestramento, l'algoritmo organizza i dati in istogrammi ($bins$) basandosi sui valori delle feature. Questo rende più efficiente l'analisi dei dati, specialmente quando ci sono molte feature e molti dati.

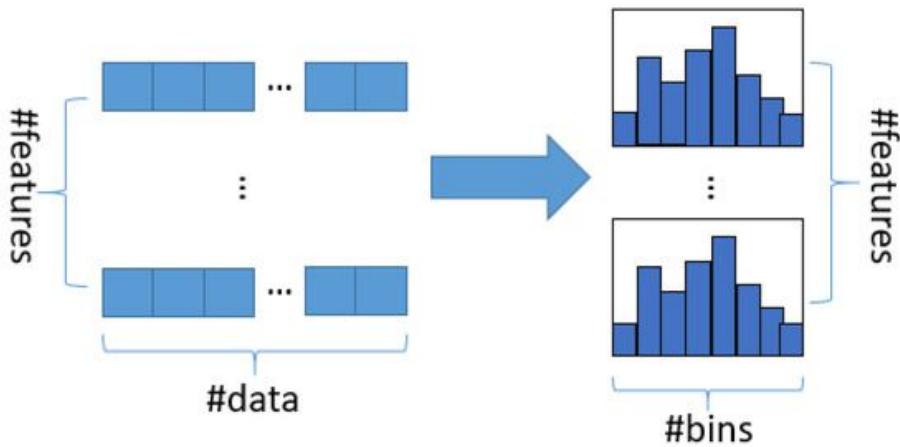


Figura 3.4: Rappresentazione schematica dell’organizzazione dei dati in *istogrammi*.

2. **Addestramento Iterativo:** L’*HistGradientBoostingClassifier* funziona in modo iterativo, creando un modello alla volta. Ogni modello successivo cerca di correggere gli errori compiuti dai modelli precedenti. Questo processo viene ripetuto per un numero specificato di iterazioni o finché il modello non raggiunge una certa precisione.
3. **Regolarizzazione:** L’algoritmo utilizza tecniche di regolarizzazione per prevenire l’*overfitting*, ovvero per evitare che il modello si adatti troppo ai dati di addestramento e non sia in grado di generalizzare bene su nuovi dati.
4. **Scelte automatiche dei parametri:** L’*HistGradientBoostingClassifier* può fare alcune scelte automatiche dei parametri, come ad esempio la profondità dell’albero, per ottimizzare le prestazioni del modello.

Questo algoritmo è particolarmente utile quando si hanno a che fare con grandi dataset e quando si desidera ottenere buone prestazioni senza dover passare attraverso complesse procedure di ottimizzazione dei parametri. Inoltre, è in grado di gestire sia dati numerici che categorici, il che lo rende versatile in molte situazioni del mondo reale. Tuttavia, è importante notare che come tutti gli algoritmi di machine learning, l’*HistGradientBoostingClassifier* richiede una buona comprensione del problema e un adeguato preprocessing dei dati per ottenere risultati significativi.

3.4.4 RidgeClassifierCV

Il *RidgeClassifierCV* [21] è un classificatore lineare basato sull'algoritmo di *ridge regression*. È utilizzato per problemi di classificazione binaria e multiclasse. La *ridge regression* è una tecnica di regressione lineare che aggiunge un termine di regolarizzazione (noto come termine di penalizzazione o termine di "ridge") alla funzione di perdita. Questo termine di regolarizzazione impedisce che i coefficienti del modello diventino troppo grandi, aiutando così a prevenire l'overfitting del modello.

Il *RidgeClassifierCV* combina questa idea con un algoritmo di classificazione. La "CV" nel nome indica che l'algoritmo utilizza la convalida incrociata (cross-validation) per selezionare automaticamente il miglior valore del parametro di regolarizzazione (chiamato anche parametro *alpha*⁶). La convalida incrociata suddivide il set di dati in più parti (fold) e addestra e valuta il modello su diverse combinazioni di dati di addestramento e di test, aiutando così a determinare quale valore di *alpha* funziona meglio per il problema specifico.

In sostanza, il *RidgeClassifierCV* è utile quando si desidera un classificatore lineare che sia regolarizzato per prevenire l'overfitting, e quando non si vuole preoccuparsi di tarare manualmente il parametro di regolarizzazione. L'utilizzo della convalida incrociata aiuta a selezionare un buon valore di *alpha* per ottenere un buon compromesso tra la precisione del modello e la sua capacità di generalizzare su nuovi dati.

⁶<https://saturncloud.io/blog/how-to-set-regularization-parameters-for-randomized-regression-in-scikitlearn/>

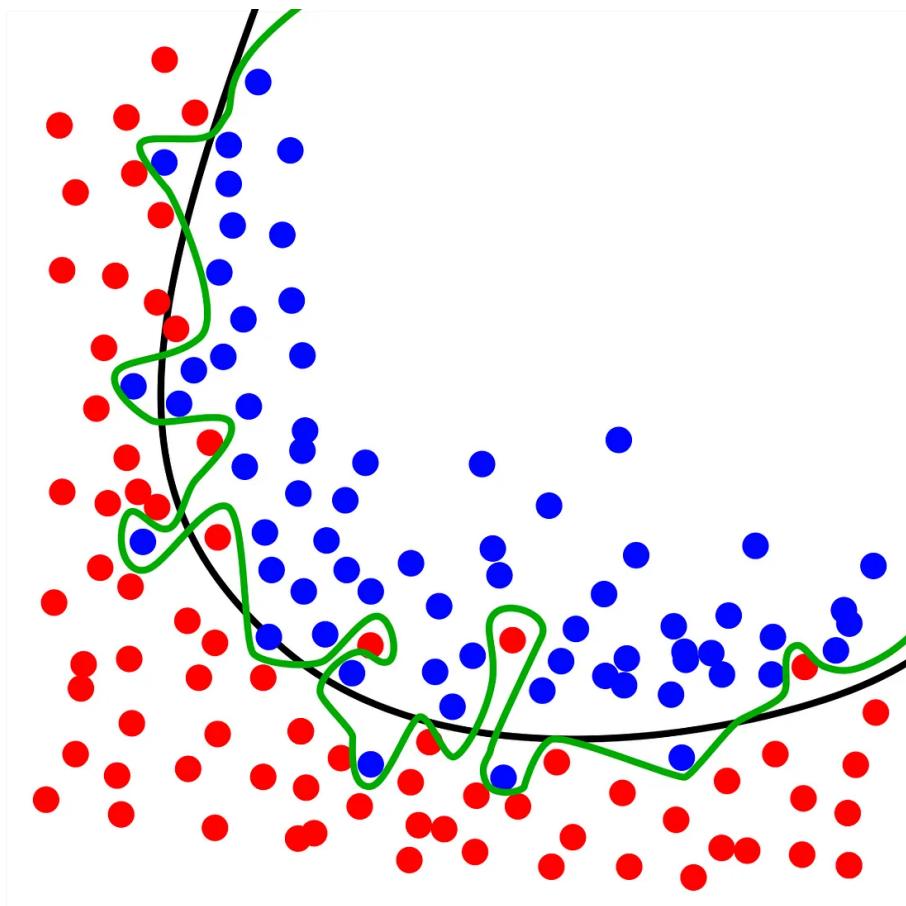


Figura 3.5: La linea verde rappresenta un **overfitted model**, mentre la linea nera rappresenta un **regularized model**. Sebbene la linea verde segua meglio i dati di addestramento, essa è troppo dipendente da tali dati ed è probabile che abbia un tasso di errore più elevato su nuovi dati non visti, rispetto alla linea nera.

Un’importante caratteristica del *RidgeClassifierCV* è che può gestire sia dati numerici che categorici, rendendolo versatile in una varietà di contesti di *machine learning*. Tuttavia, è importante notare che, come tutti gli algoritmi di *machine learning*, il *RidgeClassifierCV* richiede una cura adeguata nella preparazione dei dati e una comprensione approfondita del problema che si sta cercando di risolvere per ottenere risultati accurati e significativi.

3.4.5 NuSVC

NuSVC è un classificatore di supporto vettoriale (SVM) non lineare introdotto in *scikit-learn*, una libreria popolare di machine learning in Python. SVM è un tipo

di algoritmo di apprendimento supervisionato utilizzato per la classificazione o la regressione. L'obiettivo di un classificatore SVM è trovare un iperpiano ottimale che separi i dati di addestramento delle diverse classi nel miglior modo possibile.

