UNIVERSITÀ DI PISA

Dipartimento di Filologia, Letteratura e Linguistica

Corso di Laurea in *Informatica Umanistica*

Tesi di Laurea

Parental Control Tweet

Relatori:

Prof. Claudio Gallicchio

Prof. Alesandro Lenci

Presentata da:

Niccolò Calabrese

Anno Accademico 2022/23

A mia Nonna Rosanna

Indice

Sommario

Il progetto di tesi '**Parental Control Tweet**' consiste in un'estensione disponibile per il Browser Web (motore di ricerca) di Google Chrome. L'estensione è pensata per aggiungere una funzionalità al famoso Social Network '**Twitter**', arricchendolo di una sorta di controllo parentale su tutti i Tweet che occupano la bacheca. Il controllo avviene grazie a un modello di Machine Learning reso disponibile dall'Hub di TensorFlow [1] (repository di modelli di machine learning addestrati e implementabili ovunque) ed integrato nel progetto grazie ad un servizio API sviluppato in NodeJS [2].

Si tratta nello specifico, di un modello che analizza e classifica la tossicità presente nei testi che gli vengono passati dall'estensione. Il risultato ottenuto consiste in un sistema di offuscamento del testo, nel caso in cui si incontrino dei Tweet considerati inappropriati; è possibile anche disoffuscare il Tweet nel caso si voglia scoprire il contenuto nonostante la classificazione tossica, cliccando l'icona (occhio di visibilità) che gestisce la classe di offuscamento.

Capitolo 1

Introduzione

Alla base del progetto di tesi, vi è in primis un importante interessamento per il tema dei contenuti tossici presenti nel web. I contenuti inappropriati sono di ogni tipologia e sono frequenti in tutti gli angoli del web, ma in questo caso si parla di tossicità dei testi, all'interno di Social Network. La scelta di sviluppare un'estensione, è dettata anche da un interesse per quanto riguarda le nuove forme di tecnologia offerte da Google, tra le quali si trovano le **Estensioni Chrome**, sviluppabili e scaricabili per il browser web Google Chrome. L'affascinante possibilità di poter interagire direttamente con le applicazioni più illustri del pianeta, modificandole o aggiungendo loro funzionalità, è ciò che realmente spinge ad informarsi e studiare il campo in questione. Come si può evincere dal titolo **Parental Control Tweet**, il progetto riguarda un'estensione che aggiunge una funzionalità a Twitter [3], Social Network che offre un servizio di notizie e microblogging oramai famoso in tutto il mondo. Il servizio che offre l'estensione, può essere inteso come un controllo parentale sui Tweet, dato che l'utente ha la possibilità di oscurare i testi dei Tweet nel caso in cui quest'ultimi vengono classificati come tossici.

L'obiettivo iniziale ed introduttivo della tesi di laurea, è quello di fornire una conoscenza generica di tutte le funzionalità che compongono il progetto, così da entrare poi nel vivo di quello che è stato lo sviluppo e renderlo quanto più possibile, e comprensibile al lettore. Per quanto riguarda gli obiettivi finali, si tratta in primis di tutelare i bambini di oggi, che sono troppo spesso a contatto con i nostri smartphone, ed è appurato che la lettura di post offensivi o scurrili presenti sui Social, possa in qualche modo avere effetti negativi su di loro. In un

secondo momento, l'obiettivo è anche di avvicinare al mondo tecnologico, chi crede di esserne lontano anni luce, ma che in realtà è confinante con esso, o addirittura ne fa già parte.

La tesi è articolata in cinque capitoli: nel capitolo corrente viene fornita un'introduzione sulle tematiche affrontate e sulle tecnologie utilizzate nel progetto. Vengono esposti inoltre gli obiettivi posti ad inizio percorso, che hanno in qualche modo guidato le scelte riguardo le tecnologie utilizzate. Il secondo capitolo ha una duplice natura: innanzitutto si entra nel dettaglio delle motivazioni che hanno reso incentivante, l'intero svolgimento del progetto; dopodichè viene fatta un'introduzione generica sul Deep e Machine Learning (punti importanti del progetto), e vengono presentate le tre tecnologie Google Extension, NodeJS, e TensorFlow, colonne portanti su cui si regge l'intero sviluppo. Il terzo capitolo si concentra sulla descrizione del software che compone il progetto, quindi delle due parti che insieme permettono il funzionamento dell'estensione. Sono state quindi analizzate anche le parti più complesse dello sviluppo, attingendo a veri e propri pezzi di codice in modo da permettere una più facile comprensione. Nel quarto capitolo il focus si sposta sul funzionamento e sulle modalità di utilizzo del prodotto. Il capitolo svolge il compito di guida per l'utente, in cui vengono analizzate e spiegate le fasi necessarie per rendere il prodotto attivo e funzionante, tramite l'uso di screenshot che mostrano visivamente ciò che succede ad ogni singolo passaggio. Nel quinto capitolo, infine, si procede a presentare le conclusioni della tesi, sottolineando quali erano gli obiettivi iniziali e i risultati ottenuti alla fine del percorso. Inoltre, vengono delineati gli obiettivi futuri per il prodotto, presentando alcune 'features' aggiuntive che potrebbero rendere l'estensione più articolata e completa.

Capitolo 2

Background

In questo capitolo vengono trattate le motivazioni che hanno dettato la scelta delle tematiche progettuali ed i punti cardine che compongono tutte le conoscenze di background, necessarie per la comprensione dei capitoli successivi.

Nella sezione 2.1, vengono esposte quelle che sono le motivazioni che hanno dettato la scelta di un progetto che limitasse il linguaggio tossico sul web.

Nella sezione 2.2 si analizza il Machine Learning e il Deep Learning, esponendo le loro caratteristiche e differenze sostanziali, e viene introdotto il modello per il riconoscimento di testi tossici. Verso la fine della sezione si entra nel dettaglio degli Encoder di cui fa uso il modello.

L'ultima sezione del capitolo contiene tre ulteriori sottosezioni, che introducono le tecnologie principali utilizzate nel progetto: la funzione di Chrome 'Google Extension', NodeJS, e TensorFlow. Vedremo nel dettaglio le tecnologie nella sezione 2.3.

2.1 Motivazioni

Viviamo in un momento storico in cui la nostra quotidianità è completamente sommersa dai social media e dalla tecnologia.

Le persone di età avanzata si stanno adeguando al velocissimo sviluppo tecnologico, tutti oramai utilizzano le più famose applicazioni di messaggistica, e in pochi leggono ancora il

giornale cartaceo, ogni notizia di cui abbiamo bisogno è proprio lì, sul web. I bambini di questa generazione non hanno più nè la pazienza nè la necessità di aspettare l'orario in cui viene mandato in onda il loro cartone preferito, data la possibilità di guardare l'intera serie sui loro tablet o smartphone. Questo sviluppo rapido ed incontrastato colpisce anche gli adolescenti, e ciò che più fa specie è l'assiduità con la quale i giovani sbirciano i loro smartphone per consultare i social a cui sono iscritti e su cui passano gran parte del loro tempo libero; non necessariamente si tratta di una perdita di tempo, dato che negli ultimi anni sono nati molti mestieri legati a Social Media come Tik Tok, Instagram e molti altri.

In questo contesto in cui necessariamente tutti dobbiamo direzionarsi verso un'ottica più avanguardistica, in questo progetto si è deciso di trattare il tema dei contenuti cosidetti 'tossici', molto frequenti nel mondo del web. La tossicità nel web può avere molte sfaccettature, a partire da come questa possa diventare in qualche modo una dipendenza 'tossica', o a come il web sia popolato da materiale tossico (foto, video, o linguaggi); nel progetto di tesi si è scelto di trattare proprio il fenomeno della tossicità testuale, data la frequenza con la quale navigando sui social troviamo linguaggi che potrebbero facilmente urtare la sensibilità di molte persone, e soprattutto bambini. L'estensione sviluppata offre l'opzione di limitare i contenuti tossici, così da rendere un'idea del vantaggio che si può trarre da questa funzionalità.

2.2 Deep e Machine Learning

Introduzione generale

Nell'applicativo presentato, l'elemento che caratterizza l'estensione Chrome, è proprio la componente di **AI (Artificial Intelligence)**, rappresentata dal modello di Deep Learning utilizzato per l'identificazione di testo tossico.

Per un'introduzione esplicativa di ciò che rappresenta il Deep Learning (DL), dobbiamo necessariamente andare a retroso e sottolineare che, insieme al Machine Learning (ML), rappresentano i due concetti principali che rendono possibile l'AI. I due termini sono spesso

confusi, ma descrivono due metodi fondamentalmente diversi, ognuno con le proprie aree di applicazione.

Per quanto riguarda l'AI, sono state date moltissime definizioni, ma comunemente è considerata la scienza e l'ingegneria della creazione di macchine intelligenti, in particolare di programmi per computer intelligenti; come si intuisce dal nome, il tutto è correlato al compito simile di utilizzare computer per comprendere l'intelligenza umana, ma l'AI non deve limitarsi a metodi biologicamente osservabili. Sia il Machine Learning che il Deep Learning sono quindi sottosezioni dell'AI ed entrambi gli approcci fanno sì che i computer siano in grado di prendere decisioni intelligenti. [4]

Differenze tra ML e DL

Cerchiamo quindi di scindere i due diversi significati analizzandoli uno ad uno; quando si parla di **Machine Learning**, si intendono generalmente algoritmi che apprendono dai dati una data relazione desiderata di input-output e quindi applicano ciò che hanno appreso per prendere decisioni informate. Un semplice esempio di algoritmo di ML è rappresentato dai servizi di musica in streaming su richiesta. Affinché un servizio di questo tipo possa decidere quali nuovi brani o artisti consigliare a un ascoltatore, gli algoritmi di ML associano le preferenze dell'ascoltatore a quelle di altri ascoltatori che hanno gusti musicali simili. Questa tecnica, che viene spesso presentata semplicemente come AI, viene utilizzata in molti servizi che offrono consigli automatizzati.

Il Machine Learning sta alla base di molti tipi di attività automatizzate nei settori più disparati, dalle società che proteggono i dati dal malware, ai professionisti della finanza che desiderano scegliere le migliori operazioni di borsa. Quando si dice che una macchina è in grado di "apprendere in modo automatico", significa che svolge una funzione con i dati a sua disposizione e si migliora progressivamente nel tempo, e questo viene permesso dai calcoli matematici complessi tramite i quali è costruita. Tornando alle differenze che ci sono tra i due concetti, il **Deep Learning** è un sottoinsieme del Machine Learning, ma più specificamente, il DL è considerato un'evoluzione del ML.

Sebbene i modelli base di ML siano in grado di migliorare progressivamente le loro funzioni, hanno comunque bisogno dell'intervento umano; se un algoritmo di intelligenza artificiale

restituisce una previsione non accurata, è il programmatore di turno che deve intervenire e apportare modifiche affinchè quest'ultima diventi più attendibile. Per quanto riguarda la differenza sostanziale rispetto al Deep Learning, sta nel fatto che quest'ultimo non è propriamente un sottoinsieme, ma riguarda un modo di realizzare un algoritmo di ML, in cui l'input passa attraverso una pipeline computazionale che rappresenta trasformazioni apprese dai dati, e che semplificano progressivamente la risoluzione del problema iniziale. Questo tipo modelli viene solitamente implementato tramite reti neurali artificiali. I modelli di apprendimento profondo vengono progettati per analizzare continuamente i dati con una struttura logica simile al quella utilizzata dagli esseri umani per trarre conclusioni. Per raggiungere questo obiettivo, le applicazioni di Deep Learning si avvalgono di una struttura di algoritmi a più livelli chiamata **rete neurale artificiale**. Il design di una rete neurale artificiale si ispira alla rete neurale biologica del cervello umano, portando a un processo di apprendimento molto più efficace di quello dei modelli standard di apprendimento automatico.

