

UNIVERSITÀ DEGLI STUDI DI MILANO -
BICOCCA

Dipartimento di Economia e Statistica
Corso di Laurea Triennale in Scienze Statistiche ed
Economiche



Un'applicazione del Machine Learning per
meta-analisi in ambito clinico

Relatore: Ing. Mirko Cesarini

Relazione della prova finale di:

Julius Maliwat

Matricola 864520

Anno Accademico 2022-2023

Dedicata alla mia famiglia

Indice

| | | |
|----------|------------------------------------|-----------|
| 1 | Introduzione | 1 |
| 1.1 | Contesto | 2 |
| 1.2 | Strumenti | 3 |
| 1.3 | Struttura dell'elaborato | 4 |
| 2 | Stato dell'arte | 7 |
| 3 | Sviluppo Web App | 11 |
| 3.1 | Web App | 11 |
| 3.2 | Dash Framework | 12 |
| 3.2.1 | Dash Layout | 12 |
| 3.2.2 | Dash Components | 14 |
| 3.2.3 | Dash Callbacks | 18 |
| 3.3 | Struttura | 20 |
| 4 | Classificazione dei testi | 25 |
| 4.1 | Text Classification | 25 |
| 4.2 | Labeling Manuale | 26 |
| 4.3 | Preprocessing | 27 |
| 4.3.1 | Text Processing | 27 |
| 4.3.2 | Undersampling | 32 |
| 4.4 | Classificatori | 33 |
| 4.4.1 | Regressione Logistica | 34 |
| 4.4.2 | Support Vector Machine | 35 |
| 4.4.3 | Random Forest | 36 |
| 4.5 | TPOT | 37 |

| | | |
|----------|--|-----------|
| 4.6 | BioBert | 40 |
| 4.7 | Metriche | 40 |
| 5 | Dataset e Risultati | 45 |
| 5.1 | Descrizione del Dataset | 45 |
| 5.2 | Labeling dei dati | 48 |
| 5.3 | Hyperparameter Tuning | 50 |
| 5.3.1 | Grid Search | 50 |
| 5.3.2 | Randomized Search | 51 |
| 5.4 | Training e scelta del modello | 53 |
| 5.5 | Analisi dei risultati ottenuti | 61 |
| 6 | Conclusioni | 65 |

Capitolo 1

Introduzione

La tesi si focalizza sul lavoro svolto durante il mio stage interno presso il Dipartimento di Statistica e Metodi Quantitativi, in stretta collaborazione con il professore Mirko Cesarini. L'obiettivo principale di questo lavoro è stato lo sviluppo di una web application innovativa, progettata per affrontare una sfida di rilevanza nel campo della ricerca clinica. La sua finalità è semplificare l'analisi e la sintesi di un vasto corpus di studi al fine di agevolare la conduzione di meta-analisi cliniche. Durante il percorso di stage, ho dedicato tempo e impegno allo sviluppo dei modelli e all'implementazione e realizzazione di questa soluzione.

Oltre allo sviluppo della web app, un altro importante aspetto di questo lavoro di tesi è stato l'approfondimento dei classificatori di machine learning per la classificazione dei testi. Durante il lavoro svolto, mi sono dedicato allo studio e all'analisi di diverse tecniche di classificazione, concentrandomi in particolare sull'applicazione di tali modelli alla discriminazione di frasi relative a studi clinici randomizzati.

1.1 Contesto

Nel campo della ricerca clinica, l'analisi e la sintesi di un vasto corpus di studi rappresentano una sfida significativa per i ricercatori. La meta-analisi si configura come un metodo consolidato per combinare i risultati di studi indipendenti al fine di ottenere una valutazione globale e oggettiva dell'efficacia di un trattamento o di un intervento medico [?]. Tuttavia, l'abbondanza di studi pubblicati e la mole di informazioni disponibili possono rendere complesso il processo di selezione dei paper più rilevanti per una determinata meta-analisi. Per affrontare questa problematica, è stata sviluppata una web application per agevolare e accelerare il processo di ricerca dei paper appropriati per la meta-analisi clinica. Questa applicazione offre un'interfaccia intuitiva e funzionalità avanzate per semplificare la ricerca e l'accesso ai paper pertinenti. Uno strumento chiave integrato nella web app è il framework PICO (Popolazione, Intervento, Confronto, Outcome) [1]. La web app permette agli utenti di inserire una query basata su PubMed e specificare il numero di paper da processare e scaricare. I modelli sviluppati restituiscono, per ciascun paper, le quattro frasi più rilevanti per ciascuna lettera del framework PICO, facilitando così la valutazione della rilevanza dei documenti e accelerando il processo di selezione.

Oltre allo sviluppo della web app, mi sono dedicato all'approfondimento degli algoritmi basati sul paradigma del machine learning per la classificazione dei testi [2], concentrandomi specificamente sulla classificazione delle frasi relative agli studi clinici randomizzati.

Durante questa fase, ho analizzato e confrontato diversi algoritmi come Support Vector Machine (SVM) [3], Random Forest [4] e Regressione Logistica [5]. Ho ottimizzato le performance degli algoritmi utilizzando l'auto machine learning ed in particolare la libreria TPOT [6].

Ho analizzato i risultati ottenuti da questi modelli e li ho confrontati con

le performance di BioBert, che è un modello preaddestrato specificamente per il dominio medico [7].

Il mio scopo era quello di valutare l'efficacia dei modelli tradizionali rispetto a un approccio più avanzato come BioBert nella classificazione di frasi che descrivono studi clinici randomizzati. Attraverso questo confronto, ho cercato di comprendere se i modelli tradizionali potessero raggiungere livelli di prestazioni simili o superiori rispetto allo stato dell'arte rappresentato da BioBert.

Questo approfondimento mi ha fornito informazioni preziose sulle capacità e le limitazioni dei modelli tradizionali nel contesto specifico degli studi clinici randomizzati, e mi ha permesso di valutare come tali modelli si confrontino con lo stato dell'arte rappresentato da BioBert.

1.2 Strumenti

Per lo sviluppo della web app, è stato utilizzato il linguaggio di programmazione Python [8], insieme al framework Dash [9]. Python offre un'ampia gamma di librerie e strumenti che consentono di affrontare diverse sfide di programmazione, inclusa l'implementazione di applicazioni web, mentre Dash permette di creare interfacce utente interattive e intuitive.

Per l'analisi dei dati e l'implementazione dei modelli di classificazione di testi, è stato utilizzato Jupyter Notebook [10], un ambiente di sviluppo interattivo che agevola l'esplorazione dei dati e la prototipazione dei modelli. In particolare, sono state sfruttate le potenzialità della libreria NLTK (Natural Language Toolkit) [11] per suddividere i testi in frasi e svolgere altre operazioni di pre-elaborazione dei dati.

Per la manipolazione e l'analisi dei dati, sono state utilizzate le librerie Pandas e NumPy. Pandas offre strumenti per la gestione dei dati

strutturati [12], mentre NumPy fornisce funzionalità per il calcolo numerico efficiente [13].

Per l'implementazione dei modelli di classificazione, è stata utilizzata la libreria scikit-learn [14], che fornisce una vasta gamma di algoritmi di machine learning, tra cui Support Vector Machine (SVM), Random Forest e Regressione Logistica.

L'utilizzo combinato di questi strumenti ha consentito di sviluppare la web app per la selezione dei paper e di implementare i modelli di classificazione necessari per il confronto con lo stato dell'arte, in questo caso rappresentato da BioBert.

1.3 Struttura dell'elaborato

Il presente elaborato è strutturato in diversi capitoli al fine di fornire una visione completa del lavoro svolto durante lo stage e dei risultati ottenuti. La seguente è una panoramica della struttura dell'elaborato.

Nel capitolo 2 viene presentata un'analisi approfondita dello stato dell'arte nel campo della ricerca clinica e della meta-analisi. Vengono esplorate le tecniche e le metodologie utilizzate per l'analisi e la sintesi dei paper clinici, nonché i principali approcci di machine learning e classificazione di testi impiegati nella letteratura scientifica.

Il capitolo 3 introduce il concetto di web app e illustra la struttura e le funzionalità della web app sviluppata durante lo stage. Viene fornita una panoramica dettagliata dell'architettura della web app, dei moduli e delle interfacce utente implementate per semplificare la ricerca e l'accesso ai paper clinici pertinenti.

Nel capitolo 4 viene presentata una panoramica approfondita della classificazione di testi. Vengono descritti i principali modelli tradizionali di classificazione, come Support Vector Machine (SVM), Random Forest e Regressione Logistica, nonché le tecniche di text processing utilizzate per il pre-processing dei testi. Inoltre, vengono approfonditi la libreria TPOT e il language model BioBert come strumenti per la costruzione di modelli avanzati e per la loro valutazione.

Il capitolo 5 presenta il dataset utilizzato per l'analisi e la valutazione dei modelli di classificazione di testi. Vengono descritte le caratteristiche del dataset e le metriche utilizzate per valutare le prestazioni dei modelli. Inoltre, vengono riportati e analizzati i risultati ottenuti, confrontando i modelli tradizionali con l'utilizzo di TPOT e soprattutto con BioBert.

Il capitolo 6 riporta le principali conclusioni emerse dal lavoro svolto durante lo stage. Vengono discusse le implicazioni dei risultati ottenuti e le possibili direzioni future di ricerca nel campo della meta-analisi clinica e dell'utilizzo dei modelli di machine learning per la classificazione dei testi.

Capitolo 2

Stato dell'arte

La ricerca clinica presenta una sfida significativa per i ricercatori riguardo all'analisi e alla sintesi di un vasto corpus di studi. Per ottenere una valutazione oggettiva e completa dell'efficacia di trattamenti o interventi medici, la meta-analisi si è affermata come un metodo consolidato che combina i risultati di studi indipendenti [?]. Tuttavia, la grande quantità di studi pubblicati e l'ampia disponibilità di informazioni complicano il processo di selezione dei paper più pertinenti per una meta-analisi specifica.

