



Università di Catania

UNIVERSITÀ DEGLI STUDI DI CATANIA

DIPARTIMENTO DI MATEMATICA E INFORMATICA

CORSO DI LAUREA TRIENNALE IN INFORMATICA

Automazione del processo di creazione di classificatori su tassonomie gerarchiche

Candidato: **Sergio Maccarrone**

Relatore: **Prof. Salvatore Nicotra**

Anno accademico 2020-2021

" Dietro ogni traguardo c'è una nuova partenza.

Dietro ogni risultato c'è un'altra sfida.

Finché sei vivo sentiti vivo.

Vai avanti, anche quando tutti si aspettano che lasci perdere."

- Madre Teresa di Calcutta -

Indice

| | |
|--|----|
| 1. Abstract | 8 |
| 2. Introduzione | 9 |
| 3. Il progetto | 10 |
| 3.1. I punti del progetto | 10 |
| 3.2. Strumenti, software e tecniche di sviluppo utilizzate | 12 |
| 4. Classificazione del testo | 14 |
| 4.1. Definizione, cenni storici ed evoluzione | 14 |
| 4.2. Casi d'uso | 16 |
| 4.3. Classificazione gerarchica | 18 |
| 5. Tassonomie gerarchiche | 19 |
| 5.1. IPTC | 19 |
| 5.2. Tassonomia Media Topics | 22 |
| 5.3. Estrazione automatica delle informazioni dalla tassonomia | 24 |
| 6. Creazione Dataset | 32 |
| 6.1. Definizione e importanza dei Dataset | 32 |
| 6.2. Creazione automatica del dataset: | 33 |
| 6.2.1. Dataset per le notizie italiane | 33 |
| 6.2.2. Libreria GoogleNews | 35 |
| 6.2.3. Algoritmo sviluppato | 39 |
| 7. Preprocessing e suddivisione del testo | 45 |
| 7.1. Preprocessing del Dataset: Pipeline NPL | 45 |
| 7.2. Suddivisione del Dataset: Cross validation | 55 |
| 8. Creazione del Classificatore di testo | 60 |
| 8.1. Estrazione delle features dal testo: | 60 |
| 8.1.1. Tecniche d'estrazione delle features: | 60 |

| | | |
|------------|---|------------|
| 8.1.2. | Implementazione dell'estrazione delle features | 67 |
| 8.2. | Algoritmi di ML e approcci utilizzati per la creazione del classificatore | 71 |
| 8.2.1. | Algoritmi di ML utilizzati per classificare | 71 |
| 8.2.2. | Approcci per la realizzazione dei classificatori | 78 |
| 9. | Analisi delle performances dei due approcci | 95 |
| 10. | Sviluppi futuri..... | 101 |
| 11. | Conclusioni..... | 102 |
| 12. | Sitografia | 103 |
| 13. | Ringraziamenti | 105 |

Capitolo 1

1. Abstract

The purpose of the thesis is to demonstrate how to automate the text classification process in order to realize a classifier able to label an article, assigning it one or more categories defined within a hierarchical taxonomy.

The text classification project was carried out during the internship period at the Neodata Group S.r.l. company. The problem addressed by the latter is based on the development of a software able to classify any article available on Italian journalistic platforms on a set of predefined categories within one of the hierarchical taxonomies of IPTC (International Press Telecommunications Council).

After the internship period, I kept working on the development of the software, believing that it could be an excellent solution to the problem presented by Neodata.

What distinguishes the study and the implementation of the project is the total automation in the creation of the dataset and the text classification model, such that, for any hierarchical taxonomy, structured following a well-defined pattern, the software is able to create a dataset and a text classifier based on the chosen taxonomy.

In addition, the entire project was developed using the Italian language, which led to the creation of all the components for the creation of the dataset and the classifier, since the resources available online in Italian are few and not suitable in the context of the text classification.

The software realized has been entirely written using Python programming language.

Capitolo 2

2.Introduzione

Lo scopo della seguente tesi è quello di dimostrare come si possa automatizzare il processo di classificazione del testo al fine di realizzare un classificatore in grado di etichettare un articolo, assegnandogli una o più categorie definite all'interno di una tassonomia gerarchica.

Il progetto di classificazione del testo è stato realizzato durante il periodo di tirocinio svolto presso l'azienda Neodata Group S.r.l. Il problema posto da quest'ultima si basa sullo sviluppo di un software in grado di classificare un qualsiasi articolo presente sulle piattaforme giornalistiche italiane su un insieme di categorie predefinite all'interno di una delle tassonomie gerarchiche di IPTC (International Press Telecommunications Council).

Finito il periodo di tirocinio, ho continuato a lavorare allo sviluppo del software, convinto che esso possa essere un'ottima soluzione al problema posto da Neodata.

Ciò che distingue lo studio e l'implementazione del progetto è la totale automatizzazione nella creazione del dataset e del modello di classificazione del testo, tale che, per qualsiasi tassonomia gerarchica, strutturata seguendo uno schema ben definito, il software riesca a creare un dataset ed un classificatore di testo basati secondo la tassonomia scelta.

Inoltre, l'intero progetto è stato implementato basandosi sulla lingua italiana, la quale, ha portato a realizzare internamente tutti i componenti per la creazione del dataset e del classificatore, poiché le risorse disponibili online in lingua italiana sono poche e non utilizzabili nel contesto della classificazione del testo.

Il software realizzato è stato interamente scritto utilizzando il linguaggio di programmazione Python.

Capitolo 3

3. Il progetto

3.1. I punti del progetto

Di seguito verranno introdotti e spiegati brevemente i vari steps che sono stati svolti per sviluppare il software che crea automaticamente il dataset e il classificatore.

- **Comprensione del problema della classificazione del testo:** con il termine classificazione del testo, si intende il processo con il quale si assegna una delle categorie predefinite all'interno di una tassonomia ad un dato testo. Ad oggi, tale processo, grazie allo sviluppo di algoritmi di apprendimento automatico, può essere del tutto automatizzato, riducendo tempo e risorse.
- **Definizione e scelta della tassonomia:** con tassonomia si intende l'insieme delle categorie che possono essere assegnate ad un determinato elemento.
Nel nostro caso, una volta scelta la tassonomia (Media Topics di IPTC) più adeguata al problema della classificazione di notizie, è stata scaricata e importata all'interno del software.
- **Estrazione automatica delle keywords dalla tassonomia gerarchica:** una volta definita la tassonomia, è stato necessario creare una struttura, interna al software, che la contenga. Inoltre, sono state aggiunte ulteriori informazioni estremamente utili per gli steps successivi, come ad esempio la chiave di ricerca per gli articoli da correlare ad un nodo.

- **Creazione automatica del dataset composto da articoli di stampa italiana:** per realizzare il dataset si è dapprima creata una gerarchia di directory che rappresenta fedelmente la gerarchia della tassonomia. La logica sottostante alla creazione del dataset è stata quella di riempire automaticamente ogni directory con articoli ricercati online e correlati alla categoria rappresentata da essa. Per rendere automatica la ricerca, si è utilizzata la piattaforma Google News, la quale, attraverso una libreria di terzi, ha permesso di ricercare ed estrarre tutti i link delle pagine web contenenti gli articoli relativi ad ogni categoria. Per ogni articolo è stato estratto, attraverso web scraping, il testo, che successivamente è stato salvato in locale all'interno della relativa directory d'appartenenza.
- **Creazione del learning-set attraverso pipeline NLP per la pulitura del testo contenuto nel dataset:** il risultato dello step precedente è una gerarchia di directory, ognuna di esse contenente gli articoli correlati. Per la creazione del learning-set, si è processato ogni articolo contenuto nel dataset applicando la pipeline NLP, la quale sfruttando diversi metodi, permette di rendere il testo più semplice da processare per il classificatore.
- **Creazione del classificatore di testo per la lingua italiana:** definito il dataset ed applicata la pipeline NLP ai documenti che lo compongono, è possibile passare al processo di creazione del modello di classificazione. Tale processo si compone di diverse operazioni, quali: suddivisione del dataset in learning-set e test-set; estrazione delle features dai documenti del training-set; scelta dell'algoritmo di apprendimento automatico da utilizzare per creare il modello e fase di apprendimento e testing del modello.

Ogni step definito verrà analizzato nel dettaglio nei capitoli successivi.

3.2. Strumenti, software e tecniche di sviluppo utilizzate

Il progetto è stato interamente scritto utilizzando il linguaggio di programmazione Python sfruttando le diverse librerie offerte per tale linguaggio.

- **Sviluppo Agile:** per lo sviluppo dell'intero progetto si è fin da subito utilizzato l'approccio Agile, tale approccio si focalizza sull'obiettivo di implementare frequentemente e in tempi brevi funzionalità del software richiesto dal cliente, in modo da avere un continuo feedback da parte del commissionario del progetto. Il vantaggio della metodologia Agile sta nel continuo feedback e test da parte del cliente su funzionalità singole del software; ciò permette di avere una maggiore facilità nell'effettuare eventuali cambiamenti, poiché non si dovrà modificare l'intero software ma basterà operare sul singolo task.

All'interno del progetto di creazione del classificatore, lo sviluppo Agile ci ha permesso di implementare piccole funzionalità a cadenza settimanale ed avere, così, un continuo feedback da parte del tutor aziendale del progetto che ha approvato o suggerito delle modifiche al singolo task.

- **Python:** linguaggio ad alto livello orientato agli oggetti, noto per la sua semplicità e flessibilità, che gli permettono di essere uno dei linguaggi di programmazione più utilizzati.

Tra le caratteristiche distintive di Python spiccano le variabili non tipizzate e l'uso dell'indentazione per la sintassi delle specifiche al posto delle parentesi.

È stato scelto di sviluppare il software utilizzando tale linguaggio, in quanto, risulta essere il più utilizzato e più richiesto dal mercato.

Inoltre, essendo scalabile, si ha la possibilità di importare tutte le librerie di cui si ha bisogno in pochissimi passi, riducendo drasticamente i tempi di scrittura del codice. Avendo a disposizione una sintassi del codice user-friendly, Python permette un'intuitiva lettura del codice, riducendo al minimo i commenti alle istruzioni che si stanno scrivendo.

Infine, è stata ampiamente utilizzata la possibilità di creare singoli moduli, che hanno permesso di suddividere le funzionalità del software in singoli file Python, che, a loro volta sono stati importati come librerie nel codice principale. Tale meccanismo si sposa benissimo con la metodologia Agile, poiché ogni funzionalità implementata è stata definita come task e realizzata indipendentemente dal resto del software, rendendola testabile singolarmente e semplice da modificare.

- **Visual Studio Code:** editor di codice sorgente sviluppato da Microsoft che permette la stesura di codici in diversi linguaggi di programmazione.

Una delle funzionalità più utili di Visual Studio Code è la possibilità di scaricare diverse estensioni che possano semplificare il lavoro durante la creazione di applicazioni.

In particolare, è presente un'estensione per GitHub fondamentale per la condivisione del codice con un team di lavoro. Essa è stata fondamentale nello sviluppo del software, in quanto ha permesso di condividere il progetto con il team di sviluppo in modo da suddividere i task, rispettando in pieno quelli che sono i principi dello sviluppo Agile.

- **Librerie Python più importanti per lo sviluppo del progetto:**

- Request: libreria che permette di rendere le richieste HTTP richiamabili da codice in maniera semplice ed immediata.
- Spacy: libreria per l'elaborazione del linguaggio naturale che permette di analizzare ed elaborare il testo al fine di estrarre informazioni da esso.
- BeautifulSoup: libreria che permette di estrarre dati ed informazioni da file HTML e XML creando una struttura su cui è possibile navigare, ricercare e modificare i dati.
- Sklearn: libreria di apprendimento automatico al cui interno definisce diversi metodi per la gestione degli algoritmi di machine learning.

Capitolo 4

4. Classificazione del testo

4.1. Definizione, cenni storici ed evoluzione

Ad oggi, attraverso la continua digitalizzazione a cui stiamo andando incontro, si è diffusa sempre più la presenza di informazioni sottoforma di testo; esso lo si può trovare in diversi contesti quali: e-book, e-mail, articoli di giornale, chat, social media e siti web.

Data la grossa mole di informazioni testuali disponibili online, ci si è sempre posti il problema di etichettare tali dati, in quanto il testo contiene informazioni estremamente utili che possono permettere di semplificare diversi processi. Tale meccanismo prende il nome di classificazione del testo.

Con tale termine si intende il processo di assegnazione di tag o categorie ad un testo in base al suo contenuto.

In passato tale compito era un'operazione particolarmente difficile e costosa poiché richiedeva tempo e risorse per etichettare manualmente i dati o creare regole di etichettatura complesse.

Un esempio potrebbe essere quello di definire una lista di parole chiave per ogni categoria che si vuole identificare. Nel caso in cui le categorie fossero sport e politica, le liste di parole chiave potrebbero essere definite come segue:

- Sport [sport; calcio; tennis; atleta; gara; coppa; sportivo; campionato; olimpiade; match; vincere; medaglia; pattinaggio; squadra;]
- Politica [politica; elezione; parlamentare; governare; elettorale; referendum; candidare; costituzionale; vertice; comunale; democrazia;]

Definite le liste, si analizza il testo che si vuole classificare contando il numero di occorrenze di parole appartenenti alle liste poc'anzi descritte. Fatto ciò, si etichetta il testo con la categoria, le cui parole interne alla lista d'appartenenza sono apparse più volte.

Continuando con l'esempio, il testo potrebbe essere il seguente: "Il campionato non si ferma, Zlatan Ibrahimovic grazie alla sua tripletta ha portato alla vittoria la sua squadra nel match interno contro l'Inter".

Contando le occorrenze delle parole interne alle due liste rappresentanti le due categorie, in questo caso, per la lista Sport si hanno tre occorrenze, mentre per la lista Politica non si hanno parole presenti nel testo. Dunque, il testo stesso viene etichettato come Sport.

I sistemi basati su regole manuali risultano essere molto complessi soprattutto per quanto riguarda il processo di definizione delle liste di parole chiave che implicano un'approfondita conoscenza del dominio su cui si sta lavorando.

Inoltre, la regola che permette di etichettare un testo con una categoria piuttosto che un'altra, può diventare sempre più complessa richiedendo spreco di risorse e tempo.

Al giorno d'oggi, grazie al continuo evolversi delle tecnologie riguardanti l'apprendimento automatico, è possibile classificare il testo in maniera automatizzata, risolvendo molti dei problemi relativi alla creazione di regole manuali.

Tale classificazione automatica consiste nel far imparare al modello di classificazione come etichettare un testo. Ciò avviene attraverso una fase di apprendimento nella quale vengono passati una serie di esempi pre-etichettati, che permettono all'algoritmo di apprendimento automatico di comprendere le associazioni tra un particolare testo e una data categoria. Così facendo, una volta trasmesso al classificatore un testo mai processato, esso, riuscirà a ritornare la categoria più pertinente, in base a ciò che ha imparato dai dati passati in precedenza.

4.2. Casi d'uso

L'importanza della classificazione del testo è data dai vari ambiti in cui può essere applicata. Grazie alla totale personalizzazione del classificatore essa permette di semplificare alcuni processi utili alle aziende come:

- **Tagging di contenuti o prodotti:** processo nel quale si assegnano delle etichette/categorie a prodotti o contenuti per migliorarne la navigazione e l'identificazione. Tale processo è ampiamente utilizzato per etichettare e suddividere i prodotti interni in un e-commerce o su un forum.
- **Automatizzazione delle attività di CRM:** attraverso la classificazione del testo è possibile automatizzare molti dei processi riguardanti la gestione delle relazioni con i clienti. Ad esempio, si è in grado di analizzare la richiesta del cliente e capire autonomamente il tipo, l'importanza e la rilevanza di tale richiesta, migliorando in tal modo l'esperienza del cliente e la gestione interna delle richieste per il reparto di customer care.
- **Rilevamento della lingua:** procedura che permette di rilevare la lingua di un dato testo, utile ad esempio, per capire automaticamente se un ticket di supporto in arrivo è scritto in inglese o in italiano, permettendo di instradare automaticamente il ticket al team appropriato.
- **Sentiment Analysis:** processo mediante il quale si è in grado di capire automaticamente se un dato testo sta parlando positivamente o negativamente di un argomento, utile per monitorare la reputazione di un marchio o di un personaggio pubblico.

Inoltre, l'analisi del sentimento in un insieme di testi permette agli addetti al marketing di un'azienda di monitorare e classificare gli utenti, sulla base di testi interni a post o commenti sui social relativi ad un prodotto o ad un marchio.

- **Rilevamento dell'argomento:** processo utilizzato per l'identificazione dell'argomento di cui si sta discutendo in un dato testo. Adatto per la suddivisione di richieste di supporto interne ad un forum o per implementare la pubblicità online basata sul contenuto, permettendo di inserire pubblicità mirate in base al contenuto del testo interno ad una pagina web.
- **Estrazione di Tag interni ad una pagina Web:** processo che consente ai motori di ricerca di semplificare la scansione di un sito web al fine di estrarre tag utili per la SEO, come ad esempio il titolo o il contenuto di una pagina web.
- **Creazione di sistemi di risposta immediata alle emergenze:** le autorità possono utilizzare la classificazione del testo per capire quando all'interno dei social media sono presenti richieste di aiuto e in tal caso fornire una risposta rapida.

4.3. Classificazione gerarchica

Fin da piccoli il nostro cervello percepisce il mondo che lo circonda in strutture gerarchiche.

Prendendo in considerazione l'insieme dei mezzi di trasporto, una prima suddivisione può essere fatta in base alle caratteristiche che ogni mezzo di trasporto possiede, ad esempio considerando il numero di ruote di cui ogni mezzo è fornito una semplice classificazione gerarchica potrebbe essere la seguente:



FIGURA 1: ESEMPIO DI CLASSIFICAZIONE GERARCHICA.

Il risultato uscente dalla classificazione e suddivisione degli elementi, in base alle loro proprietà, definisce una gerarchia di categorie che prende il nome di tassonomia.

Ritornando al problema della classificazione del testo, la tassonomia che definisce le varie classi o etichette a cui possono appartenere i testi, può essere creata ad hoc in base al contesto oppure facendo riferimento ad uno degli standard già definiti, come verrà successivamente approfondito nel **paragrafo 5.2**.

Capitolo 5

5. Tassonomie gerarchiche

5.1. IPTC

Come è stato già discusso nel capitolo precedente, durante l'implementazione di un classificatore di testo, diventa essenziale scegliere un insieme di categorie che possano essere assegnate al testo. Tale insieme deve essere ben definito, motivo per il quale diversi enti hanno definito degli standard disponibili gratuitamente.

La differenza tra il definire una propria tassonomia e utilizzarne una standard dipende fortemente dall'utilizzo che se ne deve fare. Ad esempio, se si vuole implementare un classificatore di testo con l'intento di suddividere automaticamente gli articoli in vendita all'interno di un e-commerce, è più semplice utilizzare una tassonomia personalizzata che si basi sulle categorie di articoli disponibili.

Un esempio concreto è la tassonomia utilizzata da Amazon per standardizzare le categorie di articoli disponibili all'interno della piattaforma.

Viceversa, se si realizza un classificatore che deve etichettare articoli interni ad una piattaforma fruitrice di notizie [di dominio pubblico], è preferibile utilizzare una tassonomia standard così da poterne facilitare la condivisione e la gestione.

Uno degli enti più noti che ha definito degli standard per tassonomie riguardanti notizie è stato IPTC, abbreviazione di International Press Telecommunications Council, un consorzio fondato nel 1965 con sede centrale a Londra che raggruppa le maggiori agenzie di stampa del mondo. Ad oggi i membri interni ad IPTC sono circa 70 tra cui possiamo citare ANSA (IT), BBC news (UK) e il New York Times (US).

L'obiettivo primario portato avanti negli anni da IPTC è quello di creare degli standard per la condivisione di notizie e di sviluppare metadati utili per descrivere e classificare notizie, testi, foto, video e altri media.

IPTC promuove i propri standard al fine di migliorare la gestione e lo scambio di informazioni tra fornitori di notizie, intermediari e consumatori. Dunque, attraverso essi è possibile semplificare il processo di scambio di informazioni indipendentemente dalla tipologia e dalla lingua utilizzata.

Ad oggi gli standard IPTC sono 16 e vengono continuamente aggiornati in modo tale da poter stare sempre al passo con le continue evoluzioni del mondo giornalistico.

Tra questi si possono citare:

- **Photo Metadata:** standard contenente i metadati utili per l'amministrazione, descrizione e gestione del copyright delle immagini. Photo Metadata è uno degli standard più utilizzati, grazie al consenso universale tra fotografi, testate giornalistiche, distributori e sviluppatori. All'interno di esso è definita la struttura, le proprietà e i campi dei metadati in modo che le immagini stesse siano descritte in maniera ottimale e facilmente accessibili. Uno standard simile al Photo-metadata è Video Metadata Hub che definisce la struttura, le proprietà e i metadati per i Video.
- **RightsML:** standard che permette di contrassegnare i diritti di copyright sul contenuto delle diverse tipologie di media, in maniera semplice e leggibile dalle macchine. Grazie a tali metadati è possibile trasmettere i diritti e le restrizioni associati ad un determinata risorsa.
- **Ninjs e NIFT:** Ninjs standard che permette di rappresentare le notizie in formato JSON, un mezzo di scambio di informazioni leggero e facile da analizzare sia per l'uomo che per la macchina. NIFT è simile al Ninjs differenziandosi unicamente per il formato utilizzato (XML).

- **R-news:** standard definito per l'utilizzo di markup semantico per annotare i metadati specifici delle notizie interne a pagine HTML.

Grazie ad esso è possibile definire le sezioni interne alle pagine HTML, attribuendone un significato specifico per ognuna di esse. Utile in quanto, se una macchina, che analizza pagine web, deve cercare il titolo dell'articolo, in mancanza di uno standard che ne definisca un identificatore, potrebbero essere create delle regole manuali. Come, ad esempio, l'identificazione del titolo come elemento definito attraverso un dato tag HTML o attraverso delle specifiche grafiche come la dimensione del carattere.

Poiché tali regole non sono uguali per tutti i siti web, la macchina non riuscirebbe ad estrarre sempre il contenuto che si stava cercando, dunque, diventa essenziale l'utilizzo di uno standard che ne definisca le sezioni.

- **EventsML-G2 e SportsML:** due standard che definiscono la struttura per la trasmissione di informazioni riguardanti rispettivamente eventi di qualsiasi tipo ed eventi sportivi.
- **Media Topics:** standard che definisce la tassonomia gerarchica per le notizie. Esso verrà approfondito nel paragrafo successivo.