È anche però difficile fare in modo che un modello di apprendimento profondo non tragga conclusioni errate; come altri esempi di intelligenza artificiale, ha bisogno di molto allenamento per rendere corretti i processi di apprendimento. Ma quando funziona come previsto, viene spesso considerano una meraviglia scientifica, quindi fondamento della vera intelligenza artificiale [5].

Il modello Toxicity

Per il progetto di tesi, è stato utilizzato il modello ML 'Toxicity' messo a disposizione nella sezione Tensorflow Hub [1], che è composto da due modelli di codifica: uno fa uso dell'architettura del Transformer [1], mentre l'altro è formulato come 'Deep Averaging Network' (DAN) [1]. Tutti e due i modelli sono implementati in TensorFlow [6], e prendono come input stringhe in inglese e producono come output una rappresentazione dell'incorporamento di dimensione fissata della stringa. I due codificatori hanno obiettivi di progettazione diversi. Il primo, basato sull'architettura del Transformer punta a un'elevata precisione, ma con una maggiore complessità e un maggiore consumo delle risorse [7]. Il secondo mira invece a un'inferenza efficiente con una precisione leggermente ridotta. Vediamo un esempio di funzionamento nell'immagine 2.1.

Transformer

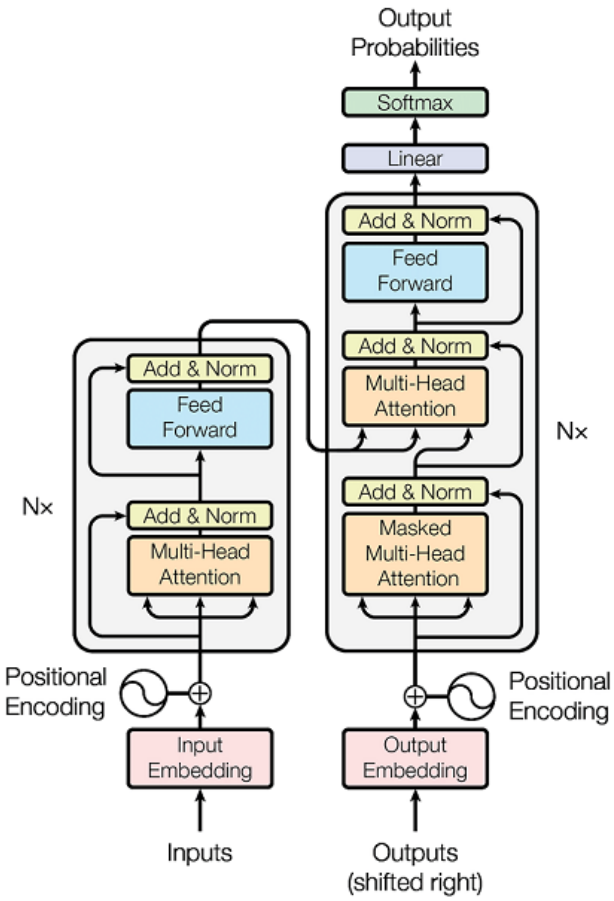

Figura 2.1: Schema dell'architettura dell'encoder Transformer, dal libro 'Machine Learning Mastery With Python' [8].

Il modello di codifica delle frasi basato sul **Transformer** costruisce incorporamenti di frasi usando il sottografico di codifica dello stesso [1]. Questo sottografico usa l'attenzione per calcolare rappresentazioni sensibili al contesto di parole in una frase che tengono conto sia dell'ordine che dell'identità di tutte le altre parole. Le rappresentazioni di parole sensibili al contesto vengono convertite in un vettore di codifica di frasi a lunghezza fissa, dopodichè si calcola la somma per elemento delle rappresentazioni, in ciascuna posizione della parola. Il codificatore prende come input una stringa minuscola tokenizzata Penn Treebank [9], e genera un vettore 512 dimensionale come incorporamento della frase. Il modello di codifica è

progettato per essere il più generico possibile, e questo grazie all'apprendimento multi-task per cui un singolo modello di codifica viene utilizzato per alimentare più attività a valle. Le attività supportate includono: un'attività simile a (Kiros et al., 2015) per l'apprendimento senza supervisione da un testo in esecuzione arbitrario; un'attività di input-risposta conversazionale per l'inclusione di dati di conversazione analizzati [1]; e compiti di classificazione per la formazione sui dati supervisionati. Il compito SkipThought sostituisce un modello di Long Short-Term Memory (LSTM) utilizzato nella formulazione originale con un modello basato sull'architettura Transformer.

L'encoder in questione basato sul Transformer, raggiunge le migliori prestazioni complessive delle attività di trasferimento [7]. Tuttavia, questo va a scapito del tempo di calcolo e dell'utilizzo della memoria. Vediamo un esempio di funzionamento nell'immagine 2.2.

Deep Averaging Network (DAN)

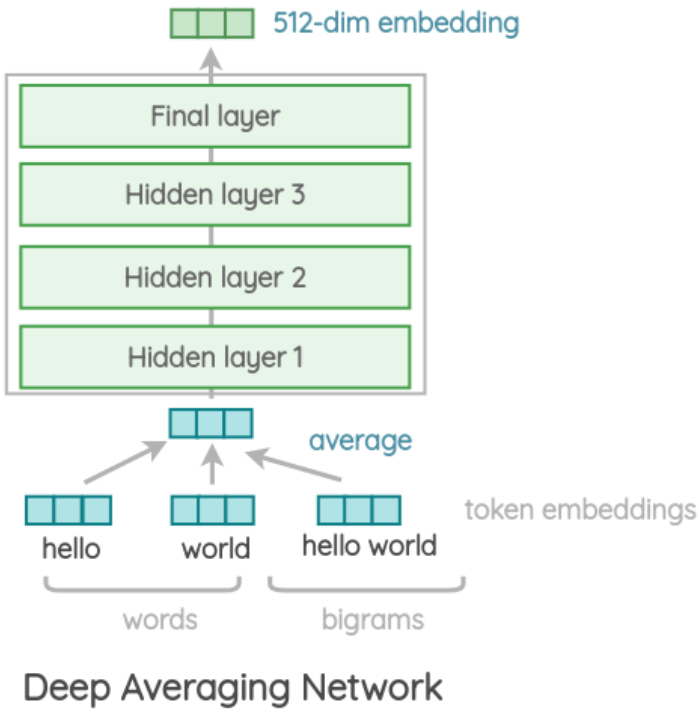

Figura 2.2: Schema dell'architettura dell'encoder DAN, dall'articolo 'Universal Sentence Encoder Visually Explained' [10].

Il secondo modello di codifica utilizza la Deep Averaging Network (DAN) [1] in cui vengono inseriti gli incorporamenti per le parole, mentre i bi-grammi vengono prima mediati insieme e poi passati attraverso una deep neural network (DNN) per produrre incorporamenti di frasi. Simile all'encoder Transformer, **l'encoder DAN** prende come input una stringa tokenizzata PTB minuscola e genera un'incorporamento di frasi a 512 dimensioni. L'encoder DAN è addestrato in modo simile all'encoder basato sul Transformer, utilizza anch'esso l'apprendimento multitasking in base al quale è presente un singolo codificatore DAN utilizzato per fornire incorporamenti di frasi per più compiti a valle. Il vantaggio principale dell'encoder DAN è il tempo di calcolo lineare nella lunghezza della sequenza di input. I nostri risultati dimostrano che i DAN ottengono ottime prestazioni di base nelle attività di classificazione del testo.

I dati di addestramento non supervisionati per i modelli di codifica delle frasi sono tratti da una varie fonti web. Le fonti principali sono in primis Wikipedia, notizie web generiche, pagine web di domanda-risposta e forum di discussione. [7]

2.3 Tecnologie Software

Questa sezione contiene tre ulteriori sottosezioni, che introducono le tecnologie principali utilizzate nel progetto. Troviamo infatti nella sottosezione 2.3.1 una descrizione della guida messa a disposizione da google [11] per creare un'estensione generica. Nella sottosezione 2.3.2 si trova una descrizione della tecnologia NodeJS [12], ed un introduzione di come si possa creare un servizio web back-end con questa tecnologia, capace di funzionare da API per un modello di Machine Learning [2]. Nella 2.3.3 viene invece presentato TensorFlow, tecnologia che possiede un Hub di modelli di ML e DL, tra i quali è presente il modello utilizzato nel progetto. [6]

2.3.1 Google Extension

Per la realizzazione del progetto, si è scelto di costruire un'estensione di Chrome da rendere open-source una volta giunta al termine, e durante la fase di sviluppo è stata necessaria la

guida che Chrome mette a disposizione nella pagina ‘Chrome Developers‘ del proprio sito. In questa sezione saranno introdotti i punti chiave delle estensioni Chrome, avvalendosi di porzioni di codice appartenenti alla guida presente online. [11]

Le estensioni di Google Chrome sono estensioni del browser che modificano Google Chrome e sono scaricabili tramite il Chrome Web Store. Dal febbraio 2010, più di 2200 estensioni sono state pubblicate dai rispettivi sviluppatori dato che tutti gli utenti con un account Google hanno la possibilità di pubblicarne di nuove, dato che Google Chrome permette tramite le API agli sviluppatori di programmare le proprie estensioni. Le estensioni sono costituite da componenti diversi, ma coesi. I componenti possono includere script in background, script di contenuto, una pagina delle opzioni, elementi dell’interfaccia utente e vari file di logica. I componenti di un’estensione vengono creati con tecnologie di sviluppo web: HTML, CSS e JavaScript. I componenti di un’estensione dipenderanno dalla sua funzionalità e potrebbero non richiedere tutte le opzioni; per quanto riguarda il progetto in questione, c’è stato bisogno di tutte le componenti sopracitate, con l’aggiunta di ulteriori tecnologie come Tensorflow e NodeJS.

Per introdurre gli utenti nello sviluppo delle estensioni, Google Chrome mette a disposizione una guida per costruire una semplice estensione, che utilizza molti dei componenti della piattaforma, in modo da fornire una dimostrazione introduttiva delle loro relazioni.

Chrome mette a disposizione tre piccole guide per creare delle semplici estensioni che trattano rispettivamente delle parti fondamentali, in modo da introdurre al meglio l’utente in questo tipo particolare di sviluppo. Il tutorial è diviso nelle tre seguenti sezioni: **Reading time**, **Focus Mode** e **Tabs Manager**; ci sarà anche una sezione quattro, in cui inserirò i passaggi per caricare l’estensione, in quanto sarebbe ripetitivo riportarli in tutte e tre le sezioni.

1. La sezione **Reading-time**, mostra all’utente come creare un’estensione che aggiunge il tempo di lettura previsto a qualsiasi estensione di Chrome e alla pagina della documentazione del Chrome Web Store. Come passo preliminare, l’utente deve creare una nuova directory chiamata ‘reading-time’ per contenere i file dell‘estensione.

    - Il primo passaggio prevede la creazione del file ‘manifest.json’, unico file che de-

ve contenere necessariamente ogni estensione che utilizza le API WebExtension. Utilizzando questo file, si può specificare i metadati di base sulla tua estensione, come il nome e la versione, si può anche specificare aspetti della funzionalità dell'estensione, come ad esempio: script in background, script di contenuto, azioni del browser, una pagina delle opzioni, elementi dell'interfaccia utente e vari file di logica. Nel listato 2.1 si trova un esempio di inizializzazione del file Manifest:

```
{
  "manifest_version": 3,
  "name": "Reading time",
  "version": "1.0",
  "description": "Add the reading time to extension",
}
```

Listing 2.1: Inizializzazione del file 'manifest.json'.