Esistono diverse vie alternative per affrontare questa problematica. Una di queste è l'approccio manuale, in cui i ricercatori svolgono una ricerca sistematica e selezionano manualmente i paper rilevanti da includere nella meta-analisi. Questo approccio richiede molto tempo e sforzo, e può essere influenzato da errori umani o da pregiudizi nell'identificazione dei paper appropriati.

Un'altra via alternativa è rappresentata dall'utilizzo di strumenti di ricerca automatica come PubMed, che consentono di filtrare gli studi in base a determinati criteri di ricerca [15]. Tuttavia, anche con l'ausilio di questi strumenti, è ancora necessario esaminare manualmente i risultati per selezionare i paper rilevanti, il che può essere un processo lungo e laborioso.

Per superare queste sfide, durante il lavoro descritto in questo elaborato

è stata sviluppata una web application focalizzata sulla ricerca dei paper appropriati per la meta-analisi clinica. Questa applicazione è stata progettata per offrire un'interfaccia semplice e intuitiva al fine di semplificare il processo di ricerca e accesso ai paper rilevanti. All'interno della web app, è possibile inserire una query basata su PubMed e specificare il numero desiderato di paper da scaricare. Attraverso l'implementazione del framework PICO (Popolazione, Intervento, Confronto, Outcome)[1] la web app restituisce poi le frasi più pertinenti per ciascuna lettera del framework, per ogni paper. L'integrazione del framework PICO rappresenta un elemento chiave della web app in quanto fornisce una struttura ben definita per la selezione dei paper. Utilizzando le frasi rilevanti per ciascuna lettera del framework, gli utenti possono valutare rapidamente la corrispondenza tra i paper identificati e i criteri specifici della meta-analisi.

La seconda parte del mio lavoro si inserisce nell'ambito della Classificazione dei testi supervisionata. L'obiettivo del lavoro infatti è quello di classificare le frasi relative a studi clinici randomizzati in maniera automatica utilizzando delle tecniche apposite. Per la classificazione dei testi è possibile scegliere diversi approcci.

Un approccio comune alla classificazione dei testi supervisionata è l'utilizzo di algoritmi di machine learning [2], come per esempio Support Vector Machine (SVM), Decision Trees[16] e K-Nearest Neighbors (KNN)[17]. Questi algoritmi vengono addestrati su un insieme di dati di addestramento annotati, in cui ogni frase è associata a una determinata classe o categoria predefinita. Una volta addestrati, questi modelli possono essere utilizzati per classificare nuove frasi in base alle informazioni apprese durante la fase di addestramento.

Negli ultimi anni, l'utilizzo delle reti neurali profonde [18] ha portato a significativi miglioramenti nelle prestazioni della classificazione dei testi.

In particolare, i modelli di elaborazione del linguaggio naturale basati su transformer, “BERT” (Bidirectional Encoder Representations from Transformers) [19], hanno ottenuto risultati eccezionali in diverse applicazioni legate alla classificazione dei testi.

Nel mio lavoro di tesi, esplorerò gli algoritmi di Machine Learning, in particolare Support Vector Machine, Random Forest e regressione logistica. L’obiettivo sarà quello di confrontare l’efficacia degli algoritmi appena elencati con lo stato dell’arte rappresentato da BioBert, modello di linguaggio preaddestrato basato su BERT, che è stato specificamente addestrato per l’elaborazione del linguaggio naturale (NLP) nel campo delle scienze biomediche e della biologia [7].

Capitolo 3

Sviluppo Web App

3.1 Web App

Una Web App, o “Web Application”, è un’applicazione software che viene eseguita su un server web e che gli utenti possono utilizzare attraverso un browser web [20]. Questo tipo di applicazione è progettato per fornire funzionalità interattive e accesso a servizi e risorse attraverso Internet.

Le Web App differiscono dalle applicazioni tradizionali installate localmente perché non richiedono l’installazione o l’esecuzione sul dispositivo dell’utente. Invece, vengono ospitate su un server remoto e possono essere accessibili da qualsiasi dispositivo connesso a Internet. Questa caratteristica offre flessibilità agli utenti, che possono utilizzare le Web App da computer desktop, laptop, smartphone, tablet o qualsiasi altro dispositivo compatibile con un browser web.

Le Web App possono essere sviluppate utilizzando una combinazione di tecnologie web, come HTML (Hypertext Markup Language) [21], CSS (Cascading Style Sheets) [22] e JavaScript [23]. Questi linguaggi consentono di creare l’interfaccia utente, gestire le interazioni con l’utente e connettersi a servizi o API esterne. Inoltre, sono disponibili diversi framework e librerie, come Angular, React e Vue.js [24], che semplificano lo sviluppo delle Web App, offrendo strumenti e componenti predefiniti per migliorare l’efficienza e la scalabilità. Nel nostro caso, andremo a

utilizzare il framework Dash (vedi Cap. 3.2).

Questo tipo di applicazioni possono offrire una vasta gamma di funzionalità, come gestione di contenuti, e-commerce, servizi di social media, strumenti di produttività, giochi online e altro ancora.

3.2 Dash Framework

Dash è un framework open source per la creazione di applicazioni web interattive e analisi dati in Python [9][25]. È progettato per semplificare lo sviluppo di applicazioni web complesse, consentendo agli sviluppatori di creare interfacce utente interattive scrivendo codice unicamente in Python (spesso si usa l'espressione *pure Python* oppure "Python Puro" per indicare ciò).

Dash consente agli sviluppatori di creare applicazioni analitiche e di data science completamente personalizzate, in cui il codice Python può essere utilizzato per effettuare calcoli complessi o applicare algoritmi di machine learning.

Gli sviluppatori possono creare applicazioni web con funzionalità di visualizzazione dei dati interattive, come grafici, tabelle e mappe. Dash offre anche componenti interattivi predefiniti, come selettori, slider e input di testo, che possono essere utilizzati per creare interfacce utente complesse e personalizzate.

Andremo a approfondire quelli che sono gli elementi principali del Framework Dash: Layout, Components, Callbacks.

3.2.1 Dash Layout

Il Layout si riferisce alla struttura e all'organizzazione degli elementi grafici e degli elementi interattivi all'interno di un'applicazione Dash. Il layout definisce come gli elementi visivi sono posizionati e disposti all'interno dell'interfaccia utente dell'applicazione web.

Un layout Dash può essere organizzato in modo flessibile e responsive¹, adattandosi alle diverse dimensioni dello schermo o ai dispositivi utilizzati dagli utenti. Ciò consente di creare interfacce utente che si adattano in modo dinamico e rispondono alle esigenze degli utenti.

L'impostazione del Layout può essere facilitato usando Bootstrap [26][27], e prevede 3 elementi principali: Container, riga, colonna.

Container

Il componente Container può essere usato per centrare e disporre orizzontalmente il contenuto dell'applicazione. Per impostazione predefinita, il contenitore ha una larghezza di pixel responsive. Si utilizza l'argomento `fluid=True` se si vuole che il contenitore riempi lo spazio orizzontale disponibile e si ridimensioni in modo fluido.

Row

Il componente Row è un involucro per le colonne. Il layout dell'applicazione dovrebbe essere costruito come una serie di righe di colonne.

Col

Il componente Col deve essere sempre usato come figlio immediato di Row ed è un involucro per il contenuto che assicura che occupi la giusta quantità di spazio orizzontale.

Il layout in Bootstrap è controllato da un sistema a griglia. La griglia di Bootstrap ha dodici colonne e sei livelli responsive (che consentono di specificare comportamenti diversi a seconda delle dimensioni dello schermo). La larghezza delle colonne può essere specificata in termini di quante delle dodici colonne della griglia deve coprire, oppure si può consentire alle

¹capacità di un sito di adattarsi in modo ottimale alle diverse dimensioni dello schermo dei dispositivi utilizzati dagli utenti, come desktop, tablet e smartphone.

colonne di espandersi o restringersi per adattarsi al loro contenuto o allo spazio disponibile nella riga.

In List. 1 si riporta un esempio di come strutturare il layout preso direttamente dalla documentazione di Bootstrap per Dash. Alle righe e alle colonne dell'esempio seguente sono stati applicati stili aggiuntivi con CSS, per garantire che gli effetti dei diversi argomenti siano chiari. In Fig. 3.1 è possibile vedere il layout ottenuto.