5.2. Tassonomia Media Topics

Tra i vari standard definiti da IPTC, Media Topics è quello che definisce la tassonomia gerarchica più recente per le notizie di tipo testuale.

Tale tassonomia è stata rilasciata nel 2010 ed è una versione aggiornata dello standard Subject Codes.

Essa è composta da 1.200 termini suddivisi in 6 livelli che vanno dalla categoria più generica a quella più specifica. Al primo livello sono presenti 17 nodi, che rappresentano le macrocategorie generiche della tassonomia.

Il vocabolario della tassonomia Media Topics è un membro della famiglia di IPTC NewsCodes che contiene un set di termini condivisi, utili per descrivere notizie e contenuti multimediali di qualsiasi natura.

Nello specifico i nodi di primo livello sono i seguenti:



FIGURA 2: GRAFICO RAPPRESENTANTE I 17 NODI DI PRIMO LIVELLO DI MEDIA TOPICS

Essendo la versione aggiornata di Subject Codes, Media Topics contiene etichette più specifiche rispetto alla tassonomia precedente che si strutturava in 3 livelli implicando la compressione di molti termini.

Lo standard è in continuo aggiornamento; ciò permette di mantenere un grado di specificità che vada di pari passo con il mondo esterno.

Per ogni nodo della tassonomia, sono specificate le seguenti informazioni:

- concept ID, cioè il codice univoco assegnato al nodo;
- la data di creazione;
- la data dell'ultima modifica;
- il nome;
- la descrizione definita da IPTC;
- il link alla pagina Wikidata che definisce un'ulteriore descrizione;
- il codice relativo alla tassonomia SubjectCode se presente;
- codice del nodo correlato.

| Metadata about the scheme/controlled vocabulary | | | | |
|--|------------------------------------|-------------------------------------|----------|--|
| Scheme URI = http://cv.iptc.org/newscodes/mediatopic/ | | | | |
| Name(en-es): Media Topic | | | | |
| Definition(en-es): Indicates a subject of an item. | | | | |
| Note(en-es): The Media Topic NewsCodes has been IPTC's primary subject taxonomy since 2010, with a focus on classification of text. The development started with our previous Subject Codes taxonomy and extended the tree to 5 levels and reused the same 17 top level terms. The terms below the top level have been revised and rearranged. Most Media Topic concepts provide a mapping back to one of the Subject Codes, and many provide a mapping to Wikidata. | | | | |
| Concept ID (QCode) = medtsp:01000000, ID (URI) = http://cv.iptc.org/newscodes/mediatopic/01000000 | created: 2009-10-22T02:00:00+00:00 | modified: 2010-12-14T21:33:19+00:00 | retired: | |
| Name(en-es): arts, culture, entertainment and media | | | | |
| Definition(en-es): All forms of arts, entertainment, cultural heritage and media | | | | |
| Related concept (skos:exactMatch): http://cv.iptc.org/newscodes/mediatopic/01000000 | | | | |
| Member of scheme: http://cv.iptc.org/newscodes/mediatopic/ | | | | |
| Concept ID (QCode) = medtsp:20000002, ID (URI) = http://cv.iptc.org/newscodes/mediatopic/20000002 | created: 2009-10-22T02:00:00+00:00 | modified: 2010-12-14T21:33:19+00:00 | retired: | |
| Type: spmat:abstract | | | | |
| Name(en-es): arts and entertainment | | | | |
| Definition(en-es): All forms of arts and entertainment | | | | |
| Broader concept: http://cv.iptc.org/newscodes/mediatopic/01000000 | | | | |
| Related concept (skos:broader): http://cv.iptc.org/newscodes/mediatopic/01000000 | | | | |
| Related concept (skos:exactMatch): https://www.wikidata.org/entity/Q2018528 | | | | |
| Related concept (skos:broadMatch): http://cv.iptc.org/newscodes/mediatopic/01000000 | | | | |
| Member of scheme: http://cv.iptc.org/newscodes/mediatopic/ | | | | |

FIGURA 3: PRIMI DUE NODI DELLA TASSONOMIA MEDIA TOPICS

Nonostante Media Topics sia la tassonomia per notizie più recente, essa non è disponibile per la lingua italiana. Quindi, all'interno del team Neodata, i colleghi tirocinanti studenti del corso di Laurea Specialistica in Scienze del Testo per le professioni digitali si sono occupati di tradurre la tassonomia e di riorganizzarla per il contesto italiano.

Ciò è stato determinante in quanto alcuni elementi della tassonomia non sono presenti nel contesto delle notizie italiane. Ad esempio, la macrocategoria *“Stili di vita e tempo libero”* è stata eliminata e i vari nodi interni ad essa sono stati inseriti all'interno di altre macrocategorie.

5.3. Estrazione automatica delle informazioni dalla tassonomia

In questo paragrafo verranno discusse le tecniche e i meccanismi implementati per l'estrazione automatica delle keywords e delle descrizioni da una tassonomia.

Scaricata e tradotta la tassonomia su cui si è deciso di basare il classificatore finale, essa è stata convertita dal formato *.xls (estensione Excel)* in formato *JSON (JavaScript Object Notation)*, così da poterla importare nel codice e gestirla in maniera standardizzata.

```
{
  "CMT": "01000000",
  "Nodo": "arte, cultura, intrattenimento e media",
  "Descrizione": "Tutte le forme di arte, di intrattenimento, del patrimonio culturale e dei media",
  "Lvl": "1"
},
{
  "CMT": "20000002",
  "Nodo": "arte e intrattenimento",
  "Descrizione": "Tutte le forme di arte e di intrattenimento",
  "Lvl": "2"
},
{
  "CMT": "20000003",
  "Nodo": "animazione",
  "Descrizione": "Narrazione di una storia attraverso disegni animati, sia lungometraggi che cortometraggi",
  "Lvl": "3"
},
{
  "CMT": "20001135",
  "Nodo": "mostre d'arte",
  "Descrizione": "Esposizione temporanea di opere d'arte in musei, saloni d'arte o gallerie d'arte",
  "Lvl": "3"
},
}
```

FIGURA 4: PRIMI QUATTRO NODI DELLA TASSONOMIA MEDIA TOPICS

All'interno del file JSON, per ogni elemento della tassonomia, è specificato:

- il codice univoco che lo rappresenta;
- il nome della categoria;
- la descrizione fornita da IPTC;
- il livello gerarchico di appartenenza.

Acquisite tali informazioni, è stato realizzato un algoritmo che legge il file JSON e salva le informazioni all'interno di un DataFrame Python che permette di gestire in maniera semplice ed intuitiva i dati inseriti.

Alle informazioni di base, per ogni nodo della tassonomia, sono stati inseriti altri importanti attributi “derivati” fondamentali per gli steps successivi.

Nello specifico le informazioni aggiuntive sono state:

- **Nodo padre:** poiché nel file scaricato rappresentante la tassonomia non si ha nessuna informazione riguardo le associazioni della gerarchia, se non per il campo che identifica il livello di appartenenza di ogni nodo, è stato necessario definire l’associazione tra quest’ultimo e il suo superiore (padre).

Dunque, per ogni elemento della tassonomia è stato inserito all’interno del DataFrame il codice univoco del nodo padre, così da poter navigare facilmente all’interno della gerarchia.

Ciò è stato implementato scorrendo la lista dei nodi e per ogni elemento è stato analizzato il suo livello. Preso tale valore, è stato assegnato come nodo padre il primo nodo precedente con livello pari al livello dell’elemento che si sta analizzando meno 1.

$$\forall \text{ nodo } N_i \text{ con Livello } L_i, \\ \text{nodo}_{\text{padre}} = \begin{cases} \text{il primo } N_j \text{ con } L_j = L_i - 1, \text{ per } j = i - 1, i - 2, \dots, \text{ se } L_i > 1 \\ 0, & \text{se } L_i = 1 \end{cases}$$

EQUAZIONE 1: FORMULA CHE DEFINISCE L’ASSEGNAZIONE DEL NODO PADRE

A livello pratico è stato sviluppato un processo iterativo che analizza i nodi dell’albero gerarchico e i suoi predecessori al fine di confrontarne il livello d’appartenenza e definirne le relazioni padre-figlio. In *figura 5* è mostrato il codice utile a tale scopo.

```
def TrovaPadre(lvl_list,CMT_list):

    index=0
    padreLv11,padreLv12,padreLv13,padreLv14,padreLv15="","","","",""
    padre_list=[]
    for item in lvl_list:
        if item=="1":
            padre_list.insert(index,"0")
            padreLv11=CMT_list[index]
        elif item=="2":
            padre_list.insert(index,padreLv11)
            padreLv12=CMT_list[index]
        elif item=="3":
            padre_list.insert(index,padreLv12)
            padreLv13=CMT_list[index]
        elif item=="4":
            padre_list.insert(index,padreLv13)
            padreLv14=CMT_list[index]
        elif item=="5":
            padre_list.insert(index,padreLv14)
            padreLv15=CMT_list[index]
        elif item=="6":
            padre_list.insert(index,padreLv15)
        index=index+1

    return padre_list
```

FIGURA 5: CODICE CHE MOSTRA IL METODO CHE SI OCCUPA DI DEFINIRE IL PADRE DI OGNI NODO DELLA TASSONOMIA.

- **Keywords:** questo è l'attributo più importante poiché contiene la stringa che viene utilizzata per la ricerca degli articoli da associare alla categoria analizzata.

Per ricavare tale stringa, è stato implementato un algoritmo che scorre i nodi interni alla tassonomia e che per ognuno di esso analizzi il nome del nodo ed esegua i seguenti controlli:

- Se esso non contiene i caratteri ",", "e" o "o" al suo interno, la keyword sarà composta solo dal nome del nodo; ciò significa che esso è composto da un singolo concetto, ad esempio i nodi *musica* o *genere musicale*.
- Se esso contiene ",", "e" o "o" essi verranno sostituiti con la parola chiave "OR" che sarà utile nella sezione di ricerca delle notizie; ciò significa che il nodo è composto da più di un concetto, ad esempio il nodo *"arte, cultura, intrattenimento e media"* diventerà *"arte OR cultura OR intrattenimento OR media"*. La parola chiave "OR" sarà utile durante la fase di ricerca automatica degli articoli che comporranno il dataset. Essa verrà discussa nel capitolo successivo.

Tale processo è stato implementato attraverso il seguente algoritmo.

Per ogni nodo della tassonomia è stata definita una stringa, che conterrà le keywords relative ad esso, inizializzata a "intitle:". Il significato della parola chiave "intitle" verrà definito nel capitolo successivo in quanto sarà utile durante la fase di ricerca degli articoli. Successivamente è stato analizzato il nome del nodo e si è verificato se, al suo interno sono presenti degli spazi; se così fosse, le istruzioni eseguite sarebbero le seguenti.

In primis si sostituiscono al nome del nodo, se presenti, tutte le "e" e le "o" con il carattere ",", dopodiché tutti i caratteri "," vengono rimpiazzati con la parola chiave "OR" ed infine la stringa ottenuta viene aggiunta alla stringa contenente le keywords.

Viceversa, invece, se non sono presenti degli spazi nel nome del nodo, esso viene aggiunto alla stringa contenente le keywords senza ulteriori controlli. Fatto ciò, in entrambi i casi si aggiunge il carattere ")", così da chiudere la stringa keywords.

```
for nodo in nodo_list:
    stringDoc=""
    stringDoc+="intitle:("
    if(Contains_whitespace(nodo)):
        nodo=nodo.replace(" e ", ", ")
        nodo=nodo.replace(" o ", ", ")
        nodo=nodo.replace(",", "OR")
        stringDoc+=nodo
    else:
        stringDoc+=nodo
    stringDoc+=")"
    keyList.append(stringDoc)
```

FIGURA 6: SERIE DI ISTRUZIONI CHE PERMETTONO LA DEFINIZIONE DEL CAMPO KEYWORDS

- **Path:** questo attributo identifica il percorso gerarchico di ogni nodo dalla root.
- Esso è stato utilizzato nella fase di creazione delle directory che compongono il dataset ed è stato definito confrontando il livello d'appartenenza del nodo che si sta analizzando con il livello del nodo precedente.

Inizializzata la stringa *path* con il percorso da cui si vuole partire, considerando che, alla prima iterazione il *livello precedente* è settato a 0 e che il *codice nodo precedente* è settato a stringa vuota, l'attributo path per ogni nodo è stato così calcolato:

- Se il livello del nodo analizzato è maggiore del livello del nodo precedente si aggiunge alla stringa *path* il suffisso delimitatore "/" e il "*codice del nodo che si sta analizzando*", vedi *figura 7*.

- Se il livello del nodo analizzato è uguale al livello del nodo precedente, in questo caso, i due nodi sono fratelli.

Dunque, considerata la stringa *path*, si è estratta la sottostringa dal carattere iniziale a quello che precede il codice del nodo precedente. Ad essa è stata aggiunta il codice del nodo che si sta analizzando, vedi *figura 8*.

- Se il livello del nodo analizzato è minore al livello del nodo precedente significa che il nodo analizzato sta al di fuori del ramo gerarchico del nodo precedente. Per definire tale path, si calcola la differenza tra il livello del nodo che si sta analizzando e il livello del nodo precedente; tale valore, incrementato di 1, è stato utilizzato per capire lo spostamento all'interno del path per trovare il padre del nodo esaminato.

Fatto ciò, si aggiunge il codice del nodo analizzato alla stringa *path*, vedi *figura 9*.

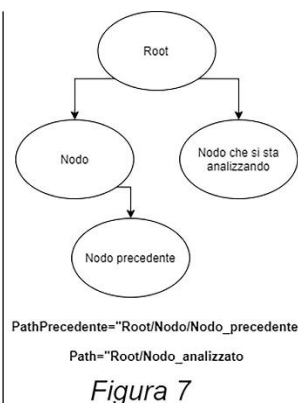
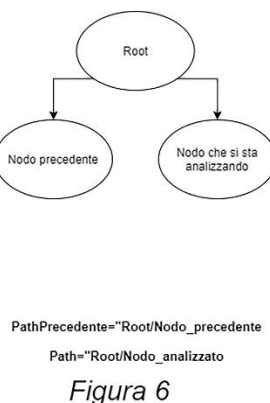
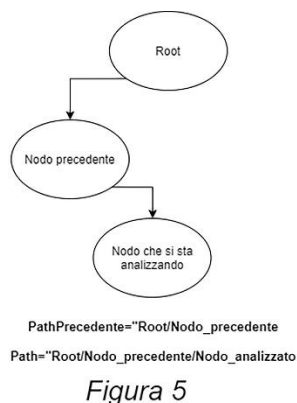


FIGURA 7

FIGURA 8

FIGURA 9

In figura 10 è mostrato il codice Python che racchiude le tre casistiche sopra definite per la definizione dell'attributo path.

Nello specifico, definite e inizializzate le variabili d'appoggio "previous_lvl" e "previous_node", per ogni nodo della tassonomia, sono stati presi in considerazione il livello d'appartenenza e il suo codice univoco. Dunque, per ogni elemento è stato confrontato il suo livello con quello del nodo precedente; tale confronto come spiegato poc'anzi, può avere tre esiti, identificati nel codice con la sequenza if-elif-elif.

```

previous_lvl=0
previous_node=""
list_Path=[]
df_final=df[['CMT','Lvl']]
for index,item in df_final.iterrows():
    if(int(item["Lvl"])>previous_lvl):
        path=path + "/" + item["CMT"]

    elif (int(item["Lvl"])==previous_lvl):
        str_tmp=path
        x=len(previous_node)
        path=str_tmp[:-x]
        path=path + item["CMT"]

    elif (int(item["Lvl"])<previous_lvl):
        diff=previous_lvl-int(item["Lvl"])
        diff=diff+1
        index_tmp=0
        str_tmp=path[::-1]
        for i in range(diff):
            index_tmp=str_tmp.find("/")
            str_tmp=str_tmp[index_tmp+1:]
        path=str_tmp[::-1]
        path=path + "/" + item["CMT"]
    previous_lvl=int(item["Lvl"])
    previous_node=item["CMT"]

list_Path.append(path)

```

FIGURA 10: CODICE IMPLEMENTATO PER DEFINIRE IL PATH DI OGNI NODO

L'inserimento di tali informazioni *"derivate"*, quali *nodo padre* e *path*, dipende fortemente dal fatto che i nodi all'interno del file JSON sono definiti in successione. Ad esempio, il nodo padre di un figlio di livello X è il primo nodo che tra i nodi precedenti ha livello X-1.

Ciò non è stata una scelta progettuale, bensì è lo standard Media Topics a fornire la tassonomia senza nessuna informazione riguardante il nodo padre, mettendo a disposizione solamente l'informazione riguardante il livello d'appartenenza.

Di seguito sono riportati i primi nodi della tassonomia con i relativi attributi di base e derivati.

| | CHT | Nodo | Descrizione |
|---|----------|--|--|
| 0 | 01000000 | arte, cultura, intrattenimento e media | Tutte le forme di arte, di intrattenimento, del patrimonio culturale e dei media |
| 1 | 20000002 | arte e intrattenimento | Tutte le forme di arte e di intrattenimento |
| 2 | 20000003 | animazione | Narrazione di una storia attraverso disegni animati, sia lungometraggi che cortometraggi |
| 3 | 20001135 | mostre d'arte | Esposizione temporanea di opere d'arte in musei, saloni d'arte o gallerie d'arte |
| 4 | 20000004 | strisce e fumetto | Disegni, come strisce e fumetti, che spesso adottano la satira e l'umorismo |

| Lvl | Padre | Keywords | Path |
|-----|----------|---|---|
| 1 | 0 | intitle:(arte OR cultura OR intrattenimento OR media) | ./classificatore/LearningSet/01000000 |
| 2 | 01000000 | intitle:(arte OR intrattenimento) | ./classificatore/LearningSet/01000000/20000002 |
| 3 | 20000002 | intitle:(animazione) | ./classificatore/LearningSet/01000000/20000002/20000003 |
| 3 | 20000002 | intitle:(mostre d'arte) | ./classificatore/LearningSet/01000000/20000002/20001135 |
| 3 | 20000002 | intitle:(strisce OR fumetto) | ./classificatore/LearningSet/01000000/20000002/20000004 |

FIGURA 11: ATTRIBUTI RIGUARDANTI I PRIMI NODI DELLA TASSONOMIA MEDIA TOPICS

Capitolo 6

6. Creazione Dataset

6.1. Definizione e importanza dei Dataset

Uno degli obiettivi posti in essere è quello di spiegare come venga creato, in maniera automatica, il dataset utile al classificatore di testo.

Per prima cosa è bene descrivere cosa sia un dataset e la sua relativa importanza.

Un dataset è un insieme di dati discreti appartenenti alla stessa natura.

Lo scopo di essi è quello di raccogliere dati su cui è possibile definire proprietà o attributi; ciò permette di studiare un certo fenomeno che si basa proprio sui valori di tali proprietà. Più esso sarà grande più saranno precise le analisi eseguite.

I dataset possono essere suddivisi in tre categorie:

- **Strutturati:** dati memorizzati seguendo un determinato schema precedentemente definito; solitamente essi sono organizzati in formato tabellare dove ogni colonna della tabella rappresenta un attributo del dato.
- **Non strutturati:** dati memorizzati che non seguono uno schema definito, poiché, per via della loro natura non possono essere organizzati in delle strutture; ad esempio, dati di tipo immagine, audio o video.
- **Semi-strutturati:** dati che possono essere organizzati in delle strutture logiche, ma che non posseggono dei limiti strutturali, ad esempio file XML e JSON.

I dataset sono fondamentali nel processo di creazione dei classificatori, poiché si collocano alla base della creazione dei modelli di apprendimento automatico.

6.2. Creazione automatica del dataset:

6.2.1. Dataset per le notizie italiane

Definita l'importanza del dataset all'interno del processo di creazione di un classificatore sono di seguito descritte le scelte che sono state prese per la creazione del dataset che è stato utilizzato all'interno del progetto.

In primis bisogna specificare la motivazione che ha portato alla scelta di creare un proprio dataset piuttosto che utilizzarne uno da terzi.

La scelta di creare un proprio dataset da zero è stata presa in considerazione poiché, su internet sono pochissime le risorse disponibili per la lingua italiana e nessuna di esse è utilizzabile per l'apprendimento di un classificatore di testo.

Inoltre, considerando la realizzazione di un modello di classificazione che si basi su di una tassonomia specifica come Media Topics, non disponibile in lingua italiana, non vi era la certezza di trovare un dataset ad hoc.

Considerando che, nessuna delle testate giornalistiche membri/partner di IPTC utilizza la tassonomia Media Topics, non è stato possibile creare un dataset che si fondasse sulle loro categorizzazioni degli articoli.

Dunque, passeremo alla spiegazione di come venga realizzato il dataset.

La logica che ne sta alla base è che una serie di dati possa essere raggruppata per una determinata caratteristica. Nel nostro caso, essendo l'output del classificatore il codice univoco definito dallo standard Media Topics per identificare una categoria, si è pensato di creare un dataset che per ogni nodo della tassonomia contenga tutti gli articoli etichettati con quella categoria.

Dal momento che il numero di nodi interni alla tassonomia è molto elevato si è fin da subito pensato di automatizzare il processo di ricerca.

Per fare ciò, si è considerato di definire una keyword di ricerca per ogni nodo della tassonomia, (il processo di creazione delle keywords è già stato trattato nel **paragrafo 5.3**), da utilizzare per la ricerca degli articoli sulle piattaforme italiane fruitrici di notizie.

Se la logica fosse quella di associare un numero variabile di articoli ad ogni categoria definita da Media Topics, ciò implicherebbe che il numero di piattaforme fruitrici di notizie da considerare sarebbe altrettanto elevato.

Per ridurre il numero di ricerche, si è pensato di utilizzare una piattaforma unica che raggruppasse le notizie, nello specifico si è scelta la piattaforma Google News fornita da Google.

Grazie al servizio offerto da esso è possibile trovare, in unica piattaforma, tutte le notizie provenienti dalle principali testate giornalistiche italiane; ciò è stato di fondamentale importanza nella creazione del dataset, poiché grazie a Google News è stato possibile effettuare le ricerche degli articoli per ogni nodo della tassonomia.

6.2.2. Libreria GoogleNews

Una volta definita la piattaforma sulla quale ricercare le notizie, è stato necessario automatizzare tale ricerca.

Per fare ciò è stata utilizzata *GoogleNews*, una libreria per il linguaggio di programmazione Python, la quale fornisce dei metodi per automatizzare le ricerche in maniera semplice e veloce.

Prima di entrare nel dettaglio sul funzionamento di tale libreria, si definisce cosa si intende con il termine librerie in informatica.

Librerie (software): esse sono dei moduli su cui sono definite un insieme di procedure atte all'espletamento di un dato compito.