- Il prossimo passaggio, spiega agli utenti come inserire le icone della propria estensione, e sebbene le icone siano facoltative durante lo sviluppo, sono obbligatorie se si prevede di distribuire l'estensione sul Chrome Web Store. L'utente deve creare una directory 'images' e inserire al suo interno quattro icone, come mostrato nel listato 2.2:

  (a) 16x16 favicon sulle pagine dell'estensione e sull'icona del menu contestuale.

  (b) 32x32 dimensione spesso richiesta dai computer Windows.

  (c) 48x48 viene visualizzata nella pagina 'Estensioni'.

  (d) 128x128 viene visualizzata durante l'installazione e nel Chrome Web Store.

```
{
  ...
  "icons": {
    "16": "images/icon-16.png",
    "32": "images/icon-32.png",
    "48": "images/icon-48.png",
    "128": "images/icon-128.png"
  }
```

```
    ...
}
```

Listing 2.2: Aggiunta delle icone nel file 'manifest.json'.

- Per il prossimo step, l'utente dovrà aggiungere i 'Content script' (script di contenuto), cioè script che possono leggere e/o modificare il contenuto di una pagina. Gli script di contenuto vivono in un ambiente isolato, il che significa che possono apportare modifiche al loro ambiente JavaScript senza entrare in conflitto con la loro pagina host o gli script di contenuto di altre estensioni.

  L'utente dovrà registrare nel Manifest la voce 'contentscripts', che al suo può contenere vari tipi di campi (come 'matches', 'css' ecc.). Nel caso seguente, i due campi di cui necessita l'utente sono il campo 'js' che specificherà i file da trattare come script, e il campo 'matches' che rende possibile la corrispondenza tra l'estensione e una url (esegue appunto il match).

  I 'matches' consentono al browser di identificare in quali siti inserire gli script di contenuto (nel caso della documentazione verranno inserite la url che punta alla guida delle estensioni di Chrome e quella della documentazione del Chrome Web Store). Il codice da aggiungere al Manifest è mostrato nel listato 2.3:

```
{
    ...
    "content_scripts": [
        {
          "js": ["scripts/content.js"],
          "matches": [
            "https://developer.chrome.com/docs/extensions/*",
            "https://developer.chrome.com/docs/webstore/*"
          ]
        }
      ]
}
```

Listing 2.3: Aggiunta dei Matches (schemi di corrispondenza) al file 'manifest.json'.

- In quest'ultimo step, l'obiettivo dell'utente sarà quello di calcolare ed inserire il tempo di lettura. Gli script di contenuto possono utilizzare il Document Object Model (DOM) standard per leggere e modificare il contenuto di una pagina. Con il quarto ed ultimo passaggio della sezione, l'estensione potrà verificare prima se la pagina contiene l'elemento 'article' , dopodichè conterà tutte le parole all'interno di questo elemento e creerà un paragrafo che mostra il tempo di lettura totale. A questo punto l'utente dovrà creare un file chiamato 'content.js' e inserirlo all'interno di una directory chiamata 'scripts' a cui punterà il campo 'js' creato nel passaggio precedente.

  Scrivendo nel file 'content.js' il codice inserito nel listato 2.4, si otterrà che se esiste un elemento 'article' nel DOM della pagina che l'utente sta visitando, verrà automaticamente calcolato il tempo di lettura (Reading time):

```
1  // 'document.querySelector' may return null,
2  // if the selector doesn't match anything.
3  if (article) {
4    const text = article.textContent;
5    const wordMatchRegExp = /[^\s]+/g; // Regular expression
6    const words = text.matchAll(wordMatchRegExp);
7    // matchAll returns an iterator, convert to array to get word
       count
8    const wordCount = [...words].length;
9    const readingTime = Math.round(wordCount / 200);
10   const badge = document.createElement("p");
11   // Use the same styling as the publish information in an article
       's header
12   badge.classList.add("color-secondary-text", "type--caption");
13   badge.textContent = `        ${readingTime} min read`;
14
15   // Support for API reference docs
16   const heading = article.querySelector("h1");
17   // Support for article docs with date
18   const date = article.querySelector("time")?.parentNode;
19
```

```
20    (date ?? heading).insertAdjacentElement("afterend", badge);
21  }
```

Listing 2.4: Calcolo del tempo di lettura di un articolo.

2. Nella sezione **Focus Mode** viene creata una nuova estensione che semplifica lo stile dell'estensione di Chrome e delle pagine della documentazione del Web Store in modo che siano più facili da leggere. Prima di entrare nel dettaglio dei passaggi, l'utente deve creare una nuova directory chiamata 'focus-mode' per contenere i file dell'estensione.

   - Il primo passaggio è il medesimo del punto 1, infatti l'utente deve creare il file 'manifest.json' con i rispettivi dati iniziali e la cartella 'images' che conterrà le quattro icone previste per l'estensione.
   - Nel secondo passaggio viene introdotto e registrato il 'Service worker'. Le estensioni sono programmi basati su eventi utilizzati per modificare o migliorare l'esperienza di navigazione di Chrome. Gli eventi sono attivatori del browser, come il passaggio a una nuova pagina, la rimozione di un segnalibro o la chiusura di una scheda. Le estensioni monitorano questi eventi utilizzando degli script in background nel ruolo di **Service worker**, che quindi reagiscono con istruzioni specificate. Una volta caricato, il Service Worker di un'estensione generalmente continua a funzionare finché esegue un'azione, ad esempio chiamando un'API di Chrome o inviando una richiesta di rete.

     L'utente può quindi registrare il Service worker con il rispettivo file di script, aggiungendo nel manifest il codice presente nel listato 2.5:

```
1  {
2    ...
3    "background": {
4      "service_worker": "background.js"
5    },
6    ...
7  }
```

Listing 2.5: Registrazione del Service Worker e del file ad esso associato. 'manifest.json'.

Successivamente l'utente deve aggiungere il pezzo di codice del listato 2.6, nel file 'background.js':

```
chrome.runtime.onInstalled.addListener(() => {
    chrome.action.setBadgeText({
        text: "OFF",
    });
});
```

Listing 2.6: Viene impostato il testo del badge per l'azione del browser.

Il primo evento che l'operatore di servizio ascolterà è runtime.onInstalled(). Questo metodo consente all'estensione di impostare uno stato iniziale o completare alcune attività durante l'installazione.

- Nel terzo passaggio deve essere abilitata l'azione dell'estensione. Quindi, ogni volta che l'utente fa clic sull'azione dell'estensione, eseguirà del codice. L'utente deve aggiungere il codice del listato 2.7, per dichiarare l'azione di estensione nel manifest.json:

```
{
  ...
  "action": {
    "default_icon": {
      "16": "images/icon-16.png",
      "32": "images/icon-32.png",
      "48": "images/icon-48.png",
      "128": "images/icon-128.png"
    }
  },
  ...
}
```

Listing 2.7: Viene dichiarata l'API Action per controllare l'icona dell'estensione.

- Il quarto passaggio prevede il tracciamento della scheda corrente; aggiungendo il codice del listato 2.8, al file 'background.js', dopo che l'utente ha fatto clic sull'azione dell'estensione, l'estensione verificherà se l'URL corrisponde a una pagina

della documentazione. Successivamente, verificherà lo stato della scheda corrente e imposterà lo stato successivo.

```
chrome.action.onClicked.addListener(async (tab) => {
if (tab.url.startsWith(extensions) || tab.url.startsWith(webstore)
    ) {
// Retrieve the action badge to check if the extension is 'ON' or
    \\'OFF'
const prevState = await chrome.action.getBadgeText({ tabId: tab.id
     });
// Next state will always be the opposite
const nextState = prevState === 'ON' ? 'OFF' : 'ON'

// Set the action badge to the next state
await chrome.action.setBadgeText({
  tabId: tab.id,
  text: nextState,
});
```

Listing 2.8: Viene verificato lo stato della scheda corrente e settato il nuovo stato.

- Nel quinto passaggio l'utente ha la possibilità di cambiare il layout della pagina. Dopo la creazione del file .css, viene utilizzata l'API di scripting dichiarando nel manifest.json l'autorizzazione 'scripting' dopo la chiave 'permissions', e deve essere aggiunto al file il seguente codice del listato 2.9, che va a lavorare sul layout:

```
body > .scaffold > :is(top-nav, navigation-rail, side-nav, footer)
    ,
main > :not(:last-child),
main > :last-child > navigation-tree,
main .toc-container {
  display: none;
}

main > :last-child {
  margin-top: min(10vmax, 10rem);
  margin-bottom: min(10vmax, 10rem);
```

```
}
```

Listing 2.9: Viene aggiunto il CSS per la modifica del layout.

- Infine l'utente deve aggiungere al file 'background.js' il codice che vediamo nel listato 2.10, per far sì che il file css che è stato creato, venga iniettato nella pagina:

```
if (nextState === "ON") {
      // Insert the CSS file when the user turns the extension on
      await chrome.scripting.insertCSS({
        files: ["focus-mode.css"],
        target: { tabId: tab.id },
      });
    } else if (nextState === "OFF") {
      // Remove the CSS file when the user turns the extension off
      await chrome.scripting.removeCSS({
        files: ["focus-mode.css"],
        target: { tabId: tab.id },
      });
    }
  }
});
```

Listing 2.10: Viene aggiunto il CSS per la modifica del layout.

3. Nella sezione **Tabs Manager** l'utente avrà la possibilità di creare una appunto un Tabs Manager (gestore di schede), che grazie all'aggiunta di un semplice popup semplificherà l'organizzazione dell'estensione di Chrome e delle schede presenti nella documentazione del Chrome Web Store. Per iniziare, l'utente deve creare una nuova directory chiamata tabs-manager per contenere i file dell'estensione.

   - Come nelle altre sezioni, il primo passo sta nella creazione del file 'manifest.json' e della cartella 'images' in cui dovranno essere scaricate le quattro icone messe a disposizione.

   - In questo passaggio, l'utente avrà modo di affrontare per la prima volta, la creazione e la modellazione del popup. Un popup è simile a una pagina web con un'eccezione:

non può eseguire JavaScript inline. L'API Action controlla l'azione dell'estensione (icona della barra degli strumenti), e come visto nella sezione 2 quando l'utente fa clic sull'azione dell'estensione, verrà eseguito del codice o si aprirà un popup, come in questo caso. L'utente dovrà innanzitutto dichiarare il popup nel 'manifest.json', come nel listato 2.11:

```
{
  ...
  "action": {
    "default_popup": "popup.html"
  },
  ...
}
```

Listing 2.11: Viene dichiarato il Popup nel file 'manifest.json'.