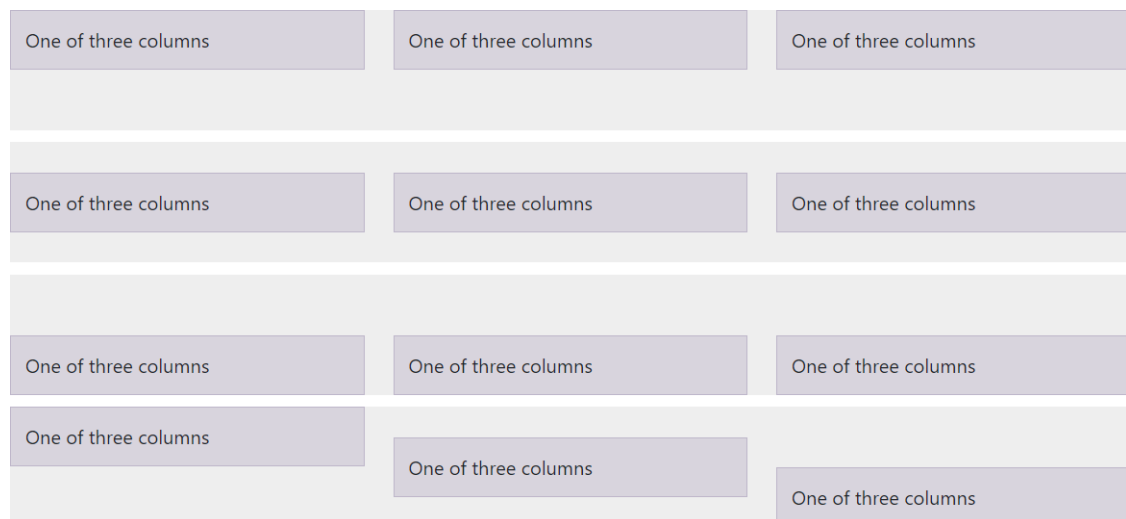


Figura 3.1: *Layout riferito a List. 1*

3.2.2 Dash Components

I "Dash Components" sono componenti interattivi predefiniti che possono essere utilizzati per creare l'interfaccia utente di un'applicazione Dash. Questi componenti offrono funzionalità specifiche, come grafici interattivi, input di testo, menu a discesa e molto altro ancora. I componenti in Dash possono essere suddivisi in due categorie principali: Dash Core Components e Dash HTML Components.

```

1  import dash_bootstrap_components as dbc
2  from dash import html
3
4  row = html.Div(
5      [
6          dbc.Row(
7              [
8                  dbc.Col(html.Div("One of three columns")),
9                  dbc.Col(html.Div("One of three columns")),
10                 dbc.Col(html.Div("One of three columns")),
11             ],
12             align="start",
13         ),
14         dbc.Row(
15             [
16                 dbc.Col(html.Div("One of three columns")),
17                 dbc.Col(html.Div("One of three columns")),
18                 dbc.Col(html.Div("One of three columns")),
19             ],
20             align="center",
21         ),
22         dbc.Row(
23             [
24                 dbc.Col(html.Div("One of three columns")),
25                 dbc.Col(html.Div("One of three columns")),
26                 dbc.Col(html.Div("One of three columns")),
27             ],
28             align="end",
29         ),
30         dbc.Row(
31             [
32                 dbc.Col(html.Div("One of three columns"), align="start"),
33                 dbc.Col(html.Div("One of three columns"), align="center"),
34                 dbc.Col(html.Div("One of three columns"), align="end"),
35             ]
36         ),
37     ],
38     className="pad-row",
39 )

```

Listing 1: Codice per strutturare Dash Layout [27]

Dash Core Components

I Dash Core Components sono disponibili attraverso il modulo `dash_core_components` di Dash. Questo modulo fornisce una vasta gamma di componenti pronti

all'uso che possono essere facilmente integrati nel layout dell'applicazione Dash.

Per avere un'idea generale di come funzionano le Dash Core Components, si riportano di seguito 2 esempi di componenti: Input e Button.

Input è una componente che consente agli utenti di inserire dati o testo all'interno dell'applicazione. Alcune delle proprietà includono:

- `id`: identificatore unico per il componente `dcc.Input`. Viene utilizzato per far riferimento al componente all'interno del codice Python
- `type`: il tipo di input. È possibile specificare il tipo di input come `"text"` per consentire all'utente di inserire del testo, o come `"number"` per consentire all'utente di inserire solo numeri.
- `placeholder`: Un testo di esempio che viene visualizzato all'interno della casella di input quando è vuota. Può fornire indicazioni agli utenti su cosa inserire.

In List. 2 si riporta un esempio di Web App utilizzando `dcc.Input` preso direttamente dal tutorial di Dash[28].

```

1  from dash import Dash, dcc, html
2
3  app = Dash(__name__)
4
5  app.layout = html.Div([
6      dcc.Input(
7          placeholder='Enter a value...',
8          type='text',
9          value=''
10     )
11 ])
12
13 if __name__ == '__main__':
14     app.run_server(debug=True)
15

```

Listing 2: Esempio Web App con Input

Button permette di creare pulsanti interattivi all’interno della Web App. I pulsanti sono utili per attivare azioni specifiche o scatenare eventi quando l’utente fa clic su di essi.

Alcune delle proprietà comuni includono:

- `id`: identificatore unico per il componente `dcc.Button`
- `children`: il testo o il contenuto del pulsante. Può essere una stringa di testo o un altro componente Dash che viene visualizzato all’interno del pulsante.
- `n_clicks`: tiene traccia del numero di volte in cui il pulsante è stato cliccato. Questa proprietà può essere utilizzata per eseguire azioni o aggiornare lo stato dell’applicazione in base ai clic del pulsante.

Un esempio di applicazione del componente Button si può vedere in List. 5, nel Cap. 3.2.3

Dash HTML Components

HTML Components sono disponibili attraverso il modulo `dash_html_components` di Dash. Questi componenti rappresentano gli elementi HTML standard

come titoli (<h1>, <h2>, ecc.), paragrafi (<p>), liste (,) e molti altri. Di seguito riporto un esempio che presenta la relazione tra Dash e HTML. In List. 3 si riporta un esempio di Web App utilizzando i Dash HTML Components.

```
1  from dash import html
2
3  html.Div([
4      html.H1('Hello Dash'),
5      html.Div([
6          html.P('Dash converts Python classes into HTML'),
7          html.P("This conversion happens behind the scenes by Dash's JavaScript front-end")
8      ])
9  ])
```

Listing 3: Esempio di Web App con Dash HTML components [29]

Il codice in List. 3 viene poi convertito (dietro le quinte) nel codice HTML con in List. 4

```
1  <div>
2      <h1>Hello Dash</h1>
3      <div>
4          <p>Dash converts Python classes into HTML</p>
5          <p>This conversion happens behind the scenes by Dash's JavaScript front-end</p>
6      </div>
7  </div>
```

Listing 4: Codice HTML riferito a List. 3

3.2.3 Dash Callbacks

Dash Callback si riferisce a una funzione Python che specifica il comportamento dinamico dell'applicazione Dash in risposta alle interazioni dell'utente. I Dash Callback consentono di collegare eventi, come clic su pulsanti o modifiche in input di testo, a azioni specifiche che possono includere l'aggiornamento di componenti, la manipolazione dei dati o il calcolo di risultati.

I Dash Callback vengono definiti utilizzando il decoratore “@app.callback” seguita dalla specifica degli input e degli output della funzione di callback.

Gli input sono gli eventi o le proprietà dei componenti che attivano la callback, mentre gli output sono i componenti che verranno aggiornati come risultato della callback.

Inoltre è possibile specificare “State”, che sono gli argomenti aggiuntivi che sono passati alla funzione di callback. La differenza con Input è che non attivano immediatamente la funzione di callback quando tali componenti vengono modificati.

In List. 5 si riporta un esempio di Callback in Dash.

```
1
2  @app.callback(
3      Output('output', 'children'),
4      [Input('submit-button', 'n_clicks')],
5      [State('input-1', 'value')]
6  )
7  def update_output(n_clicks, input1_value):
8      if n_clicks > 0:
9          return f'The button has been clicked {n_clicks} times, and the input value is "{input1_value}".'
10     else:
11         return 'Click the button to submit.'
12
13
14
```

Listing 5: Callback in Dash

Nell’esempio riportato in List. 5 abbiamo un input di testo (id = ‘input-1’), un bottone (id = ‘submit-button’) e un elemento di output (id = ‘output’). La funzione di Callback viene attivata solo quando il bottone viene cliccato.

Quando il bottone viene cliccato, la funzione di callback controlla se il numero di clic è maggiore di zero e restituisce un messaggio corrispon-

dente con il valore dell'input di testo. Se il bottone non è stato ancora cliccato, viene visualizzato un messaggio di invito.

3.3 Struttura

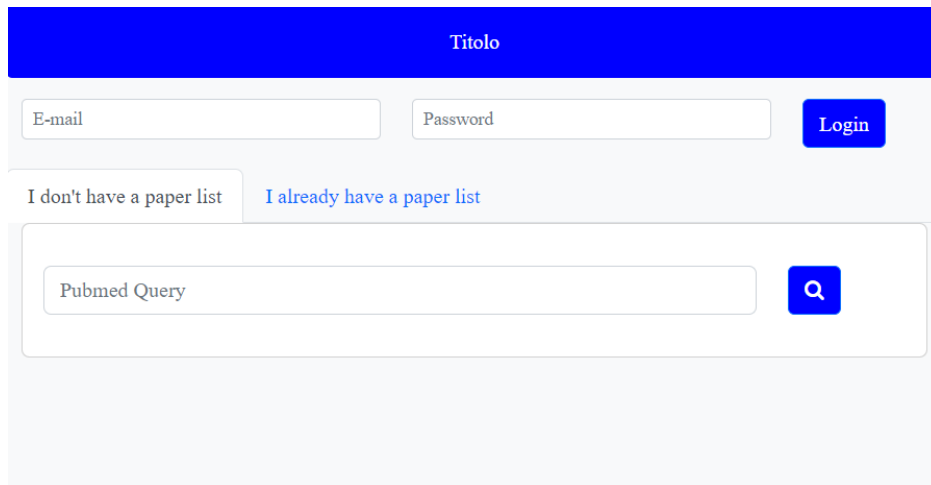
La web app è stata sviluppata durante lo stage interno in università con il supporto del prof. Cesarini. L'obiettivo principale di questo lavoro è stato quello di fornire ai ricercatori uno strumento per semplificare e accelerare il processo di individuazione dei paper rilevanti per le meta-analisi in ambito clinico [?]. È stata progettata per offrire una piattaforma intuitiva e user-friendly che permetta agli utenti di interagire facilmente con il framework PICO (Popolazione, Intervento, Confronto, Outcome). Il framework PICO è un acronimo utilizzato nel campo della ricerca clinica per strutturare le domande di ricerca e facilitare la selezione dei paper pertinenti ad una meta analisi [1]. Di seguito riporto una breve spiegazione delle sue componenti:

- Popolazione: si riferisce al gruppo di individui o pazienti che sono oggetto di studio.
- Intervento: indica il trattamento o l'intervento medico che viene valutato nella ricerca.
- Confronto: si riferisce al confronto tra l'intervento in questione e un altro trattamento o una condizione di controllo.
- Outcome: rappresenta il risultato o l'effetto che viene valutato per determinare l'efficacia dell'intervento.