L'approccio di *NuSVC* è simile al classificatore SVM standard, ma con una differenza chiave: SVM standard richiede la specifica di un parametro chiamato C , che controlla il trade-off tra l'errore di classificazione e la larghezza della "striscia di sicurezza" attorno all'iperpiano. Tuttavia, *NuSVC* utilizza un diverso parametro chiamato nu , che rappresenta una stima superiore del numero totale di errori e una stima inferiore della frazione di support vector.

In sostanza, nu è un valore tra 0 e 1 che controlla la quantità di support vector e gli errori di classificazione che il modello è disposto ad accettare. Un valore di nu più basso implica una stima più stringente degli errori e una maggiore complessità del modello, mentre un valore più alto di nu permette una maggiore flessibilità nella definizione dell'iperpiano e una stima più ampia degli errori.

L'utilizzo di *NuSVC* è utile quando si desidera una maggiore flessibilità nella gestione degli errori di classificazione, il che è spesso il caso quando si hanno dati rumorosi o problemi di classificazione difficili. Tuttavia, è importante notare che la corretta taratura del parametro nu è essenziale per ottenere prestazioni ottimali dal modello. Come con tutti gli algoritmi di *machine learning*, una buona comprensione del problema e un adeguato preprocessing dei dati sono fondamentali per ottenere risultati significativi.

3.4.6 CalibratedClassifierCV

CalibratedClassifierCV è una tecnica utilizzata per migliorare le previsioni delle probabilità generate da un classificatore. Questa tecnica è spesso utile quando è necessario ottenere stime di probabilità accurate e affidabili, soprattutto in applicazioni come la classificazione di casi medici o la valutazione del rischio in finanza, dove è essenziale avere stime di probabilità ben calibrate.

La calibrazione delle probabilità si riferisce al processo di aggiustamento delle previsioni di probabilità fatte da un classificatore in modo che riflettano in modo accurato la probabilità reale dell'appartenenza di un'osservazione a una determinata

classe. Alcuni classificatori, specialmente quelli basati su modelli complessi come le reti neurali o gli alberi decisionali profondi, potrebbero produrre previsioni di probabilità che non sono ben calibrate, il che significa che le probabilità stimate potrebbero non essere realistiche.

CalibratedClassifierCV risolve questo problema applicando la calibrazione probabilistica ai risultati del classificatore. Può essere utilizzato con diversi classificatori di base (come *Support Vector Machines*, *Logistic Regression*, o *Random Forest*) e utilizza la tecnica di calibrazione *Platt* o *Isotonic* per migliorare le stime di probabilità. La calibrazione Platt è basata su una regressione logistica, mentre la calibrazione Isotonic è basata su una regressione non parametrica.

L'idea principale è che *CalibratedClassifierCV* prende le previsioni di probabilità del classificatore di base e le regola in modo che riflettano meglio la distribuzione reale delle classi nel set di dati. Questo è particolarmente utile in situazioni in cui la precisione delle probabilità predette è di primaria importanza, come ad esempio nel campo della medicina, in cui le decisioni basate sulle probabilità richiedono una grande accuratezza.

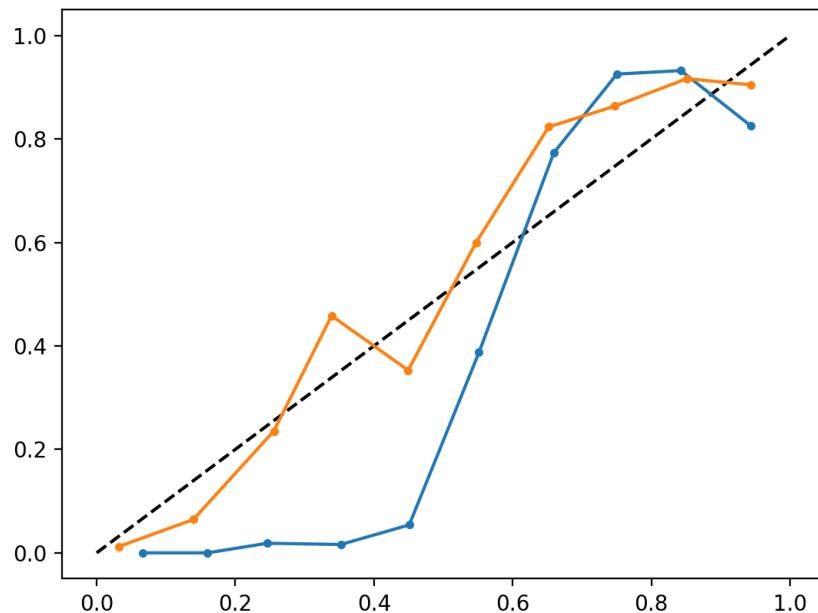


Figura 3.6: *Reliability Diagram* che mostra sia le *probabilità calibrate* (in arancione) che le *probabilità non calibrate* (in blu) sull'algoritmo SVM.

In generale, *CalibratedClassifierCV* è uno strumento utile per migliorare la calibrazione delle previsioni di probabilità dei classificatori, garantendo che siano affidabili e utilizzabili per prendere decisioni importanti nelle applicazioni del mondo reale.

3.4.7 GaussianProcessClassifier

Il *GaussianProcessClassifier* è un algoritmo di classificazione basato su processi gaussiani (Gaussian Processes, GP) nell’ambito dell’apprendimento automatico. Un processo gaussiano è un tipo di modello probabilistico non parametrico che può essere utilizzato per effettuare previsioni probabilistiche su dati complessi e incerti.

Ecco come funziona il *GaussianProcessClassifier*:

- **Processi Gaussiani:** Un processo gaussiano è una generalizzazione di una distribuzione gaussiana (normale) a un numero infinito di variabili. In questo contesto, si tratta di un modo per modellare le relazioni tra le variabili di *input* e *output* senza assumere una forma specifica per la funzione sottostante.
- **Apprendimento Supervisionato:** Quando applicato alla classificazione, il *GaussianProcessClassifier* utilizza processi gaussiani per modellare la relazione tra i dati di *input* e le classi di *output*. L’obiettivo è trovare una distribuzione di probabilità sulle funzioni che mappano gli *input* alle classi. Questa distribuzione cattura l’incertezza nel modello, consentendo di fare previsioni probabilistiche.
- **Incertezza nelle previsioni:** Una delle caratteristiche distintive dei processi gaussiani è la capacità di fornire non solo una previsione, ma anche una stima dell’incertezza associata a quella previsione. Questo è particolarmente utile in situazioni in cui è importante conoscere la fiducia che si può avere nelle previsioni del modello.
- **Funzioni di Kernel:** Il *GaussianProcessClassifier* utilizza le funzioni di Kernel per misurare la similarità tra le coppie di punti nei dati di *input*. La scelta del kernel influisce sulla flessibilità del modello: Kernel diversi catturano relazioni diverse nei dati.

Il *GaussianProcessClassifier* è utile in situazioni in cui si hanno dati di addestramento limitati e dove è importante catturare l'incertezza nei dati. Tuttavia, è computazionalmente più impegnativo rispetto ad alcuni altri algoritmi di classificazione, specialmente quando il numero di osservazioni di addestramento è grande.

Questo modello è spesso usato in applicazioni dove è necessario considerare l'incertezza nelle previsioni, come nella modellazione finanziaria, nelle scienze ambientali e in medicina. La capacità di fornire previsioni probabilistiche e stime di incertezza lo rende una scelta potente in contesti dove la sicurezza delle decisioni è cruciale.

3.5 Scelta delle tecniche di Word Embedding

3.5.1 FastText

FastText è un algoritmo di rappresentazione del linguaggio naturale (NLP) sviluppato da Facebook AI Research (FAIR). FastText è un modello di *word embedding*, il che significa che converte le parole in vettori numerici in uno spazio multidimensionale. Questi vettori catturano le relazioni semantiche tra le parole basandosi sul contesto in cui appaiono.