Successivamente, dovrà creare un nuovo file denominato 'popup.html' ed inserirvi il codice del listato 2.12:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<meta http-equiv="X-UA-Compatible" content="IE=edge" />
<meta name="viewport" content="width=device-width, initial-scale
    =1.0" />
<link rel="stylesheet" href="./popup.css" />
</head>
<body>
<template id="li_template">
  <li>
    <a>
      <h3 class="title">Tab Title</h3>
      <p class="pathname">Tab Pathname</p>
    </a>
  </li>
</template>
```

```
18
19 <h1>Google Dev Docs</h1>
20 <button>Group Tabs</button>
21 <ul></ul>
22
23 <script src="./popup.js" type="module"></script>
24 </body>
25 </html>
```

Listing 2.12: Viene creato il file 'popup.html'.

Facoltativamente, l'utente può aggiungere un foglio di stile 'popup.css', come riportato nel listato 2.13:

```
1  body {
2    width: 20rem;
3  }
4
5  ul {
6    list-style-type: none;
7    padding-inline-start: 0;
8    margin: 1rem 0;
9  }
10
11 li {
12   padding: 0.25rem;
13 }
14 li:nth-child(odd) {
15   background: #80808030;
16 }
17 li:nth-child(even) {
18   background: #ffffff;
19 }
20
21 h3,
22 p {
23   margin: 0;
```

```
}
```

Listing 2.13: Viene aggiunto dello stile CSS nel file 'popup.css'.

- Nel terzo passaggio, l'utente ha la possibilità di interagire con la gestione delle schede grazie all'API Tabs, che consente a un'estensione di creare, interrogare, modificare e riorganizzare le schede nel browser.

  Molti metodi nell'API Tabs possono essere utilizzati senza richiedere alcuna autorizzazione. Tuttavia, abbiamo bisogno dell'accesso alle URL delle schede. Potremmo richiedere l'autorizzazione 'tabs', ma ciò darebbe accesso alle proprietà sensibili di tutte le schede. Poiché gestiamo solo le schede di un sito specifico, richiederemo autorizzazioni host limitate.

  Le autorizzazioni host limitate ci consentono di proteggere la privacy degli utenti concedendo autorizzazioni elevate a siti specifici. Ciò garantirà l'accesso alle proprietà title e URL richieste, oltre a funzionalità aggiuntive. Per completare la procedura l'utente deve aggiungere il codice del listato 2.14, al 'manifest.json':

```
{
  ...
  "host_permissions": [
    "https://developer.chrome.com/*"
  ],
  ...
}
```

Listing 2.14: Viene aggiunta l'autorizzazione di accesso alle proprietà 'title' e URL.

- Successivamente viene mostrato un metodo per interrogare le schede; si può infatti recuperare le schede da URL specifici utilizzando il metodo tabs.query(). L'utente deve creare un file 'popup.js' e aggiungere il codice presente nel listato 2.15:

```
const tabs = await chrome.tabs.query({
  url: [
    "https://developer.chrome.com/docs/webstore/*",
    "https://developer.chrome.com/docs/extensions/*",
```

```
  ],
});
...
```

Listing 2.15: Viene aggiunta l'autorizzazione di accesso alle proprietà 'title' e URL.

- Il prossimo step permette all'utente di concentrarsi su una scheda, l'estensione ordinerà i nomi delle schede (i titoli delle pagine HTML contenute) in ordine alfabetico. Quindi, quando si fa clic su un elemento dell'elenco, si concentrerà su quella scheda utilizzando il metodo tabs.update() e porterà la finestra in primo piano utilizzando windows.update(). Deve essere aggiunto al file 'popup.js' il codice del listato 2.16:

```
...
const collator = new Intl.Collator();
tabs.sort((a, b) => collator.compare(a.title, b.title));

const template = document.getElementById("li_template");
const elements = new Set();
for (const tab of tabs) {
  const element = template.content.firstElementChild.cloneNode(
    true);

  const title = tab.title.split("-")[0].trim();
  const pathname = new URL(tab.url).pathname.slice("/docs".length)
    ;

  element.querySelector(".title").textContent = title;
  element.querySelector(".pathname").textContent = pathname;
  element.querySelector("a").addEventListener("click", async () =>
    {
    // need to focus window as well as the active tab
    await chrome.tabs.update(tab.id, { active: true });
    await chrome.windows.update(tab.windowId, { focused: true });
  });

```

```
21    elements.add(element);
22  }
23  document.querySelector("ul").append(...elements);
24  ...
```

Listing 2.16: Vengono ordinate le schede e viene aggiunta la modalità di concentrarsi su una sola scheda.

- L'API TabGroups consente di assegnare un nome al gruppo e scegliere un colore di sfondo. L'utente dovrà aggiungere l'autorizzazione "tabGroups" al manifest alla voce 'permissions', e aggiornare il codice di 'popup.js' con quello del listato 2.17, per creare un pulsante che raggrupperà tutte le schede utilizzando tabs.group() e le sposterà nella finestra corrente:

```
1  ...
2  const button = document.querySelector("button");
3  button.addEventListener("click", async () => {
4    const tabIds = tabs.map(({ id }) => id);
5    const group = await chrome.tabs.group({ tabIds });
6    await chrome.tabGroups.update(group, { title: "DOCS" });
7  });
8  ...
```

Listing 2.17: Viene creato un pulsante che raggrupperà tutte le schede utilizzando tabs.group() e le sposterà nella finestra corrente.

4. Nella quarta sezione, affrontiamo il tema del caricamento dell'estensione, da svolgere ogni volta che viene terminata ogni sezione precedente.

   - Per prima cosa l'utente deve navigare la pagina 'chrome://extensions' in una nuova scheda, così da raggiungere la pagina delle estensioni. In alternativa, fai clic sul pulsante puzzle del menu Estensioni e seleziona Gestisci estensioni nella parte inferiore del menu.

   - Giunti nella pagina dedicata alle estensioni, l'utente dovrà inserire la modalità sviluppatore facendo clic sull'interruttore accanto alla modalità sviluppatore.

- L'ultimo passaggio prevede di cliccare sul pulsante 'Carica decompresso' e selezionare la directory dell'estensione, creata all'inizio di ogni sezione. L'estensione a questo punto sarà installata correttamente. Poiché nel manifest non sono state incluse icone di estensione, verrà creata un'icona generica per l'estensione.

2.3.2 NodeJS

Node.js è un ambiente di runtime open source e multipiattaforma basato sul motore JavaScript V8 di Chrome per l'esecuzione di codice JavaScript al di fuori di un browser. NodeJS non è un framework e non è un linguaggio di programmazione, fornisce un ambiente di runtime multipiattaforma I/O guidato da eventi (asincroni), per la creazione di applicazioni lato server altamente scalabili, utilizzando JavaScript. Molto spesso Node.js è utilizzato per creare servizi di back-end come API, app Web o app mobili [12]. In tal caso, essendo TensorFlow.js una libreria JavaScript, ci consente non solo di eseguirla nel browser ma anche come applicazione di back-end utilizzando appunto Node.js.

La funzione di cui si è fatto uso grazie a questa tecnologia, è quella di sviluppare una sorta di servizio web che permette l'integrazione del modello di rilevamento della tossicità. L'integrazione del modello grazie all'API NodeJS consentirà di effettuare una chiamata POST al servizio, per ottenere classificazioni di tossicità sui Tweet.

Una volta installato NodeJS, deve essere creata una directory di lavoro nella posizione che si preferisce. Quindi in quella posizione, si crea un file package.json, che è un documento contenente i metadati. Successivamente viene fatto il set del comando 'npm run serve' in modo che l'esecuzione verrà eseguita per il file index.js (file che contiene tutto il codice del servizio). Infine, si installa le dipendenze del progetto necessarie al funzionamento del servizio, quidni il modello di tossicità, così come altri framework e librerie come Express.js, un framework per creare applicazioni web.

Il prossimo passaggio è la vera e propria stesura del codice di 'index.js', il nucleo del servizio. Nel file Index viene quindi inserito nel codice il modello di tossicità e viene creato il server web con Express.js, utilizzando il metodo 'express.json()' che serve per analizzare le richieste JSON in arrivo. Questo permette di creare l'unico endpoint (che supporta una chiamata POST)

del servizio e la relativa funzione di gestione; all'interno di questa funzione, si utilizza il metodo 'classify()' del modello, con un argomento che è un elenco contenente la proprietà 'sentence' della Request. Subito dopo classify(), si utilizza una funzione di callback che prende in carico l'oggetto 'prediction' restituito da classify() per inviare la previsione richiesta all'utente.

Il JSON restituito da classify() ha una chiave 'prediction', con un elenco delle etichette (che sono i tipi di tossicità) e un'altra chiave 'results', i cui valori sono le probabilità che la frase abbia quell'etichetta e una chiave 'match', cioè un booleano che è 'true' se la probabilità della frase appartenente all'etichetta è sopra la soglia (quindi tossica).

A questo punto la struttura del servizio è funzionante, e permette di testare alcune frasi eseguendo il comando 'npm run serve', e controllare l'output risultante [2].

2.3.3 TensorFlow

TensorFlow è una libreria che di base può essere utilizzata per creare da zero modelli di DL e ML, o utilizzando librerie wrapper che semplificano il processo, basate su TensorFlow [6]. Creata dal team di Google Brain e inizialmente rilasciata al pubblico nel 2015, **TensorFlow** è una libreria open source per il calcolo numerico e l'apprendimento automatico su larga scala. TensorFlow raggruppa una serie di modelli e algoritmi di machine learning e deep learning e li rende utili tramite metafore programmatiche comuni. Utilizza Python o JavaScript per fornire una comoda API front-end per la creazione di applicazioni, mentre esegue tali applicazioni in C++ ad alte prestazioni. [13]

TensorFlow offre la possibilità di addestrare ed eseguire reti neurali profonde per la classificazione delle cifre scritte a mano, il riconoscimento di immagini, l'incorporamento di parole, le reti neurali ricorrenti, i modelli da sequenza a sequenza per la traduzione automatica, l'elaborazione del linguaggio naturale e simulazioni basate su PDE (equazione differenziale parziale). Soprattutto, TensorFlow supporta la previsione della produzione su larga scala, con gli stessi modelli utilizzati per la formazione. Un'altra importante caratteristica che lo rende tra gli altri competitor, uno tra i più famosi, è l'ampia libreria di modelli pre-addestrati messi a disposizione del client, che possono essere utilizzati nei propri progetti. Il modello di rilevamento della tossicità è appunto un modello pre-addestrato che rileva sei tipi di contenuti testuali tossici,

più una classe di tossicità complessiva, che accetta un booleano (basato sulle precedenti 6).[1] I sei tipi sono attacco di identità, insulto, oscenità, grave tossicità, sessualmente esplicito e minaccia. Ad esempio, secondo il modello, la frase ”fai schifo” è un insulto e nel risultato finale sarà una frase tossica. [2]

Features aggiuntive

TensorFlow consente agli sviluppatori di creare grafici del flusso di dati, strutture che descrivono come i dati si spostano attraverso un grafico o una serie di nodi di elaborazione. Ogni nodo nel grafico rappresenta un'operazione matematica e ogni connessione o spigolo tra i nodi è una matrice di dati multidimensionale, o tensore. Le applicazioni TensorFlow possono essere eseguite sulla maggior parte di qualsiasi device : una macchina locale, un cluster nel cloud, dispositivi iOS e Android, CPU o GPU.

TensorFlow 2.0, rilasciato nell'ottobre 2019, ha rinnovato il framework in molti modi in base al feedback degli utenti, per renderlo più semplice da utilizzare (ad esempio, utilizzando l'API Keras relativamente semplice per l'addestramento del modello) e più performante. La formazione distribuita è più facile da eseguire grazie a una nuova API e il supporto per TensorFlow Lite consente di distribuire modelli su una maggiore varietà di piattaforme.[14] Tuttavia, il codice scritto per le versioni precedenti di TensorFlow deve essere riscritto, a volte solo leggermente, a volte in modo significativo, per sfruttare al massimo le nuove funzionalità di TensorFlow 2.0. [13]

TensorFlow con Python

TensorFlow fornisce tutte queste funzionalità tramite il linguaggio Python. Python è facile da imparare e fornisce modi convenienti per esprimere come le astrazioni di alto livello possono essere accoppiate insieme. TensorFlow è supportato su Python nelle versioni dalla 3.7 alla 3.10 e, sebbene possa funzionare su versioni precedenti di Python, non è garantito che lo faccia.

I nodi e i tensori in TensorFlow sono oggetti Python e le applicazioni TensorFlow sono esse stesse applicazioni Python. Le effettive operazioni matematiche, tuttavia, non vengono eseguite in Python. Le librerie di trasformazioni disponibili tramite TensorFlow sono scritte come

binari C++ ad alte prestazioni. Python dirige semplicemente il traffico tra i pezzi e fornisce astrazioni di programmazione di alto livello per collegarli insieme.

Il lavoro di alto livello in TensorFlow, la creazione di nodi e livelli e il loro collegamento tra loro, utilizza la libreria Keras . L'API Keras è apparentemente semplice; un modello base con tre livelli può essere definito in meno di 10 righe di codice e il codice di addestramento per lo stesso richiede solo poche righe di codice in più. [13]

TensorFlow con Javascript

Python è il linguaggio più popolare per lavorare con TensorFlow e l'apprendimento automatico in generale. Ma anche JavaScript si sta dimostrando un linguaggio di prima classe per TensorFlow e uno degli enormi vantaggi di JavaScript è che funziona ovunque ci sia un browser web. TensorFlow.js, come viene chiamata la libreria JavaScript TensorFlow, utilizza l'API WebGL per accelerare i calcoli tramite qualsiasi GPU disponibile nel sistema.[6] È anche possibile utilizzare un back-end WebAssembly per l'esecuzione ed è più veloce del normale back-end JavaScript se si esegue solo su una CPU. I modelli predefiniti consentono di iniziare a lavorare con progetti semplici e avere un'idea di come funziona l'API. [13]

Installazione di TensorFlow

È possibile seguire le istruzioni per il download e l'installazione sul sito Web di TensorFlow. L'installazione è probabilmente la più semplice tramite PyPI e le istruzioni specifiche del comando pip da utilizzare per la tua piattaforma Linux, o Mac OS X, si trovano nella pagina Web di download e configurazione. Ci sono anche 'virtualenv' e immagini Docker che si può utilizzare, ma nel caso più semplice, si deve solo inserire il codice del listato 2.18, nella riga di comando:

```
pip install tensorflow
```

Listing 2.18: Comando per installare TensorFlow da linea di comando.

Un'eccezione avverrebbe installandolo sul nuovo Mac con una CPU Apple Silicon. Il nome del pacchetto per questa specifica architettura è tensorflow-macos, e il comando da utilizzare si trova nel listato 2.19:

```
pip install tensorflow-macos
```

Listing 2.19: Comando per installare TensorFlow con una CPU Apple Silicon.

Capitolo 3

Realizzazione Software

In questo capitolo sarà spiegato il software, quindi il codice che compone l'estensione. Questo è costituito da due parti distinte, la parte dedicata allo sviluppo dell'estensione, e quella dedicata al servizio web in NodeJS che rappresenta l'API tramite la quale l'estensione comunica con il modello passandogli i Tweet da controllare.

Estensione Chrome

Andando in ordine di sviluppo, la prima parte costituita dall'estensione è stata necessaria per la costruzione del servizio API grazie al quale il modello analizzerà i testi dei Tweet; pertanto sarà analizzata prima l'estensione.

Sviluppo del file 'manifest.json'

Per lo sviluppo iniziale, è stato necessario inizializzare l'estensione tramite la creazione del file 'manifest.json', che permette di specificare i metadati di base, come il nome, la versione, e vari aspetti della funzionalità dell'estensione. Come prima cosa, vengono inizializzati i metadati iniziali come nell'esempio del listato 3.1.

```
{
  "name": "Parental Control Tweet",
  "action": {},
  "manifest_version": 3,
  "version": "0.1",
```

```
6   "description": "Estensione che permette l'oscuramento di testi tossici,
      durante la navigazione di Twitter",
7 }
```

Listing 3.1: Dichiarati i metadati di base dell'estensione.

Una volta dichiarati i metadati, c'è stato bisogno di ottenere i permessi per effettuare alcune azioni all'interno dell'estensione, arricchendo delle chiavi che vediamo nel listato 3.2, il file 'manifest.json'.

```
1 "permissions": [
2     "storage",
3     "activeTab",
4     "scripting",
5     "debugger"
6   ]
```

Listing 3.2: Dichiarati i permessi necessari per le funzioni da sviluppare.

Andando in ordine, saranno spiegati i significati delle chiavi utilizzate per l'estensione;

- l'API **chrome.storage**, viene utilizzata per archiviare, recuperare e tenere traccia delle modifiche ai dati utente.
- Per quanto riguarda l'autorizzazione **activeTab**, questa fornisce a un'estensione l'accesso temporaneo alla scheda attualmente attiva quando l'utente richiama l'estensione. L'accesso alla scheda dura finché l'utente si trova su quella pagina e viene revocato quando l'utente esce o chiude la scheda.
- L'API **chrome.scripting** è utilizzata per iniettare JavaScript e CSS nei siti Web. Questo è simile a quello si può realizzare con i Content Scripts, ma usando l'API chrome.scripting, le estensioni possono prendere decisioni in fase di runtime.
- Infine l'API **chrome.debugger** funge da trasporto alternativo per il protocollo di debug remoto di Chrome. Viene utilizzato per collegarsi a una o più schede per strumentare l'interazione di rete, eseguire il debug di JavaScript, mutare DOM o CSS.

Nel listato 3.3, è stata aggiunta la chiave **Content Scripts** (script di contenuto), cioè file che vengono eseguiti nel contesto delle pagine Web. Utilizzando il Document Object Model (DOM) standard, sono in grado di leggere i dettagli delle pagine Web visitate dal browser, apportare modifiche e trasmettere informazioni all'interno principale.

```
"content_scripts": [
    {
      "run_at": "document_idle",
      "matches": [
        "https://*.twitter.com/*",
        "*://*/*"
      ],
      "js": [
        "content.js"
      ]
    }
  ]
```

Listing 3.3: Dichiarata la chiave che permette lo script di contenuto.

In questo pezzo di codice, sono stati aggiunti il campo 'js' che contiene il file da iniettare nelle pagine, il campo 'matches' che specifica in quali pagine verrà inserito questo script di contenuto, e possiamo vedere come sia presente il dominio appartenente a **Twitter**. Infine, Il campo 'run_at', che ha come compito quello di controllare quando i file JavaScript vengono inseriti nella pagina web, e gli script in esecuzione su 'document_idle' vengono immediatamente eseguiti dopo il completamento del DOM. La scelta di quest'ultimo deriva dal fatto che abbiamo la necessità di catturare i testi dei post presenti (vedremo successivamente la funzione), non appena si carica la pagina Twitter, così da controllarne l'eventuale tossicità.

Sviluppo del file 'content.js'

A questo punto, una volta che il file Manifest ha ottenuto tutti i permessi necessari, è stato possibile sviluppare il file che contiene tutto il funzionamento dell'estensione, il file di script 'content.js'. Questo file è composto da varie funzioni che permettono il funzionamento dell'estensione; la funzione **'parental_twitting_dom_loaded()'**, stampata nel listato 3.4, ha

il compito di catturare l'elemento 'section' del DOM che contiene tutti i Tweet testuali della pagina.

Funzione 'parental_twitting_dom_loaded()'

```
1   const observer = new MutationObserver(list => {
2     obfuscate_text();
3 });
4 const parental_twitting = setInterval(parental_twitting_dom_loaded, 300);
    // chiama la funzione parental_twitting_dom_loaded ogni 3 decimi di
    secondo fino a che non viene cancellato
5
6 function parental_twitting_dom_loaded() {
7     var tweet_container = document.querySelector('section.css-1dbjc4n[role
    ="region"]'); // recupera l'elemento section che contiene i tweet
8     if (tweet_container) { // se la pagina contiene gia l'elemento section
    dei tweet
9         clearInterval(parental_twitting);// cancello la 'l' che controlla
    la presenza del dom (e del tag section)
10
11         obfuscate_text();// chiamata alla funzione che recupera i tweet
12         observer.observe(document.querySelector('section.css-1dbjc4n[role="
    region"]'), { // observe controlla se dentro a section sono arrivati
    nuovi tweet (se ci sono cambiamenti dentro alla section chiama
    obfuscate_text che gli abbiamo detto alla riga 4)
13             attributes: false,
14             childList: true,// se dentro alla section cambia la lista dei
    figli
15             subtree: true// se dentro alla section cambia l'alberatura
16         });
17     }
18 }
19 }
```

Listing 3.4: Funzione che si salva in una variabile il contenitore dei Tweet ed invoca la funzione di offuscamento.

Innanzitutto, si è dovuto fronteggiare il problema dovuto al continuo ricaricamento della pagina che Twitter effettua; nella pagina avviene infatti un 'refresh' ogni pochi millisecondi, che ricarica i nuovi Tweet appena pubblicati, motivo per il quale è stato necessario ricercare un sistema di monitoraggio dell'elemento 'section'.

Alla riga 1 del listato 3.4, viene inizializzato il costruttore 'MutationObserver' [15], che offre la possibilità di controllare le modifiche apportate all'albero DOM, invocando una funzione di callback specificata quando si verificano cambiamenti. Mentre, all'interno della funzione alla riga 10, viene applicato sull'oggetto il metodo di istanza 'observe()' [15], standard de facto per per iniziare a ricevere notifiche in seguito a delle modifiche sul DOM. Alla riga 12-13 viene definito quando notificare il cambiamento, quindi inerenti alla lista dei figli ('childList') e all'alberatura ('subtree'). Se avviene uno di questi cambiamenti, il costruttore (come definito nella riga 1), richiama nuovamente la funzione 'obfuscate_text()', che ha il compito di offuscare i Tweet in attesa del controllo del modello (che disoffuscherà i Tweet non tossici)..

Alla riga 4 viene creata una variabile 'parental_tweeting', che grazie al metodo **'setInterval()'** chiama la funzione 'parental_twitting_dom_loaded' ogni 3 decimi di secondo. Questo avviene finchè la funzione entra nell'**if** (cioè fin quando non esiste l'elemento contenente i Tweet), e viene cancellato l'intervallo grazie al metodo **'clearInterval()'**.

Analizzando invece la funzione, vediamo che entra nell'**if** solo se l'elemento 'section' è stato caricato, e dopo aver chiamato il 'clearInterval()', invoca la funzione **'obfuscate_text()'**, che vedremo nel listato 3.5. Successivamente, come già anticipato, grazie al metodo **'observe()'** viene definito il comportamento dell'oggetto 'Observer', che deve osservare i cambiamenti dell'elemento 'section'.

Funzione 'obfuscate_text()'

```
1     var extension_times = 3;
2 function obfuscate_text() {
3     spans = document.querySelectorAll('[data-testid="tweetText"]'); //
    recupera tutti i tweet
4     let arr_tweet = [];
5     for (var i = 0; i < spans.length; i++) {//per ogni tweet
```

```
6           if (i > 5) {//se sono piu di 5 tweet non ne controllo altri
7               break;
8           }
9           arr_tweet.push(spans[i]);
10      }
11
12      if (arr_tweet.length > 0) {
13          arr_tweet.forEach(mlCall);
14
15          if (extension_times <= 0) { // if creato per effettuare il
    controllo dell'observer solo 3 volte.
16              observer.disconnect();
17          } else {
18              extension_times--;
19          }
20      }
21  }
```

Listing 3.5: Funzione che raccoglie in un array i Tweet e invoca la funzione che li analizzerà.

Questa funzione ha il compito di ciclare tutti i Tweet (elementi che hanno l'attributo **'data-testid="tweetText"**) e per ogni Tweet chiama la funzione mlCall.

La fuzione inizialmente salva in una variabile 'spans' l'elemento del DOM contenente i testi dei Tweet (cinque alla volta, per motivi tecnici); successivamente 'spans' viene ciclato e ad ogni iterazione, i suoi elementi vengono inseriti tramite il metodo 'push()' all'interno dell'array **'arr_tweet'**, creato appositamente per contenere i Tweet che verranno inviati al modello di Machine Learning. Il prossimo step consiste nel verificare che l'array 'arr_tweet' esista e che sia popolato dai Tweet, se questo si verifica, viene eseguita la funzione **'mlCall'** per ogni elemento dell'array (quindi per ogni Tweet). Alla fine della funzione troviamo un'altra istruzione 'if' in cui la variabile **'extension_time'** è un contatore impostato a 3, e serve per controllare che il controllo del metodo 'observer()' venga effettuato solo tre volte, altrimenti verrebbero caricati Tweet all'infinito.

Un'altra importantissima funzione del file Content è la 'mlCall', che come già visto viene invocata dalla funzione 'obfuscate_text' per passare al modello i Tweet da analizzare, chiaman-

do l'API (la chiamata è effettuata in Ajax [16]) passando il testo di 'element' come argomento. Nel listato 3.6, verrà analizzata questa funzione, una tra le più importanti del progetto.

Funzione 'mlCall'