Lo scopo finale della web app è quello di fornire all'utente un output dettagliato, che comprende quattro frasi rilevanti per ciascuna componente del framework PICO per ogni paper analizzato.

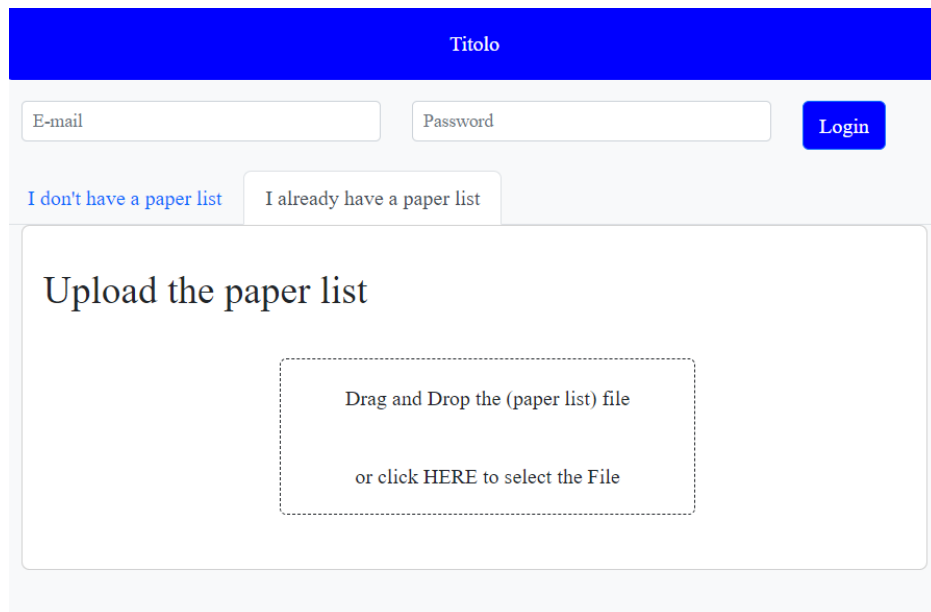
Durante lo sviluppo dell'applicazione, Alex Alborghetti, studente magistrale in biostatistica, si è occupato dello sviluppo dei modelli di machine learning e NLP utilizzati per estrarre le frasi rilevanti secondo il

framework PICO. Alex ha utilizzato alcuni dei dataset da me prodotti. L'interazione tra l'utente e la web app avviene nel seguente modo. Inizialmente l'utente ha la possibilità di utilizzare 2 modalità: inserire una query di ricerca specifica collegata a PubMed (vedi Fig. 3.2) o caricare direttamente la lista dei PMID (in formato json) dei paper da processare (vedi Fig. 3.3). Nel secondo caso, si passa direttamente al passaggio successivo.



The screenshot shows a web application interface. At the top is a blue header bar with the word "Titolo" in white. Below the header, there are two input fields: "E-mail" and "Password", followed by a blue "Login" button. Below these fields, there are two tabs: "I don't have a paper list" (selected) and "I already have a paper list". Under the selected tab, there is a large input field labeled "Pubmed Query" and a blue search button with a magnifying glass icon.

Figura 3.2: *Schermata iniziale con query*



The screenshot shows the same web application interface as Figure 3.2, but with the "I already have a paper list" tab selected. The main content area is titled "Upload the paper list" and contains a dashed box with the text "Drag and Drop the (paper list) file" and "or click HERE to select the File".

Figura 3.3: *Schermata iniziale per upload*

Nel primo caso, viene visualizzato il numero di paper disponibili e si chiede all'utente il numero di paper da processare (consigliato 50-100 articoli per motivi computazionali).

Una volta scelto il numero di paper da processare, oppure dopo aver caricato la lista dei PMID dei paper da processare, si chiede all'utente un'insieme di Keywords positive e negative (quelle negative sono specificate con il simbolo “-” davanti) divisi per categoria. Sono presenti quattro righe composte da 2 input in cui nel primo si inserisce il nome della categoria, mentre nel secondo si inseriscono le keywords separate da uno spazio.

Titolo

E-mail Password Login

I don't have a paper list I already have a paper list

(randomized controlled trial[Publication Type] OR randomized clinical trial[Pu

Result: 773349 paper(s)

How many papers to process? 5 Download and Process Papers

5 paper(s) to process - from download

Insert Category Insert Keywords

Insert Category Insert Keywords

Insert Category Insert Keywords

Insert Category Insert Keywords

PICO and Keywords

Figura 3.4: Schermata finale con Keywords da inserire

Una volta inserite le keywords (non è obbligatorio inserirle) l'utente clicca sul bottone "PICO and Keywords" per processare i paper. L'utente quindi scarica un foglio Excel contenente i risultati ottenuti col Framework PICO (vedi Fig. 3.4). Il file Excel include le quattro frasi più probabili per ogni lettera del framework PICO, per ciascun paper. Questo consente agli utenti di ottenere una panoramica rapida delle informazioni chiave presenti nei loro paper senza doverli leggere integralmente. Inoltre, se l'utente ha inserito delle keywords, il file excel include le quattro frasi più probabili per ogni categoria inserita.

L'estrazione delle frasi per il framework PICO è stata ottenuta utilizzando dei classificatori binari descritti nel capitolo successivo. L'estrazione delle frasi a seguito di parole chiave utilizza un algoritmo basato su Colbert [30] che mi è stato fornito a Alex Alborghetti, ed è stato da lui sviluppato assieme al prof. Cesarini. Per maggiori dettagli si rimanda a [31]

Capitolo 4

Classificazione dei testi

4.1 Text Classification

La classificazione dei testi, detta anche Text Classification, è un processo che consiste nel classificare documenti di testo in categorie predefinite in base al loro contenuto [2]. I classificatori di testi possono strutturare, organizzare e classificare una grande varietà di testi, da articoli scientifici a recensioni e testi trovati nel Web.

I testi sono un tipo di dato non strutturato, che non ha un modello predefinito e non può essere organizzato in righe e colonne.

“I dati non strutturati costituiscono la stragrande maggioranza dei dati presenti in un’organizzazione, secondo alcune stime fino all’80%.”

afferma Merrill, banca d’investimento con sede a New York [32]. Altre fonti riportano simili o percentuali più alte di dati non strutturati[33][34]. Data la mancanza di struttura nei testi, analizzare, capire, organizzare e ordinare dati testuali risulta essere difficile, e quindi la maggior parte delle aziende fallisce nello sfruttare al massimo il potenziale di questo tipo di dato.

La via tradizionale sarebbe quella di processare il dato manualmente, anche se ciò potrebbe richiedere una grande quantità di tempo.

Ecco perché entrano in gioco strumenti di classificazione dei testi le quali combinano tecniche di Machine Learning e Natural Language Processing per organizzare e analizzare un numero elevato di testi in maniera efficiente e sostenibile. Ciò significa poter classificare articoli in base all'argomento, o anche valutare le opinioni dei clienti sui social analizzando un'elevata quantità di post e commenti che parlano di un prodotto o servizio. Questo permette alle aziende di salvare tempo e prendere decisioni in base alle informazioni estratte da questo tipo di dati, in questo caso testi.

4.2 Labeling Manuale

Per la classificazione dei testi, l'approccio utilizzato sarà il machine learning supervisionato, che prevede l'addestramento di un modello di classificazione (vedi Cap. 4.4) utilizzando dati etichettati, cioè osservazioni nel dataset che sono associate a classi o etichette predefinite. Nella maggior parte dei casi, quando ci si avvicina a un nuovo problema di classificazione, le etichette o le classi associate alle osservazioni del dataset non sono disponibili o non sono state precedentemente assegnate, e quindi è necessaria un'operazione iniziale di Labeling manuale. Ciò consiste nell'etichettare o assegnare manualmente una determinata classe a osservazioni di un Dataset, in base al proprio giudizio o ai criteri specifici del problema. Il Labeling manuale può essere un compito oneroso e richiede una certa quantità di tempo e risorse umane, tuttavia è necessario quando non è disponibile un Dataset già annotato [35].

Ci sono diverse modalità in cui viene effettuato il labeling manuale, come ad esempio utilizzando strumenti specifici o interfacce grafiche che semplificano il processo. Nel mio caso, andrò a utilizzare uno strumento basato sull'approccio semi-supervised learning fornito dal prof. Cesarini che sarà introdotto nel Cap. 5.2

4.3 Preprocessing

La fase di Preprocessing dei dati in ambito Machine Learning comprende una serie di operazioni e trasformazioni applicate ai dati grezzi per permettere l'impiego di un modello di classificazione. Questa fase è fondamentale nella pipeline di sviluppo di un modello e può avere un impatto significativo sulle performance del modello finale [36].

La libreria scikit-learn [14] ha un intero pacchetto dedicato al Preprocessing dei dati.

Per il problema in questione la fase di Preprocessing è stata suddivisa in 2 parti: Text Processing e Undersampling. Verranno riportate di seguito le definizioni di questi 2 processi e diverse funzioni di libreria utilizzate.

4.3.1 Text Processing

Nel contesto della classificazione dei testi, Text Processing si riferisce all'insieme di operazioni eseguite sui testi prima di poterli utilizzare per addestrare un modello di classificazione [37].

Questa fase è fondamentale per migliorare le prestazioni dei classificatori. Le principali operazioni di Text Processing utilizzate sono Tokenization, Rimozione delle Stopwords, Stemming, Vectorization e saranno dettagliate nei paragrafi seguenti.

Tokenization

Tokenization consiste nel suddividere il testo in unità più piccole chiamate token. Questi possono essere parole o frasi. Inoltre Tokenization può comprendere anche l'eliminazione della punteggiatura e la normalizzazione delle parole (Esempio: convertire tutto in minuscolo) [38]. Per spiegare meglio il concetto di Tokenization si procede con un esempio. Si consideri la frase:

“Tanti auguri a Maria”

Possiamo utilizzare una suddivisione basata sugli spazi per separare le parole, ottenendo i seguenti token: “Tanti”, “auguri”, “a”, “Maria”.