I vantaggi nell'uso di librerie all'interno di un software sono:

- **Suddivisione logica e fisica del codice:** definendo una libreria si possono estrarre e raggruppare tutti i metodi definiti all'interno del software che risolvono un dato problema specifico, con il vantaggio di modificare ciò che compone la libreria stessa distintamente dal codice del software, permettendo una manutenzione più semplice e veloce.
- **Riuso del codice:** definendo una libreria, i vari metodi al suo interno potranno essere richiamati dall'esterno tramite una semplice chiamata al metodo, permettendo di evitare la riscrittura dell'intera sezione di istruzioni e riducendo il numero di righe finali di codice all'interno del software.

Nel caso specifico della libreria *GoogleNews*, i metodi al suo interno definiti permettono di estrarre informazioni dalla piattaforma Google News e basteranno poche righe di codice per avere la lista di articoli correlati alla ricerca che si sta effettuando.

L'estrazione delle informazioni avviene tramite un algoritmo il cui funzionamento è il seguente: data una determinata keyword da cercare sulla piattaforma Google News, i metodi interni alla libreria si occupano di comporre uno specifico link che contenga la keyword ricercata.

Ad esempio: "<https://news.google.com/search?q=Simpson&hl=it>".

Dunque, definito il link, la libreria si occupa di eseguire una richiesta HTTP, passando come Url il link dapprima composto; tale richiesta ritornerà il codice HTML della pagina Google News al cui interno sono presenti tutti gli articoli relativi alla keyword scelta, vedi *Figura 12*.

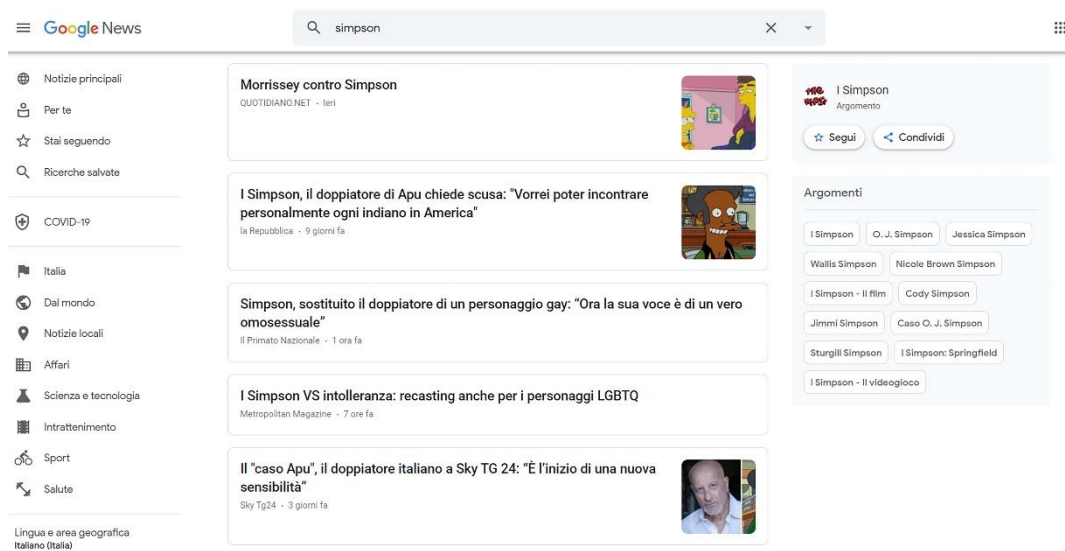


FIGURA 12: ARTICOLI MOSTRATI DA GOOGLE NEWS DATA LA CHIAVE DI RICERCA "SIMPSON"

Ottenuto il codice HTML della pagina, i metodi della libreria GoogleNews si occupano di estrarre le informazioni relative ad ogni articolo presente tramite l'approccio di Web-Scraping, che verrà definito nel dettaglio nel paragrafo successivo.

In *figura 13* si vuole mostrare un esempio di utilizzo della libreria `GoogleNews` (per installare il modulo `GoogleNews` basterà lanciare il seguente comando nel terminale Python: `pip install GoogleNews`), la quale, al suo interno definisce la classe `GoogleNews`.

Ciò implica che la prima istruzione, dopo l'importazione della libreria, è la creazione di un'istanza dell'oggetto al cui costruttore possano essere passati dei parametri, quali ad esempio: la lingua, il periodo di ricerca e la codifica, che altrimenti verrebbero settati con valori di default.

Una volta creato l'oggetto, si richiama il metodo `get_news()` che prende come parametro di riferimento una stringa rappresentante la key da ricercare su Google News.

Le informazioni relative agli articoli trovati in seguito alla ricerca saranno disponibili richiamando il metodo `results()` al quale è possibile passare il parametro `"sort=True"`, affinché gli articoli siano ordinati per data di pubblicazione.

```
from GoogleNews import GoogleNews

googlenews = GoogleNews(lang='it',period='180d',encode='utf-8')
googlenews.get_news('Simpson')
results_gnews=googlenews.results(sort=True)
```

FIGURA 13: CODICE PYTHON CHE SIMULA L'UTILIZZO DELLA LIBRERIA `GOOGLENEWS`.

```
https://news.google.com/search?q=Simpson+when:180d&hl=it
{'date': '7 ore fa',
 'datetime': datetime.datetime(2021, 4, 7, 10, 2, 56),
 'desc': "I videogiochi e i Simpson, un'accoppiata vincente. Non solo Homer e "
         'famiglia sono diventati protagonisti di tutta una serie di '
         'videogame, ma nelle tante stagioni ...',
 'img': 'https://lh3.googleusercontent.com/Hwp5a1ZV917Tr1I-6eatUA3-vs7_ny7EhJhLRcraQM8ffGR8VZw520Rm6cgS0B10xhQgrsAf3v0Xz1cS3E8=-p-df-h100-w100',
 'link': 'news.google.com/./articles/CATiEPd78nBHxsBdb403o1nqqQMqGQgEKhAIACoHCAowlDqGCzDy4IQMKAJowY?hl=it&gl=IT&ceid=IT%3Ait',
 'media': None,
 'site': 'GQ Italia',
 'title': 'I videogiochi inventati dai Simpson che ci piacerebbe tanto provare'}
```

FIGURA 14: OUTPUT DATO DALL'ESECUZIONE DEL CODICE ILLUSTRATO IN FIGURA 13

L'output mostrato in *figura 14* è composto da una serie di informazioni riguardanti un articolo.

Nel dettaglio i dati ritornati riguardano: il tempo passato dalla pubblicazione dell'articolo; la data di pubblicazione stessa; la descrizione dell'articolo; il link dell'immagine principale; il link al sito contenente l'articolo; il nome del sito stesso e il titolo dell'articolo.

All'interno del software del progetto è stata realizzata un'ulteriore libreria, vedi *figura 15*, al cui interno è stata definita la classe *GoogleScraping* per creare un unico metodo che quando richiamato permette: di inizializzare un oggetto di tipo *GoogleNews*, far partire le ricerche di notizie su Google News per una data keyword e di ritornare la lista dei links relativi agli articoli trovati.

```
from bin.CreateLearningSet.GoogleNewsLib import GoogleNews
class GoogleScraping:
    def __init__(self,period="180d"):
        self.lang="it"
        self.period=period #default
        self.encode="utf-8"
        self.googleObj = None
        self.googleObj=GoogleNews( lang=self.lang,
                                   period=self.period,
                                   encode=self.encode)

    def GetLinks(self,results):
        links=[]
        for num,page in enumerate(results):
            links.append("https://"+page['link'])
        return links

    def GetNews(self,keywords):
        self.__init__()
        self.googleObj.get_news(keywords)
        results_gnews=self.googleObj.results(sort=True)
        links=self.GetLinks(results_gnews)
        return links
```

FIGURA 15: CODICE UTILIZZATO PER RAGGRUPPARE TUTTE LE CHIAMATE UTILE DELLA LIBRERIA *GOOGLENEWS*

6.2.3. Algoritmo sviluppato

Definiti gli strumenti utili per automatizzare la ricerca degli articoli, si può passare alla spiegazione dell'algoritmo che è stato implementato per svolgere tale operazione.

- Per ogni nodo della tassonomia è stata presa la lista di keywords definita in precedenza nel **paragrafo 5.3**, alle quali sono state inserite delle parole chiave: *"intitle"* e *"OR"* che rappresentano degli operatori logici utilizzabili nelle ricerche su Google News. Nello specifico:
 - **intitle("key")**: operatore che permette di ricercare su Google News tutti gli articoli che contengono nel titolo le parole definite all'interno delle parentesi tonde. Ad esempio: *"intitle:(musica)"* ritornerà tutti gli articoli che contengono la parola *musica* all'interno del titolo.
 - **OR**: operatore che permette di combinare più parole da ricercare negli articoli. Ad esempio: la ricerca di *"musica OR teatro"* darà come risultato tutti gli articoli relativi alla parola *"musica"* o *"teatro"* o entrambe.
 - **AND**: operatore che permette di fare delle ricerche di articoli combinando più parole. Ad esempio *"musica AND teatro"*, ciò permette di ricercare tutti gli articoli relativi ad entrambe le parole.
L'operatore AND si può omettere poiché Google News automaticamente sostituisce lo spazio tra due parole vicine con la parola chiave "AND".
 - **When**: operatore che permette di filtrare gli articoli in base alla data di pubblicazione. Ad esempio *"musica when:180d"*, permette di avere tutti gli articoli correlati alla ricerca della parola chiave musica che siano stati pubblicati non oltre 180 giorni da quando si è fatta partire la ricerca.

Un esempio di keyword utilizzata per la ricerca di articoli correlati ad una data categoria della tassonomia Media Topics è mostrata in *figura 16*.

```
intitle:(arte OR cultura OR intrattenimento OR media)
```

FIGURA 16: KEYWORD UTILIZZATA PER LA RICERCA DEGLI ARTICOLI RELATIVI ALLA CATEGORIA “ARTE, CULTURA, INTRATTENIMENTO E MEDIA”

- Definite tutte le stringhe di ricerca, per ogni nodo della tassonomia, si è richiamato il metodo `getNews()`, dichiarato all'interno della classe mostrata in *figura 15*, il quale, data una key, ne ritorna tutti i link degli articoli correlati estratti da Google News

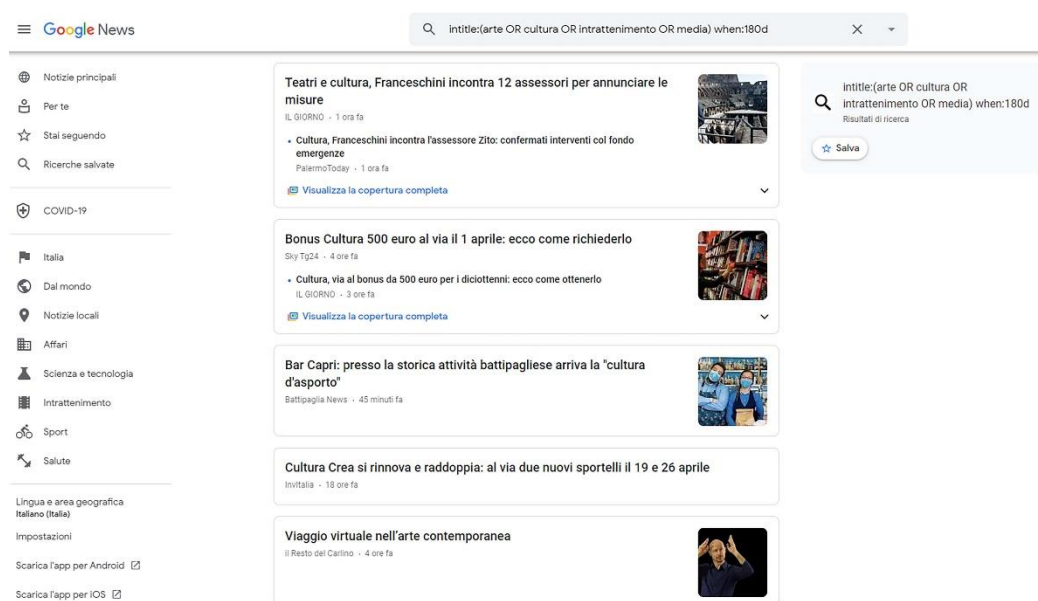


FIGURA 17: ALCUNI DEGLI ARTICOLI RITORNATI DA GOOGLE NEWS CERCANDO LA STRINGA “INTITLE:(ARTE OR CULTURA OR INTRATTENIMENTO OR MEDIA) WHEN:180D”.

- Una volta ottenuta la lista dei link degli articoli, si è effettuata la tecnica del Web-Scraping al fine di estrarre i testi delle notizie contenute all'interno delle pagine web.

Con Web-Scraping si indica la tecnica informatica che permette di estrarre informazioni da un sito web tramite programmi software per scopi di analisi e valutazione.

Per effettuare lo scraping delle pagine contenenti gli articoli, sono state utilizzate le librerie Python Request e BeautifulSoup, grazie alle quali è stato possibile ottenere il codice HTML delle pagine web ed estrarne il testo.

Nello specifico, per ogni link si è richiamato il metodo *request.get(url)* della libreria Request, il quale, dato un Url, ne estrae il contenuto della pagina.

Ottenuto il codice HTML della pagina contenente l'articolo, si è passato all'esecuzione del metodo *BeautifulSoup(content,"html.parser)* della libreria BeautifulSoup, che, preso del codice HTML ne crea automaticamente un oggetto su cui sono definiti diversi metodi per l'accesso diretto ai tag che compongono la pagina web.

Per ogni oggetto creato con *BeautifulSoup* sono state eseguite le seguenti operazioni:

- Rimozione del contenuto interno ai tag *<footer> </footer>* (essi contengono le informazioni di contatto o di riepilogo del sito) e di tutti gli elementi che hanno all'interno del nome della classe CSS la parola *"hidden"*, così da non considerare il testo interno a sezioni nascoste della pagina.
- Estrazione del titolo, descrizione e keywords dalla pagina HTML: tali attributi è possibile trovarli nella maggior parte delle pagine web e sono di norma situati all'interno della sezione *<head>* (sezione nella quale si inseriscono tutte le informazioni relative al sito) e nello specifico nel *content* dei tag *<meta>* (i tag meta contengono i metadati relativi ad un documento HTML).

Una volta estratto il contenuto è stato applicato un algoritmo che: ha rimosso le parole inutili e la punteggiatura, ha suddiviso in singole parole il contenuto, inserendole all'interno di una lista ed escludendone i duplicati.

- Dopo aver ottenuto la lista definita nello step precedente, sono stati estratti dall'oggetto *BeautifulSoup* tutti i testi interni ai tag *<p>*, i quali di norma contengono il testo di un paragrafo.

Per ogni testo, si è applicato un filtro interno che controlla se il testo passato ha una lunghezza maggiore di 35 caratteri e se l'applicazione della seguente espressione regolare

`"[.\s/[cC]ookie[s]*/©/@[Cc]opyright/http[s]*:"` non riporti nessun match.

Tale filtro viene applicato dato che, esaminare un testo con lunghezza inferiore ai 35 caratteri si è dimostrato essere inutile perché si considererebbero testi inutili quali inserzioni o testi appartenenti alle sezioni che definiscono la struttura del sito.

L'applicazione dell'espressione regolare si è dimostrata essere utilissima nel caso in cui la pagina web non definisca la sezione footer con i tag *<footer>*, poiché grazie al controllo è possibile capire se il testo che si sta analizzando contenga parole chiave di norma presenti in un footer.

- Se il testo risulta idoneo al primo filtro viene applicato un ulteriore controllo che verifica se all'interno del testo sia presente almeno una delle parole appartenenti alla lista di parole estratte da titolo, descrizione e keywords.
- Ogni testo estratto dalla pagina web che passa entrambi i controlli viene scritto in un file di testo salvato in locale nel percorso definito dall'attributo *path* (attributo definito nel **paragrafo 5.3** che indica il path interno alla gerarchia di directory).

Tale file ha come denominazione la data in cui esso viene creato, seguita da un valore auto-incrementante. Esso conterrà oltre al testo estratto anche il titolo, la descrizione e le keywords definite all'interno della pagina web.

La scelta di settare il nome del documento con “data + indice” è stata implementata per eventuali aggiornamenti futuri del dataset.

- Per ogni file creato in locale, viene applicato un ulteriore controllo che consiste nel verificare che la dimensione del file sia maggiore di 0 (file vuoto) e minore di 20 KB, valore elevato nella maggior parte dei casi che implica l'estrazione di un testo errato. Medesimo discorso per quanto riguarda i file che contengono più di 50 righe.

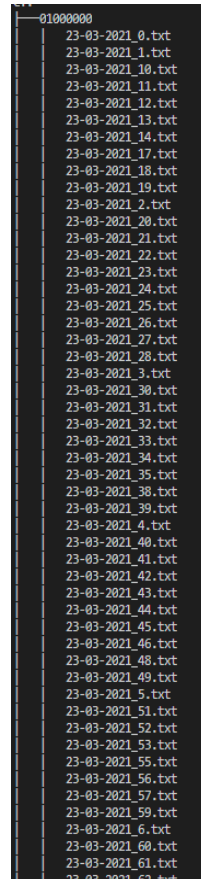
I valori precedenti sono stati definiti a seguito di diversi test.

- Il risultato dell'esecuzione dell'algoritmo che si occupa degli steps precedenti produce una serie di file di testo contenente ognuno una notizia appartenente ad un determinato nodo della tassonomia.

```
Link: https://milano.repubblica.it/cronaca/2021/04/22/news/gorle\_campo\_da\_basket\_street\_art-297523634/  
Titolo: campo basket diventa opera arte Repubblica  
Keywords: basket lucia provincia bergamo Landoni  
Descrizione: paese street artist realizzato opera ambito dell  
Il campo da basket di Gorle (nella Bergamasca) si è trasformato in un'opera d'arte: da qualche giorno tra i  
due canestri è apparsa l'immagine di una Calathea Makoyana, "una grande pianta purificatrice dell'aria capace  
di trasformare le sostanze nocive in ossigeno". Merito dell'artista torinese Fabio Petani, che ha realizzato l'opera  
"Carbon Dioxide & Calathea Makoyana" nell'ambito dello StreetArtBall Project, nato per coniugare street art e sport a  
favore della rigenerazione urbana in vari Comuni della provincia di Bergamo.  
"Si tratta di un'iniziativa che veicola due messaggi molto importanti: da un lato la rinascita dopo il terribile periodo  
della pandemia, da cui la nostra zona è stata colpita con particolare durezza, e dall'altro l'attenzione all'ambiente -  
sottolinea il sindaco Giovanni Testa - Il campo da basket, uno dei luoghi di aggregazione più frequentati e amati dai nostri  
ragazzi, assume così un'ulteriore valenza educativa. Oltre a trasmettere i valori dello sport, ricorda ai ragazzi l'importanza  
della sostenibilità ambientale". Temi importanti, che "solo l'arte riesce a sintetizzare con questa chiarezza e immediatezza  
- continua il primo cittadino - Per Gorle è un onore far parte dei cinque Comuni selezionati per partecipare allo StreetArtBall Project".  
Sulla pagina Facebook ufficiale del progetto si trova la spiegazione dell'opera, che segue "la linea artistica e concettuale  
di tutti i lavori realizzati da Fabio Petani in giro per il mondo. I titoli delle sue opere si compongono sempre di un elemento  
chimico e di una pianta, entrambi citati nella loro versione latina. L'arte, in tutte le sue forme, trasmette messaggi. Se poi  
questi messaggi 'rimangono vivi' grazie a opere di arte urbana, a cielo aperto, gratuite e addirittura usufruibili, acquistano ancora più forza".  
Soprattutto alla luce del fatto che il campo da basket di Gorle si trova in una zona del paese proiettata al futuro: "Lì vicino c'è  
un parco dove vengono piantati gli 'alberi della vita', ovvero le piantine donate dall'amministrazione comunale alle famiglie di  
tutti i neonati di Gorle - conclude Giovanni Testa - Quei bambini rappresentano il futuro della nostra comunità e stiamo lavorando  
per fare in modo che quel futuro sia il più possibile green".
```

**FIGURA 18: FIGURA MOSTRANTE UN ESEMPIO DI TESTO ESTRATTO DA UNA PAGINA WEB
CONTENENTE UN ARTICOLO RELATIVO ALLA CATEGORIA “ARTE, CULTURA,
INTRATTENIMENTO E MEDIA”.**

Il risultato delle precedenti operazioni è stato un dataset composto da 864 directory, ognuna rappresentante una categoria della tassonomia, al cui interno sono stati salvati 56.162 file, ognuno contenente un articolo relativo ad una determinata categoria.



```
01000000
23-03-2021_0.txt
23-03-2021_1.txt
23-03-2021_10.txt
23-03-2021_11.txt
23-03-2021_12.txt
23-03-2021_13.txt
23-03-2021_14.txt
23-03-2021_17.txt
23-03-2021_18.txt
23-03-2021_19.txt
23-03-2021_2.txt
23-03-2021_20.txt
23-03-2021_21.txt
23-03-2021_22.txt
23-03-2021_23.txt
23-03-2021_24.txt
23-03-2021_25.txt
23-03-2021_26.txt
23-03-2021_27.txt
23-03-2021_28.txt
23-03-2021_3.txt
23-03-2021_30.txt
23-03-2021_31.txt
23-03-2021_32.txt
23-03-2021_33.txt
23-03-2021_34.txt
23-03-2021_35.txt
23-03-2021_38.txt
23-03-2021_39.txt
23-03-2021_4.txt
23-03-2021_40.txt
23-03-2021_41.txt
23-03-2021_42.txt
23-03-2021_43.txt
23-03-2021_44.txt
23-03-2021_45.txt
23-03-2021_46.txt
23-03-2021_48.txt
23-03-2021_49.txt
23-03-2021_5.txt
23-03-2021_51.txt
23-03-2021_52.txt
23-03-2021_53.txt
23-03-2021_55.txt
23-03-2021_56.txt
23-03-2021_57.txt
23-03-2021_59.txt
23-03-2021_6.txt
23-03-2021_60.txt
23-03-2021_61.txt
23-03-2021_62.txt
```

FIGURA 19: FIGURA MOSTRANTE PARTE DEL DATASET

Capitolo 7

7.Preprocessing e suddivisione del testo

7.1. Preprocessing del Dataset: Pipeline NPL

In questo paragrafo verrà definito il processo di pulizia del dataset al fine di renderlo processabile dalla macchina ed utilizzabile per la fase di apprendimento e testing del modello di classificazione.

Per implementare tale processo è stato realizzato un modulo Python nel quale sono state utilizzate delle tecniche di pulizia del testo interne alla pipeline NLP.