Subword Embeddings

La caratteristica chiave di FastText è che gestisce le parole come una serie di sottoinsiemi più piccoli chiamati "*n-grammi*" o "*subword*" [22]. Ad esempio, la parola "apple" può essere suddivisa in "ap", "app", "ppl", "ple". Questo approccio consente a FastText di catturare il significato delle parole anche quando non sono presenti nel suo vocabolario, separando le parole in parti più piccole e imparando dai frammenti. Ad esempio, se "apple" è nel vocabolario ma "apples" no, FastText può comunque rappresentare "apples" attraverso il vettore delle sue subword come "app", "ppl", "ple" e la combinazione di questi vettori può rappresentare il significato di "apples".

FastText può gestire parole che non ha mai visto durante l'addestramento, scomponendole in subword e creando rappresentazioni significative basate su queste parti più piccole [23].

Vettorializzazione

Ricapitolando, il concetto chiave dietro FastText e altri modelli di *word embedding* è la rappresentazione distribuita delle parole. Invece di rappresentare le parole come singoli token, FastText le rappresenta come vettori numerici in uno spazio multidimensionale, in cui parole simili sono posizionate vicine l'una all'altra nello spazio vettoriale. Ecco come funziona il processo di creazione di queste rappresentazioni vettoriali:

Durante la fase di addestramento, FastText crea un vocabolario di tutte le parole nel corpus di testo. Ogni parola nel vocabolario è rappresentata come un vettore iniziale casuale. Successivamente, FastText apprende il contesto in cui le parole appaiono nel corpus di testo. Considera il contesto non solo delle parole intere, ma anche delle loro subword. FastText utilizza un algoritmo di apprendimento per ottimizzare i vettori delle parole nel modo che posizioni parole simili vicine nello spazio vettoriale. Questi vettori diventano le rappresentazioni numeriche delle parole. Ogni parola è ora rappresentata come un vettore numerico in uno spazio multidimensionale. Questo spazio multidimensionale cattura le relazioni semantiche tra le parole basandosi sul contesto in cui compaiono nel corpus di testo.

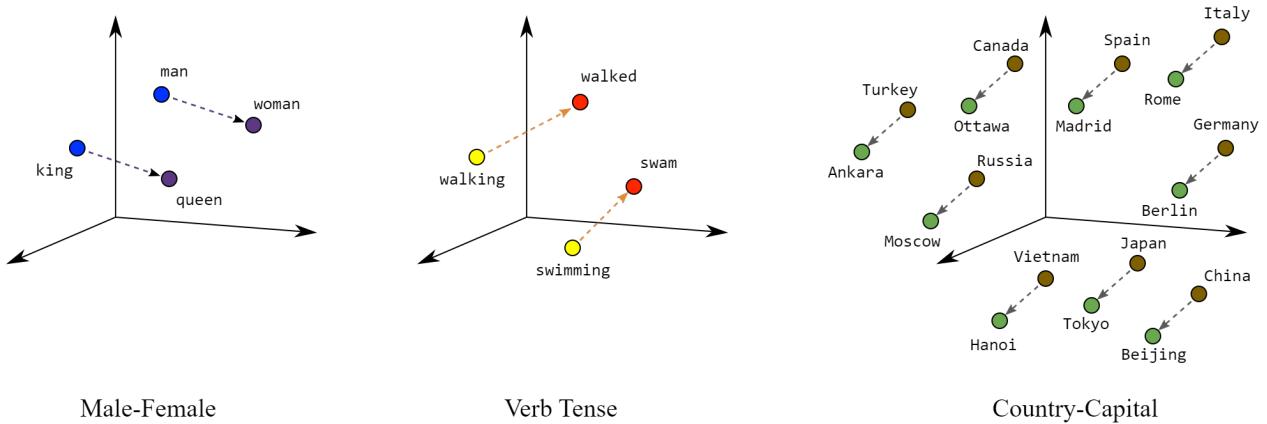


Figura 3.7: Visualizzazioni dei *word embedding*: mostrano relazioni geometriche che catturano legami semantici, come la relazione tra un paese e la sua capitale.

3.5.2 Word2Vec

Word2Vec è un algoritmo di apprendimento automatico utilizzato per la creazione di rappresentazioni vettoriali di parole a partire da grandi quantità di testo. L'obiettivo principale di Word2Vec è catturare il significato semantico delle parole in modo che parole simili siano rappresentate come vettori vicini nello spazio vettoriale. Word2Vec utilizza due approcci principali per creare i vettori di parole: Skip-gram e Continuous Bag of Words (CBOW). Entrambi i modelli coinvolgono la previsione delle parole circostanti date le parole di input (nel caso di CBOW) o previsione delle parole di input date le parole circostanti (nel caso di Skip-gram):

- **CBOW:** In questo approccio, il modello cerca di prevedere una parola di destinazione (target) basandosi sul contesto circostante, cioè le parole di input. Ad esempio, dato il contesto "Il gatto sta sul tavolo", il modello cerca di prevedere la parola *tavolo* sulla base delle parole circostanti *Il, gatto, sta, sul*.
- **Skip-gram:** In questo approccio, il modello fa l'opposto. Prende una parola di input e cerca di prevedere le parole circostanti. Utilizzando la stessa frase di esempio, il modello riceve *tavolo* come input e cerca di prevedere le parole circostanti *Il, gatto, sta, sul*.

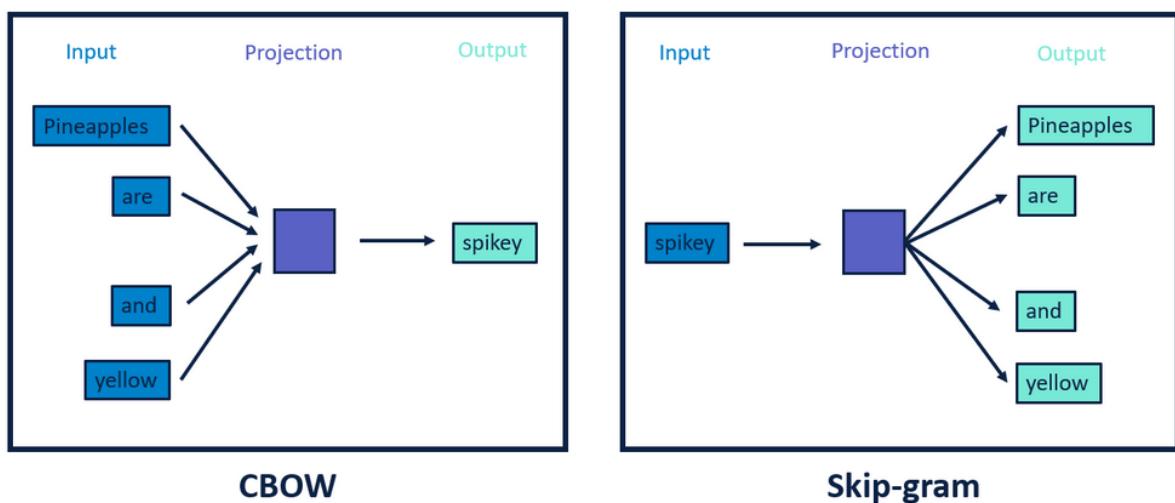


Figura 3.8: Rappresentazione visiva e intuitiva della differenza nell'approccio utilizzato da CBOW e da Skip-gram.

Skip-gram

È stato scelto Skip-gram poiché a differenza di modelli più semplici (come il bag of words), cattura il contesto flessibile delle parole. Questo significa che una parola può avere rappresentazioni diverse basate sul suo contesto circostante, consentendo una maggiore precisione nelle rappresentazioni semantiche.

La funzione obiettivo⁷ del modello Skip-gram può essere rappresentata nel modo seguente:

$$\frac{1}{T} \sum_{t=1}^T \sum_{-c \leq j \leq c, j \neq 0} \log p(w_{t+j} | w_t)$$

- T è il numero totale di parole nel corpus.
- c è la dimensione della finestra di contesto, indicando la massima distanza tra la parola target w_t e le parole di contesto w_{t+j} considerate durante l'addestramento.
- $p(w_{t+j} | w_t)$ è la probabilità di prevedere la parola di contesto w_{t+j} dato che è nota la parola target w_t .

$$\frac{1}{T} \sum_{t=1}^T$$

Questa parte significa che stiamo facendo una media (\sum indica somma e T è il numero totale di parole nel nostro testo) dell'obiettivo per ogni parola nel nostro corpus. La media è presa per assicurarci che l'obiettivo sia bilanciato su tutte le parole nel corpus.

$$\sum_{-c \leq j \leq c, j \neq 0}$$

Qui, si sommano tutte le parole nel contesto della parola target w_t . L'indice j rappresenta la distanza tra la parola target e le parole nel suo contesto. c rappresenta la dimensione della finestra di contesto - quante parole a sinistra e quante a destra sono considerate intorno alla parola target.

$$\log p(w_{t+j} | w_t)$$

⁷<https://www.tensorflow.org/text/tutorials/word2vec>

Questo è il logaritmo della probabilità che la parola nel contesto w_{t+j} sia predetta dato che conosciamo la parola target w_t . In altre parole, stiamo cercando di massimizzare la probabilità di predire le parole nel contesto quando conosciamo la parola target.