Per applicare Tokenization in Python è possibile utilizzare la funzione `tokenize` della libreria `nlTK` [39], oppure attraverso una funzione che viene riportata in List. 6.

```
1 class Tokenizer(BaseWrapper):
2     def manageSentence(self, sentence):
3         """This method turn a single document (i.e., a string)
4         into a list of single words (i.e., tokens).
5         The parameter "sentence" is expected to be a string,
6         this method returns a list of strings whereas each string
7         is a tokenized word. This method replaces all the punctuation
8         with spaces.
9         Two or more consecutive spaces are reduced to a single space.
10        Then the string is splitted in substring using the spaces as
11        split markers"""
12
13        if sentence==None:
14            return []
15        # if the parameter is not the right type, the execution
16        is interrupted
17        assert type(sentence)==type('') or type(sentence)==type(u''), "Tokenizer Assertion Error"
18        punteggiatura=u'!{}[]?""",;.-<>|/\\"*+=+-% \n\t\r()' +u'\"' +u'\u2019'+u'\u2018'
19        #\r and \n can be used as "new line"
20        # Unicode Character 'RIGHT SINGLE QUOTATION MARK' (U+2019)
21        #
22        for l in punteggiatura:
23            #print(s)
24            sentence=sentence.replace(l,u" ") #replacing all punctuation characters with spaces
25
26        # loop untill all double spaces are removed
27        while sentence.find(u" ")!=1:
28            sentence=sentence.replace(u" ",u" ") #replacing double spaces with a single one
29        return sentence.split(u' ') #e.g., "a b c d".split(' ') returns ['a','b','c','d']
30
```

Listing 6: Codice per effettuare Tokenizer

Rimozione delle Stopwords

Consiste nel rimuovere le parole comuni ma poco informative da un testo. Queste parole, come articoli, preposizioni e congiunzioni, non apportano un significato sostanziale alla comprensione del testo e possono essere rimosse per ridurre la complessità e il rumore durante l’elaborazione [40].

Si riporta un esempio per chiarire meglio il concetto delle Stopwords. Consideriamo la frase seguente:

“Il gatto nero è saltato sul muro”

Nella lingua italiana, alcune parole come “il”, “è” e “sul” sono Stopwords comuni che possono essere rimosse senza perdere significato. L’obiettivo della rimozione delle Stopwords è mantenere solo le parole che sono più informative per l’analisi. Dopo la rimozione delle Stopwords, la frase potrebbe apparire così:

“gatto nero saltato muro”

Questa operazione ha semplificato il testo e consente di concentrarsi sulle parole chiave che potrebbero portare a una migliore comprensione o modellazione del testo.

La rimozione delle Stopwords può avvenire utilizzando una lista predefinita di parole specifiche per la lingua, oppure è possibile costruire una lista personalizzata in base alle esigenze specifiche del progetto.

Si riporta in List. 7 una classe per la rimozione delle Stopwords, dove inoltre è possibile notare che in questo caso la lista delle Stopwords è stata inserita manualmente.

Stemming

L’operazione di Stemming consiste nel ridurre le parole alla loro radice o forma base, nota come “stem”. L’obiettivo è di trasformare le parole in modo da trattare parole simili come se fossero identiche, semplificando l’analisi del testo [41].

Riprendendo la frase prodotta nell’esempio precedente di rimozione delle Stopwords, applicando lo Stemming otteniamo il risultato seguente:

“gatt ner salt mur”

```

1  class EnglishStopWordsRemover(BaseWrapper):
2      def getStopWords(self):
3          """This method returns a list of English stop words. Stop words can be added to the list"""
4          return ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves',
5                  'you', 'your', 'yours', 'yourself', 'yourselves',
6                  'he', 'him', 'his', 'himself', 'she', 'her', 'hers', 'herself',
7                  'it', 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves',
8                  'what', 'which', 'who', 'whom', 'this', 'that', 'these', 'those',
9                  'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being',
10                 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing',
11                 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as',
12                 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about',
13                 'against', 'between', 'into', 'through', 'during', 'before',
14                 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in',
15                 'out', 'on', 'off', 'over', 'under', 'again', 'further',
16                 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how',
17                 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other',
18                 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same',
19                 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just',
20                 'don', 'should', 'now', 'd', 'll', 'm', 'o', 're', 've', 'y',
21                 'ain', 'aren', 'couldn', 'didn', 'doesn', 'hadn', 'hasn', 'haven', 'isn',
22                 'ma', 'mightn', 'mustn', 'needn', 'shan', 'shouldn', 'wasn', 'weren',
23                 'won', 'wouldn']
24
25     def manageSentence(self, sentence):
26         """sentence is expected to be a list of words (a list where each item is a string containing a single word),
27         this method returns the input list where the stop words are removed """
28         assert type(sentence)==type([]) , "EnglishStopWordsRemover, Assertion Error"
29         stopWords = self.getStopWords()
30         return [w for w in sentence if w not in stopWords]

```

Listing 7: Codice per effettuare la rimozione delle Stopwords

In questo caso lo stemming semplifica la frase riducendo le parole alle loro radici. È importante però sottolineare alcuni svantaggi dell'operazione di Stemming. Uno di questi è che potrebbe generare radici che potrebbero essere ambigue in alcuni contesti oppure potrebbero non corrispondere a parole reali. È possibile applicare l'operazione di Stemming in Python con il pacchetto stem presente nella libreria nltk (vedi [42]), come presentato in List. 8.

Vectorization

L'operazione di Vectorization consiste nel rappresentare i dati non strutturati in vettori numerici, in modo da essere in un formato compatibile con gli algoritmi di Machine Learning, che richiedono input numerici.

```

1  class EnglishStemmer(BaseWrapper):
2      def __init__(self):
3          """Load the NLTK English stemmer. A stemmer is an algorithm that reduces a word
4              to its base form e.g., "books" is reduced to "book",
5              'children' is reduced to 'child'. """
6          self.st = nltk.stem.SnowballStemmer("english") # loading the NLTK stemmer
7      def manageSentence(self, sentence):
8          """sentence is expected to be a list of words (a list where each item is a
9              string containing a single word), this method returns a list of stemmed words"""
10         assert type(sentence)==type([]), "EnglishStemmer, Assertion Error"
11         return [self.st.stem(w) for w in sentence]
12

```

Listing 8: Codice per effettuare lo Stemming

Una delle tecniche più utilizzate è la “Bag of Words (BoW)” [43].

BoW prende il documento come un insieme non ordinato di parole, ignorando la struttura grammaticale e l’ordine delle parole nel testo. Dopo l’operazione di Tokenization viene creato un vocabolario che rappresenta l’insieme di tutte le parole uniche presenti nei documenti. Ogni parola del vocabolario rappresenta una variabile o “feature” del vettore finale. Per ogni documento viene creato poi un vettore numerico in cui per ogni variabile, che corrisponde a una parola del vocabolario, si inserisce il valore che può essere il conteggio delle occorrenze della parola nel documento oppure può essere una misura normalizzata della parola nel documento, come TF-IDF. TF-IDF è una tecnica che misura l’importanza di una parola all’interno di un documento in base alla sua frequenza all’interno del documento stesso e alla sua rarità nel corpus globale. Il valore TF-IDF di una parola aumenta quando compare frequentemente in un documento specifico e raramente negli altri documenti del corpus.

L’approccio BoW si può applicare in Python grazie alla classe Tfidf-Vectorizer della libreria scikit-learn. Un esempio di applicazione si può ricavare direttamente dal sito ufficiale di Scikit-learn, presentato in List.

```

1  from sklearn.feature_extraction.text import TfidfVectorizer
2  corpus = [
3      'This is the first document.',
4      'This document is the second document.',
5      'And this is the third one.',
6      'Is this the first document?',
7  ]
8  vectorizer = TfidfVectorizer()
9  X = vectorizer.fit_transform(corpus)
10 vectorizer.get_feature_names_out()
11
12
13 print(X.shape)
14

```

Listing 9: Codice per applicare Bag of Words

4.3.2 Undersampling

L'Undersampling è una tecnica utilizzata in ambito di classificazione nel caso in cui si ha un problema di dati non bilanciati. Questo problema si verifica quando una delle categorie predefinite è rappresentata da un numero molto inferiore di osservazioni rispetto alle altre categorie [44].

L'uso di dataset sbilanciati durante l'addestramento può portare alla creazione di un modello con una scarsa capacità di classificare correttamente la classe o le classi di minoranza.

L'undersampling consiste nel ridurre la quantità di osservazioni nella categoria maggioritaria in modo da renderla più bilanciata rispetto alla categoria minoritaria. Ciò viene fatto eliminando casualmente osservazioni dalla classe maggioritaria fino a raggiungere il rapporto di bilanciamento desiderato tra le classi.

Questa operazione può migliorare la capacità del modello a classificare correttamente la classe di minoranza. E' importante notare però che questa tecnica comporta una perdita di informazione poiché si eliminano dati dalla classe maggioritaria. Pertanto è necessario valutare attentamente i trade-off tra la riduzione dello sbilanciamento tra classi e la perdita di dati prima di applicare l'undersampling.