```
function mlCall(element) {
    let obfuscate_test_div = element.parentNode.querySelector("div.parental_twitter_obfuscate_div");
    if(!obfuscate_test_div) {
        element.innerHTML = "<div class='parental_twitter_obfuscate_div'>" + element.innerText + "</div>";
    }

    let xmlhttp = new XMLHttpRequest;

    xmlhttp.open("POST", "http://127.0.0.1:3000");
    xmlhttp.setRequestHeader("Content-Type", "application/json"); // impostiamo il content-type su json perch  l'API accetta un formato .json
    xmlhttp.onload = function (e) {
        if (xmlhttp.readyState === 4) {
            if (xmlhttp.status === 200) {//se l'API ritorna 200 OK
                try {
                    let json_arr = JSON.parse(xmlhttp.response);// recupera la risposta della API
                    if (json_arr[json_arr.length - 1]["results"][0]["match"]) { // recupero ultimo elemento dell'oggetto response (perche contiene un booleano, indicante la tossicita o meno)
                        let parent_div = null;// conterra il div contenitore del tweet
                        let parent_element = element.parentNode;
                        while (parent_element && parent_element.nodeName.toLowerCase() != 'body') {
                            if (parent_element.getAttribute("data-testid") && parent_element.getAttribute("data-testid") == "cellInnerDiv") {
                                parent_div = parent_element;
```

```
                                break;
                            }

                            parent_element = parent_element.parentNode;
                        }

                        let tox_child = parent_div.getElementsByClassName("tox_img");

                        if (parent_div && tox_child.length == 0) { // se non ha un immagine 'occhio di visibilita', allora aggiungo
                            const eye = document.createElement("img");
                            eye.src = " string"
                            eye.style.cssText += 'position: absolute; right: 9%;top: 13%;height: 1.2em;cursor:pointer;';
                            eye.classList.add("tox_img");
                            eye.addEventListener("click", function () {
                                toggleObfuscation(this);
                            });
                            parent_div.appendChild(eye);
                        }
                    } else {
                        obfuscate_test_div.classList.add('parental_twitter_nobfuscate_div');
                        obfuscate_test_div.classList.remove('parental_twitter_obfuscate_div');
                    }
                } catch (e) {
                    console.log(e);
                }
            } else {

            }
        }
    };
    xmlhttp.send(JSON.stringify([element.innerText]));
```

```
53 }
```

Listing 3.6: Dichiarata la chiave che permette lo script di contenuto.

All'inizio della funzione, viene data una classe personalizzata 'parental_twitter_obfuscate_div' all'elemento padre del Tweet (il div che lo contiene). Alla riga 9 comincia la chiamata Ajax al servizio web NodeJS che funziona da API; innanziztutto viene specificata una chiamata in 'POST' per la porta 3000 del server locale. Dopodichè viene impostato il 'content-type' su JSON [17], perchè l'API accetta come input un formato '.json'.

Se la **'Request'** ha successo (quindi se lo 'status' ha valore === 200), si esegue il corpo dell'istruzione **'try'**, che salva in una variabile **'json_arr'**, la risposta ('Response') dell'API dopo aver trasformato in un array l'output JSON, grazie al metodo **'parse()'** [18]. Nella riga 16 c'è un ulteriore istruzione 'if' in cui, in caso di match tossico con il , si recupera l'ultimo elemento dell'oggetto 'Response' (Tweet ritornato dall'API), perche contiene il booleano che indica se il Tweet è tossico o meno. In caso negativo si entra nell'istruzione 'else', in cui viene aggiunta al 'div' la classe **'parental_twitter_nobfuscate_div'**, che rende quindi possibile la visione del Tweet.

Nella variaile **'parent_div'** sarà asseganto l'elemento 'div', nonchè padre dell'element (che contiene il Tweet), e questa assegnazione è resa possibile dall'istruzione 'while' alla riga 19, che risale tutti gli elementi padre del Tweet, finchè non incontra un 'div' con un attributo 'data-testid', e con valore 'cellInnerDiv', a quel punto lo assegna alla variabile 'parent_div'. Nell'istruzione 'if' alla riga 30, si verifica che esista un elemento 'parent_div' e che questo non contenga già un'immagine associata **'tox_child'**, che corrisponde a un'icona cliccabile presente in ogni Tweet tossico. Se si entra nell'if viene quindi creata la classe 'eye' che contiene l'immagine (dalla riga 32 a 34), mentre invece grazie al metodo **'addEventListener()'**, al click dell'immagine viene invocata la funzione **'toggleObfuscation()'** che vedremo nel listato 3.7, e che renderà inizialmente offuscato il testo dell'intero Tweet. Alla riga 36 verrà assegnata la classe 'eye' all'elemento 'parent_div'.

Andiamo a questo punto ad analizzare la funzione 'toggleObfuscation()':

Funzione 'toggleObfuscation()'

```
1  function toggleObfuscation(el) {
2      let obfuscate_test_div = el.parentNode.querySelector("div.
       parental_twitter_obfuscate_div");
3      if (obfuscate_test_div) {
4          el.src = "_string_1";
5          obfuscate_test_div.classList.add('parental_twitter_nobfuscate_div')
       ;
6          obfuscate_test_div.classList.remove('parental_twitter_obfuscate_div
       ');
7      } else {
8          let nobfuscate_test_div = el.parentNode.querySelector("div.
       parental_twitter_nobfuscate_div");
9          if (nobfuscate_test_div) {
10             el.src = " image_string_2;
11             nobfuscate_test_div.classList.remove('
       parental_twitter_nobfuscate_div');
12             nobfuscate_test_div.classList.add('
       parental_twitter_obfuscate_div');
13         }
14     }
15 }
```

Listing 3.7: Funzione che gestisce il click dell'icona.

Questa funzione svolge semplicemente il compito di rendere l'icona o cliccata (quindi il Tweet offuscato), o non cliccata (quindi il Tweet visibile). Abbiamo visto (nel listato 3.6) che questa funzione viene invocata al click dell'icona 'tox_img'; quindi se si entra nell' istruzione 'if' alla riga 3 significa che il 'div' ha la classe **'parental_twitter_obfuscate_div'** ed è attualmente offuscato, deve essere quindi rimossa questa classe e aggiunta la classe **'div.parental_twitter_nobfuscate_div'** che lo renderà visibile (quindi verrà aggiornata l'icona 'non sbarrata'). Se invece si entra nel secondo 'if' alla riga 8, avviene l'esatto contrario, dato che l'icona cliccata in questo caso, appartiene a un 'div' che sicuramente ha la classe di 'non offuscamento', e verrà quindi offuscato (quindi verrà inserita l'icona 'sbarrata').

API NodeJS

Dopo aver sviluppato l'estensione ed aver testato le varie funzioni grazie alla console di Chrome, l'ultimo passaggio è stato lo sviluppo del servizio web NodeJS che si comporta come un'API per l'estensione. Per creare questo servizio è stato necessario innanzitutto installare le librerie che compongono il modello **'Toxicity'** all'interno della cartella route del servizio; successivamente è stato creato il file portante del servizio **'index.js'** tramite il quale ci si mette in ascolto sulla porta 3000 del server locale (porta di destinazione della chiamata Ajax nella funzione 'mlcall()'), ed è stata definita una route che accetta chiamate con il metodo 'POST'. Vedremo nel listato 3.8, come è strutturato il file 'index.js', per capire come gestisce la chiamata 'POST' e come viene fatta la chiamata al modello di Machine Learning.

Sviluppo del file 'index.js'

```
1  const express = require('express');
2  const app = express();
3  const port = 3000;//porta su cui node sta in ascolto
4
5  const tf = require('@tensorflow/tfjs');//libreria tensorflow
6  const tf_tox = require('@tensorflow-models/toxicity'); //libreria toxicity
7
8  let model_result = null;
9
10 app.use(express.json())//middleware di express che recupera la request e la
       restituisce gia in formato json
11 app.post('/', async (req, res) => {//route principale della API (in post)
12     model_result = null;
13     /*
14     * async e await si basano sul meccanismo delle Promise e il loro
       risultato e' compatibile con qualsiasi API che utilizza le Promise.
15     * La parola chiave await sospende l'esecuzione di una funzione in
       attesa che la Promise associata ad un'attivita asincrona venga risolta o
       rigettata.
16     * */
17     await model_call(req.body);//chiamata della funzione "model_call" in
```

```
    modalit  sincrona (await) passando come parametro la richiesta
18     res.send(model_result);//invio della risposta
19 });
```

Listing 3.8: File principale dell'API.

Per lo sviluppo del servizio, si è utilizzato **'Express.js'** (o semplicemente Express), cioè un framework per applicazioni web per Node.js che fornisce uno strato di funzionalità di base per le applicazioni web, senza nascondere le funzioni Node.js. Inoltre fornisce molti metodi di utilità HTTP, middleware e per la creazione di API affidabili [19].

Nelle prime righe del listato 3.8 vengono caricati i moduli del framework con il metodo **'require()'** e viene inizializzata l'app Express assegnando la funzione 'express()', a una variabile 'app', che diventa un oggetto tramite il quale è possibile accedere a metodi e proprietà della stessa. Successivamente viene impostata la porta su cui ci si mette in ascolto (la porta 3000), e istanziate le due librerie che compongono il modello. Alla riga 10 si utilizza una funzione middleware [20] di express che recupera la 'request' e la restituisce già in formato json.

Fatte le iniziali importazioni ed inizializzazioni di variabili, con la riga 11 si entra nel vivo del codice, infatti viene creata la route principale dell'API, che accetta chiamate 'POST'. All'interno della route viene utilizzata la parole chiave **'await'** per effettuare la chiamata in modalità sincrona alla funzione 'model_call()' (che vedremo nel listato 3.9), passando come parametro la 'request'. La funzione rimarrà in attesa dei risultati che passano dall'esecuzione di altre funzioni, dopodichè sarà effettuato l'invio della risposta.

Per quanto riguarda le due parole chiave 'async' e 'await', servono per abilitare la gestione di funzioni asincrone eseguite tramite un approccio sincrono. Si basano sul meccanismo delle Promise e il loro risultato è compatibile con qualsiasi API che utilizza le Promise. La parola chiave 'await' utilizzata nella route, sospende l'esecuzione di una funzione, in attesa che la Promise associata ad un'attività asincrona venga risolta o rigettata [21].

Funzione 'model_call()'