Conclusioni

Nel modello *skip-gram*, l’obiettivo è massimizzare questa probabilità media del logaritmo, il che significa massimizzare la probabilità di prevedere le parole di contesto date le parole target in tutto il corpus di addestramento. Questo obiettivo incoraggia il modello a imparare le rappresentazioni vettoriali delle parole in modo che le parole di contesto siano probabili quando è nota la parola target.

3.5.3 GloVe

GloVe, acronimo di *Global Vectors for Word Representation*, è un algoritmo di *word embedding* che converte parole in vettori numerici densi in modo che possano essere utilizzati in algoritmi di apprendimento automatico. I *word embeddings* sono rappresentazioni vettoriali di parole in uno spazio multidimensionale, dove parole semanticamente simili sono collocate vicine l’una all’altra. Questi embedding sono utili in molte applicazioni di linguistica computazionale, come il riconoscimento di entità, la traduzione automatica, l’analisi del sentimento e altre attività di elaborazione del linguaggio naturale.

GloVe è stato sviluppato da *Jeffrey Pennington, Richard Socher e Christopher D. Manning* presso la *Stanford University*. L’algoritmo si basa su statistiche globali del corpus di testo per catturare relazioni semantiche tra le parole. A differenza di altri approcci di word embedding, come *word2vec*, GloVe non si basa solo su informazioni locali (come il contesto di una parola), ma utilizza l’intero corpus di testo per catturare le relazioni semantiche.

Il processo di creazione di word embedding con GloVe avviene attraverso i seguenti passaggi:

Matrice di Co-occorrenza

GloVe inizia creando una matrice di co-occorrenza delle parole nel corpus di testo. Questa matrice tiene traccia di quante volte le parole appaiono insieme in contesti specifici.

Ottimizzazione del modello

Successivamente, l'algoritmo ottimizza una funzione di costo che misura la discrepanza tra i prodotti scalari dei vettori delle parole (nella forma di logaritmi delle probabilità di co-occorrenza) e i valori della matrice di co-occorrenza.

Creazione degli embedding

Una volta che il modello è addestrato, i vettori risultanti rappresentano le parole nel corpus in uno spazio vettoriale. Questi vettori catturano le relazioni semantiche tra le parole in base ai loro contesti di co-occorrenza.

Conclusioni

GloVe ha dimostrato di produrre embedding di parole di alta qualità e può essere utilizzato per migliorare le prestazioni di vari compiti di linguistica computazionale. I vettori di parole GloVe possono essere preaddestrati su grandi corpora di testo e utilizzati come input per modelli di apprendimento automatico nelle applicazioni di *elaborazione del linguaggio naturale*.

3.5.4 BERT

BERT, acronimo di Bidirectional Encoder Representations from Transformers, è un modello di linguaggio basato su trasformatori (**Transformers**) sviluppato da *Google*. È stato introdotto nel paper "*BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*" [24] nel 2018. BERT è stato progettato per migliorare la comprensione del contesto nelle applicazioni di elaborazione del linguaggio naturale (NLP). La sua principale innovazione risiede nell'approccio bidirezionale all'analisi del testo, il che significa che il modello può comprendere le parole in un contesto sia precedente che successivo nella sequenza, il che migliora significativamente la sua capacità di comprendere il significato delle parole e delle frasi in modo più accurato.

I punti chiave di BERT sono il *Pretraining* e il *Fine-Tuning*:

Pretraining

BERT viene preallenato su grandi quantità di testo senza supervisione. Questo significa che il modello impara rappresentazioni linguistiche generali dal testo fornito, senza avere etichette specifiche per compiti particolari. Il processo di *pretraining* di BERT coinvolge l’addestramento di un modello bidirezionale su vasti corpus testuali.

Fine-tuning

Dopo il preallenamento, BERT può essere adattato a compiti specifici attraverso un processo chiamato *fine-tuning*. In questa fase, il modello viene addestrato su un set di dati specifico per un’attività particolare, come la classificazione del testo o il riconoscimento delle entità nominate. Questo adattamento consente a BERT di apprendere dettagli specifici del compito.

Struttura del modello

BERT utilizza l’architettura del Transformer, che è composta da vari strati di *encoder* e *decoder*. Tuttavia, BERT si concentra solo sugli encoder. Gli encoder di BERT sono impilati insieme per creare una rete neurale profonda. Ogni encoder è basato su meccanismi di autoattenzione (*self-attention*) che consentono al modello di dare più peso alle parti rilevanti del testo durante il processo di apprendimento.

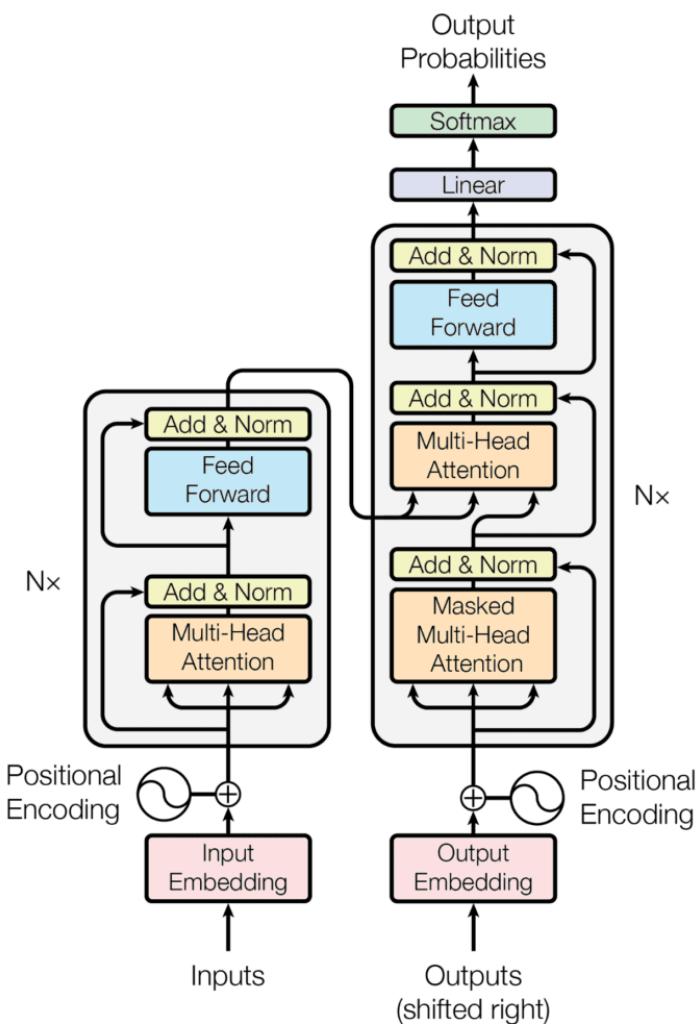


Figura 3.9: L'architettura del **Transformer** segue una struttura *encoder-decoder*. Il compito dell'*encoder*, nella metà a sinistra, è quello di mappare una sequenza di input in una sequenza di rappresentazioni continue, che viene poi alimentata al decoder. Il *decoder*, nella metà a destra, riceve l'output dell'encoder insieme all'output del decoder al passo temporale precedente per generare una sequenza di output.

Tokenizzazione

BERT utilizza una tecnica di tokenizzazione chiamata *WordPiece Tokenization*, che suddivide le parole in sotto-parti chiamate "*subword tokens*". Questo è utile per gestire parole complesse e lingue con una vasta varietà di parole.