La funzione utilizzata per applicare l'undersampling su Python è riportata in List. 10

```
1 def underSample2Min(df, labelName):
2     ''' The dataset is undersampled so that all label groups will have the same size,
3         corresponding to the (original) minimal label set.
4         The parameter labelName is the DataFrame column hosting the labels'''
5
6     vc = df.loc[:,labelName].value_counts() # Counting label frequencies
7     lab2freq = dict(zip(vc.index.tolist(), vc.values.tolist()))
8     #print(lab2freq) # if you want to see lab2freq, please uncomment this command
9     #print(min(lab2freq.values()))
10    minfreq = min(lab2freq.values())
11    #print(minfreq)
12    idxSample=[]
13    for selectedLabel, actualFreq in lab2freq.items():
14        selIndexes=df.loc[df.loc[:,labelName]==selectedLabel, :].sample(n=minfreq, random_state = 42).index.tolist()
15        idxSample+=selIndexes
16    idxSample.sort()
17    #print(type(idxSample), idxSample)
18    #print(list(df.index)[:5])
19
20    df2 = df.loc[idxSample, :]
21    #print(len(idxSample), df2.shape);exit()
22    df2 = df2.reset_index() # otherwise missing index may cause problem
23    return df2
24
```

Listing 10: Codice per effettuare lo Stemming

4.4 Classificatori

Per classificare i testi in modo automatico, accurato e veloce vengono utilizzati gli algoritmi di Machine Learning per la classificazione [45]. Questi algoritmi imparano a classificare le osservazioni utilizzando osservazioni passate con la classe già nota. In questo modo possono apprendere le diverse associazioni che ci sono tra le caratteristiche (features) delle osservazioni e la classe. Per risolvere il problema in questione sono stati allenati 3 modelli di Machine Learning: regressione logistica, Support Vector Machine e Random Forest, maggiori dettagli saranno forniti nei paragrafi seguenti.

4.4.1 Regressione Logistica

La regressione logistica è il primo algoritmo di classificazione che sarà utilizzato per la classificazione dei testi. Nonostante il nome "regressione", la regressione logistica viene comunemente utilizzata per risolvere problemi di classificazione binaria [5].

L'obiettivo della regressione logistica è quello di stimare la probabilità che un'osservazione faccia parte di una classe specifica, data una serie di variabili indipendenti (le sue caratteristiche o features). L'algoritmo si basa su un modello logistico che utilizza una funzione logistica, o sigmoide, per mappare l'input a un valore compreso tra 0 e 1.

Nella fase di Training vengono utilizzati i dati di addestramento per apprendere i pesi o i coefficienti delle variabili indipendenti. Questi pesi vengono quindi utilizzati per calcolare la probabilità di appartenenza a una classe specifica per nuovi dati non osservati.

Una volta addestrato, per fare previsioni su nuovi dati in input, il modello calcola la probabilità che un dato appartenga a una determinata classe e, a seconda di una soglia di decisione scelta, assegna l'etichetta di classe corrispondente.

L'iperparametro che sarà validato in fase di addestramento è l'iperparametro C , che controlla l'importanza data alla riduzione dell'errore sui dati di training rispetto alla complessità del modello.

Se il valore di C è alto, la penalizzazione per gli errori sarà forte e il modello tenderà a cercare di adattarsi perfettamente ai dati di addestramento, anche a rischio di overfitting. Questo significa che il modello potrebbe essere troppo sensibile al rumore o alle peculiarità del dataset di addestramento e potrebbe non generalizzare bene su nuovi dati.

D'altra parte, se il valore di C è basso, la penalizzazione per gli errori sarà debole e il modello sarà più flessibile. Ciò può portare a un modello più generale e meno sensibile ai dati di addestramento, ma potrebbe anche risultare in un'accuratezza inferiore sui dati di addestramento.

4.4.2 Support Vector Machine

Support Vector Machine(SVM) è il secondo algoritmo di classificazione che sarà usato per classificare i testi [3].

Le SVM svolgono la classificazione trovando un iperpiano ottimale che separa i dati di addestramento in due classi (vedi Fig. 4.1). L'obiettivo delle SVM è quello di massimizzare la distanza tra l'iperpiano e i punti di addestramento più vicini di entrambe le classi, noti come Support Vector, al fine di ottenere un confine di decisione ottimale.

Per classificare nuovi punti, le SVM utilizzano la posizione di questi punti rispetto all'iperpiano di separazione. Se un punto si trova su un lato dell'iperpiano associato ad una classe, viene classificato come appartenente a quella classe, mentre se si trova sull'altro lato, viene classificato come appartenente all'altra classe.

Inoltre, le SVM possono essere estese per gestire problemi di classificazione non lineari attraverso l'uso di funzioni di kernel, che mappano i dati in uno spazio di dimensioni superiori dove la separazione lineare potrebbe essere possibile.

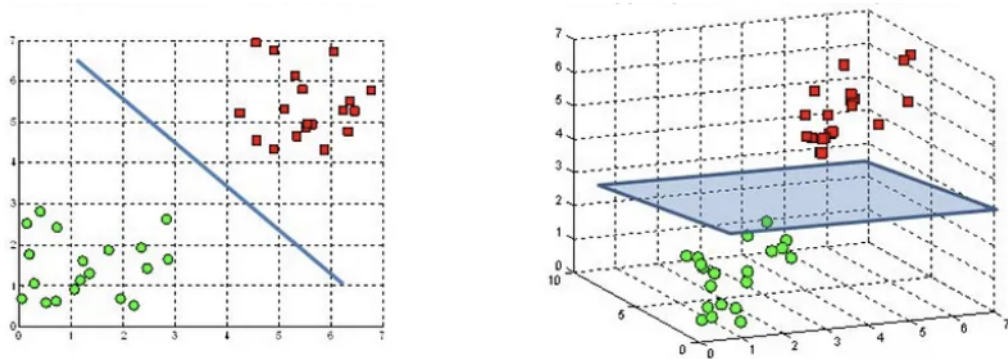


Figura 4.1: *Esempi di iperpiano in SVM [46]*

Gli iperparametri che saranno validati in fase di addestramento sono i seguenti:

- C: controlla il trade-off tra l'accuratezza del modello e la sua capacità di generalizzazione. Un valore più alto di C implica una penalizzazione più severa per gli errori di classificazione nel dataset

di Training, il che significa che il modello cercherà di classificare correttamente la maggior parte dei punti di addestramento. Un valore più basso di C permette al modello di avere un margine di separazione più ampio, anche se a costo di alcune violazioni degli esempi di addestramento.

- Gamma: regola la sensibilità del modello ai punti di addestramento.

Un valore più elevato di Gamma implica che il modello darà più importanza ai punti di addestramento più vicini all'iperpiano e considererà solo quelli nella definizione del confine di decisione, e questo può portare a un iperpiano più complesso che aderirà più strettamente ai dati.

Un valore più basso di Gamma implica che il modello darà meno importanza ai punti di addestramento individuali e considererà anche quelli più lontani dall'iperpiano nella definizione del confine di decisione, e questo può portare a un iperpiano più lineare e meno sensibile ai dati.

4.4.3 Random Forest

Random Forest è un insieme di Decision Trees (per approfondire gli alberi decisionali, vedi [16]) in cui ogni albero viene addestrato su un sottoinsieme casuale dei dati di training [4]. Inoltre, durante la costruzione di ogni albero, viene utilizzato solo un sottoinsieme casuale delle caratteristiche disponibili.

Una volta addestrati i Decision Trees, Random Forest aggrega le previsioni di ciascun albero per ottenere una previsione finale. In un problema di classificazione, ciò viene solitamente fatto attraverso un processo di voto a maggioranza, in cui la classe predetta dal maggior numero di alberi viene selezionata come classe predetta (vedi Fig. 4.2).

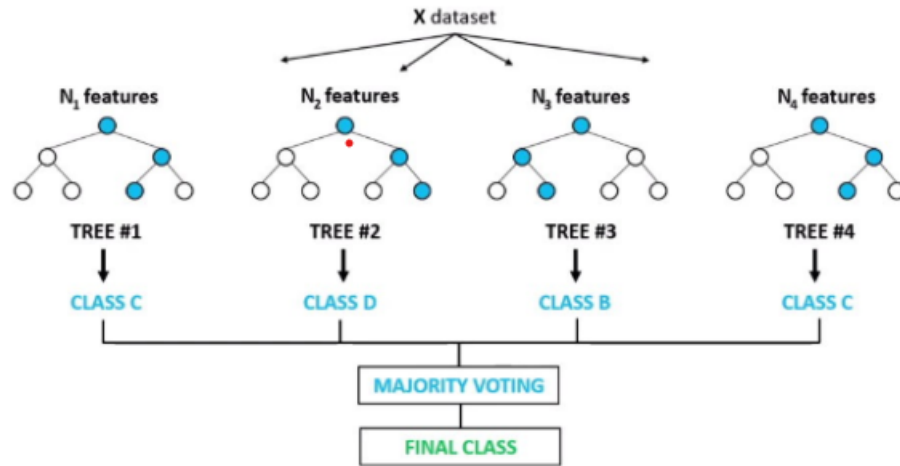


Figura 4.2: *Esempio di Random Forest [47]*

Gli iperparametri che saranno validati in fase di addestramento sono i seguenti:

- `min_samples_split`: specifica il numero minimo di osservazioni richieste per suddividere un nodo durante la costruzione dei Decision Trees. Aumentare il valore dell'iperparametro può portare a alberi più semplici con meno divisioni e ridurre il rischio di overfitting, ma potrebbe anche portare a un modello meno flessibile che potrebbe non essere in grado di catturare la complessità nei dati.
- `max_features`: specifica il numero massimo di caratteristiche (features) da considerare per la divisione di un nodo durante la costruzione dei Decision Trees. Riducendo il numero di caratteristiche considerate, si riduce la complessità del modello e si riduce il rischio di overfitting.

4.5 TPOT

TPOT, acronimo di Tree-based Pipeline Optimization Tool, è un algoritmo automatizzato di Machine Learning che esegue la selezione del modello e l'ottimizzazione degli iperparametri in modo completamente automatico [6] a partire da un insieme elevato di algoritmi e relativi in-

siemi di iperparametri. Utilizza la programmazione genetica per creare e ottimizzare Pipelines di Machine Learning. TPOT rappresenta pipeline di processing attraverso "alberi" dove un nodo dell'albero rappresenta un operatore applicato e un arco rappresenta un percorso di elaborazione del dato che coinvolge due operatori. Un esempio di albero è rappresentato in Fig. 4.3.