Con NLP [Natural Language Processing] si intende il processo di elaborazione automatica di informazioni scritte o parlate utilizzando un calcolatore elettronico. Esso è di fondamentale importanza quando bisogna realizzare un software che deve rendere processabile e comprensibile, un testo scritto in linguaggio naturale, quindi, non strutturato, da una macchina.

Al giorno d'oggi, le applicazioni del NLP sono molte e utilizzate in diversi contesti come ad esempio:

- **Riconoscimento ottico dei caratteri (OCR):** data un'immagine, con del testo stampato, attraverso l'elaborazione del testo, se ne può estrarre e digitalizzare il contenuto. Le applicazioni dell'OCR sono svariate come ad esempio: il riconoscimento automatico di targhe o segnali stradali, la digitalizzazione di libri e l'assistenza per persone non vedenti.
- **Segmentazione delle parole (tokenizzazione):** processo utile per suddividere del testo in singole parole. Nonostante sembri un'operazione banale, poiché nelle lingue come l'italiano o l'inglese basterà spezzare le parole ad ogni spazio, per lingue come il cinese o il

giapponese che non posseggono dei confini tra le parole, tale processo diventa fondamentale per l'elaborazione del testo.

- **Lemmatizzazione e Stemming:** processi simili che permettono di trasformare ogni parola nel testo nel suo lemma. Ad esempio, prese in considerazione le parole “chiudere”, “chiuso” o “chiusura” esse verranno convertite nel loro lemma “chiudi”. La differenza tra i due meccanismi sta nell'utilizzo di un dizionario predefinito per la lemmatizzazione e di regole automatiche per lo stemming.
- **Etichettatura di parti del discorso:** processo che permette di determinare parti del discorso da una data frase.
Dunque, dato un testo, è possibile analizzare ogni singola parola e definire automaticamente a quale delle parti del discorso essa appartiene.
Nella grammatica italiana esse sono: nome, aggettivo, articolo, pronomi, verbo, avverbio, preposizione, congiunzione e intersezione.
- **Creazione dell'albero sintattico di una frase:** processo che permette di realizzare un albero sintattico di una frase prendendo in analisi le parole che la compongono. Di norma si analizzano le dipendenze delle singole parole contrassegnandole come oggetti primari e predicati.
- **Riconoscimento dell'entità:** processo che permette di estrarre ed etichettare le entità interne ad un testo. Tra di esse possiamo citare: nome proprio, persona, luogo, date o organizzazioni.
- **Sentiment analysis:** processo che permette di comprendere informazioni soggettive da un dato documento al fine di definire il sentimento espresso dal soggetto che ha scritto il testo.

- **Estrazione delle relazioni:** processo che permette di estrarre le relazioni tra entità presenti all'interno di un testo. Ad esempio, capire la relazione di parentela tra soggetti interni ad una frase.

Per implementare alcuni dei processi interni a NLP è disponibile, in Python, Spacy una libreria open source per l'elaborazione del linguaggio naturale disponibile per oltre 64 lingue, tra cui l'italiano.

Di seguito verranno analizzati nel dettaglio alcuni dei metodi disponibili all'interno del modulo Spacy:

- **Tokenizzazione:** metodo che permette di suddividere un testo in singole parole.
- **POS (Part Of Speech) tagging:** metodo per il quale, dato un testo se ne analizzano le singole parole assegnando un tag che ne identifichi la funzione all'interno della frase.

```
import spacy
nlp = spacy.load("it_core_news_lg")

doc = nlp("Esempio di utilizzo della libreria Spacy")
for token in doc:
    print(token.text, token.pos_)
```

FIGURA 20: CODICE UTILE PER STAMPARE I TAG POS DELLE PAROLE CHE COMPONGONO UNA FRASE.

```
Frase: Esempio di utilizzo della libreria Spacy
```

| | Token | Tag |
|---|----------|-------|
| 0 | Esempio | NOUN |
| 1 | di | ADP |
| 2 | utilizzo | NOUN |
| 3 | della | ADP |
| 4 | libreria | NOUN |
| 5 | Spacy | PROPN |

FIGURA 21: OUTPUT DEL CODICE MOSTRATO IN FIGURA 20.

- **Lemmatizzazione:** metodo che, dato un testo, ne trasforma le parole nei rispettivi lemma.

```
import spacy
nlp = spacy.load("it_core_news_lg")
lemmatizer = nlp.get_pipe("lemmatizer")
doc = nlp("Sto eseguendo un test della lemmatizzazione con la libreria Spacy")
print("Frase: ",doc)
print("Lemma",[token.lemma_ for token in doc])
```

FIGURA 22: FIGURA CHE MOSTRA IL CODICE NECESSARIO PER APPLICARE LA LEMMATIZZAZIONE AD UN TESTO.

```
Frase: Sto eseguendo un test della lemmatizzazione con la libreria Spacy
Lemma ['Sto', 'eseguire', 'un', 'test', 'della', 'lemmatizzazione', 'con', 'la', 'libreria', 'Spacy']
```

FIGURA 23: OUTPUT DEL CODICE MOSTRATO IN FIGURA 22

- **NER (Named Entity Recognition):** processo che permette di estrarre le entità da un testo.

```
import spacy
nlp = spacy.load("it_core_news_lg")
doc = nlp("Il presidente del consiglio Draghi si trova a Catania per un convegno")
print("Frase: ",doc)
for ent in doc.ents:
    print(ent.text, ent.label_)
```

FIGURA 24: CODICE CHE MOSTRA L'APPLICAZIONE DELLA NER AD UNA FRASE.

```
Frase: Il presidente del consiglio Draghi si trova a Catania per un convegno
Draghi PER
Catania LOC
```

FIGURA 25: OUTPUT DEL CODICE MOSTRATO IN FIGURA 24, NELLO SPECIFICO È VISIBILE COME LE ENTITÀ "DRAGHI" E "CATANIA" SIANO STATE ESTRATTE E TAGGATE RISPETTIVAMENTE COME PERSONA E LUOGO.

- **Identificazione delle StopWords:** processo che permette di identificare le stop words, cioè le parole non utili alla comprensione del testo all'interno di un documento.

```
import spacy
nlp = spacy.load("it_core_news_lg")
doc = nlp("Esempio di utilizzo della libreria Spacy")
for token in doc:
    print(token.text, token.is_stop)
```

FIGURA 26: CODICE UTILE PER IDENTIFICARE LE STOP WORDS ALL'INTERNO DI UN TESTO.

Frase: Esempio di utilizzo della libreria Spacy

| | Token | IsStop |
|---|----------|--------|
| 0 | Esempio | True |
| 1 | di | True |
| 2 | utilizzo | False |
| 3 | della | True |
| 4 | libreria | False |
| 5 | Spacy | False |

FIGURA 27: OUTPUT DEL CODICE MOSTRATO IN FIGURA 26, PER OGNI PAROLA DEL TESTO, VIENE STAMPATO IL VALORE *TRUE* SE LA PAROLA ANALIZZATA È UNA STOP WORD, VICEVERSA VIENE RITORNATO *FALSE*.

La libreria appena definita è stata utilizzata all'interno del progetto per ripulire gli articoli che compongono il dataset.

I metodi interni alla pipeline NLP implementati sono stati: la rimozione delle stop word e la lemmatizzazione. Inoltre, sono stati implementati ulteriori metodi per la rimozione di caratteri non utili.

Nello specifico, è stato prodotto un algoritmo che ha analizzato tutti gli articoli estratti negli steps precedenti, al fine di creare un Python Dataframe contenente tutte le informazioni utili per la fase di apprendimento e testing del modello di classificazione.

Dunque, per ogni articolo salvato in locale, sono state effettuate le seguenti operazioni:

- Preso il testo dell'articolo che si sta analizzando, è stato implementato un metodo *RemoveTrash()* che, dato un testo, lo ripulisce rimuovendone tutti i caratteri inutili.

In primis, sono stati rimossi dal testo il link, il titolo, la descrizione, e le keywords, che erano stati inseriti per eventuali controlli nel dataset. Successivamente, sono stati sostituiti dal testo tutti i numeri e caratteri speciali `[!."'"'«»\`#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n]` con il carattere spazio. Infine, poiché le varie sostituzioni potrebbero aver portato alla presenza di doppi spazi contigui, è stato realizzato e applicato un metodo che li rimuovesse. Il metodo si occuperà di ritornare il testo ripulito.

```
def RemoveTrashChar(text):  
    #remove link titolo, descrizione and key  
    tmp_str="Descrizione: "  
    i=text.find(tmp_str)  
    text=text[i+len(tmp_str):].replace("\n"," ").replace(" "," ").replace(u'\xa0', u'').strip()  
    # remove the characters [\], ['] and ["]  
    text = re.sub(r"[0-9]", "", text)  
    text = re.sub(r"^[^ \nA-Za-z0-9A-00-00-y]+", "", text)  
    text.replace('\xa0', ' ').replace(" ", " ")  
    # replace punctuation characters with spaces  
    filters='!."'"'«»\`#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n'  
    translate_dict = dict((c, " ") for c in filters)  
    translate_map = str.maketrans(translate_dict)  
    text = text.translate(translate_map)  
    #remove double space  
    text = RemoveDoubleSpace(text)  
    return text  
  
def RemoveDoubleSpace(text):  
    i=0  
    for i in range(text.count(" ")):  
        text=text.replace(" ", " ")  
        if text.find(" ") == -1:  
            break  
        i=i+1  
    return text
```

FIGURA 28: CODICE IMPLEMENTATO PER LA RIMOZIONE DEI CARATTERI INUTILI E DEI POSSIBILI DOPPI SPAZI CHE SI POTREBBERO CREARE.

- Preso il nome della categoria d'appartenenza, è stato controllato se tale stringa sia contenuta almeno una volta all'interno del testo.

Poiché alcuni nodi potrebbero essere composti da più concetti, il nome del nodo è stato suddiviso in singole parole, per passare poi il controllo basterà che una delle parole sia presente nel testo.

Tale scelta è stata sviluppata al fine di verificare la correlazione tra l'articolo preso in analisi con la categoria a cui è stato assegnato.

Se il nome della categoria non è presente all'interno del testo, di norma esso sarà un Outliers (elemento assegnato erroneamente ad una categoria). Se così fosse, il file verrebbe eliminato e non verranno eseguite le successive operazioni. Viceversa, si continuerà con l'algoritmo.

```
def ControlRelatedText(dirpath,text):
    nodo=translator.GetKey(dirpath[-8:])
    keyNodo_list = nodo.split(" ")

    index=0
    for key in keyNodo_list:
        keyNodo_list[index]=RemoveTrashChar(key)
        index=index+1

    if not words_in_string(keyNodo_list,text):
        os.remove(pathFile)
        return False

    return True

def words_in_string(keyNodo_list, text):
    textTmp=text
    for key in keyNodo_list:
        if textTmp.lower().find(key.lower()) != -1:
            return True
    return False
```

FIGURA 29: CODICE CHE SI OCCUPA DI PRENDERE IL NOME DEL NODO, DIVIDERLO IN SINGOLE PAROLE, RIPULIRLO DA EVENTUALI CARATTERI INUTILI E VERIFICARE SE ALMENO UNA DELLE PAROLE È PRESENTE ALL'INTERNO DEL TESTO.

- Se il testo passa il precedente controllo, esso viene dato in input ad un ulteriore metodo che si occupa della rimozione delle stop words. Nel caso specifico, esso, dato un testo, lo suddivide in singole parole e controlla se esse siano delle stop word attraverso il metodo interno a Spacy.

Inoltre, tale metodo si occupa di verificare che la parola in analisi sia composta da più di un carattere non ripetuto.

Ad esempio, se si sta analizzando la parola “aaa”, il metodo si occupa di considerare i caratteri senza ripetizioni, così che la stringa diventerebbe “a”. Se tale modifica porta ad avere una dimensione della stringa minore di 2, significa che la parola analizzata è una stop word e dunque verrà rimossa.

```
def RemoveStopWord(text):  
    doc=nlp(text)  
    cleanedList=[]  
    for token in doc:  
        if not token.is_stop:  
            cleanedList.append(token)  
  
    listWithoutRepetitions=[]  
    for strTmp in cleanedList:  
        result = "".join(dict.fromkeys(strTmp))  
        if len(result)>=2 or len(strTmp)==1:  
            listWithoutRepetitions.append(strTmp)  
  
    return listWithoutRepetitions
```

FIGURA 30: FIGURA CHE MOSTRA IL METODO CHE SI OCCUPA DI RIMUOVERE LE STOP WORDS E LE PAROLE CHE CONTENGONO SOLO UN CARATTERE RIPETUTO.

- Ottenuta la lista delle parole non stop words del testo, esse attraverso Spacy, vengono trasformate nel loro lemma. Poiché lo step precedente ha fornito una lista, essa è stata riconvertita in testo così da poter essere passato a Spacy.

Successivamente, si è applicato il metodo Spacy che permette di ottenere il lemma dalle parole che compongono il testo. Visto che anche tale metodo fornisce una lista di parole, esse sono state riunite al fine di ritornare il testo lemmatizzato.

```
def Nlplemmatization(listText):  
    listLemmaWords=[]  
    strText=" ".join([str(item) for item in listText])  
    doc = nlp(strText)  
    for words in doc:  
        listLemmaWords.append(words.lemma_)  
    strTextLemma=" ".join([str(item) for item in listLemmaWords])  
    return strTextLemma
```

FIGURA 31: CODICE IMPLEMENTATO PER APPLICARE LA LEMMATIZZAZIONE ALLE PAROLE CHE COMPONGONO IL TESTO CHE GLI VIENE PASSATO.

- Infine, si è realizzato un Python Dataframe che contiene tutte le informazioni utili per la fase di apprendimento e testing del modello di classificazione. Nello specifico, all'interno del Dataframe, per ogni articolo, sono stati salvati: il codice univoco della categoria d'appartenenza, il testo senza modifiche e il testo ripulito dalle stop words e lemmatizzato.

In *figura 32* è possibile vedere un esempio della pipeline che è stata eseguita al fine di ottenere, per ogni articolo, un testo ripulito e lemmatizzato.

```
Articolo: Link: https://www.lanazione.it/prato/cronaca/quando-il-riuso-diventa-arte-la-sfida-circolare-si-gioca-qui-1.6177331
Titolo: riuso diventa arte sfida circolare gioca lanazione.it
Keywords: none
Descrizione: diretta streaming dialogo istituzioni imprese
Terzo appuntamento domani mattina con la rassegna "Le città del futuro". A partire dalle 11 sulla pagina Facebook e sul canale Youtube CittàdiPrato in diretta streaming si parlerà della sfida del riuso dei materiali per l'economia e il contributo dell'arte. Nel corso di questo incontro, partendo dal contesto della globalizzazione economica, caratterizzata da crescita dei consumi, scarsità di risorse e crescente produzione di rifiuti, si discuterà sui possibili orizzonti di sviluppo per le imprese, sulle possibili azioni delle amministrazioni pubbliche e sul ruolo dell'arte contemporanea nel sensibilizzare e stimolare la popolazione per favorire una nuova coscienza delle potenzialità insite nell'economia circolare. L'ospite principale dell ...

Articolo ripulito: riuso diventa arte sfida circolare gioca lanazione it diretta streaming dialogo istituzioni imprese Terzo appuntamento domani mattina con la rassegna Le città del futuro A partire dalle sulla pagina Facebook e sul canale Youtube CittàdiPrato in diretta streaming si parlerà della sfida del riuso dei materiali per l economia e il contributo dell arte Nel corso di questo incontro partendo dal contesto della globalizzazione economica caratterizzata da crescita dei consumi la scarsità di risorse e crescente produzione di rifiuti si discuterà sui possibili orizzonti di sviluppo per le imprese sulle possibili azioni delle amministrazioni pubbliche e sul ruolo dell arte contemporanea nel sensibilizzare e stimolare la popolazione per favorire una nuova coscienza delle e potenzialità insite nell economia circolare L ospite principale della conferenza on line sarà l artista David Tremlett foto noto per le sue policromie su grandi superfici realizzate con la tecnica del cosiddetto wall drawing che punta ...

Articolo Lemmatizzato: riuso arte sfidare circolare giocare lanazione it dirigere streaming dialogare istituzione impresa terzo appuntamento domani mattina rassegnare città partito pagina facebook canale youtube cittàdiPrato dirigere streaming parlare sfidare riuso materiale economia contributo arte correre incontrare contestare globalizzazione economico caratterizzare crescita consumo scarsità risorsa crescere produzione rifiuto discutere possibile orizzonte sviluppare impresa possibili azione amministrazione pubblico ruolo arte contemporaneo sensibilizzare stimolare popolazione favorire nuovo coscienza potenzialità insito economia circolare ospite principale conferenza on line e essere artista david tremlett foto notare policromie grande superficie realizzare tecnico cosiddetto wall drawing punto ridefinire oggetto dipinto collocazione spaziare tremlett esporre molto importare museo arte contemporaneo centrare peccati mostrare dedicare intervento permanere visibile pa rete importare museo edificio pubblico priv ...
```

FIGURA 32: ESEMPIO DI TESTO A CUI È STATA APPLICATA LA PIPELINE NLP.

7.2. Suddivisione del Dataset: Cross validation

Una volta realizzata la struttura contenente tutti gli articoli ripuliti e lemmatizzati, prima di passare alla creazione del modello di classificazione testuale, è necessario suddividere il dataset in due sottostrutture: il training-set e il test-set.

Tale processo è di fondamentale importanza poiché se si utilizzasse lo stesso set di dati sia per l'apprendimento sia per il testing, il classificatore darebbe un punteggio perfetto in termini di performance, ma sarebbe un punteggio falsato, in quanto si limiterebbe a dare in output le stesse etichette dei campioni da cui ha imparato a classificare.

Dunque, bisogna sempre testare il classificatore su dati che non ha mai processato.

Senza entrare troppo nel dettaglio, i due sottoprocessi training e testing, sono alla base della creazione del classificatore, poiché:

- **Training (apprendimento del modello):** è il processo che permette al modello di imparare ad associare un testo ad una data etichetta e di definirne le associazioni che legano i due elementi. Ciò viene svolto dall'algoritmo di apprendimento automatico andando a modificare e ad ottimizzare i propri parametri interni detti hyperparametri.
- **Testing (test del modello):** è il processo nel quale si testa il modello appena creato facendogli elaborare degli esempi mai visti. Ciò permette di calcolarne le performance e di capirne il corretto apprendimento nella fase di training.
Di norma, se nel testing le performance non sono ottimali, si ritorna al training, rieseguendolo con parametri diversi.

Definite le due fasi più rilevanti per la creazione di un modello di apprendimento automatico, è semplice capire quali dei due subset sia utilizzato nella fase di training e quale nella fase di testing.

Ritornando al processo di suddivisione del dataset in training-set e test-set, esistono principalmente due approcci di norma utilizzati per tale scopo.

Un primo approccio, quello più banale, è la suddivisione del dataset in due parti ben precise, una utilizzata per il training e l'altra per il testing. Tale suddivisione può essere parametrizzata definendone delle percentuali.

I due sottoinsiemi del dataset che si creeranno devono essere omogenei, cioè rappresentare al meglio l'interezza del dataset, poiché si rischierebbe di testare il modello con esempi legati ad etichette mai viste o di non testare il modello su tutte le categorie che ha imparato ad associare.

Lo svantaggio di tale approccio è che la suddivisione può ridurre drasticamente la dimensione dei dati di training.

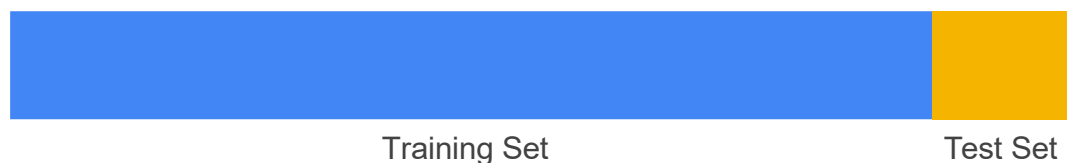


FIGURA 33: SUDDIVISIONE FISSA DEL DATASET.

Un secondo approccio che risolve il problema della riduzione del training-set, è la validazione incrociata o più comunemente chiamata Cross Validation o k-fold.

Essa consiste nella suddivisione del dataset in k parti, chiamati folds, della stessa dimensione. Il processo di assegnazioni delle folds al training-set o al test-set avviene in maniera iterativa, dunque ad ogni iterazione si assegnano k-1 folds al set di addestramento, ed 1 fold al set di testing.

In questo modo, il dataset verrà interamente sfruttato sia per il training e sia per il testing, poiché tutte le folds verranno assegnate almeno una volta al training-set e al test-set.

Tale approccio può essere costoso in termini di prestazioni, ma ha il vantaggio di risolvere il problema della perdita di dati per il training.

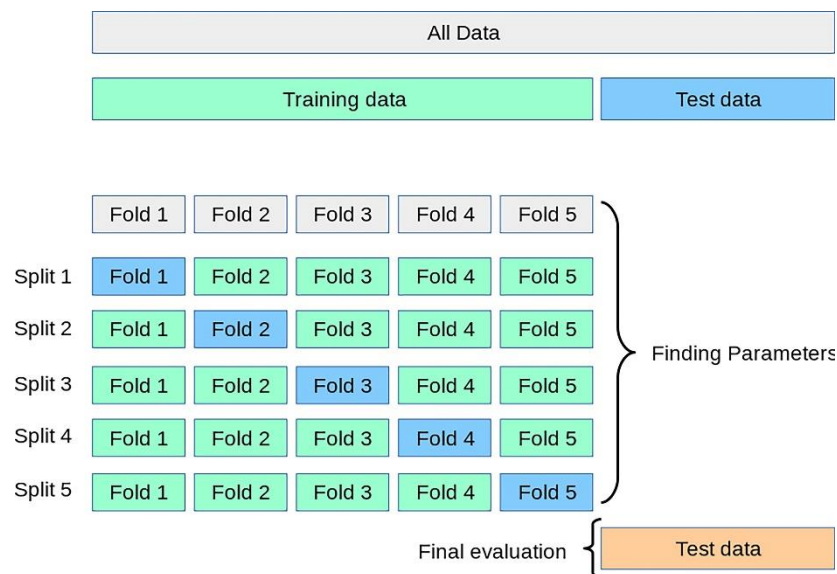


FIGURA 34: IMMAGINE MOSTRANTE LA SUDDIVISIONE DI UN DATASET UTILIZZANDO IL K-FOLD.

All'interno del progetto è stato utilizzato l'approccio della Cross Validation, ciò è stato possibile attraverso la libreria *sklearn* di Python, che mette a disposizione diversi metodi per tale scopo.