Applicazioni

BERT è stato ampiamente utilizzato in una serie di applicazioni di elaborazione del linguaggio naturale, tra cui la comprensione del linguaggio, la traduzione automatica, la generazione di testo e molto altro. A causa della sua capacità di comprendere il contesto in modo più profondo rispetto ai modelli precedenti, BERT ha portato a miglioramenti significativi in molte attività di NLP.

In breve, BERT è un modello di linguaggio basato su trasformatori che ha rivoluzionato il campo dell'elaborazione del linguaggio naturale grazie alla sua capacità di comprendere il contesto bidirezionale nelle sequenze di testo, migliorando così le prestazioni in una varietà di compiti di NLP.

3.6 Tecniche di Validazione

Come tecnica di valutazione, è stata utilizzata la *cross-validation* con una divisione stratificata dei dati in 5 fold.

Cross Validation

La *cross-validation* è una tecnica utilizzata per valutare le prestazioni di un modello di *machine learning*. Invece di dividere il dataset in un set di allenamento e uno di test una sola volta, la *cross-validation* prevede di suddividere il dataset in K parti uguali (dove K è il numero di fold) e di addestrare il modello K volte. In ogni iterazione, una delle K parti viene utilizzata come set di test e le rimanenti K-1 parti vengono utilizzate come set di allenamento. Questo aiuta a ottenere una stima più affidabile delle prestazioni del modello, poiché il modello viene valutato su dati diversi in ogni iterazione.

Divisione Stratificata

La divisione stratificata è una tecnica di campionamento in cui i dati vengono suddivisi in modo che ogni piega (fold) abbia una distribuzione simile delle classi del dataset originale. Questo è particolarmente importante quando si lavora con dataset sbilanciati, cioè dataset in cui alcune classi sono rappresentate da un numero significativamente inferiore di campioni rispetto ad altre. La divisione stratificata aiuta a garantire che ogni fold contenga una proporzione adeguata di esempi da

ciascuna classe, migliorando così l'affidabilità delle valutazioni delle prestazioni del modello.

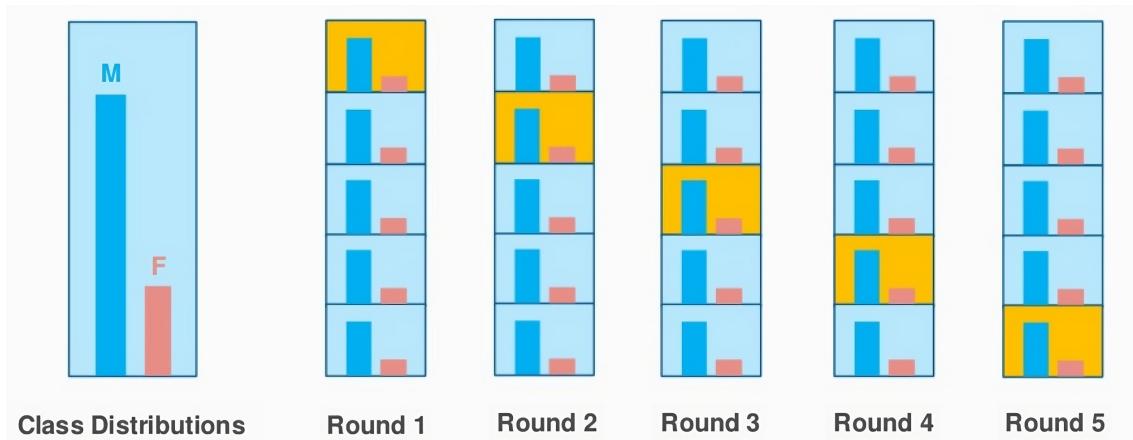


Figura 3.10: Esempio di una 5-fold Cross Validation. In azzurro è raffigurato il *training data*, mentre in giallo è mostrato il *validation data*.

Quindi, è stata utilizzata la tecnica di cross-validation con una divisione stratificata dei dati in 5 fold. Ogni fold viene utilizzato come set di test esattamente una volta, mentre i restanti fold vengono utilizzati come set di allenamento. Questo processo viene ripetuto 5 volte, con una diversa piega come set di test in ogni iterazione, al fine di ottenere una stima robusta delle prestazioni del modello.

3.7 Metriche di Validazione

Per valutare l'efficacia dei modelli di machine learning, sono comunemente utilizzate diverse metriche, tra cui *precision*, *recall*, *F1-score*, *Balanced Accuracy* e *ROC AUC*. Queste metriche sono calcolate considerando le seguenti categorie di predizioni del modello:

- **True Positive (TP):** Numero di previsioni positive correttamente classificate.
- **False Positive (FP):** Numero di previsioni positive erroneamente classificate.
- **True Negative (TN):** Numero di previsioni negative correttamente classificate.
- **False Negative (FN):** Numero di previsioni negative erroneamente classificate.

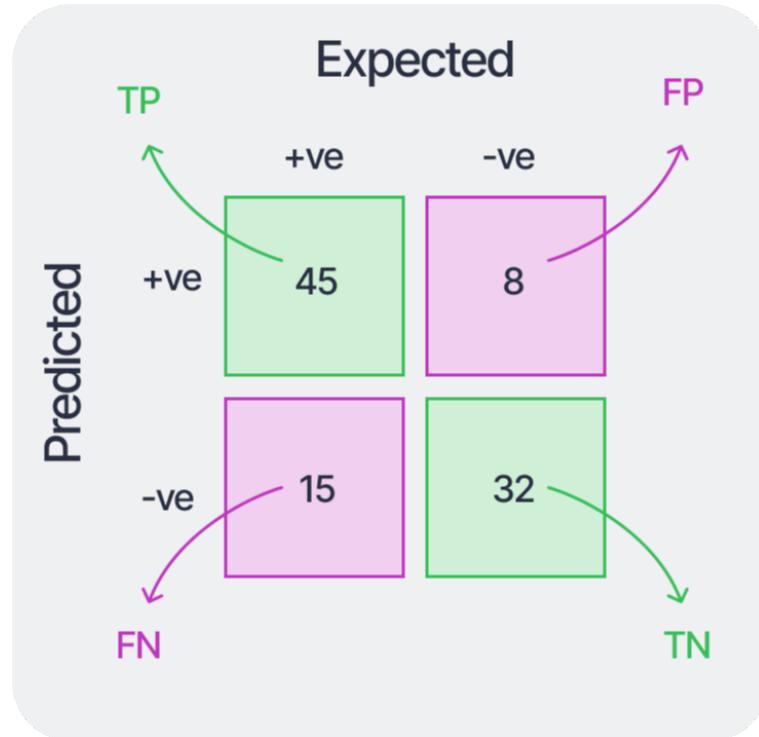


Figura 3.11: Matrice di Confusione per un dataset di classificazione binaria.

La **precision** è definita come il rapporto tra il numero di previsioni positive corrette (TP) e il totale delle volte in cui il modello ha previsto quella classe:

$$\text{Precision, P} = \frac{TP}{TP + FP}$$

La **recall**, o sensibilità, rappresenta il rapporto tra le previsioni positive corrette (TP) e il totale dei casi effettivamente positivi:

$$\text{Recall, R} = \frac{TP}{TP + FN}$$

L'**accuracy** è il rapporto tra le classificazioni corrette (TP e TN) e il totale delle previsioni effettuate dal modello:

$$\text{Accuracy, A} = \frac{TP + TN}{TP + TN + FP + FN}$$

La **Balanced Accuracy** è una versione modificata dell'accuratezza che tiene conto del bilanciamento tra le classi nel dataset, utile quando le classi sono sbilanciate. Questa metrica considera la media delle sensibilità (*recall*) di ciascuna classe:

$$\text{Balanced Accuracy} = \frac{\text{Sensibilità di Classe 1} + \text{Sensibilità di Classe 2} + \dots}{\text{Numero di Classi}}$$

L'area sotto la curva ROC (**ROC AUC**) rappresenta la capacità del modello di distinguere tra classi positive e negative su un insieme di dati. Una ROC AUC vicina a 1 indica un modello molto accurato.

In questo determinato contesto, la *recall* assume maggiore importanza rispetto alla *precision*, poiché l'obiettivo è rilevare tutti i Requisiti Software. Ad ogni modo, la *precision* è comunque importante, poiché una precisione bassa potrebbe generare parecchi falsi positivi.