```
1 async function model_call(sentences) {
2     try {
```

```
3         const threshold = 0.9;//definizione della soglia per il
    riconoscimento della tossicita da parte del modello di ML Toxicity
4         await tf_tox.load(threshold).then(model => {//chiamata in modalita
    sincrona della funzione load di Toxicity
5             let tox_result = tox_call(model, sentences);//chiamata della
    funzione tox_call passando come argomento il modello ML (Toxicity) e la
    frase da controllare
6             return tox_result;
7         }).catch((error) => {
8             console.error(error);
9         });
10    } catch (e) {
11        console.log(e);
12    }
13}
```

Listing 3.9: Funzione che effettua la chiamata del modello.

La funzione prende come argomento un oggetto 'sentences', che corrisponde al Tweet da analizzare, dopodichè si trova un costrutto 'try-catch' [22], in cui inizialmente nella variabile 'threshold' viene definita la soglia per il riconoscimento della tossicità da parte del modello. Alla riga 4 viene effettuata la chiamata sincrona della funzione 'load()' di 'Toxicity' passando come argomento la variabile (soglia) 'threshold'.

A questo punto si effettua la chiamata alla funzione **'tox_call'** (che analizzeremo nel listato 3.10), passando come argomento il modello 'Toxicity' e la frase da controllare; i risultati vengono salvati in una variabile 'tox_result', che viene ritornata.

Funzione 'tox_call()'

```
1 async function tox_call(model, sentences) {
2     try {
3         if(sentences && sentences.length > 0 && sentences["0"]){//se esiste
    la frase e non e' vuota
4             await model.classify(sentences).then(predictions => {//
    invocazione del metodo classify di Toxicity; in prediction vengono
    passati i risultati del ML Toxicity
```