Durante lo sviluppo è stato scelto di utilizzare il metodo *StratifiedShuffleSplit*.

Esso è un'unione dello *StratifiedKFold* e dello *ShuffleSplit*, due metodi che garantiscono rispettivamente: il bilanciamento di ogni fold, cioè la stessa proporzione di elementi con la stessa etichetta e il campionamento casuale dell'intero dataset.

Dunque, grazie allo *StratifiedShuffleSplit* si ha la garanzia di un campionamento casuale e bilanciato.

Esso accetta i seguenti parametri:

- `n_splits`: il numero di iterazioni e suddivisioni che devono essere applicati.
- `test_size`: la proporzione del dataset da includere nel test-set.
- `train_size`: la proporzione del dataset da includere nel training-set.
- `random_state`: seme utilizzato per la generazione degli indici casuali.

Una volta definito l'oggetto *StratifiedShuffleSplit* sul quale sono stati settati opportunamente i parametri, si può richiamare il metodo *split()*, che, data una struttura contenente dati ed una lista di label (utile nel caso in cui si voglia una suddivisione bilanciata), restituisce due liste di indici, che rappresentano gli elementi che compongono il training-set e il test-set in quella iterazione.

In figura 35 è mostrato il codice utilizzato per la creazione dell'oggetto *StratifiedShuffleSplit* dove il dataset passato deve essere suddiviso in 4 folds e il numero di elementi assegnati al test-set, per ogni categoria, deve essere il 25%. Successivamente, è stato richiamato il metodo *split()* sull'oggetto appena creato, passandogli il DataFrame contenente tutti i dati relativi agli articoli e la lista dei codici univoci dei nodi interni alla tassonomia, tale che la suddivisione possa rispettare il bilanciamento tra training-set e test-set.

Il metodo *split()* ritornerà ad ogni iterazione (per un totale di 4, valore definito nel parametro *n_splits* dell'oggetto *StratifiedShuffleSplit*) la lista degli indici assegnati al training-set e test-set.

Essi, infine, verranno utilizzati per estrarre dal DataFrame le due sottostrutture *train* e *test* e le relative label.

```
kf = StratifiedShuffleSplit(n_splits=4, test_size=0.25, random_state=123)
for train_index, test_index in kf.split(df_data, y=custom_y):
    train = df_data.iloc[train_index]
    test = df_data.iloc[test_index]
    y_train = df_data.iloc[train_index].loc[:, 'CMT']
    y_test = df_data.iloc[test_index].loc[:, 'CMT']
```

FIGURA 35: CODICE UTILIZZATO PER L'IMPLEMENTAZIONE DELLO STRATIFIEDSHUFFLESPLIT.

Capitolo 8

8. Creazione del Classificatore di testo

8.1. Estrazione delle features dal testo:

8.1.1. Tecniche d'estrazione delle features:

L'ultimo step da eseguire prima di passare alla creazione e al training del modello di classificazione è la riduzione della dimensionalità del set di dati che vengono passati al classificatore.

Essa è fondamentale poiché, se i dati passati in input al classificatore sono troppo grandi, la probabilità di avere informazioni ridondanti o inutili aumenta e di conseguenza ne risente anche il costo computazionale.

Tale riduzione viene implementata grazie a dei meccanismi di estrazione delle features o caratteristiche, i quali data un serie di dati grezzi, riescono ad estrarne informazioni non ridondanti rappresentanti al meglio l'insieme dei dati passati.

Essa è fondamentale poiché, un'analisi con un numero elevato di informazioni richiede generalmente un grande spreco in quantità di memoria e di potenza computazionale utilizzata. Inoltre, può accadere che l'algoritmo di apprendimento automatico si adatti troppo ai campioni del training-set, portando ad una predizione erranea di campioni mai visti.

Dunque, ciò che verrà dato in output dall'estrazione di features sarà un insieme di caratteristiche in grado di riassumere al meglio le informazioni contenute nel set iniziale.

Per effettuare l'estrazione delle features esistono diverse tecniche che possono essere raggruppate in due macrocategorie in base all'approccio utilizzato che può essere frequentistico o semantico.

- **Estrazione delle features basata su approccio frequentistico:** tale approccio utilizza la frequenza delle parole, interne ad un testo, per estrarne le features.

Le più comuni tecniche che seguono questo principio sono:

- **BOW (Bag Of Words):** tale tecnica si basa sulla creazione di vettori a lunghezza fissa utili alla rappresentazione dei documenti.

Il primo step è la realizzazione di un vocabolario composto dalle parole uniche presenti nel set di documenti passato (a cui è già stata applicata la pipeline NLP).

Definito il vocabolario, ogni documento può essere rappresentato come:

$$d = \{t_i | t_i \in V\}_i$$

EQUAZIONE 2: FORMULA CHE RAPPRESENTA LA COMPOSIZIONE DI UN DOCUMENTO UTILIZZANDO LA BOW, DOVE OGNI DOCUMENTO È COMPOSTO DALL'INSIEME DI TERMINI CHE LO COMPONGONO PRESENTI ALL'INTERNO DEL VOCABOLARIO.

Il secondo step è la creazione di una matrice di caratteristiche in cui le colonne saranno le parole del vocabolario e le righe i documenti presi in analisi; mentre i valori interni alla matrice sono il numero di occorrenze dell'i-esima parola nel j-esimo documento.

Quindi, ogni riga della matrice rappresenterà il vettore del documento preso in analisi che può essere definito come segue:

$$bow(d)_i = count(d, t_i)$$

EQUAZIONE 3: FORMULA CHE DEFINISCE IL METODO DI CREAZIONE DEL VETTORE DI RAPPRESENTAZIONE DI UN DOCUMENTO D CONSIDERANDO LA FREQUENZA DEL TERMINE T IN D.

La somma dei valori interni a tale vettore identifica la lunghezza, in termini di parole, del documento.

Ad esempio, date le due frasi “A Sergio piace la musica rap” e “Aldo preferisce la musica classica”, una volta applicata la pipeline NLP, il vocabolario delle parole sarà il seguente:

["sergio", "piacere", "musica", "rap", "aldo", "preferire", "classica"].

Di conseguenza la matrice di caratteristiche sarà:

| | sergio | piacere | musica | rap | aldo | preferire | classica |
|---------|--------|---------|--------|-----|------|-----------|----------|
| Frase 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| Frase 2 | 0 | 0 | 1 | 0 | 1 | 1 | 1 |

GRAFICO 1: MATRICE DI CARATTERISTICHE.

Infine, i vettori di rappresentazione per le due frasi saranno:

Frase 1=[1,1,1,1,0,0,0] e Frase2=[0,0,1,0,1,1,1].

Poiché attraverso questa rappresentazione si perde l'ordine delle parole (da qui il nome Bag Of Words) è possibile considerare coppie, terzine o qualsiasi altro raggruppamento di parole che prendono il nome di n-grammi, con n che rappresenta il numero di parole raggruppate.

Gli svantaggi di tale approccio sono: l'utilizzo degli n-grammi, che fa aumentare notevolmente il numero di features; l'inserimento di un nuovo documento fa aumentare la dimensione del vocabolario e di conseguenza anche i vettori di rappresentazione dei documenti; i vettori di rappresentazione dei documenti conterranno molti 0 che porta ad avere informazioni non utili.

Quindi, ricollegandosi al processo di estrazione delle features, esso può essere implementato nel seguente modo:

- Si definiscono i vettori di rappresentazione dal set di documenti su cui si vogliono estrarre le features;
 - Si sommano le occorrenze delle singole parole in tutti i documenti;
 - Ottenute le occorrenze totali delle parole interne al vocabolario, è possibile settare delle soglie di frequenze che permettano di decidere se accettare o scartare un termine. I termini che supereranno la soglia prefissata saranno le features che rappresenteranno il set di dati.
-
- **TF-IDF:** essa è una tecnica che si basa sulla frequenza delle parole, le quali avranno un peso maggiore se la loro frequenza è minore, contrariamente alla BOW. Ciò è utile, perché le parole la cui frequenza è alta risultano essere più comuni, e dunque, meno significative. Poiché l'obiettivo dell'estrazione delle features è quello di definire le caratteristiche, nel nostro caso parole che meglio rappresentino il set di dati preso in analisi, diventa necessario considerare solo quelle non ridondanti all'interno dei documenti. Dunque, la logica su cui si basa tale tecnica è la seguente: la frequenza di una parola del vocabolario, per un dato documento, è data dalla frequenza del termine nel documento, divisa per il numero di documenti in cui essa è presente. In questo modo se una parola comune ha una frequenza alta su un dato documento, il valore di essa verrà ridotta se la parola risulta presente in molti documenti. Per fare ciò si identifichi l'insieme dei documenti come $\{d_i\}$, per ogni documento d_i il numero di termini contenuti in esso è uguale a:

$$m_i = \sum_j count(d_i, t_j)$$

EQUAZIONE 4. NUMERO DI TERMINI CONTENUTI IN UN DOCUMENTO.

Dunque, la frequenza di un dato termine in un dato documento sarà uguale alla frequenza del termine nel documento diviso il numero di parole contenute nello stesso.

Tale formula può essere definita quanto segue:

$$tf(d_i, t_j) = \frac{count(d_i, t_j)}{m_i}$$

EQUAZIONE 5: FORMULA PER CALCOLARE LA TERM FREQUENCY

Fatto ciò, si definiscano due ulteriori funzioni:

$$p(d_i, t_j) = \begin{cases} 1 & \text{se } count(d_j, t_j) > 0 \\ 0 & \text{altrimenti} \end{cases} \quad c(t_j) = \sum_j p(d_i, t_j)$$

EQUAZIONI 6 E 7.

Esse permettono rispettivamente di verificare se un dato termine è presente in un documento e di contare in quanti di essi un dato termine è presente.

Grazie a tali funzioni è possibile definire la frequenza inversa dei documenti attraverso la formula:

$$idf(t_j) = \log \frac{n}{c(t_j)} \text{ con } n = \#documenti$$

EQUAZIONE 8: FREQUENZA INVERSA DEI DOCUMENTI.

Infine, la rappresentazione bow finale di un dato documento sarà uguale a:

$$TfIdf(t_j, d_i) = tf(d_i, t_j) \cdot idf(t_j)$$

EQUAZIONE 9: FREQUENZA INVERSA DI UN TERMINE NEI DOCUMENTI

Cioè il prodotto tra la frequenza del termine e la frequenza inversa del documento. Essa avrà un valore alto quando la parola analizzata ha un'alta frequenza in un documento, e una bassa frequenza nei restanti casi.

Mentre avrà un valore basso quando la parola è presente più volte in molti documenti.

Calcolata la TF-IDF per ogni termine del vocabolario, è necessario settare una soglia sopra la quale le parole analizzate facciano parte delle features finali.

- **Estrazione delle features basata su approccio semantico:** tale approccio utilizza la semantica delle parole al fine di rappresentare i documenti in uno spazio vettoriale. Differentemente dagli approcci frequentistici, gli approcci semantici definiscono delle relazioni tra le parole appartenenti a contesti simili.

La tecnica più comune che segue il suddetto approccio è la Word2Vec.

- **Word2Vec:** tale tecnica si basa sulla mappatura delle parole appartenenti al testo in uno spazio vettoriale, ciò consente di convertire ogni singola parola in un vettore (da qui il nome word 2 vec) dando la possibilità di poter applicare qualsiasi funzione matematica, come la somma, la sottrazione o la distanza tra vettori.

Ciò che fa il Word2Vec è definire vettori simili tra parole che sono presenti in contesti simili.

Questo è possibile tramite due algoritmi interni ad esso che sono: il CBOW, che permette di predire una parola in base ad un dato contesto (un insieme di parole) e lo Skip Gram che permette di predire, data una parola, il contesto a cui può essere legata.

Creare, dunque, i vettori di ogni parola tramite il Word2Vec, si estraggono come features finali quelle più simili al contesto dei documenti presi in analisi.

Utilizzare Word2Vec implica un grande impiego di risorse in termini di memoria e di tempo, per questo è il meno utilizzato quando si ha a che fare con molti documenti.

8.1.2. Implementazione dell'estrazione delle features

In questo paragrafo verranno definite le scelte implementative che sono state prese per effettuare l'estrazione delle features all'interno del software realizzato.

Nel progetto si è utilizzato l'approccio frequentistico e nello specifico si è deciso di utilizzare il TF-IDF, poiché a seguito di diversi test si è mostrato essere il più performante in termini di utilizzo di memoria e tempo.

Tale processo è stato implementato attraverso l'oggetto "*feature_extraction.text.TfidfVectorizer()*" della libreria Sklearn che permette di trasformare un dato set di documenti in una matrice di features tramite TF-IDF.

Tra i vari parametri accettabili dal costruttore dell'oggetto in questione, sono stati utilizzati principalmente i seguenti:

- **sublinear_tf**: che se settata a "True" permette di applicare un ridimensionamento ai risultati della TF-IDF, andando a sostituire la frequenza di un dato termine con " $1 + \log(\text{frequenza_termine})$ ". Esso è utile quando si ha a che fare con documenti di lunghezza diversa permettendo di gestirli in maniera omogenea.
- **min_df**: permette di definire il numero minimo di documenti in cui deve apparire la parola per essere campionata come features. Se tale valore è un float, allora si considera la porzione definita di documenti minima su cui le features devono apparire.
- **max_df**: indica il numero massimo di documenti in cui la parola può apparire affinché possa essere una feature. Se il valore è un float esso indica la proporzione di documenti massima su cui quel termine può apparire.
- **vocabulary**: esse permette di calcolare direttamente i valori del TF-IDF tra i documenti analizzati e i termini passati all'interno del parametro *vocabulary*. Se essa è settata non avverrà nessuna estrazione delle features, poiché utilizzerà quelle passate.

Una volta richiamato il costruttore `feature_extraction.text.TfidfVectorizer()` verrà dato in output un oggetto su cui è possibile chiamare diversi metodi.

Quelli utilizzati all'interno del progetto sono stati:

- **fit(documenti)**: metodo che permette all'oggetto di estrarre dai documenti passati le features applicandone la TF-IDF, rispettando le condizioni passate all'interno del costruttore dell'oggetto.
Le features estratte verranno memorizzate all'interno di un vocabolario interno all'oggetto.
- **get_feature_names()**: metodo che permette di ottenere tutte le features che sono state estratte nel processo di fitting dell'oggetto.
- **transform(documenti)**: metodo che permette di trasformare i documenti passati in una matrice di features, utilizzando il vocabolario e le frequenze apprese dal metodo `fit()`.

La logica utilizzata all'interno del progetto è stata quella di utilizzare l'applicazione del TF-IDF due volte, nello specifico:

- Il primo utilizzo è stato il seguente: data la lista delle categorie, su cui si deve applicare l'estrazione delle features, per ognuna di esse è stato richiamato il metodo `FeatureExtraction_tfidf(train,nodeCMT)`.
Esso, dato il subset *train*, contenente i documenti etichettati come appartenenti al training-set e la categoria che si sta analizzando, richiama il costruttore `feature_extraction.text.TfidfVectorizer()` che ritorna un oggetto su cui è possibile applicare i metodi precedentemente citati. Dunque, estratti dal *train* i testi su cui è stata applicata la pipeline NLP appartenenti alla categoria passata, è stato applicato il `fit()`, così da far creare il dizionario di features interno all'oggetto.
Esse sono state ottenute attraverso il metodo `get_feature_names()` e ritornate al metodo.

Successivamente verranno inserite all'interno di una lista che raccoglierà le features di tutte le categorie analizzate. Vedi *figura 36* e *figura 37*.

- Il secondo utilizzo è stato il seguente: data la lista delle features estratte da ogni categoria, sono state eliminate quelle ripetute e passate al costruttore "*feature_extraction.text.TfidfVectorizer()*", all'interno del parametro *vocabulary*; il quale ritorna l'oggetto su cui successivamente è stato applicato il *fit()* dell'intero *train*, così da avere un oggetto sul quale è possibile richiamare il metodo *transform()* ed ottenere così la matrice contenente i valori della TF-IDF dati dai documenti e features. Vedi *figura 38*.
- Ciò è stato fatto poiché, se si applicasse direttamente la TF-IDF a tutti i documenti delle categorie che si vogliono classificare si rischierebbe che alcune categorie non abbiano abbastanza features da poter essere identificate univocamente poiché il numero di documenti non è uguale per tutti.

```
featureList = []
for nodeCMT in path_list:
    X_names=FeatureExtraction_tfidf(train,nodeCMT)
    for feature in X_names:
        featureList.append(feature)
```

FIGURA 36: CODICE CHE PER OGNI CATEGORIA DA CLASSIFICARE RICHIAMA IL METODO *FEATUREEXTRACTION_TFIDF()*.

```
def FeatureExtraction_tfidf(train,CMT):
    vectorizer = feature_extraction.text.TfidfVectorizer(sublinear_tf=True, min_df=5 )
    data_corpus = train[train["CMT"] == CMT]["Lemma"]

    vectorizer.fit(data_corpus)

    features = vectorizer.get_feature_names()
    return features
```

FIGURA 37: METODO *FEATUREEXTRACTION_TFIDF()* CHE SI OCCUPA DI APPLICARE L'ESTRAZIONE DELLE FEATURES TRAMITE TF-IDF

```
featureList = list(dict.fromkeys(featureList))

vectorizer = feature_extraction.text.TfidfVectorizer(vocabulary=featureList, sublinear_tf=True)
data_corpus = train["Lemma"]
vectorizer.fit(data_corpus)
X_train = vectorizer.transform(data_corpus)
```

FIGURA 38: CODICE CHE SI OCCUPA DI APPLICARE LA TF-IDF A TUTTI I DOCUMENTI, APPARTENENTI ALLE CATEGORIE DA CLASSIFICARE, UTILIZZANDO LE FEATURES ESTRATTE IN PRECEDENZA.

Di seguito verranno mostrate le prime 4 features estratte per ogni categoria di primo livello appartenenti alla tassonomia Media Topics.

| | |
|---|--|
| Cat:01000000 Nome: arte, cultura, intrattenimento e mass-media arte artista museo cultura | Cat:12000000 Nome: religione cattolico religione religioso dio |
| Cat:02000000 Nome: criminalità, legge e giustizia giustiziare indagato udienza tribunale | Cat:13000000 Nome: Scienze e tecnologia tecnologia scienza geologia ingegneria |
| Cat:04000000 Nome: economia, affari e finanza finanza mercato economia scambio | Cat:14000000 Nome: sociale sociale dipendenza inclusione donazione |
| Cat:05000000 Nome: istruzione istruzione scolastico scuola studente | Cat:15000000 Nome: sport sport atleta gara coppa |
| Cat:06000000 Nome: ambiente ambiente inquinamento biodiversità acqua | Cat:16000000 Nome: conflitto, guerra e pace guerra pace bombardamento profugo |
| Cat:07000000 Nome: salute medicina paziente malattia medicare | Cat:17000000 Nome: meteo temperatura nuvoloso pioggia perturbazione |
| Cat:09000000 Nome: lavoro lavoratore lavorare contratto datore | |
| Cat:10000000 Nome: lifestyle e hobby hobby passatempo giardinaggio giocare | |
| Cat:11000000 Nome: politica politico elezione parlamentare referendum | |

FIGURA 39B: FEATURES ESTRATTE PER LE CATEGORIA DA 12 A 17.

FIGURA 39A: FEATURES ESTRATTE PER LE CATEGORIA DA 01 A 11.

8.2. Algoritmi di ML e approcci utilizzati per la creazione del classificatore

8.2.1. Algoritmi di ML utilizzati per classificare

Una volta estratte le features da tutte le categorie su cui si vuole basare il classificatore, si può passare alla realizzazione del modello di classificazione.

Per tale scopo sono di seguito definiti i due algoritmi di Machine Learning che meglio si prestano alla risoluzione del problema di classificazione.

SVM (Support-Vector Machine):

Algoritmo di apprendimento automatico supervisionato per scopi di classificazione e regressione. Esso può essere utilizzato sia per problemi di classificazione binaria che multiclasse.

Nel caso di una classificazione binaria, l'SVM si occuperà di trovare un iperpiano che divida al meglio il set di dati analizzato in due gruppi.

Il concetto di iperpiano varia in base al numero di classi da identificare, nel caso in cui si hanno solo due classi l'iperpiano sarà rappresentato da una retta che divide il set di dati (vedi *figura 40a*). Nel caso di 3 classi esso sarà identificato da un piano geometrico (vedi *figura 40b*), mentre per qualsiasi altro numero di dimensioni si generalizza il concetto di iperpiano identificandolo come una figura geometrica che suddivide il set di dati.

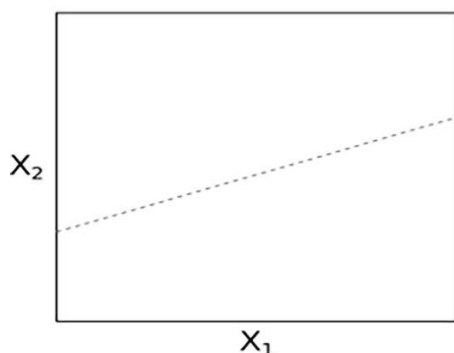


FIGURA 40A: IPERPIANO A 2 DIMENSIONI

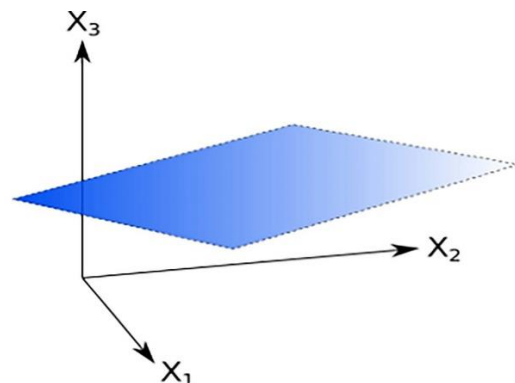


FIGURA 40B: IPERPIANO A 3 DIMENSIONI

Per l'identificazione dell'iperpiano si devono dapprima introdurre i concetti di vettori di supporto e margine.

Si definiscono vettori di supporto, nel caso di spazio a due dimensioni, i due elementi appartenenti alle due classi più vicini tra loro, vedi *Figura 41a*.

Con Margine si indica la distanza tra i vettori di supporto delle due classi, vedi *Figura 41b*.

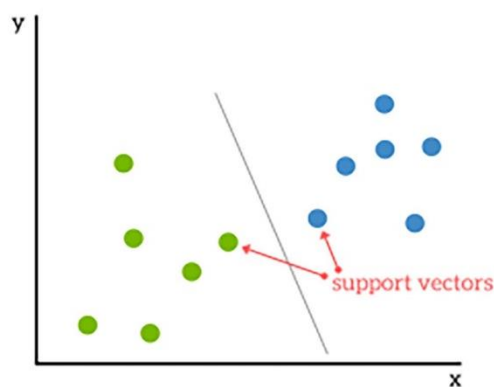


FIGURA 41A: VETTORI DI SUPPORTO.