CAPITOLO 4

Risultati

4.1 Risultati delle tecniche di Word Embedding

In totale, sono stati utilizzati 30 algoritmi di classificazione, e 4 tecniche di Word Embedding. Per ognuna di esse, saranno riportati i tre algoritmi che hanno ottenuto le metriche più alte. Mediamente, la tecnica di word embedding dalle prestazioni inferiori è stata FastText. A seguire, vi sono Word2Vec, poi GloVe e infine BERT, che è risultato essere il migliore.

Tutti i risultati delle metriche dei 30 classifier, per ogni tecnica di word embedding, sono disponibili al link della *repository* di **GitHub**:

<https://github.com/GiovanniBorrelli/Software-Requirements-Classification.git>

4.1.1 fastText

Classifier	Accuracy	F1-Score	Recall	Precision	Time Taken
NuSVC	0,81	0,79	0,82	0,76	0,11s
LinearDiscriminantAnalysis	0,79	0,76	0,76	0,76	0,19s
LogisticRegressionCV	0,74	0,65	0,54	0,80	2,45s

Tabella 4.1: Risultati dei migliori classificatori con fastText

Utilizzando fastText, la maggior parte degli algoritmi ha dimostrato scarse prestazioni, tranne i 3 riportati nella tabella.

Per quanto riguarda **LogisticRegressionCV**, esso è robusto quando le relazioni tra le caratteristiche e le classi sono lineari. L'uso della validazione incrociata aiuta a selezionare un buon parametro di regolarizzazione, migliorando così la generalizzazione del modello.

LinearDiscriminantAnalysis aiuta a trovare un piano (o un iperpiano in spazi multidimensionali) che massimizza la separazione tra classi in modo lineare. Questa tecnica è comunemente usata in problemi di classificazione, specialmente quando le classi sono ben separabili linearmente.

NuSVC è adatto a trovare una sorta di "*linea di separazione*" tra diverse categorie nei dati. Questa linea è disegnata in modo tale da massimizzare la distanza tra le categorie, il che rende più facile distinguere tra di esse.

4.1.2 word2vec

Classifier	Accuracy	F1-Score	Recall	Precision	Time Taken
LogisticRegressionCV	0,78	0,74	0,72	0,76	3,46s
NuSVC	0,75	0,71	0,72	0,71	0,10s
ExtraTreesClassifier	0,74	0,66	0,58	0,76	1,82s

Tabella 4.2: Risultati dei migliori classificatori con word2vec

Il miglior algoritmo con word2vec ha avuto prestazioni leggermente minori del miglior algoritmo con fastText, tuttavia la media delle prestazioni di tutti gli algoritmi con word2vec è più alta di quella con fastText.

LogisticRegression e **NuSVC** riappaiono anche utilizzando word2vec.

ExtraTreesClassifier è un algoritmo basato su alberi decisionali. È simile a un albero decisionale, ma anziché cercare il miglior split, utilizza una selezione casuale di attributi per trovare il miglior split. Questo lo rende più robusto contro l'overfitting. *ExtraTreesClassifier* è efficace quando si hanno dataset rumorosi. La sua natura robusta lo rende adatto a una varietà di dati.

4.1.3 GloVe

Classifier	Accuracy	F1-Score	Recall	Precision	Time Taken
NuSVC	0,84	0,82	0,83	0,81	0,08s
RidgeClassifier	0,83	0,81	0,81	0,81	0,10s
MLPClassifier	0,82	0,80	0,81	0,79	3,09s

Tabella 4.3: Risultati dei migliori classificatori con GloVe

GloVe ha ottenuto generalmente buone prestazioni per la maggior parte degli algoritmi. Ciò potrebbe essere dovuto al fatto che utilizza anche vettori di parole preaddestrati, e cattura anche relazioni contestuali tra le parole in base alla loro co-occorrenza nei dati di addestramento.

NuSVC compare per la terza volta tra i migliori algoritmi. Sembra essere particolarmente efficace in questa tipologia di problemi.

RidgeClassifier è efficace quando si desidera prevenire l'overfitting in un modello di regressione logistica. La regolarizzazione di Ridge può aiutare a mantenere stabile il modello anche con dataset di dimensioni ridotte.

MLPClassifier è un algoritmo che utilizza reti neurali artificiali, ed è particolarmente indicato per problemi complessi. Ha impiegato molto più tempo per essere eseguito rispetto agli altri algoritmi.

4.1.4 BERT

Classifier	Accuracy	F1-Score	Recall	Precision	Time Taken
SVC	0,90	0,89	0,89	0,88	0,16s
MLPClassifier	0,90	0,88	0,9	0,86	3,55s
LogisticRegression	0,89	0,87	0,87	0,87	0,26s

Tabella 4.4: Risultati dei migliori classificatori con BERT

BERT si è dimostrato il migliore, in quanto tutti gli algoritmi hanno mostrato ottime prestazioni (tranne DummyClassifier).

Grazie al suo pre-addestramento su grandi corpus di testo, BERT apprende rappresentazioni linguistiche generali che possono essere trasferite a una vasta gamma di task di NLP. Questo trasferimento di conoscenze aiuta BERT a raggiungere prestazioni elevate anche su dataset relativamente piccoli o specifici.

LogisticRegression appare per la terza volta, quindi sembra essere decisamente appropriato in questo caso.

MLPClassifier non ha avuto una buona performance nel caso di fastText e word2vec. Invece, si è dimostrato molto efficace nel caso di BERT e di GloVe. Ciò, probabilmente, è dovuto al fatto che con questi ultimi si ha a che fare con modelli preaddestrati.

SVC si dimostra il modello più efficace e allo stesso tempo più veloce. La sua semplicità lo rende particolarmente adatto per risolvere questo tipo di task. Per questo motivo è stato scelto come algoritmo finale, insieme a BERT.

4.2 Risultati del testing sui Dataset

Da un punto di vista più ampio, è stata fatta un'analisi sia di discussioni nelle fasi iniziali dello sviluppo del software (*ChatGPT discussions* e la conversazione fornita da *REConSum*), sia di discussioni quotidiane durante lo sviluppo del software (*Slack dataset*), sia di meeting tramite *ELITR*.

4.2.1 Slack Conversations

Il dataset è formato da messaggi del seguente tipo: "*Huh. Got it*", "*Frameworks are evil*", "*Okay.*". Poi erano presenti altri messaggi particolarmente specifici con del codice: "*class Foo: def bar(self): do work return self Foo().bar().baz().json*", "*return super().len() is much nicer... slight smile*". La presenza di Requisiti Software è abbastanza scarsa. Dopo aver analizzato il dataset delle conversazioni sulla programmazione in Python, e il dataset sulla programmazione in Clojure, È stato osservato che le discussioni sono molto tecniche e spesso un messaggio è un vero e proprio blocco di codice, oppure una risposta secca a un altro messaggio. Una delle poche frasi che il modello ha identificato come requisito software, è: "*how can I get the list of protocols which a type implements?*". Anche se non si tratta di un requisito software diretto, è collegato alla funzionalità del software. Potrebbe essere considerato come un requisito implicito per una funzionalità che permetta all'utente di accedere alle informazioni su "*protocols implementations*". Probabilmente il modello potrebbe averlo riconosciuto come un requisito software anche per la presenza di termini tecnici come "*list of protocols*" e "*type implements*".

4.2.2 REConSum Conversation

È stata analizzata la conversazione fornita da REConSum. A differenza sua, il nostro modello ha identificato come requisiti software anche delle frasi che non sono domande, come "*It was mentioned, for example, that it should be accessed by 50,000 people uh simultaneously and fast.*". Identificare questo requisito non sarebbe stato possibile utilizzando il loro modello.

4.2.3 ELITR Minuting Corpus

È stato analizzato il dataset ELITR, il quale era organizzato nel modo seguente: alcune conversazioni sono state utilizzate per il *test*, altre per il *training*, altre per la *validation*. È stata esaminata una conversazione da ogni set per controllare se fossero diversi da loro, ma alla fine si sono rivelati essere molto simili nella forma e nei contenuti. Il numero di requisiti software identificati è stato più basso, e ciò è dovuto

al fatto che le conversazioni non sono troppo focalizzate sull'ingegneria del software ma su argomenti più ampi. Una frase interessante è "*A new message is a new subtitle to be displayed, and it can substitute the biggest one or update the let's say the button of the text field in order to let <unintelligible/>*." Questa frase potrebbe risultare ambigua a causa del "*<unintelligible/>*": che indica una porzione di testo che non è stata inserita. tuttavia il modello ha deciso di classificare la frase come requisito software in base al testo riammennente che è presente nella frase. Probabilmente questo significa che il modello ha riconosciuto il contesto e il significato della frase senza avere tutte le parole a disposizione.