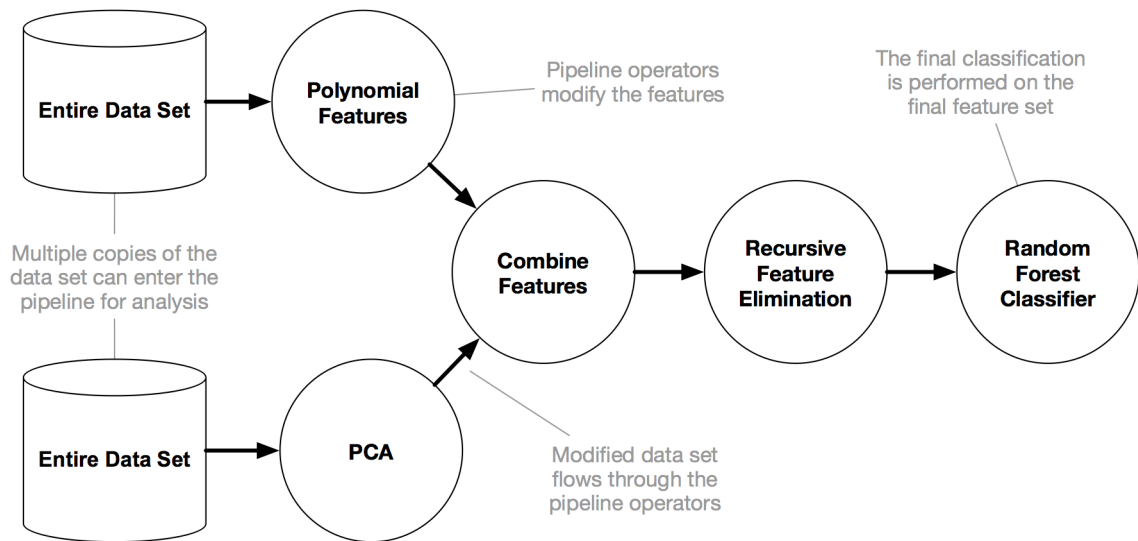


Figura 4.3: *Esempio di albero in TPOT [6]*

Le fasi di TPOT sono le seguenti:

1. Creazione della prima popolazione generando un numero fisso di alberi (ogni albero rappresenta una Pipeline di Machine Learning).
2. Valutazione delle pipeline: ogni pipeline viene valutata utilizzando una "Fitness function" come per esempio l'accuracy e la complessità della pipeline (a parità di prestazione una pipeline più complessa viene penalizzata).
3. Evoluzione delle pipeline: si applicano principi di programmazione genetica per evolvere le pipeline. Vengono selezionate le pipeline con risultati migliori e si applicano operazioni genetiche come mutazione, ricombinazione e selezione per generare nuove pipeline. I passi 2 e 3 si ripetono per un numero fisso di generazioni.

4. Selezione del modello finale: durante i processi di valutazione, la singola pipeline con le performance migliori viene selezionata e conservata separatamente, e viene poi utilizzata come output del processo di ricerca della pipeline migliore.

Senza conoscenza a priori del problema, è stato dimostrato che TPOT risulta avere spesso performance migliori rispetto a un'analisi di Machine Learning standard.

È possibile applicare TPOT su Python installando il pacchetto `tpot` [48], come in List. 11.

```
1  pip install tpot
2  from tpot import TPOTClassifier
3
4  tpot = TPOTClassifier(
5      generations=2, #Number of iterations to the run pipeline optimization process
6                  #2 is to keep the computation time limited.
7                  # Better 100 to get optimal (and stable) values.
8      population_size=50, # default=100 # Number of individuals to retain in the
9                      # genetic programming population every generation.
10                     # Generally, TPOT will work better when you give it
11                     # more individuals with which to optimize the pipeline.
12      verbosity=2, # How many explanation output will be printed during simulation
13      scoring='accuracy', # Metric score to optimize
14      cv=5, # cross-validation, default=5
15      n_jobs=-1, # Number of processes to use in parallel for evaluating pipelines
16              # during the TPOT optimization process. Default = 1 (1 processor/core)
17              # -1 is all available processors
18              # -2 is all available processors minus one (useful if you are running tpot
19              # on your pc and you want to work on something else during the computation)
20      config_dict='TPOT sparse', # The set of initial classifiers
21                      # and preprocessing component evaluated. In this set,
22                      # every preprocessing element can handle sparse matrices.
23      random_state=42 # The seed to make the process repeatable
24  )
25
```

Listing 11: Codice per applicare TPOT

Si riporta in Fig. 4.4 anche un esempio di Output di TPOT dopo il training.

```
tpot.fit(xTrainDtm, yTrain)
```

Generation 1 - Current best internal CV score: 0.9156010230179028

Generation 2 - Current best internal CV score: 0.9184995737425405

Best pipeline: MultinomialNB(input_matrix, alpha=1.0, fit_prior=False)

```
TPOTClassifier
TPOTClassifier(config_dict='TPOT sparse', generations=2, n_jobs=-1,
                population_size=50, random_state=42, scoring='accuracy',
                verbosity=2)
```

Figura 4.4: *Esempio do Output di TPOT*

4.6 BioBert

BioBERT è un modello di linguaggio preaddestrato basato su BERT (Bidirectional Encoder Representations from Transformers) [19], che è stato specificamente addestrato per l'elaborazione del linguaggio naturale (NLP) nel campo delle scienze biomediche e della biologia [7]. Il fine-tuning è stato effettuato da Alex Alborghetti, studente magistrale di Biostatistica. Lo scopo è quello di confrontare i risultati dei classificatori introdotti nel Capitolo 4.4 con il modello BERT, che rappresenta lo stato dell'arte in molti compiti di NLP.

4.7 Metriche

Le metriche che saranno utilizzate per valutare le performance dei classificatori sono i seguenti: Accuracy, Precision, Recall [49], F1 score [50]. Per dare una definizione delle metriche è necessario introdurre le seguenti definizioni:

- True Positives (TP): rappresenta il numero di campioni positivi correttamente classificati come positivi dal modello

- False Positives (FP): rappresenta il numero di campioni negativi erroneamente classificati come positivi dal modello
- True Negatives (TN): rappresenta il numero di campioni negativi correttamente classificati come negativi dal modello
- False Negatives (FN): rappresenta il numero di campioni positivi erroneamente classificati come negativi dal modello

In Fig. 4.5 è possibile avere una visione più chiara delle definizioni appena riportate sopra, mentre in Fig. 4.6 è possibile visualizzare la relazione tra Precision, Recall e Accuracy.

| | | Predicted Class | |
|--------------|---|----------------------|----------------------|
| | | P | N |
| Actual Class | P | True Positives (TP) | False Negatives (FN) |
| | N | False Positives (FP) | True Negatives (TN) |

Figura 4.5: *Confusion Matrix* [51]

Accuracy

L'Accuracy è una misura che indica la percentuale di predizioni corrette rispetto al numero totale di predizioni effettuate dal modello. Si ottiene dividendo il numero di predizioni corrette per il numero di predizioni fatte. Dato che il numero di predizioni corrette è dato da $TP + TN$, l'accuracy si può scrivere nel seguente modo:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (4.1)$$

Precision

Precision è una metrica che misura la proporzione di predizioni positive corrette rispetto al totale delle predizioni positive fatte da un modello. La

Precision è calcolata dividendo il numero di predizioni positive corrette per il numero totale di predizioni positive fatte, sia corrette che errate.

La formula per calcolare la Precision è:

$$Precision = \frac{TP}{TP + FP} \quad (4.2)$$

Recall

Recall è una metrica che misura la proporzione di esempi positivi correttamente identificati dal modello rispetto al totale dei veri esempi positivi presenti nel dataset. Il Recall è calcolato dividendo il numero di predizioni positive corrette per il numero totale di osservazioni positive. La formula per calcolare il Recall è:

$$Recall = \frac{TP}{TP + FN} \quad (4.3)$$

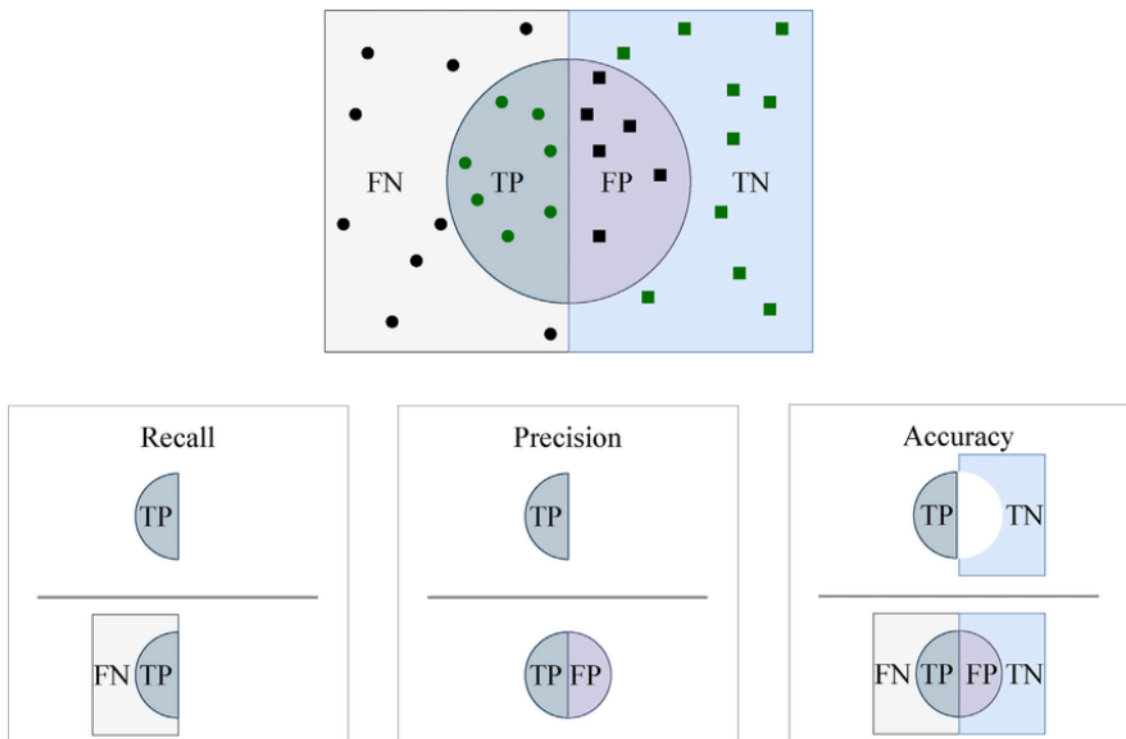


Figura 4.6: Rappresentazione di Recall, Precision e Accuracy [52]

F1 Score

L’F1-score è una metrica che combina Precision e Recall per fornire una misura complessiva delle prestazioni di un modello di classificazione nel Machine Learning. Esiste un trade-off tra Precision e Recall, tali per cui in alcuni classificatori è possibile migliorare la precision a scapito della recall e viceversa.