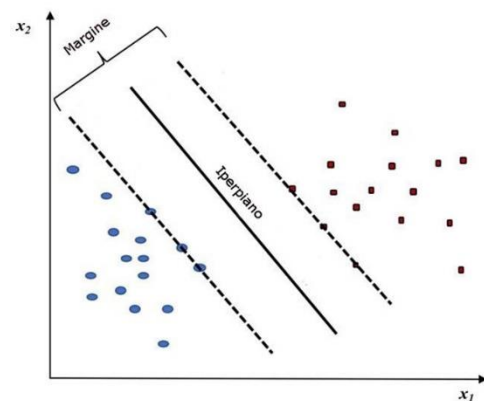


FIGURA 41B: MARGINE.

Una volta definito il valore del margine, alla metà di esso verrà tracciato l'iperpiano. Poiché possono essere identificati infiniti iperpiani all'interno dello spazio, si predilige sempre l'iperpiano che abbia i vettori di supporto delle due classi il più distanti possibili, così da migliorare classificazione di un nuovo elemento mai visto.

Finché il dataset passato è lineare, l'identificazione dell'iperpiano è semplice, ma se il dataset non è lineare si deve utilizzare il metodo del kernel, il quale permette di passare ad uno spazio di dimensione superiore. In questo modo, ritornando allo spazio a due dimensioni non si avrà più un iperpiano rappresentato da una retta ma bensì da un'altra figura geometrica, vedi *figura 42a* e *figura 42b*.

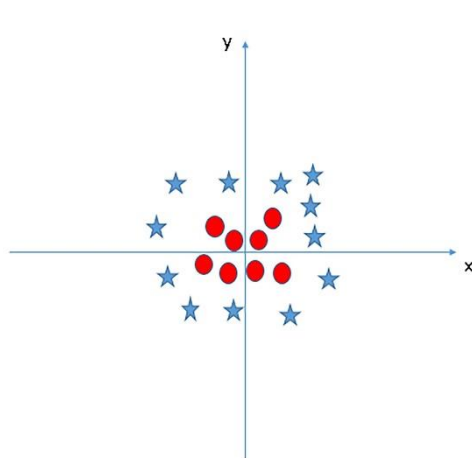


FIGURA 42A: DATASET RAPPRESENTATO IN DUE DIMENSIONI CHE NON PERMETTE L'IDENTIFICAZIONE DI UN IPERPIANO.

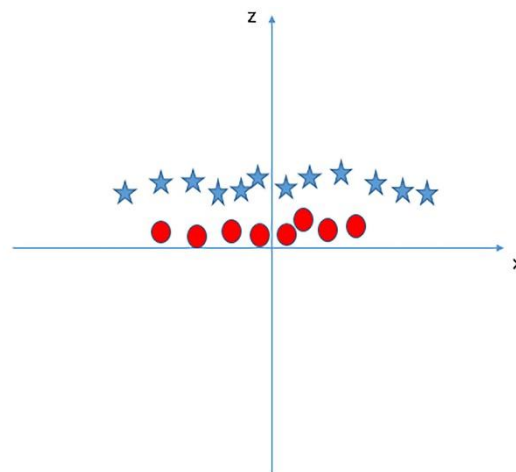


FIGURA 42A: DATASET RAPPRESENTATO IN TRE DIMENSIONI PERMETTENDO L'IDENTIFICAZIONE DELL'IPERPIANO.

Per rappresentare i dati all'interno dello spazio ad N dimensioni l'SVM utilizza un insieme di funzioni matematiche che prende il nome di Kernel.

Le funzioni Kernel più diffuse sono:

- **Kernel lineare:** esso permette di rappresentare in maniera ottimale i dati all'interno dello spazio quando il dataset è lineare.
- **Kernel polinomiale:** permette di rappresentare i dati non lineari, utilizzando il metodo del Kernel. Esso è meno performante rispetto al Kernel lineare.
- **Kernel RBF o Gaussiano:** anch'esso viene utilizzato nel caso in cui i dati non seguano una distribuzione lineare.

Nel caso si volesse realizzare un classificatore multiclasse con l'SVM, si prendono in considerazione tutte le classi in esame e si suddivide il problema in numerosi a classificazione binaria.

Per fare ciò esistono diverse tecniche: ad esempio, classificatori che analizzano una classe con tutte il resto oppure tanti classificatori che analizzano le classi prese a coppie.

In *figura 43* è possibile vedere un esempio di utilizzo dell'algoritmo SVM fornito dalla libreria "sklearn". Per la creazione del classificatore basterà richiamare il metodo "svm.SVC()" su cui è possibile specificare il kernel da utilizzare. Una volta ottenuto l'oggetto SVM dalla chiamata, è possibile tramite il metodo "fit()" far sì che il modello di classificazione impari dai dati passati.

Poiché nell'esempio proposto si sta utilizzando un approccio supervisionato, oltre alla matrice della TF-IDF (x_train) viene passata la lista delle relative classi d'appartenenza. Tali variabili vengono definite durante il processo d'estrazione delle features dai documenti.

Una volta "fittato" il classificatore è possibile passare un elemento mai visto, e, attraverso il metodo "predict()" ottenere la classe che secondo il classificatore meglio si correla all'elemento.

```
from sklearn import svm
clf=svm.SVC(kernel="linear")
clf.fit(x_train,y_train)
y_pred=clf.predict(x_test)
```

FIGURA 43: CODICE CHE IMPLEMENTA UN CLASSIFICATORE SVM.

K-NN (K Nearest Neighbors):

Esso è uno degli algoritmi di machine learning più conosciuti, grazie ai molteplici casi d'uso in cui può essere applicato.

Il K-NN è un algoritmo di apprendimento automatico supervisionato che si occupa di predire la classe di appartenenza di un elemento analizzando i dati che ha studiato in fase di apprendimento.

Nello specifico il K-NN rappresenta l'insieme dei dati di apprendimento in uno spazio a n dimensioni, con n uguale al numero di caratteristiche scelte per rappresentare i dati.

Durante la fase di predizione della classe di un nuovo elemento, l'algoritmo si occupa di inserire l'elemento all'interno dello spazio e successivamente calcolerà la distanza euclidea tra l'elemento analizzato e il resto degli elementi nello spazio.

In questo algoritmo, è necessario fissare un valore per il parametro K che indica il numero di elementi più vicini all'elemento da predire che devono essere presi in considerazione durante la fase di assegnazione di una classe all'elemento analizzato.

Dunque, fissato K , e calcolate le distanze tra l'elemento e gli elementi passati in fase di apprendimento, si prenderanno in analisi le classi dei K elementi con distanza minore. L'algoritmo K-NN assegnerà come classe d'appartenenza al nuovo l'elemento, la classe con più occorrenze tra i K elementi presi in considerazione, vedi *figura 44*.

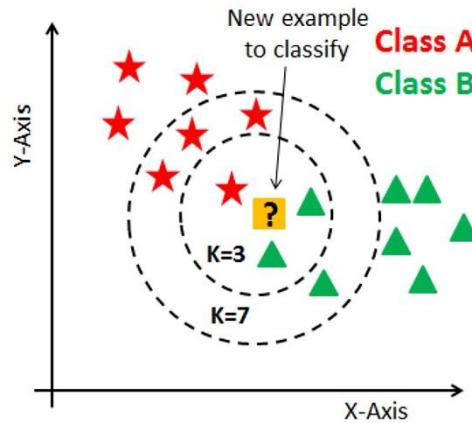


FIGURA 44: FIGURA CHE MOSTRA UN ESEMPIO DI FUNZIONAMENTO DEL K -NN, IN QUESTO CASO IL NUOVO ELEMENTO, IN BASE AL VALORE DEL PARAMETRO K VERRÀ ASSEGNATO ALLA CLASSE B PER $K=3$, MENTRE PER $K=7$ VERRÀ ASSEGNATO ALLA A.

Un valore di K pari ad 1 fa sì che la classe assegnata all'elemento sarà quella d'appartenenza dell'elemento più vicino. Quindi, un valore di K molto piccolo, implica che la regione presa in analisi dal classificatore sia ridotta e quindi non prenderebbe in considerazione la distribuzione generale del dataset.

Mentre un valore di K molto alto fa sì che il l'algoritmo prenda in considerazione molti elementi del dataset, il che può portare ad ignorare piccoli dettagli nelle caratteristiche degli elementi.

Poiché il valore di K è così importante, bisogna utilizzare un meccanismo che permetta di ottenere, in base al dataset su cui si sta lavorando, un valore che riduca al minimo l'errore di assegnazione della categoria.

Tra i tanti metodi utili per tale scopo, il più diffuso è la cross validation, introdotta nel **paragrafo 7.2**.

Dunque, ad ogni iterazione della cross validation si testa il classificatore sul test-set, variando il valore di K e salvando il numero di predizioni errate che ha effettuato. Infine, effettuate le N iterazioni, si sceglierà come valore di K quello che ha portato il numero minore di errori.

Il metodo della cross validation ha lo svantaggio di richiedere molto tempo e memoria, quindi, se si conosce la tipologia dei dati e il loro andamento si può assegnare un valore di K definito a priori.

Il vantaggio più importante nell'utilizzo del K-NN, è che tale metodo non ha bisogno di nessuna ipotesi sulla distribuzione dei dati e permette di lavorare con qualsiasi tipologia di dataset indipendentemente dalla loro distribuzione. Ciò rende il K-NN l'algoritmo migliore quando non si ha conoscenza sull'andamento dei dati.

Di seguito nella *figura 45* è mostrato un esempio di utilizzo dell'algoritmo K-NN fornito dalla libreria "sklearn".

Per la creazione del classificatore basterà richiamare il metodo "neighbors.KNeighborsClassifier()" su cui è possibile specificare il valore della costante K settando il parametro "n_neighbors".

Una volta ottenuto l'oggetto KNeighborsClassifier dalla chiamata, è possibile "fittare" il modello tramite la chiamata al metodo "fit()" a cui viene passata la matrice TF-IDF (x_train) e la lista delle relative classi d'appartenenza.

Come nel caso dell'esempio in *figura 43*, tali variabili vengono definite durante il processo d'estrazione delle features.

Effettuato il fitting del modello è possibile testarlo passandogli un elemento mai visto, ciò avviene attraverso il metodo "predict()" che ritorna in output la classe d'appartenenza dell'elemento passato.

```
from sklearn import neighbors.KNeighborsClassifier()
K=3
clf=neighbors.KNeighborsClassifier(n_neighbors=K)
clf.fit(x_train,y_train)
y_predict=clf.predict(x_test)
```

FIGURA 45: CODICE CHE MOSTRA LE ISTRUZIONI UTILI ALLA CREAZIONE DI UN MODELLO DI CLASSIFICAZIONE BASATO SULL'ALGORITMO K-NN.

8.2.2. Approcci per la realizzazione dei classificatori

Nei capitoli precedenti si è più volte detto che lo scopo finale del progetto sarebbe stato la creazione automatica di un classificatore di articoli di giornale, ma durante lo sviluppo del codice utile al suddetto compito, ci si è resi conto che la creazione di un tale classificatore, sarebbe stata molto costosa in termini di tempo e memoria. Ciò è dovuto dall'elevato numero di documenti contenuti all'interno del learning-set, che, se pur suddiviso in training-set e test-set, risulta essere comunque di dimensioni elevate.

Quindi, si è pensato di implementare un gruppo di classificatori che si occupassero di etichettare un singolo sottoinsieme delle categorie.

Nello specifico sono stati utilizzati due diversi approcci che creano, con logiche differenti, un gruppo di classificatori, rendendo così la loro realizzazione più veloce. Inoltre, tale suddivisione ha il vantaggio di rendere il processo di classificazione automatica più simile alla logica umana.

La decisione di implementare due approcci differenti permette di avere due diverse combinazioni di classificatori su cui è possibile confrontare le performance.

Essi, oltre ad avere una logica differente, utilizzano due algoritmi di Machine Learning diversi.

I due approcci verranno di seguito discussi.

Primo approccio: classificatore a 2 livelli

Con questo metodo si è deciso di realizzare un primo classificatore in grado di categorizzare un testo assegnandogli una delle categorie di primo livello della tassonomia ed un classificatore per ognuno dei nodi di primo livello che prenda in analisi tutte le categorie relative ad esso.

Nello specifico:

- Classificatore di primo livello: esso dovrà essere in grado, dato un testo, di capire quale dei 16 nodi di primo livello è la macrocategoria che meglio si appresta a identificare il documento.
- Classificatori di secondo livello: per ognuno dei 16 nodi di primo livello, si realizza un classificatore che sia in grado di etichettare un documento assegnandogli una delle categorie discendente dalla categoria di primo livello. Ciò implica che tutte le categorie dal secondo livello in poi verranno portate allo stesso livello.

Dunque, si avranno 17 classificatori totali, il primo dei quali si occuperà di capire con quale macrocategoria di primo livello il documento debba essere etichettato, mentre i restanti 16 si occuperanno di classificare il documento con una delle categorie interne alla macrocategoria d'ingresso.

Ciò è utile in quanto, se il classificatore di primo livello etichetta il documento con una data macrocategoria, il successivo da applicare sarà solo quello che contiene tutti i nodi discendenti da essa, riducendo così il numero di etichette da analizzare e rendendo il classificatore più accurato e veloce.

Poiché, quando si dovrà classificare un documento, gli verrà applicato il classificatore di primo livello, esso dovrà essere in grado di etichettare un qualsiasi articolo che può avere un contesto le cui features non compaiano nei documenti relativi alle macrocategorie. Dunque, è necessario, in questo approccio, che il primo classificatore in fase di apprendimento abbia a disposizione anche i documenti appartenenti ai vari nodi discendenti dalle prime 16 categorie.

Prima di passare alla spiegazione del codice si vuole concludere la spiegazione logica dell'approccio in questione mostrandone una immagine esplicativa, vedi *figura 46*.

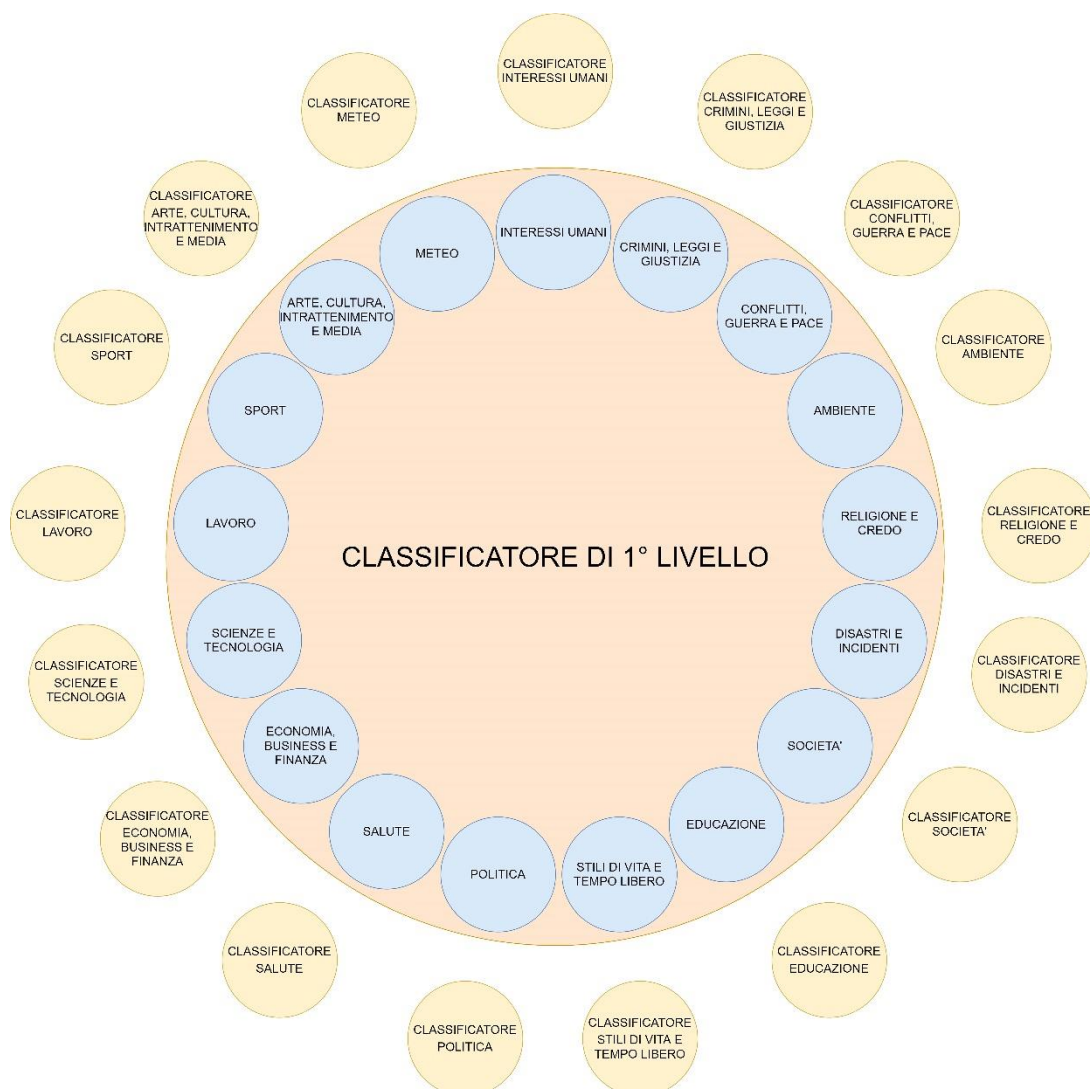


FIGURA 46: IMMAGINE MOSTRANTE IL GRUPPO DI CLASSIFICATORI CREATI UTILIZZANDO L'APPROCCIO A DUE LIVELLI. IN BASE ALL'OUTPUT USCENTE FUORI DALL'APPLICAZIONE DEL CLASSIFICATORE DI 1° LIVELLO AL DOCUMENTO, VERRÀ APPLICATO IL RELATIVO CLASSIFICATORE DI 2° LIVELLO.

Di seguito verranno mostrate e discusse le immagini mostranti i codici utili all'implementazione dell'approccio definito poc'anzi.

In figura 47 è mostrato il codice utile a far iniziare la pipeline che realizzerà automaticamente l'insieme dei classificatori.

Nello specifico, il metodo “k_cross()” verrà richiamato una volta per la realizzazione del classificatore di primo livello e 16 volte per la creazione dei classificatori di secondo livello. All'interno di tale metodo viene effettuato lo split del dataset in training-set e test-set che vengono passati al metodo che si occuperà di estrarre le features dai documenti e creare il modello.

Una volta ottenuta l'accuracy dei classificatori testati sul test-set per tutte le iterazioni del K-fold, si eliminano tutti i modelli salvati che hanno ottenuto accuracy inferiore. Da notare come, nel caso del primo classificatore viene richiamato il metodo “ExpandStructures()” che permette di aggiungere al training-set e al test-set alcuni dei documenti relativi ai nodi figli di nodi di primo livello.

```
def k_cross(df_doc,path_list,CMT):
    df_data = df_doc.loc[df_doc["CMT"].isin(path_list)]
    custom_y=df_data["CMT"].values
    #Suddivisione del dataset, test_size mi assicura che il 25% dei documenti finisca nel test-set
    n_splits=4
    kf = StratifiedShuffleSplit(n_splits=n_splits,test_size=0.25,random_state=123)
    accuracy=[]
    i=0
    for train_index, test_index in kf.split(df_data,y=custom_y):
        #Documenti del training-set e relative categorie d'appartenenza
        train = df_data.iloc[train_index]
        y_train = df_data.iloc[train_index].loc[:, 'CMT']
        #Documenti del test-set e relative categorie d'appartenenza
        test = df_data.iloc[test_index]
        y_test = df_data.iloc[test_index].loc[:, 'CMT']

        #Se si sta realizzando il primo classificatore bisogna integrare alcuni dei documenti dei nodi figli
        #così da avere un punto d'ingresso sicuro per nodi che non hanno features in comune con il padre
        if CMT=="0":
            expandator = Expandator()
            #Metodo che si occuperà di estendere il training-set e il test-set
            train,y_train,test,y_test=expandator.ExpandStructures(train,y_train,test,y_test)

        #Metodo che fa partire la pipeline per la creazione del classificatore
        score=CreateClassifier(train,y_train,test,y_test,CMT,i)
        accuracy.append(score)
        i=i+1
    print("Accuracy: ",accuracy)
    print("Max: ",max(accuracy)," Index: ",accuracy.index(max(accuracy)))
    bestModelIndex=accuracy.index(max(accuracy))
    for index in range(n_splits):
        if index != bestModelIndex:
            os.remove("./resources/models_stored/svmClf"+CMT[0:2]+str(index)+".pkl")
def GetAllChildsFromCmt(CMT):...

if __name__ == "__main__":
    CmtNodeFirstLevel=["01000000","02000000","03000000","04000000","05000000","06000000","07000000","08000000","09000000","10000000","11000000","12000000","13000000","14000000","15000000","16000000"]
    #Caricamento DataFrame contenente i documenti a cui è applicata la pipeline NLP
    df_data = pd.read_pickle("./resources/textClean.pkl")

    #Primo livello: classificatore per i 16 nodi di primo livello
    k_cross(df_data,CmtNodeFirstLevel,"0")

    #Secondo livello: 1 classificatore per ognuno dei 16 nodi di primo livello
    for child2Lvl in CmtNodeFirstLevel:
        #Metodo che ritorna la lista di tutti i nodi discendenti da un nodo di primo lvl
        CmtNodeSecondLevel = GetAllChildsFromCmt(child2Lvl)
        k_cross(df_data,CmtNodeSecondLevel,child2Lvl)
```

FIGURA 47: CODICE CHE SI OCCUPA DI PREPROCESSARE I DATI AL FINE DI RICHIAMARE I METODI PER LA REALIZZAZIONE DEI CLASSIFICATORI.

In *figura 48* vi è mostrato il codice che racchiude le chiamate ai due metodi fondamentali per la realizzazione dei modelli di classificazione, cioè l'estrazione delle features e la creazione e memorizzazione dei classificatori.

```
def CreateClassifier(train,y_train,test,y_test,CMT,i):  
  
    #Estrazione delle features, due casi, classificatore di 1° livello, classificatori di 2° livello  
    if CMT=="0":  
        vectorizer,data_corpus=GetTopFeatures(train)  
    else:  
        vectorizer,data_corpus=GetTopFeaturesSubNode(train,CMT)  
  
    #Creazione MODELLO  
    score=Svm_model(vectorizer,y_train,test,y_test,data_corpus,CMT,i)  
  
    return score
```

FIGURA 48: CODICE CHE SI OCCUPA DI LANCIARE CONSECUTIVAMENTE L'ESTRAZIONE DELLE FEATURES E LA CREAZIONE DEL MODELLO.