4.3 Analisi dei limiti tramite ChatGPT

Infine, con l'aiuto di ChatGPT è stata generata una conversazione fittizia tra un Software Engineer e un Client. È stato testato il modello su questa conversazione, e sono emersi dei risultati interessanti.

il modello è riuscito a identificare correttamente delle frasi che contenevano un po' di "rumore", cioè parole inutili al fine del riconoscimento dei requisiti (Ad esempio: "*That's a great thing!*", "*Huh*", "*By the way...*"). Il modello è riuscito a ragionare sulle frasi senza aver bisogno di parole o trigger specifici (Ad esempio "*The system must be...*" o "*The software needs...*", che sono spesso presenti nel dataset creato per addestrare il modello).

4.3.1 Filosofia dei Requisiti Software

Esistono questioni aperte intrinseche alla natura del modello che devono essere ancora affrontate. Il modello ha riconosciuto come requisiti funzionali frasi ambigue come: "*The software needs to be user-friendly.*" e "*The system needs to be flexible and adaptable.*". In questo caso la questione rientra più nel piano filosofico. Quanto può essere utile sapere questo tipo di "Requisiti"? Non c'è una risposta definitiva, infatti è un problema ancora aperto. Dipende anche dall'obiettivo che ci si pone.

La mia opinione personale è la seguente: dire che "*il sistema è user-friendly*" è una frase molto utilizzata al giorno d'oggi, soprattutto da persone che presentano un loro

software o prodotto per pubblicizzarlo. Di conseguenza questa frase perde un po' l'intensità del suo significato; ma non la perde completamente. Sapere che il cliente vuole un sistema user-friendly, per quanto possa essere un'informazione che non dice molto su cosa voglia di preciso, resta comunque un'informazione utile per lo sviluppo del software: perché un software si può davvero rendere user-friendly, e non è solo un modo di pubblicizzare il proprio prodotto. Quindi, una frase usata spesso non le fa perdere il significato.

Inoltre, il modello tende ad essere più severo nel ritenere che una frase sia un requisito software. Infatti, se identifica una frase come requisito funzionale, essa lo è quasi certamente. Mentre se identifica una frase come non-requisito, c'è una probabilità leggermente più alta che quella frase potrebbe contenere qualche requisito nascosto o detto in maniera implicita. Questo probabilmente è dovuto ai documenti (*PURE*) utilizzati per la creazione del dataset di addestramento, che contengono frasi molto specifiche in ambito di ingegneria del software. Ciò significa che il modello è stato addestrato su frasi molto dettagliate e su argomenti avanzati e specifici (ma che comunque hanno una certa varietà perché sono stati inseriti vari tipi di requisiti diversi). Se la conversazione tra Cliente e Software Engineer dovesse trattare di argomenti più ampi che trattano anche di altri ambiti che non sono solo Software, il modello potrebbe avere più difficoltà a trovare requisiti funzionali, perché meno esplicativi.

4.3.2 Ambiguità dei Requisiti Software

Infine, è stato chiesto a ChatGPT di generare delle frasi ambigue o particolari per testare i limiti del modello. Tra le frasi che sono state classificate come requisiti software vi sono:

- "**And do you have any preferences for the programming language?**". questa frase menziona la preferenza per uno specifico linguaggio di programmazione. Anche se potrebbe non essere un requisito software diretto, è un requisito essenziale per lo sviluppo di un'applicazione.
- "**We can use React Native or Flutter for that purpose.**". Anche se non rappresenta esplicitamente un requisito software, suggerisce una possibile scelta

di programmazione per un bisogno specifico, che è rilevante per un requisito software.

La seguente frase, invece, è stata classificata come non-requisito software:

- **"How about we go with MySQL or PostgreSQL?"**. Il motivo potrebbe essere che una richiesta riguardo a delle scelte su un database potrebbe non essere direttamente un requisito software. Oppure sarebbe potuto essere classificato come requisito software se fossero state utilizzate espressioni più dirette come "we should" or "we need."

Complessivamente, questi sono i risultati, dopo aver analizzato tutti i dataset disponibili. Il modello riesce a riconoscere facilmente requisiti software espressi chiaramente, mentre se la cava abbastanza bene su frasi ambigue o requisiti impliciti.

4.4 Risultati e Confronto con REConSum

È stato testato il modello di Reconsum su *ELITR*, *Slack Chats* e la conversazione fornita da *REConSum*, in modo da poter fare un confronto tra il loro e il nostro approccio.

Innanzitutto, REConSum analizza solamente la rilevanza delle domande. Ciò diminuisce notevolmente la possibilità di trovare tutti i requisiti software all'interno della conversazione.

Seconda limitazione: il loro modello indica la conversazione (in questo caso intesa come "*insieme di frasi*") in cui è presente il requisito software, ma non la frase specifica.

Inoltre, la rilevanza viene stabilita utilizzando un TF-IDF Dataset, cioè si stabilisce la rilevanza di una frase in base alla frequenza con cui appaiono certi termini. Se ad esempio "*System*" è una parola ad alta frequenza, ed appare in una conversazione analizzata dal modello, c'è una probabilità più alta che il modello riconosca quella conversazione come contenente un requisito software.

Per quanto riguarda le *Slack Chats*, così come il nostro modello ha individuato pochi requisiti software, così è stato anche per il loro. Questo è dovuto al fatto che le *Slack Chats* sono più improntate alla programmazione e al codice.

ELITR è formato da conversazioni di carattere generale, e quindi non concentrate direttamente in ambito di ingegneria del software. Questo comporta che il modello identifichi un numero medio/basso di requisiti, proprio perché la loro presenza è minore.

Per quanto riguarda *REConSum*, quando vengono applicate ulteriori condizioni ai requisiti, come ad esempio la necessità di essere formulati come domande, il loro numero diminuisce significativamente. In questo contesto, *REConSum* ha individuato solo un numero limitato di requisiti. Questa situazione è dovuta non tanto alle capacità del modello, ma piuttosto alle decisioni prese durante la progettazione del sistema, le quali hanno influenzato negativamente le prestazioni del sistema su questo tipo di dataset.

Conclusioni

Il nostro modello presenta diverse differenze rispetto al loro:

- Esamina tutte le frasi, non solo quelle formulate come domande. Naturalmente, considerare un insieme V , rappresentato da tutte le frasi, ci consente di avere un maggior numero di elementi rispetto a un suo sottoinsieme S , a meno che non si verifichi un caso particolare in cui V e S siano uguali.
- Il nostro modello è stato addestrato utilizzando un dataset in cui ogni frase è correlata all'ingegneria del software, a differenza del loro metodo *TD-IDF* in cui alcune parole ad alta frequenza potrebbero risultare fuorvianti.
- Il nostro modello è in grado di identificare la frase specifica in cui è contenuto il requisito software, offrendo così una maggiore precisione nel contesto dell'analisi dei requisiti.

CAPITOLO 5

Conclusioni

Il lavoro svolto in questa tesi descrive lo sviluppo di un tool per identificare e individuare requisiti software all'interno di una conversazione. Utilizzando diversi dataset, è stata fatta un'analisi sia di discussioni nelle fasi iniziali dello sviluppo del software, sia di discussioni quotidiane durante lo sviluppo del software, sia di meeting. Sono stati utilizzati diversi modelli di machine learning e diverse tecniche di word embedding. Alla fine, come tecnica di *word embedding* è stato scelto **BERT**, mentre come algoritmo di *machine learning* è stato selezionato **SVC**, che ha *recall*, *precision*, *accuracy* e *F1-Score* che si aggirano intorno al **90%**.

Rispetto a *REConSum*, attraverso il nostro tool è possibile individuare la frase specifica in cui è presente un requisito software, e con maggior precisione. Tuttavia, è possibile apportare dei miglioramenti in futuro:

- l'utilizzo di un dataset di addestramento con più dati, o appartenenti a un dominio applicativo più ampio.
- La possibilità di indicare anche la categoria a cui appartiene il requisito software.