```
5                   model_result = predictions;
6                   return predictions;
7               }).catch((error) => {
8                   console.error(error);
9               });
10          }
11      } catch (e) {
12          console.log(e);
13      }
14  }
15
16  app.listen(port, () => console.log(`Parental Twitter control running at
        port: ${port}!`))//avvio di nodejs in ascolto sulla porta indicata
```

Listing 3.10: Funzione che valuta la tossicità della frase da analizzare.

In quest'ultima funzione, verrà valutata la tossicità della frase passata nella precedente chiamata. Troviamo nuovamente alla riga 2, un'istruzione 'try-catch', e non appena entrati nel 'try' viene valuatata una condizione per accedere ad un'istruzione 'if': se esiste la frase ('sentences'), e non è una stringa vuota viene eseguito il corpo. Una volta verificata l'esistenza della frase, alla riga 4 viene invocato il metodo 'classify()' del modello passando la frase come argomento, e vengono poi passati i risultati in 'predictions', che viene ritornato.

Capitolo 4

Funzionamento dell'estensione Chrome

Nel capitolo precedente abbiamo visto come si arriva ad ottenere l'analisi di tossicità dei Tweet, mentre in questo capitolo, sarà mostrato tramite alcuni screenshot, il funzionamento lato Client dell'estensione. Per testare l'estensione e far vedere il risultato ottenuto, è stato necessario creare un account 'test' in cui sono stati pubblicati dei Tweet considerati tossici dal modello, così da non sovraccaricare di chiamate l'API, alla ricerca di Tweet inappropriati.

L'utilizzo di 'Parental Control Tweet'

Il primo passo per avviare l'estensione, dopo aver attivato la 'modalità sviluppatore' e caricato correttamente la directory dall'apposita sezione alla URL 'chrome://extensions/', è avviare l'estensione cliccando sull'interrutore di attivazione che vedremo nella figura 4.1.

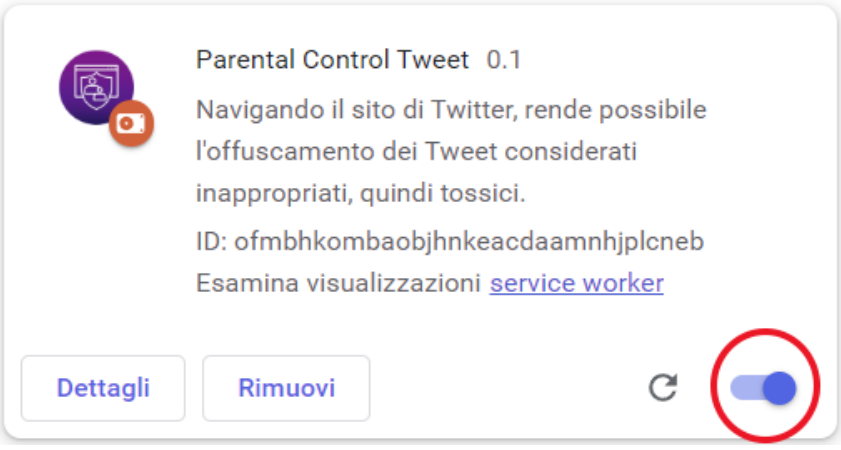

Figura 4.1: Attivare l'estensione Parental Control Tweet.

Dopo l'attivazione, è conveniente usufruire della funzione di Chrome di fissare accanto alla barra di ricerca le estensioni più utilizzate, per un'interazione più veloce con le stesse. Possiamo vedere nell'immagine 4.2, come clicccando l'icona del puzzle (icona delle estensioni Chrome), si apre un popup che ci permette di fissare in alto l'estensione.

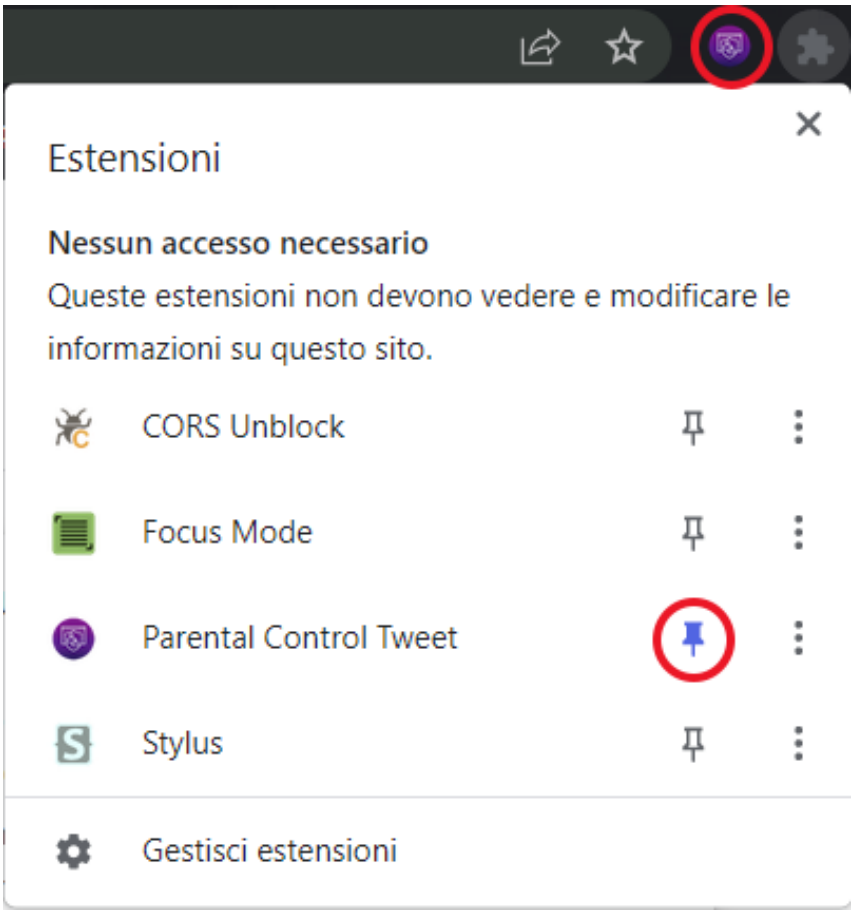

Figura 4.2: Figura che mostra come fissare in alto l'estensione.

Una volta cliccato l'interruttore, se viene navigato un dominio presente nel campo 'matches' del Manifest (in questo caso 'www.twitter.com'), l'estensione mette in pratica le azioni che abbiamo specificato nel codice. Nella figura 4.3 vedremo come dopo aver effettuato il login con il proprio account all'interno di Twitter, e ricercato l'account test, sarà catturato tramite il DOM l'elemento che contiene i Tweet, per poi analizzarli attraverso il modello di Machine Learning, 'Toxicity'.

Figura 4.3: Figura in cui viene mostrata la funzione dell'offuscamento di Tweet inappropriati.

Nella prossima figura (4.4), vedremo in azione la funzione di 'disoffuscamento', che si attiva una volta cliccata l'icona evidenziata di un Tweet considerato tossico dal modello.

Figura 4.4: Figura in cui viene mostrato il contenuto dei Tweet inappropriati.

In questo modo viene proposto all'utente una diretta interazione con i Tweet, in cui si ha la scelta di scoprire contenuti che sappiamo essere quantomeno scurrili. Oltre al funzionamento lato Client che abbiamo visto, in figura 4.5 viene mostrata la stampa della classificazione che compie il modello quando gli viene passato in input il Tweet.

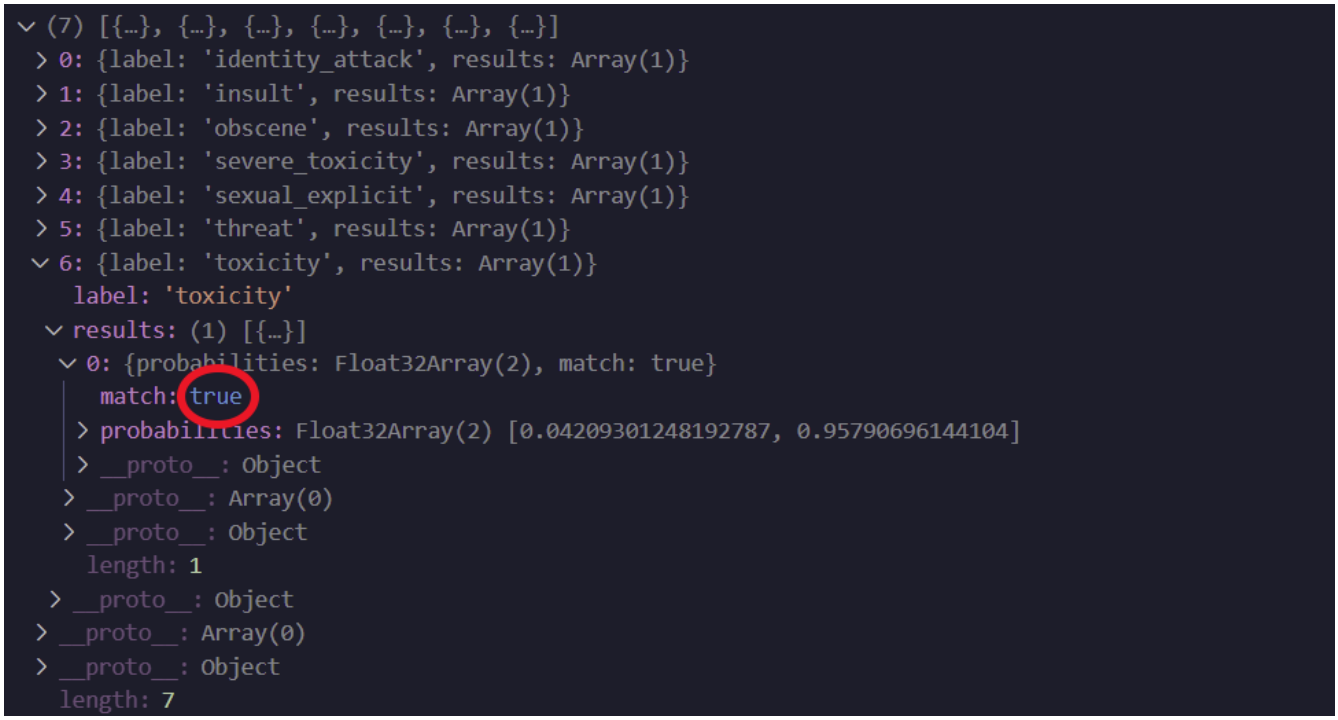

Figura 4.5: Figura che mostra la stampa della classificazione del Tweet lato API.

Come si puo notare, l'elemento 6 ('toxicity') dell'array risultante dall'analisi del modello, è stato espanso, e vediamo che è composto da due campi: il primo è 'label' che è l'etichetta dell'elemento appunto; il secondo campo è 'results', e al suo interno questo campo contiene una chiave 'matches', booleano (true o false) che indica se il testo passato in input ha tossicità **true** o **false**.

Se analizzando il Tweet, fa match almeno una delle 5 tipologie di tossicità ('identity_attack', 'insult' ecc.), automaticamente l'elemento 'toxicity' risulterà di valore **true** (quindi tossico, come nella figura 4.5). La funzione 'mlCall' dell'estensione va infatti ad analizzare l'elemento 'toxicity', per sapere se un Tweet è tossico o meno.

Nella prossima figura (4.6), si avrà la possibilità di osservare come tramite il metodo Javascript 'console.log()' [23], venga stampato nella console di Chrome (DevTools [24]), il risultato che il modello passa dopo aver effettuato l'analisi.

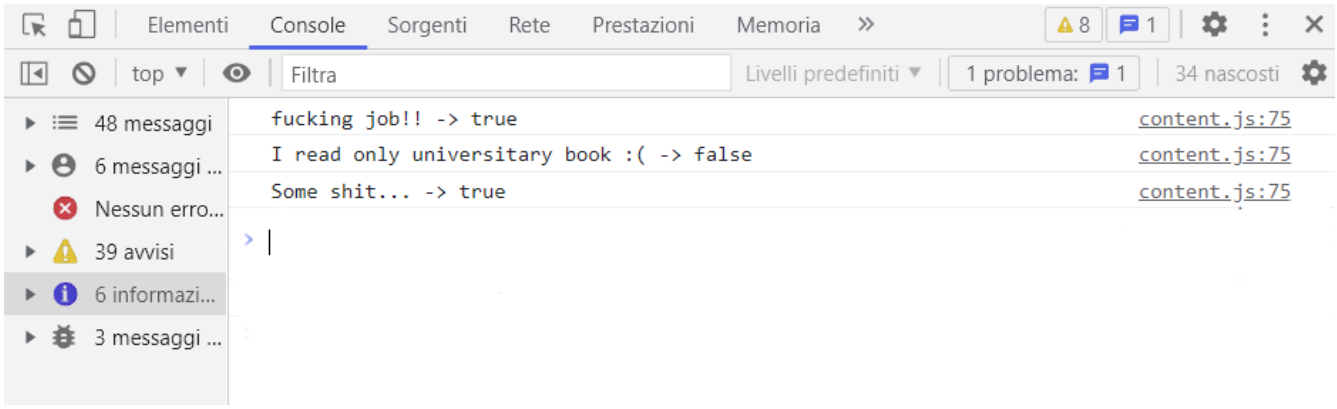

Figura 4.6: Figura che mostra il risultato dell'analisi dei Tweet visualizzato nella console.

A questo punto l'estensione è funzionante e svolge il compito di dare la possibilità di limitare la visione di specifici Tweet.

Capitolo 5

Conclusioni

In questo capitolo conclusivo, verranno sottolineati gli obiettivi che hanno spinto allo sviluppo di questo progetto, e i risultati ottenuti, nella sezione 5.1; mentre nella sezione 5.2 saranno analizzati nel dettaglio gli sviluppi che sono previsti per il futuro dell'estensione creata.

5.1 Highlights

Questa tesi ha introdotto un metodo di semplice utilizzo per gestire e limitare il contenuto tossico online, ed in particolare sulle piattaforme social. In particolare, è stato portato come esempio quello di Twitter, data la grande quantità di post testuali al suo interno.

L'obiettivo principale era quello di creare un sistema di limitazione del linguaggio tossico, e rendere disponibile un controllo dei contenuti sulle piattaforme social vista la frequenza con cui i bambini maneggiano i nostri smartphone. Lo sviluppo ha riguardato la creazione di un'estensione per il motore di ricerca di Google Chrome (di cui la maggior parte degli utenti fa uso), utilizzando la funzione che offre Chrome agli sviluppatori per realizzare un'estensione personalizzata e unica. Successivamente è stato sviluppato un servizio API in NodeJS [2] che al suo interno contiene un modello ML per la classificazione di tossicità dei testi, e la logica di codice che rende possibile l'integrazione di quest'ultimo con l'estensione.

Lo sviluppo del progetto di tesi ha portato come risultato un'estensione Chrome che funziona navigando Twitter dal Browser di Chrome, e svolge il compito di analizzare ogni Tweet presente nella pagina, (interfacciandosi con l'API del modello) e una volta ottenuti i risultati

agisce sui testi dei Tweet, offuscando quelli ritenuti tossici. Inoltre viene aggiunta un'icona cliccabile (occhio di visibilità) ad ogni Tweet offuscato, e al click dell'icona la classe di offuscamento viene rimossa ed il testo diventa momentaneamente leggibile.

5.2 Sviluppi futuri

Il risultato sperato con lo sviluppo di tale progetto, era quello di creare un'estensione che fosse più attuale possibile, quindi realmente utilizzabile e utile per gli utenti. La maggior parte del tempo che è stato investito nello sviluppo, è stato incentrato nell'obiettivo di creare la funzione di oscuramento dei Tweet, curando quanto possibile la velocità di risposta del modello, e la linea grafica che cerca di rendere al meglio l'idea di controllo, sia tramite i colori che tramite le icone scelte. L'idea futura riguardo l'estensione, non si limita però alla pubblicazione dell'attuale versione, ma ad un'accurata revisione, durante la quale verrano migliorate le funzionalità esistenti, ed aggiunte di nuove. Le features aggiuntive previste, attualmente sono tre:

- La prima funzionalità, riguarda un pannello di gestione dell'estensione, in cui appunto decidere in base alle proprie esigenze, se mostrare l'icona per rimuovere l'offuscamento, o se addirittura mettere fuori uso la funzione, e ricevere un semplice avvertimento (in caso di Tweet tossico) senza però limitare la visione del Tweet.

- Un'altra importante feature da aggiungere, che sarebbe di supporto alla precedente, è un semaforo di tossicità. Sarebbe necessario espandere il controllo eseguito dal modello, e creare tre gradi di tossicità (grave, medio, nullo) rappresentati da un semaforo, differentemente da quello attuale (che è un booleano), che ci dice solamente se il testo è tossico, o meno. In questo modo si aggiungerebbe al pannello una funzionalità in cui ci si affida al semaforo per l'oscuramento, magari impostando la funzione solo per i casi 'gravi'.

- L'ultima feature in programma, permetterà all'estensione di espandere il campo di azione, integrando la logica applicativa per comunicare ed agire su altri Social Network, come Facebook e Instagram (o dividendo le estensioni, in base al Social su cui agisce). In tal caso non basterà solo l'utilizzo del modello ML 'Toxicity' che agisce esclusivamente

sui testi, ma ci sarà la necessità di integrare un modello che si basi sullo studio delle immagini e dei video, e che ne identifichi eventualmente la tossicità.

Queste funzionalità permetterebbero all'utente, intanto una diretta interazione con le funzioni che l'estensione offre, personalizzando il controllo in base alle situazioni personali, e alzando quindi il valore di usabilità. Inoltre sarà incrementato l'uso dell'estensione, grazie alla feature che la renderà utilizzabile anche per altri Social Network.

5.3 Ringraziamenti

Giunti alle conclusioni dell'elaborato, ritengo doveroso sottolineare chi ha contribuito a portare a compimento questa splendida esperienza. Ringrazio in primis il mio relatore, il Professor Gallicchio, che mi ha guidato e supportato nella fase più importante, fornendomi preziosi consigli e trasmettendomi conoscenze indispensabili per trattare gli argomenti affrontati.

Un ringraziamento speciale va alla mia famiglia, che non manca mai l'occasione di farmi sentire al primo posto e di essere presente in ogni momento della vita, anche negli errori e nelle difficoltà; non avrei desiderato niente di più perfetto di voi. In particolare voglio ringraziare i miei genitori, per avermi permesso di affrontare questa esperienza con il giusto focus, ma soprattutto per essere sempre al mio fianco, in prima linea, pronti a sostenermi in ogni mia scelta.

Ringrazio Chiara, la mia fidanzata, la mia compagna di squadra, sempre pronta a fare del suo meglio per me e per i miei obiettivi. Ringrazio tutti i miei amici, quelli di una vita, e quelli incontrati lungo la strada di cui ho subito capito il valore, ma in particolare ringrazio Nicola, Carmine, e Vittorio, che sono una parte di me, e sono stati fondamentali in questo viaggio. Infine, ringrazio la mia azienda, che mi ha permesso di svolgere al meglio il percorso intrapreso.

Bibliografia

[1] TensorFlow. *TensorFlow Hub*. URL: https://tfhub.dev/tensorflow/tfjs-model/toxicity/1/default/1.

[2] Juan De Dios Santos. *Distribuisci un modello TensorFlow.js pre-addestrato utilizzando Node in Cloud Run*. URL: https://juandes.com/tensorflowjs-node-cloudrun/.

[3] Wikipedia. *Twitter*. URL: https://it.wikipedia.org/wiki/Twitter.

[4] IBM. *Artificial Intelligence (AI)*. URL: https://www.ibm.com/cloud/learn/what-is-artificial-intelligence.

[5] Zendesk. *Apprendimento profondo e apprendimento automatico*. URL: https://www.zendesk.com/it/blog/machine-learning-and-deep-learning/#georedirect.

[6] TensorFlow. *TensorFlow*. URL: https://www.tensorflow.org/.

[7] NPM. *@tensorflow-modelli/tossicità*. URL: https://www.npmjs.com/package/@tensorflow-models/toxicity.

[8] Jason Brownlee. *Machine Learning Mastery With Python*. 2021.

[9] Wikipedia. *Treebank*. URL: https://en.wikipedia.org/wiki/Treebank.

[10] Amit Chaudhary. "Universal Sentence Encoder Visually Explained". In: *Annalen der Physik* (2020). DOI: https://amitness.com/2020/06/universal-sentence-encoder/.

[11] Google. *Getting Started Guides*. URL: https://developer.chrome.com/docs/extensions/mv3/getstarted/.

[12] GeeksforGeeks. *NodeJS*. URL: https://www.geeksforgeeks.org/nodejs/.

[13] Team I.A. Italia. *TensorFlow la libreria Python per il Deep Learning*. URL: https://www.intelligenzaartificialeitalia.net/post/tensorflow-la-libreria-python-per-il-deep-learning.

[14] TensorFlow. *TensorFlow Lite*. URL: https://www.tensorflow.org/lite.

[15] MDN. *MutuationObserver - API Web*. URL: https://developer.mozilla.org/en-US/docs/Web/API/MutationObserver.

[16] W3schools. *Ajax$_d$oc*. URL: https://www.w3schools.com/js/js ajax intro.asp.

[17] MDN. *JSON*. URL: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON.

[18] MDN. *json.parse()*. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/GlobalObjects/JSON/parse?retiredLocale=it.

[19] Express. *Express documentation*. URL: https://expressjs.com/.

[20] Express. *Express middleware*. URL: https://expressjs.com/it/guide/using-middleware.html.

[21] HTML.it. *Funzioni asincrone con async/await*. URL: https://www.html.it/pag/69778/funzioni-asincrone-con-asyncawait/.

[22] MDN. *try...catch*. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/try...catch.

[23] MDN. *console.log()*. URL: https://developer.mozilla.org/en-US/docs/Web/API/console/log.

[24] Google Chrome. *Chrome DevTools*. URL: https://developer.chrome.com/docs/devtools/.