In *figura 49a* è mostrato il codice utile all'estrazione delle features dai documenti che compongono il training-set per il classificatore di 1° livello.

Nello specifico, effettuata una prima estrazione delle features per i documenti della stessa categoria vengono memorizzate all'interno di una lista tutte le features uniche estratte.

Successivamente, viene riapplicato il TF-IDF su tutti i documenti del training-set con le features estratte in precedenza.

Infine, vengono ritornati l'oggetto *vectorizer* e i testi dei documenti appartenenti al train.

Mentre in *figura 49b* è mostrato il codice utile all'estrazione delle features dai documenti che compongono il training-set per i classificatori di 2° livello.

Il funzionamento è identico a quello descritto in *figura 49a* con la differenza che, prima di applicare l'estrazione delle features dei documenti, si deve definire la lista di nodi discendenti dalla macrocategoria che si sta processando.

Inoltre, il numero minimo di documenti su cui deve apparire un termine per essere ritenuto una features è di 0.05; ciò significa che esso debba essere presente in almeno il 5% dei documenti.

```
def FeatureExtraction_tfid(train,CMT):  
    #Si prendono tutti quei termini che compaiono in almeno 5 documenti diversi  
    vectorizer = feature_extraction.text.TfidfVectorizer(sublinear_tf=True, min_df=5)  
    data_corpus = train[train["CMT"] == CMT]["Lemma"] #confrontare con text  
  
    vectorizer.fit(data_corpus)  
    features = vectorizer.get_feature_names()  
  
    return features  
  
def GetTopFeatures(train,pathList=[]): #versione del tf-idf avanzata con refit secondo le top features  
    if pathList:  
        path_list=pathList  
    else:  
        path_list=["01000000","02000000","03000000","04000000","05000000","06000000","07000000","09000000"]  
  
    #Estrazione delle features prendendo in analisi solo i documenti della stessa categoria  
    featureList = []  
    for nodeCMT in path_list:  
        X_names=FeatureExtraction_tfid(train,nodeCMT)  
        for feature in X_names:  
            featureList.append(feature)  
  
    #Rimozione dei doppioni dalla lista di features estratte  
    featureList = list(dict.fromkeys(featureList))  
  
    #Rielaborazione del TF-IDF su tutto il training-set con le features estratte  
    vectorizer = feature_extraction.text.TfidfVectorizer(vocabulary=featureList, sublinear_tf=True)  
    data_corpus = train["Lemma"] #confrontare con text  
    vectorizer.fit(data_corpus)  
  
    return vectorizer,data_corpus
```

FIGURA 49A: CODICE CHE SI OCCUPA DELL'ESTRAZIONE DELLE FEATURES DAI DOCUMENTI CHE COMPONGONO IL TRAINING-SET PER IL CLASSIFICATORE DI 1° LIVELLO.

```

def FeatureExtraction_tfidfSubNode(train,CMT):

    data_corpus = train[train["CMT"] == CMT][["Lemma"]]
    vectorizer = feature_extraction.text.TfidfVectorizer(sublinear_tf=True,min_df=0.05)

    vectorizer.fit(data_corpus)
    features = vectorizer.get_feature_names()

    return features

def GetTopFeaturesSubNode(train,CMT):

    #Estrazione di tutti i nodi figli di un nodo di 1° livello
    listPaths=[]
    src = "./Classificatore/LearningSet/"+CMT+"/"
    for dirpath, dirnames,files in os.walk(src):

        if len(files) > 3:
            listPaths.append(dirpath[-8:])
    listPaths.pop(0)

    #Estrazione delle features prendendo in analisi solo i documenti della stessa categoria
    featureList = []
    for nodeCmt in listPaths:
        X_names=FeatureExtraction_tfidfSubNode(train,nodeCmt)
        for feature in X_names:
            featureList.append(feature)

    #Rimozione dei doppioni dalla lista di features estratte
    featureList = list(dict.fromkeys(featureList))

    #Rielaborazione del TF-IDF su tutto il training-set con le features estratte
    vectorizer = feature_extraction.text.TfidfVectorizer(vocabulary=featureList, sublinear_tf=True)
    data_corpus = train["Lemma"]
    vectorizer.fit(data_corpus)

    return vectorizer,data_corpus

```

FIGURA 49B: CODICE UTILE ALL'ESTRAZIONE DELLE FEATURES DAI DOCUMENTI CHE COMPONGONO IL TRAINING-SET PER I CLASSIFICATORI DI 2° LIVELLO.

Infine, in *figura 50* vi è il codice utile alla realizzazione e memorizzazione in locale del modello di classificazione.

Nello specifico si verifica che il modello non sia già stato creato e in tal caso lo si crea attraverso i seguenti step: creazione dell'oggetto SVM; fitting del modello passando la matrice TF-IDF e le relative categorie; creazione di un'oggetto "pipeline" che permette di racchiudere in un singolo elemento l'estrazione delle features e il modello di classificazione.

Successivamente, il modello viene testato attraverso il metodo "predict()", il quale ritorna la lista di categorie che ha predetto per i documenti del test-set.

Tale lista viene passata al metodo “accuracy_score()” insieme alla lista y_test contenente le reali categorie d’appartenenza dei documenti del test-set, al fine di definirne un punteggio utile a capirne l’accuratezza del modello.

```
def Svm_model(vectorizer,y_train,test,y_test,data_corpus,CMT,i):
    modelPath = "./resources/models_stored/svmClf"+CMT+str(i)+".pkl"
    #Matrice documenti-features
    X_train = vectorizer.transform(data_corpus)

    if path.exists(modelPath):
        print("Modello già esistente")
        with open(modelPath, 'rb') as file:
            model = pickle.load(file)

    else:
        print("Creazione del modello")
        clf = svm.SVC(probability=True)
        clf.fit(X_train,y_train)
        model = pipeline.Pipeline([("vectorizer", vectorizer),("classifier", clf)])
        with open(modelPath, 'wb') as file:
            pickle.dump(model, file)

    #Testing del modello di classificazione sul test-set
    X_test = test["Lemma"].values
    predicted = model.predict(X_test)
    score = accuracy_score(predicted,y_test)

    return score
```

FIGURA 50: CODICE UTILE ALLA REALIZZAZIONE E MEMORIZZAZIONE IN LOCALE DEL MODELLO DI CLASSIFICAZIONE.

Da notare che il codice riportato in *figura 50*, viene effettuato K volte, con K definito durante l’implementazione della cross validation. Tuttavia, con il controllo presente in *figura 47* tra i K classificatori creati per lo stesso gruppo di categorie, K-1 verranno eliminati, cioè quelli che hanno avuto una accuracy inferiore.

L'esecuzione del codice appena discusso ha portato alla realizzazione dei seguenti classificatori:

```
svmClf00.pkl  
svmClf010000000.pkl  
svmClf020000003.pkl  
svmClf030000002.pkl  
svmClf040000001.pkl  
svmClf050000001.pkl  
svmClf060000001.pkl  
svmClf070000003.pkl  
svmClf090000003.pkl  
svmClf100000001.pkl  
svmClf110000000.pkl  
svmClf120000002.pkl  
svmClf130000002.pkl  
svmClf140000003.pkl  
svmClf150000001.pkl  
svmClf160000001.pkl  
svmClf170000000.pkl
```

FIGURA 51: LISTA DEI CLASSIFICATORI REALIZZATI. DA NOTARE CHE L'ULTIMO NUMERO PRIMA DELL'ESTENSIONE ".PKL" INDICA L'ITERAZIONE DEL K-FOLD CHE HA PORTATO AD AVERE UNA ACCURACY MIGLIORE.

Secondo approccio: classificatore gerarchico

In questo secondo approccio si è realizzato un classificatore per ogni insieme di nodi che condividono lo stesso padre, in questo modo si avrà una gerarchia di classificatori che rispecchiano fedelmente la tassonomia gerarchica.

Nel nostro caso si avranno 130 classificatori che permettono di avere una accuratezza maggiore e di ridurre ulteriormente il tempo necessario all'esecuzione del processo di classificazione in quanto il numero di etichette candidate all'assegnazione del documento si riducono.

Il processo sarà il seguente: dato un testo, vi si applica il primo classificatore che permette di individuare quale tra le prime 16 categorie sia quella più pertinente. Definita tale etichetta, si applica il classificatore relativo ad essa che permette di identificare quale tra i nodi figli sia la categoria che meglio descriva il documento. Tale processo viene ripetuto fino all'arrivo di un nodo che non sia padre di nessun altro o che l'applicazione del classificatore abbia portato una probabilità di assegnazione della categoria inferiore ad una data soglia.

In questo approccio, a differenza del primo, è necessario aggiungere i documenti dei nodi discendenti per tutti i nodi non foglia. Tale operazione è necessaria in quanto durante il processo di estrazione delle features si devono avere a disposizione anche i documenti relativi ai nodi discendenti dalla gerarchia. Ciò permette di considerare anche le loro features ed arrivare ad avere un punto d'ingresso anche per i documenti che appartengono ad una categoria che ha features diverse da quelle del nodo padre.

Nell'implementazione di questo approccio si effettua un controllo per ogni nodo che permette di evitare la creazione di un classificatore se il livello che si sta analizzando contiene solo un nodo. In tal caso, esso diventa fratello del padre e viene considerato nella creazione del classificatore del padre stesso.

Prima di passare alla spiegazione del codice utile all'implementazione dei classificatori, si vuole concludere con un'immagine che possa spiegare al meglio la logica utilizzata nel suddetto approccio.

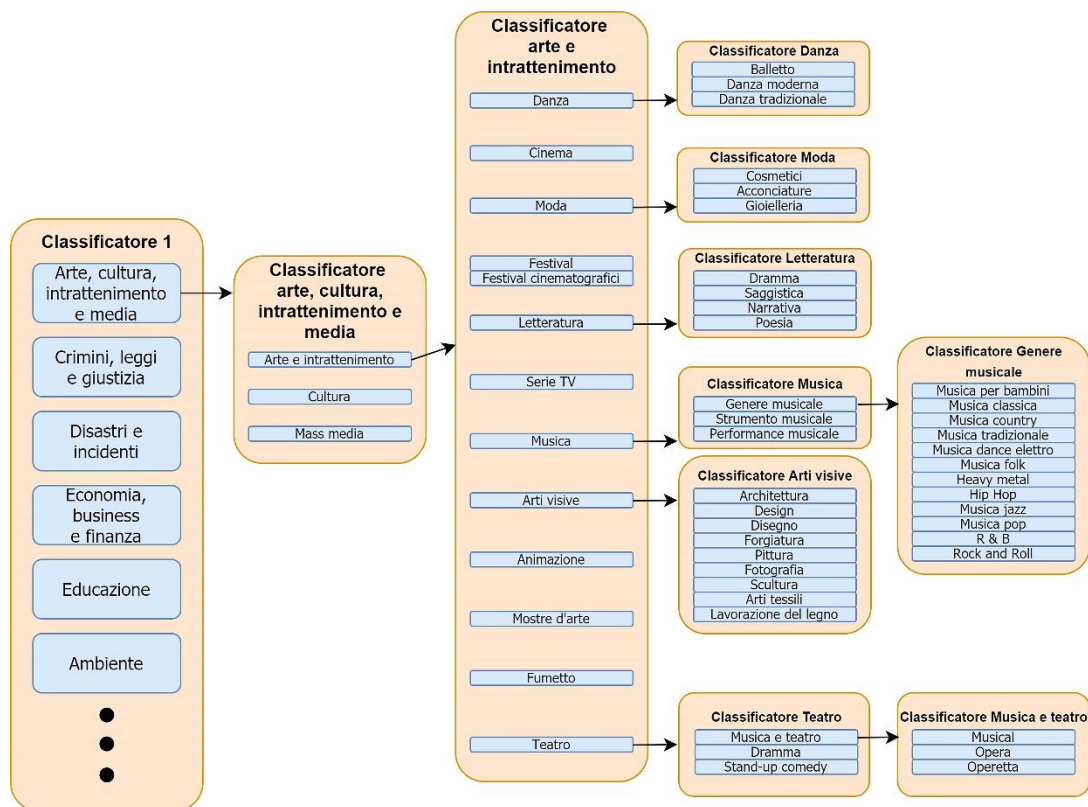


FIGURA 51: IMMAGINE MOSTRANTE ALCUNI DEI CLASSIFICATORI CREATI UTILIZZANDO L'APPROCCIO GERARCHICO. DA NOTARE COME IL NODO "FESTIVAL CINEMATOGRAFICI" ESSENDO IL SOLO FIGLIO DI "FESTIVAL" VIENE PORTATO AL LIVELLO DEL PADRE E DUNQUE PRESO IN CONSIDERAZIONE ALL'INTERNO DEL CLASSIFICATORE "ARTE E INTRATTENIMENTO"

Di seguito verrà mostrato il codice scritto per l'implementazione dell'approccio gerarchico.

In figura 52 è mostrato il codice che si occupa di far partire la pipeline che porterà alla creazione dell'insieme dei classificatori.

Nello specifico, è stato istanziato un oggetto *Predictor* dapprima definito e per ogni nodo di primo livello è stato richiamato il metodo *StartPipeline()* che si occupa di: realizzare la directory che conterrà tutti i classificatori relativi al nodo che si sta analizzando; definire la lista dei figli e controllare se essi siano più di uno ed in tal caso passarli al metodo *CreateClf()*. Quest'ultimo controllo è utile poiché permette di evitare la creazione di un classificatore che analizzi solo un nodo.

```
def __init__(self,entryCMT="",df_path="./resources/textClean.pkl"):
    print("INIT")
    self.fastObj = fastClass()
    self.cmtMaster = entryCMT
    self.df_data = pd.read_pickle(df_path)
    self.expandator = Expandator()
    self.Path=""

def StartPipeline(self,CMT,buildDf):

    #Creazione Dir che conterrà tutti i classificatori del nodo di 1° livello che si sta analizzando
    self.Path = "./resources/gerarchia_modelli/"+CMT+"/"
    if not path.exists(self.Path):
        os.makedirs(self.Path)

    src = "./Classificatore/LearningSet/"+CMT+"/"

    for dirpath, dirnames,files in os.walk(src):
        #Estrazione dei figli del nodo che si sta analizzando
        node=dirpath[-8:]
        if node[7]==" ":
            node=dirpath[-9:-1]
        childs=buildDf.loc[buildDf["Padre"]==node]

        if not childs.empty:
            #Salvataggio del CMT del nodo su cui si deve creare il classificatore in grado
            #di assegnare il documento ad una delle categorie figlie
            self.cmtMaster = childs.iloc[0]["Padre"]
            figli = (childs["CMT"].values).tolist()
            paths = (childs["Path"].values).tolist()

            #Verifico che il nodo che si sta analizzando abbia più di un figlio
            #Ciò è utile perché si evita di creare un classificatore per solo un nodo.
            #Se il nodo ha un solo figlio, esso viene portato allo stesso livello del padre.
            if(len(figli)>1):
                self.CreateClf(figli,buildDf,paths)
            else:
                print("Nodo con un solo un figlio, passo")

if __name__ == "__main__":
    predictObj = Predictor()
    buildDf = pd.read_pickle("./resources/final_media.pkl" )[[ "CMT","Lvl","Padre","Path"]]
    path_FirstLevellist=["01000000","02000000","03000000","04000000","05000000","06000000","07000000","08000000","09000000"]

    for cmt in path_FirstLevellist:
        predictObj.StartPipeline(cmt,buildDf)
```

FIGURA 52: CODICE UTILE A FAR PARTIRE LA PIPELINE CHE CREERÀ L'INSIEME DEI CLASSIFICATORI.

In figura 53 vi è mostrato il metodo “CreateClf()” che, presa la lista di figli del nodo che si sta analizzando su cui si dovrà creare il classificatore, si occupa di verificare se essi, a sua volta, sono padri di un solo figlio ed in tal caso il metodo si occupa di portare tali singoli nodi al livello del padre.

Fatto ciò, vengono eseguite le seguenti operazioni: estrazione dei documenti relativi ai nodi presi in analisi; aggiunta dei documenti relativi a tutti i nodi discendenti; suddivisione del dataset in training-set e test-set; estrazione delle features attraverso i metodi definiti in figura 49a, relativa all’approccio a 2 livelli e chiamata al metodo “KNN_model()” che si occupa di creare il modello di classificazione.

```
def splitData(self,df_child): ...

def checkAnotherChild(self,cmt,buildDf): ...

def CreateClf(self,figli,buildDf,paths):

    #Controllo se il nodo ha un solo figlio := se così fosse esso da figlio diventa fratello del padre
    list_child_childCmt=[]
    list_child_childPath=[]
    for figlio in figli:
        #Metodo che ritorna la lista dei figli del nodo passato
        df_sonOfSon = self.checkAnotherChild(figlio,buildDf)
        if len(df_sonOfSon) == 1:
            list_child_childCmt.append(df_sonOfSon["CMT"].item())
            list_child_childPath.append(df_sonOfSon["Path"].item())
    figli.extend(list_child_childCmt)
    paths.extend(list_child_childPath)

    #Estraggo i documenti delle categorie su cui si deve creare il classificatore
    df_child = self.df_data.loc[self.df_data["CMT"].isin(figli)]

    #Effettuo la suddivisione del dataset in train e test set
    train,test,y_train,y_test=self.splitData(df_child)

    #Espando il train e il test set con i documenti dei nodi discendenti
    train,y_train,test,y_test=self.expandator.ExpandStructures(train,y_train,test,y_test,figli,paths)

    #Estrazione delle features con metodo uguale all'approccio a 2 livelli
    vectorizer,data_corpus=GetTopFeatures(train,figli)

    #Creazione del classificatore
    self.KNN_model(train,vectorizer,y_train,test,y_test,data_corpus)
```

FIGURA 53: CODICE UTILE AD ESTRARRE E SUDDIVIDERE IL DATASET RELATIVO AI NODI SU CUI SI STA CREANDO IL CLASSIFICATORE SU CUI VERRÀ APPLICATA L’ESTRAZIONE DELLE FEATURES.

In figura 54 è raffigurata l’ultima parte di codice che si occupa della creazione del classificatore per un determinato gruppo di nodi della gerarchia.

Nello specifico, definita la matrice contenente i valori della TF-IDF, si è impostata una soglia massima di K pari a 40.

Se i documenti del training-set superano tale soglia, sarà uguale a 40, altrimenti essa sarà pari alla dimensione del training-set.

Successivamente, si è implementato un ciclo da 1 a K che ad ogni iterazione crea un oggetto K-NN, ne fa il fitting passando la matrice TF-IDF e le etichette relative ad ogni documento, testa il modello sul test-set e ne salva il tasso d'errore.

Poiché ad ogni iterazione il parametro K (numero di vicini da analizzare in fase di predict) varia, si va ad estrarre il valore di K che ha portato un tasso d'errore minore e lo si utilizza per creare il classificatore finale.

Come nell'approccio a 2 livelli, l'ultimo step da eseguire prima, di salvare il modello in locale, è la creazione di un oggetto di tipo "pipeline" che permette di unire le operazioni "estrazione delle features" ed "applicazione del modello di classificazione" così da poterle eseguire una dopo l'altra quando verrà eseguito il *predict*.

```
def KNN_model(self,train,vectorizer,y_train,test,y_test,data_corpus):
    X_train = vectorizer.transform(data_corpus)

    #Imposto limite massimo del parametro K
    #se ci sono meno di 40 elementi sul train si basa su quelli che ha
    if(len(y_train)<40):
        limit= len(y_train)
    else:
        limit = 40

    error_rate = []
    #Creerà 40 classificatori e li testerà con un K che va da 1 a 40
    #Per ogni classificatore creato avverrà una fase di testing che permetterà
    #di capire il tasso d'errore del modello
    for i in range(1,limit):
        clf = KNeighborsClassifier(n_neighbors=i)
        clf.fit(X_train,y_train)
        model = pipeline.Pipeline([("vectorizer", vectorizer),("classifier", clf)])
        X_test = test["Lemma"].values
        predicted = model.predict(X_test)
        error_rate.append(np.mean(predicted != y_test))
    print("ErrorRate:",error_rate)

    #Preso l'indice dell'errore minore si potrà settare la K che ha portato il miglior risultato
    #Dunque, si potrà realizzare e salvare il classificatore con quella K
    k = error_rate.index(min(error_rate))
    model =KNeighborsClassifier(n_neighbors=k)
    model.fit(X_train,y_train)
    customPipe = pipeline.Pipeline([("vectorizer", vectorizer),("classifier", model)])
    print("Accuracy:",accuracy_score(y_test, predicted))
    modelPath=self.Path+self.cmtMaster+".pkl"
    with open(modelPath, 'wb') as file:
        pickle.dump(customPipe, file)

    return
```

FIGURA 54: CODICE CHE IMPLEMENTA LA RICERCA DEL PARAMETRO *K* MIGLIORE PER CREARE IL CLASSIFICATORE. TROVATO ESSO LO SI UTILIZZA PER REALIZZARE E SALVARE IL CLASSIFICATORE PER I NODI IN ANALISI.