La *repository* di **Github** contenente tutte le informazioni relative al progetto è disponibile al seguente link:

<https://github.com/GiovanniBorrelli/Software-Requirements-Classification>

Bibliografia

- [1] D. A. López-Hernández, J. O. Ocharán-Hernández, E. Mezura-Montes, and Ángel J. Sánchez-García, “Automatic classification of software requirements using artificial neural networks: A systematic literature review,” *9th International Conference in Software Engineering Research and Innovation (CONISOFT)*, 2021. [Online]. Available: <https://ieeexplore.ieee.org/document/9653417> (Citato a pagina 6)
- [2] J. Winkler and A. Vogelsang, “Automatic classification of requirements based on convolutional neural networks,” in *Proceedings- 2016 IEEE 24th International Requirements Engineering Conference Workshops REW 2016*, 2017, pp. 39–45. (Citato a pagina 6)
- [3] R. Chatterjee, A. Ahmed, and P. R. Anish, “Identification and classification of architecturally significant functional requirements,” in *2020 IEEE Seventh International Workshop on Artificial Intelligence for Requirements Engineering (AIRE)*, 2020, pp. 9–17. (Citato a pagina 6)
- [4] A. Dekhtyar and V. Fong, “Re data challenge: Requirements identification with word2vec and tensorflow,” in *Proceedings- 2017 IEEE 25th International Requirements Engineering Conference RE 2017*, 2017, pp. 484–489. (Citato a pagina 6)

- [5] Z. S. H. Abad, O. Karras, P. Ghazi, M. Glinz, G. Ruhe, and K. Schneider, "What works better? a study of classifying requirements," in *Proceedings - 2017 IEEE 25th International Requirements Engineering Conference RE 2017*, no. 1, 2017, pp. 496–501. (Citato a pagina 6)
- [6] M. A. Haque, M. A. Rahman, and M. S. Siddik, "Non-functional requirements classification with feature extraction and machine learning: An empirical study," in *1st International Conference on Advances in Science Engineering and Robotics Technology 2019 ICASERT 2019*, vol. 2019, no. Icasert, 2019. (Citato a pagina 7)
- [7] V. Silva-Rodríguez, S. E. Nava-Muñoz, L. A. Castro, F. E. Martínez-Pérez, H. G. Pérez-González, and F. Torres-Reyes, "Classifying design-level requirements using machine learning for a recommender of interaction design patterns," *IET Software*, vol. 14, no. 5, pp. 544–552, 2020. (Citato a pagina 7)
- [8] C. S. R. K. Surana, S. D. B. Gupta, and S. P. Shankar, "Intelligent chatbot for requirements elicitation and classification," in *2019 4th IEEE International Conference on Recent Trends on Electronics Information Communication and Technology RTEICT 2019 - Proceedings*. IEEE, 2019, pp. 866–870. (Citato a pagina 7)
- [9] T. Spijkman, X. de Bondt, F. Dalpiaz, and S. Brinkkemper, "Summarization of elicitation conversations to locate requirements-relevant information," *Dept. of Information and Computing Sciences, Utrecht University, The Netherlands*, 2023. [Online]. Available: <https://webspace.science.uu.nl/~dalpi001/papers/spij-bond-dalp-brin-23-refsq.pdf> (Citato a pagina 7)
- [10] V. Ivanov, A. Sadovskykh, A. Naumchev, A. Bagnato, and K. Yakovlev, "Extracting software requirements from unstructured documents," *Innopolis University, Innopolis, Russia*, 2022. [Online]. Available: <https://arxiv.org/pdf/2202.02135.pdf> (Citato a pagina 9)
- [11] A. Nedoluzhko, M. Singh, M. Hledíková, T. Ghosal, and O. Bojar, "ELITR Minuting Corpus: A novel dataset for automatic minuting from multi-party meetings in English and Czech," in *Proceedings of the 13th International Conference on Language Resources and Evaluation LREC 2019*, no. 1, 2019, pp. 496–501. (Citato a pagina 9)

- ge Resources and Evaluation (LREC-2022).* Marseille, France: European Language Resources Association (ELRA), June 2022, in print. (Citato a pagina 10)
- [12] R. Pupale, "Support vector machines(svm) — an overview," *Towards Data Science*, 2018. [Online]. Available: <https://towardsdatascience.com/https-medium-com-pupalershikesh-svm-f4b42800e989> (Citato a pagina 12)
- [13] R. Misra, "Support vector machines — soft margin formulation and kernel trick," *Towards Data Science*, 2019. [Online]. Available: <https://towardsdatascience.com/support-vector-machines-soft-margin-formulation-and-kernel-trick-4c9729dc8efe> (Citato a pagina 13)
- [14] S. Sreenivasa, "Radial basis function (rbf) kernel: The go-to kernel," *Towards Data Science*, 2020. [Online]. Available: <https://towardsdatascience.com/radial-basis-function-rbf-kernel-the-go-to-kernel-acf0d22c798a> (Citato a pagina 13)
- [15] A. Al-Masri, "How does backpropagation in a neural network work?" 2023. [Online]. Available: <https://builtin.com/machine-learning/backpropagation-neural-network> (Citato a pagina 15)
- [16] D. Jurafsky and J. H. Martin, "Speech and language processing," 2023. [Online]. Available: <https://web.stanford.edu/~jurafsky/slp3/5.pdf> (Citato a pagina 16)
- [17] A. Wickramarachchi, "Logistic regression and decision boundary," *Towards Data Science*, 2020. [Online]. Available: <https://towardsdatascience.com/logistic-regression-and-decision-boundary-eab6e00c1e8> (Citato a pagina 16)
- [18] J. Brownlee, "Machine learning mastery," 2020. [Online]. Available: <https://machinelearningmastery.com/cross-entropy-for-machine-learning/> (Citato a pagina 17)
- [19] A. Shrivastava, "Lecture 5: Deep learning: Logistic regression," 2022. [Online]. Available: https://www.cs.rice.edu/~as143/COMP642_Spring22/Scribes/Lecture5 (Citato a pagina 17)
- [20] V. Joshi, "Activation functions," *GeeksforGeeks*, 2019. [Online]. Available: <https://www.geeksforgeeks.org/activation-functions/> (Citato a pagina 18)

- [21] A. Provino, “Ridge regression,” 2019. [Online]. Available: <https://www.andrea-provino.it/ridge-regression> (Citato a pagina 22)
- [22] N. Subedi, “Fasttext: Under the hood,” *Towards Data Science*. [Online]. Available: <https://towardsdatascience.com/fasttext-under-the-hood-11efc57b2b3> (Citato a pagina 27)
- [23] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, “Enriching word vectors with subword information,” *Facebook AI Research*, 2017. [Online]. Available: <https://arxiv.org/pdf/1607.04606.pdf> (Citato a pagina 27)
- [24] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” 2019. (Citato a pagina 32)

Ringraziamenti

Grazie al prof. *Fabio Palomba* per avermi incuriosito ad approfondire questo campo, attraverso il corso di Fondamenti di Intelligenza Artificiale. Nel momento in cui si sceglie di fare il professore, automaticamente si riceve una forte responsabilità; secondo me, si può studiare autonomamente qualsiasi argomento, ma è diritto e dovere del professore rendere l'introduzione – a un campo qualsiasi – interessante e leggera.

Grazie al dott. *Francesco Casillo* per gli utili consigli che mi ha fornito durante lo svolgimento del tirocinio. Inoltre, mi ha permesso di lavorare dandomi un buon grado di libertà e flessibilità.

Grazie a *mamma* e a *papà*, che mi hanno permesso di dedicare quanto più tempo possibile all'università. In questo modo, ho avuto a disposizione tutto il tempo di cui avevo bisogno.

Grazie al mio amico *Francesco*, conosciuto in università, che mi ha sempre ascoltato pazientemente nei momenti più difficili. Spero che, dopo aver conseguito la laurea, lui possa fare il lavoro che desidera nel posto che preferisce, proprio come spero di fare anch'io.

Grazie a *me stesso*, perché ho affrontato alcuni momenti di forte indecisione sul futuro della mia vita. *Ma nonostante tutto, sono ancora qui.*