Di seguito viene mostrata la lista dei 130 modelli creati dall'approccio di creazione dei classificatori gerarchico.

| | | |
|---|--|--|
| <pre> 01000000 01000000.pkl 20000002.pkl 20000007.pkl 20000011.pkl 20000013.pkl 20000018.pkl 20000021.pkl 20000029.pkl 20000030.pkl 20000031.pkl 20000038.pkl 20000045.pkl 02000000 02000000.pkl 20000082.pkl 20000087.pkl 20000106.pkl 20000107.pkl 20000111.pkl 20000121.pkl 20000126.pkl 20000129.pkl 03000000 03000000.pkl 20000139.pkl 20000143.pkl 20000148.pkl 20000151.pkl 04000000 04000000.pkl 20000170.pkl 20000171.pkl 20000192.pkl 20000209.pkl 20000217.pkl 20000225.pkl 20000235.pkl 20000243.pkl 20000256.pkl 20000271.pkl 20000294.pkl 20000304.pkl 20000316.pkl 20000322.pkl 20000344.pkl 20000346.pkl 20000385.pkl 20001244.pkl 05000000 05000000.pkl 20000400.pkl 20000411.pkl </pre> | <pre> 06000000 06000000.pkl 20000420.pkl 20000424.pkl 20000430.pkl 20000432.pkl 20000437.pkl 20000441.pkl 07000000 07000000.pkl 20000446.pkl 20000448.pkl 20000449.pkl 20000464.pkl 20000470.pkl 20000479.pkl 20000485.pkl 20000487.pkl 20000493.pkl 09000000 09000000.pkl 20000509.pkl 20000524.pkl 10000000 10000000.pkl 20000538.pkl 20000540.pkl 20000550.pkl 20000551.pkl 20000552.pkl 20000553.pkl 20000560.pkl 20000565.pkl 20000568.pkl 20000570.pkl 20001250.pkl 11000000 11000000.pkl 20000345.pkl 20000574.pkl 20000587.pkl 20000593.pkl 20000598.pkl 20000615.pkl 20000621.pkl 20000626.pkl 20000638.pkl 20000639.pkl 20000649.pkl 20000653.pkl </pre> | <pre> 12000000 12000000.pkl 20000657.pkl 20000659.pkl 20000689.pkl 20000690.pkl 20000697.pkl 20000705.pkl 13000000 13000000.pkl 20000717.pkl 20000719.pkl 20000731.pkl 20000735.pkl 20000742.pkl 20000756.pkl 14000000 14000000.pkl 20000497.pkl 20000501.pkl 20000502.pkl 20000504.pkl 20000775.pkl 20000780.pkl 20000784.pkl 20000788.pkl 20000799.pkl 20000802.pkl 20000808.pkl 20000810.pkl 20000817.pkl 15000000 15000000.pkl 20000822.pkl 20001108.pkl 16000000 16000000.pkl 20000053.pkl 20000056.pkl 20000065.pkl 20000077.pkl 17000000 17000000.pkl </pre> |
| <p>FIGURA 55A: MODELLI PER LA GERARCHIA DEI NODI DA 01 A 05.</p> | <p>FIGURA 55B: MODELLI PER LA GERARCHIA DEI NODI DA 06 A 11.</p> | <p>FIGURA 55C: MODELLI PER LA GERARCHIA DEI NODI DA 12 A 17.</p> |

Per concludere con la spiegazione dei due approcci, si vuole spiegare la motivazione che ha portato ad utilizzare l'algoritmo di machine learning SVM per l'approccio a 2 livelli e il K-NN per l'approccio gerarchico.

Per arrivare a tale decisione sono stati effettuati dei confronti in termini di tempo e accuratezza e, per ognuno dei due approcci, sono stati implementati entrambi gli algoritmi di apprendimento automatico, i quali hanno portato le seguenti performance:

- **Approccio a 2 livelli:**
 - **SVM:**
 - Tempo per la creazione dei 17 classificatori: 17,00 minuti.
 - Accuratezza media pari al 58%.
 - **K-NN:**
 - Tempo per la creazione dei 17 classificatori: 3,00 minuti.
 - Accuratezza media pari al 46%.
- **Approccio gerarchico:**
 - **SVM:**
 - Tempo per la creazione dei 130 classificatori: 12,26 minuti.
 - Accuratezza media pari al 72%.
 - **K-NN:**
 - Tempo per la creazione dei 130 classificatori: 3,46 minuti.
 - Accuratezza media pari al 65%.

Mostrati i seguenti valori sembrerebbe che i due algoritmi si completino, in quanto i modelli creati con l'SVM hanno portato una accuracy media più alta con lo svantaggio di impiegare molto tempo; viceversa, i modelli creati con il K-NN sono risultati essere molto più veloci nella creazione ma con un valore medio di accuracy più basso.

Dunque, sulla base di questi risultati si è scelto di utilizzare l'SVM per quanto riguarda il primo approccio, in quanto ha portato ad avere una accuratezza media più alta del K-NN nonostante impieghi più tempo per la realizzazione dei modelli.

Ciò è stato deciso poiché, avendo solo 17 classificatori relativi a molte categorie, si è preferito dare più importanza all'accuratezza piuttosto che al tempo, viceversa, nel secondo approccio, si è scelto il K-NN poiché i modelli di classificazione sono 130 ed ognuno di essi si occupa di classificare poche categorie.

Per tale motivo si è preferito dare più importanza al tempo piuttosto che all'accuratezza che, in questo caso varia di poco.

Poiché l'accuratezza dell'SVM è risultata essere la migliore, si è deciso di utilizzare il primo classificatore dell'approccio a due livelli anche per quello gerarchico. Dunque, entrambi condividono lo stesso modello di classificazione per quanto riguarda i 16 nodi di primo livello.

Capitolo 9

9. Analisi e confronto delle performances

Una volta implementati entrambi gli approcci si è deciso di testarli su un'ulteriore test-set contenente coppie così composte:

$$\{link ; categorie\}$$

Ogni coppia è composta da un link rappresentante l'url dell'articolo e una lista contenente le categorie Media Topics che potrebbero essere assegnate ad esso.

Si è deciso di passare più etichette, poiché nel mondo reale una notizia può essere associata a più categorie contemporaneamente.

Tale test-set, diversamente da quello utilizzato all'interno del software utile all'implementazione dei due approcci, non è stato creato automaticamente tramite codice bensì, è stato realizzato internamente dal team di progetto.

Ciò è stato necessario in quanto si può confrontare l'accuratezza dei due insiemi di classificatori con le effettive categorie che assegnerebbe un'ipotetica persona che si debba occupare del suddetto compito.

Dunque, per definire l'accuratezza dei due approcci sono stati implementati per ogni coppia del test-set i seguenti steps:

1. **Applicazione dello scraping alla pagina web identificata dal link interno alla coppia**, attraverso i metodi spiegati nel paragrafo 6.2.3. Effettuata l'estrazione della notizia, il testo è stato salvato in un file in locale.
2. **Applicazione della pipeline NLP al testo della notizia estratta**, attraverso i metodi definiti nel paragrafo 7.1.

3. **Applicazione al testo ripulito del primo modello di classificazione**, chiamato classificatore padre, in grado di assegnare una delle 16 categorie di primo livello al testo. Dall'output sono state estratte le prime 3 categorie con probabilità più alta e prese in analisi per gli step successivi solo quelle la cui probabilità risulti essere maggiore di 0,10.
4. **Applicazione dei modelli di classificazione dei due approcci** aventi come punto d'ingresso le categorie date in output dal classificatore padre.
5. **Approccio a due livelli**: per ogni categoria ritornata dallo step 3, è stato importato e applicato al testo il modello relativo. L'output uscente fuori dalla predizione del modello saranno le tre categorie predette con probabilità più alta. Esse saranno salvate all'interno di una lista che conterrà anche le categorie definite dal classificatore padre.
6. **Approccio gerarchico**: per ogni categoria data in output dallo step 3, è stato importato ed applicato al testo in analisi il relativo modello di classificazione.
Esso darà in output le prime 3 categorie la cui probabilità risulti essere maggiore di 0,10.
Ricorsivamente verranno applicati i classificatori relativi alle categorie date in output. Il ciclo ricorsivo finirà quando, per la categoria che si sta analizzando, non esiste nessun classificatore salvato in locale.
Tutte le categorie che sono state analizzate all'interno di questo step sono state salvate all'interno di una lista.

7. **Definizione del path:** letta la categoria assegnata all'articolo, definita all'interno del test-set, ne è stato definito il path all'interno della gerarchia della tassonomia Media Topics.

Le categorie interne al path sono state salvate in una lista che permetterà di definire uno score per i due approcci.

8. **Calcolo accuracy:** ottenute le liste degli steps 5, 6 e 7 contenenti le categorie con probabilità più alte predette dai due approcci (step 5 e 6) e le categorie che formano il path dell'etichetta assegnata all'articolo (step 7), è stato definito il punteggio dell'accuratezza dei due approcci nel seguente modo:

○ Inizializzate le variabili:

- $score_2lvl, score_gerarchico = 0.$
- $score = 1.$
- $list_2lvl = lista\ step\ 5$
- $list_gerarchico = lista\ step\ 6$
- $list_path_label = lista\ step\ 7$
- $score_single = \frac{score}{len(list_path_label)}$

○ Se la categoria assegnata all'articolo è presente nella lista $list_2lvl$:

$$score_2lvl = score_2lvl + score$$

○ Se la categoria assegnata all'articolo non è presente nella lista $list_2lvl$: per ogni categoria interna alla lista $list_path_label$ si controlla se essa è presente nella lista $list_2lvl$, se così fosse

$$score_2lvl = score_2lvl + score_single.$$

- Se la categoria assegnata all'articolo è presente nella lista *list_gerarchico*:

$$score_{gerarchico} = score_{gerarchico} + score$$

- Se la categoria assegnata all'articolo non è presente nella lista *list_gerarchico*: per ogni categoria interna alla lista *list_path_label* si controlla se essa è presente nella lista *list_2lvl*, se così fosse

$$score_{gerarchico} = score_{gerarchico} + score_{single}.$$

Il controllo, che verifica la presenza di una delle categorie interne al path nella lista di categorie predette dai classificatori, permette di premiare l'insieme di classificatori dei due approcci anche quando non predicono la categoria corretta, ma ci si avvicinano, tornando ad esempio una categoria con cui l'etichetta assegnata alla notizia condivide lo stesso padre.

- Applicato lo step 8 per tutti i link interni al test-set, è possibile ottenere l'accuratezza dei due approcci dividendo lo *score_2lvl* e lo *score_gerarchico* per il numero di articoli analizzati.

Un esempio di output del codice che implementa gli steps sopra definiti è il seguente:

```
[ 'https://www.finestresullarte.info/arte-base/giotto-vita-opere-principali-importanza' ] [ '01000000' ]
Effettuo lo scraping => salvo il testo estratto in un file => leggo il file => applico pipeline NPL => passo il testo al classificatore padre
=====
CLASSIFICATORE PADRE
Predizioni con probabilità >= 0.01
=====
CMT      Prob      Cat
0 01000000 0.664252 arte, cultura, intrattenimento e mass-media
1 12000000 0.194152 religione
2 14000000 0.035069 sociale
=====
CLASSIFICATORE GERARCHICO
=====
Prendo il classificatore gerarchico: 01000000

d = 1 CMT: 20000002 - arte e intrattenimento - Prob : 0.875
  d = 2 CMT: 20000031 - arti visive - Prob : 0.5806451612903226
    d = 3 CMT: 20000035 - pittura - Prob : 0.5
    d = 3 CMT: 20000037 - scultura - Prob : 0.3333333333333333
    d = 3 CMT: 20000033 - design (arti visive) - Prob : 0.16666666666666666
  d = 2 CMT: 20001135 - mostre d'arte - Prob : 0.1935483870967742

d = 1 CMT: 20000038 - cultura - Prob : 0.125
  d = 2 CMT: 20000040 - costumi e tradizione - Prob : 0.3157894736842105
  d = 2 CMT: 20000042 - lingua - Prob : 0.21052631578947367
  d = 2 CMT: 20000043 - biblioteca e museo - Prob : 0.21052631578947367

Prendo il classificatore gerarchico: 12000000

d = 1 CMT: 20000657 - credo religioso - Prob : 0.2857142857142857
d = 1 CMT: 20000703 - leader religioso - Prob : 0.2857142857142857
d = 1 CMT: 20000704 - papa - Prob : 0.2857142857142857

=====
CLASSIFICATORE 2 LIVELLI
=====
Prendo il classificatore 2 LIVELLI 01000000
Prendo il classificatore 2 LIVELLI 12000000
=====
CMT      Prob      Cat
0 20000035 0.305800 pittura
1 20000002 0.112553 arte e intrattenimento
2 20000028 0.058393 opera
=====
CMT      Prob      Cat
0 20000674 0.117393 culto e setta
1 20000689 0.110325 evento religioso
2 20001271 0.085979 Giorno di Ognissanti
=====
Label true: 01000000 arte, cultura, intrattenimento e mass-media
Lista categorie predette CLF GER: [ '01000000', '20000002', '20000031', '20000035', '20000037', '20000033', '20001135', '20000038', '20000040', '20000042',
'20000043', '12000000', '20000657', '20000703', '20000704' ]
Lista categorie predette CLF 2LVL: [ '01000000', '20000035', '20000002', '20000028', '12000000', '20000674', '20000689', '20001271' ]
Path label: [ '01000000' ]
SCORE 2LVL: 1 GER: 1
Accuracy Cl Gerarchico: 100.0
Accuracy Cl 2 Livelli: 100.0
```

FIGURA 56: OUTPUT CHE MOSTRA L'ESECUZIONE DEGLI STEPS DEFINITI PER CALCOLARE LE PERFORMANCES DEI CLASSIFICATORI REALIZZATI TRAMITE I DUE APPROCCI.

Infine, l'insieme dei classificatori è stato testato su tutto il test-set ed i valori dell'accuratezza dei due approcci basata su 616 articoli è la seguente:

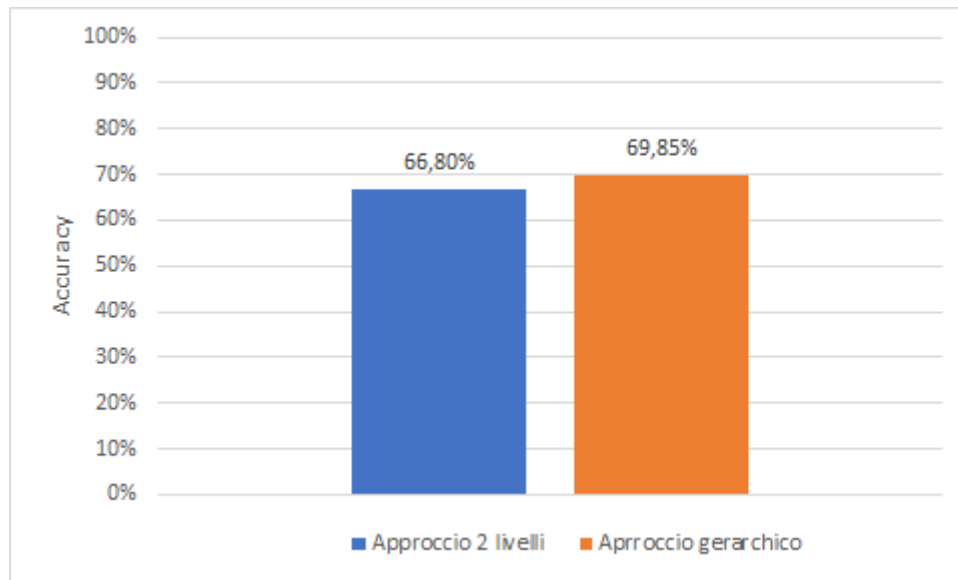


FIGURA 57: IMMAGINE MOSTRANTE I VALORI DELL'ACCURATEZZA CALCOLATA SULL'APPROCCIO A 2 LIVELLI E L'APPROCCIO GERARCHICO.

Dai risultati ottenuti si può notare che, se pur di poco, l'approccio gerarchico risulta essere il migliore.

Capitolo 10

10. Sviluppi futuri

Ottenuto l'insieme di classificatori la cui accuratezza è risultata la migliore, si conclude la spiegazione logica e pratica del progetto.

Se pur i risultati ottenuti dal software implementato siano soddisfacenti, essendo una prima versione del progetto, sicuramente ci sono vari processi al suo interno che possono essere affinati, in particolare:

- **Rilevamento degli outliers interni al dataset:** nonostante i molti controlli implementati per l'estrazione dei documenti dalla piattaforma Google News, sono presenti, se pur con frequenza bassa, degli outliers all'interno del dataset, cioè documenti/notizie che sono stati erroneamente assegnati ad una categoria. Quindi, un miglioramento all'interno di questo processo potrebbe essere quello di rilevarli e rimuoverli o riassegnarli ad un'altra categoria automaticamente, rendendo il dataset più accurato.
- **Verifica delle features estratte:** per via degli outliers interni al dataset, alcuni nodi, molto specifici e con pochi documenti, durante la fase di estrazione delle features ne hanno dato in output alcune che si discostano molto dal contesto che rappresentano. Dunque, un miglioramento in tal senso potrebbe essere l'applicazione di un ulteriore controllo che vada a determinare quanto un termine sia correlato alla label che si sta analizzando.
- **Ottimizzazione degli hyperparametri interni ai modelli:** un miglioramento sostanziale per quanta riguarda la capacità di classificare correttamente un testo è impostare e testare diversi hyperparametri interni agli algoritmi di apprendimento. Ciò porterà a definirne i migliori da utilizzare per rendere il modello stesso più accurato in base ai dati su cui si sta lavorando.

Capitolo 11

11. Conclusioni

Si conclude così il processo di sviluppo del software in grado di realizzare automaticamente un dataset e un insieme di classificatori in grado di etichettare in maniera automatica un articolo di stampa italiana.

Grazie alla progettazione e allo sviluppo di questo software, ho avuto la possibilità di poter mettere in pratica tutti i concetti teorici e pratici che ho acquisito in questi anni di percorso universitario.

Nello specifico, ho avuto l'occasione di migliorare le mie competenze nell'utilizzo del linguaggio di programmazione Python e nello sviluppo logico-pratico di classificatori.

Infine, ho avuto la possibilità di affinare quelle competenze di base, quali: lavoro in team; utilizzo di ambienti di lavoro finalizzati alla collaborazione da remoto; la metodologia Agile per lo sviluppo di progetti; competenze di problem-solving e rispetto delle tempistiche prestabilite per portare a termine i task assegnatimi.

Capitolo 12

12. Sitografia

- Introduzione alla classificazione del testo:
<https://towardsdatascience.com/https-medium-com-noa-weiss-the-hitchhikers-guide-to-hierarchical-classification-f8428ea1e076>
- Casi d'uso della classificazione del testo:
<https://towardsdatascience.com/text-classification-applications-and-use-cases-beab4bfe2e62>
- IPTC:
<https://iptc.org/>
- Media Topics:
<https://iptc.org/standards/media-topics/>
- Documentazione libreria GoogleNews:
<https://pypi.org/project/GoogleNews/>
- Lista operatori logici per le ricerche Google:
<https://ahrefs.com/blog/google-advanced-search-operators/>
- Documentazione libreria BeautifulSoup:
<https://www.crummy.com/software/BeautifulSoup/bs4/doc/>
- Documentazione NLP:
https://en.wikipedia.org/wiki/Natural_language_processing
- Documentazione libreria Spacy:
<https://spacy.io/usage/linguistic-features#named-entities>
- Documentazione CrossValidation:
https://scikit-learn.org/stable/modules/cross_validation.html
- Documentazione BOW:
https://en.wikipedia.org/wiki/Bag-of-words_model
- Documentazione TF-IDF:
<https://en.wikipedia.org/wiki/Tf%E2%80%93idf>

- Documentazione Word2Vec:
<https://medium.com/wisio/a-gentle-introduction-to-doc2vec-db3e8c0cce5e>
- Documentazione per l'implementazione del TF-IDF:
https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html
- Documentazione SVM:
<https://www.lorenzogovoni.com/support-vector-machine/>
- Documentazione per l'implementazione di un classificatore SVM:
<https://scikit-learn.org/stable/modules/svm.html>
- Documentazione K-NN:
<https://www.lorenzogovoni.com/knn/>
- Documentazione per l'implementazione di un classificatore K-NN:
<https://scikit-learn.org/stable/modules/neighbors.html>

Capitolo 13

13. Ringraziamenti

Ringrazio il **Professore Salvatore Nicotra**, nonché relatore della suddetta, il quale mi ha dato la possibilità di svolgere il tirocinio all'interno dell'azienda Neodata s.r.l. e di continuare a poter lavorare costantemente al progetto in quanto appassionati delle tecnologie utilizzate.

In particolar modo, essendo il primo tesista del Prof. Nicotra, vorrei sottolineare l'importanza che ha avuto per me seguire le lezioni da lui sostenute, poiché, grazie alla grande passione trasmessa mi ha fatto avvicinare molto ad un insieme di nuove tecnologie, da me sconosciuto, tanto da volerne basare il mio futuro lavorativo.

Ciò dimostra quanto sia importante per noi studenti stare in contatto professori che riescano a trasmettere non solo i canonici contenuti didattici ma anche la passione che li ha portati ad essere docenti.

Ringrazio il **Dott. Emanuele Mambelli**, tutor aziendale del tirocinio, il quale mi ha seguito durante lo sviluppo del software, fornendomi continui feedback per il conseguimento ottimale del progetto.

Ringrazio tutti i docenti del dipartimento di Informatica, i quali mi hanno dato la possibilità di apprendere nuove conoscenze che mi daranno la possibilità di ampliare il mio bagaglio culturale e lavorativo.

Ringrazio il collega, amico e a breve **Dottore Aldo Fiorito** con il quale ho condiviso il mio percorso di studi dal primo superiore, studiando insieme per qualsiasi interrogazione, compito o esame che abbiamo sostenuto, ripetendoci sempre il nostro motto “O lo passiamo O la morte”.

Caro Aldo, dopo più di un anno dove ci siamo sentiti giorno dopo giorno davanti ad un computer, anche questo traguardo siamo riusciti ad ottenerlo insieme. Colgo l'occasione per dirti che ti ringrazio per tutte le volte in cui avevo bisogno d'aiuto e ci sei stato, dimostrandoti più che un amico di studio ma bensì un amico fraterno.

Ti auguro un futuro pieno di soddisfazioni con la speranza di poter lavorare sempre fianco a fianco.

Ringrazio i miei amici **Damiano, Francesco, Davide e Simone** per avermi sempre sostenuto ed aiutato in tutti i momenti della mia vita.

Nonostante la mia presenza nella vostra vita non sia sempre stata costante, fate parte della mia vita e ci sarò sempre per voi.

Siete importantissimi per me e anche se non ve lo dico spesso, vi voglio bene.

Ringrazio **Marta**, che fa parte della mia vita da molti anni e mi ha sostenuto nei momenti più importanti.

Ci siamo conosciuti quando eravamo dei ragazzini e siamo cresciuti insieme anno dopo anno. Ti ringrazio per esserci sempre stata e per il sostegno che mi hai dato in questi anni, spero di poter continuare a condividere insieme tutte le gioie e gli obiettivi futuri.

Ringrazio mia cugina **Teresa**, che ha sempre creduto in me sostenendomi sia nei momenti belli sia in quelli meno belli.

Ti voglio molto bene sorellina.

Ringrazio tutti gli amici e parenti che sono presenti in questo giorno così importante per me.

Infine, ringrazio la mia famiglia, mia madre **Tiziana**, mio padre **Alfio** e mio fratello **Marco** che mi sono sempre stati vicino sostenendomi e dandomi la forza per oltrepassare qualsiasi problema. In particolar modo li ringrazio per credere continuamente in me e avermi dato sempre la possibilità di prendere le mie scelte senza essere condizionato in alcun modo.

Vi ringrazio, siete la famiglia migliore in cui potessi crescere, parte di questo traguardo è vostro, vi amo.