È particolarmente utile quando si hanno classi sbilanciate o quando si desidera trovare un equilibrio tra la Precision e il Recall. È calcolato come la media armonica tra la Precision e il Recall ed è espressa come un valore compreso tra 0 e 1, dove un valore più alto indica prestazioni migliori.

La formula per calcolare l’F1-score è:

$$F1Score = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{TP}{TP + \frac{1}{2}(FP + FN)} \quad (4.4)$$

Capitolo 5

Dataset e Risultati

5.1 Descrizione del Dataset

In questo capitolo, verrà descritto il processo di creazione del dataset utilizzato per il lavoro di tesi, che ha come obiettivo l’addestramento di classificatori in grado di categorizzare frasi collegate a studi clinici randomizzati. Sarà fornita una dettagliata descrizione del dataset, comprese le sue colonne e il numero di righe. Inoltre, sarà illustrato come il dataset è stato utilizzato per addestrare i modelli di machine learning.

Per iniziare, sono stati raccolti 200 articoli scientifici relativi a studi clinici randomizzati. Per fare ciò, è stata utilizzata una query PubMed specifica all’interno della Web App (per approfondire, vedi Cap. 3.3), che ha consentito di scaricare un foglio Excel contenente le informazioni relative ai 200 articoli. Tra queste informazioni erano incluse il PMID, l’abstract e il testo completo (se disponibile).

La query utilizzata, fornita da Alex Alborghetti e validata da alcuni esperti in meta-analisi, era la seguente:

“(randomized controlled trial[Publication Type] OR randomized clinical trial[Publication Type] OR randomized controlled trials[MeSH Terms] OR randomized clinical trials[MeSH Terms])”

Successivamente, per ogni articolo è stato effettuato il processo di suddivisione del testo (composto da abstract + fulltext) in frasi utilizzando il tokenizer fornito dalla libreria nltk [39], come illustrato in List. 12.

```

1  sentLi = [] #Lista per contenere le frasi
2  for a in fulltext: #fulltext è la lista con i testi completi
3      li = sent_tokenize(a) #funzione per dividere il testo in una lista di frasi
4      sentLi = sentLi + li #aggiungo le frasi nella lista
5

```

Listing 12: Codice per dividere il testo in una lista di frasi

In questo modo, è stato creato un dataset contenente gli indici e le frasi estratte.

Successivamente, è stato utilizzato uno strumento basato sull’approccio semi-supervised learning [53] fornito dal prof. Cesarini per l’operazione di labeling manuale. Il concetto di labeling manuale è approfondito nel Cap. 4.2, mentre l’uso dello strumento è spiegato Cap. 5.2.

Il dataset ricavato dopo l’operazione di labeling manuale è costituito da 3 colonne e 200 righe. Riporto in Fig. 5.1 l’head del dataset:

| | Index | text | y |
|---|-------|---|---|
| 0 | 6565 | Methods \n \n Trial design and s... | 1 |
| 1 | 28256 | Ethics and Study Design \n The lactofe... | 1 |
| 2 | 13877 | Study Design \n This study was a prosp... | 1 |
| 3 | 24481 | Study design and participants \n This ... | 1 |
| 4 | 15211 | Methods \n \n Trial design \n ... | 1 |

Figura 5.1: Head del dataset contenente le frasi etichettate

Di seguito si descrivono le colonne del dataset:

- index: rappresenta l’indice delle frasi nel dataset. L’indice deriva dalle frasi durante la Tokenization.
- text: contiene le frasi estratte, che saranno utilizzate per addestrare i modelli.

- label: contiene le etichette assegnate manualmente a ciascuna frase. Un'etichetta è considerata positiva se la frase riguarda il concetto di randomizzazione, mentre è negativa se la frase non è rilevante per l'argomento.

Si tratta di un dataset sbilanciato, con 99 frasi etichettate come positive e 168 frasi etichettate come negative. È importante notare che il dataset non contiene valori mancanti (missing values), garantendo così una completezza dei dati.

L'uso di dataset sbilanciati durante l'addestramento può portare alla creazione di un modello con una scarsa capacità di classificare correttamente la classe o le classi di minoranza.

Per affrontare il problema dello sbilanciamento, è stata adottata la strategia dell'undersampling [54].

L'undersampling consiste nel ridurre la quantità di osservazioni nella categoria maggioritaria in modo da renderla più bilanciata rispetto alla categoria minoritaria. Nel nostro caso, abbiamo ridotto la numerosità del campione delle frasi etichettate come negative per renderlo equivalente a quello delle frasi positive. Questo ci ha permesso di rappresentare entrambe le classi in modo equo. Come risultato, il dataset prodotto presenta 198 righe, di cui 99 positive e 99 negative.

Per la classificazione delle frasi, il dataset è stato diviso in due parti: un set di addestramento (training set) e un set di test (test set) [55]. Per lo split è stata utilizzata la funzione `train_test_split` di scikit-learn, come riportato in List. 13.

Il training set rappresenta il 70% del dataset originale (138 osservazioni, mentre il test set rappresenta il restante 30% (60 osservazioni). Questa suddivisione è stata effettuata in modo casuale, garantendo che entrambi i set contengano una distribuzione equa di esempi positivi e negativi.

```

1  # Suddivido xAll e yAll in due sottoinsiemi, rispettivamente di training e test
2  xTrainVec, xTestVec, yTrain, yTest = train_test_split(
3      xAll, yAll, # the x and y to be partitioned
4      test_size=0.30, # the test set size will be 30% of the original dataset,
5                      # i.e. training size will be 70%
6      random_state=0, # random_state is the seed to make random number
7                      # generation reproducible (and hence the split into train and test)
8      stratify=yAll # stratify: tries to ensure a proportional
9                      # distribution of labels among train and test set
10 )

```

Listing 13: Codice per applicare train_test_split

Il training set viene utilizzato per addestrare i modelli di machine learning, consentendo all'algoritmo di apprendere le relazioni nei dati etichettati. Durante il processo di addestramento, l'algoritmo cerca di sviluppare il modello migliore per fare previsioni accurate sul training set. Una volta terminato l'addestramento, il modello viene testato sul test set, che è costituito da dati “mai visti” dal modello durante l'addestramento. L'obiettivo è valutare la capacità di generalizzazione, cioè la capacità di fare previsioni accurate su nuovi dati non annotati.

La divisione del dataset in train e test è fondamentale per valutare in modo imparziale l'efficacia del modello di classificazione. Utilizzando il test set, possiamo calcolare metriche come l'accuratezza, la precisione, la recall e l'F1-score per valutare le prestazioni del modello e confrontarle con le etichette di test note.

5.2 Labeling dei dati

Per semplificare e accelerare la procedura di Labeling dei dati ho utilizzato un notebook fornito dal prof Cesarini. Di seguito riporto le fasi nel dettaglio per effettuare il labeling con l'approccio sem-supervised learning implementato nel notebook:

1. Insieme ad Alex, ho stabilito un insieme iniziale di parole chiave sulla base della conoscenza preliminare degli studi randomizzati.

Le parole scelte sono state randomized, controlled, trial, groups, arms, randomization, assigned, treatment, control, placebo, clinical e allocation.

2. Il professore ha fornito le funzioni per il preprocessing delle frasi e il classificatore Support Vector Machine, per supporto al labeling.
3. Abbiamo utilizzato le parole chiave per individuare dei testi con i quali effettuare un primo addestramento del classificatore basato sul conteggio delle parole. Così facendo, il classificatore ha restituito le prime 20 frasi ognuna con una proposta di etichettatura, di cui 10 positive e 10 negative.
4. Successivamente, ho verificato le etichette proposte e corretto eventuali errori di assegnazione. Inoltre, è stata introdotta una categoria "non classificabile", a cui ho assegnato le frasi che non contenevano informazioni sufficienti per essere correttamente classificate. Ho quindi nuovamente addestrato il classificatore utilizzando sia le frasi precedenti, sia le frasi con le etichette validate e corrette manualmente.
5. Il classificatore ha restituito altre 20 frasi con una proposta di etichetta per ognuna, di cui 10 positive e 10 negative.
6. Ho ripetuto i passaggi 4 e 5 per un totale di 19-20 iterazioni, fino a raggiungere un totale di 98 frasi positive e 168 frasi negative nel dataset etichettato.

Attraverso questo processo iterativo di labeling manuale, abbiamo creato un dataset etichettato di alta qualità.

Grazie al notebook fornito, il processo di labeling manuale è stato facilitato molto, tuttavia ha richiesto comunque un notevole impegno di tempo. Questo è stato principalmente dovuto alla necessità di consentire al classificatore di etichettare le nuove frasi e alla mia parte di revisione e correzione manuale delle etichette generate dal classificatore.

5.3 Hyperparameter Tuning

Per migliorare le performance dei modelli, è stato effettuato il tuning dei loro parametri, noto anche come Hyperparameter Tuning [56].

Gli iperparametri sono i parametri del modello che non vengono appresi durante il processo di addestramento, ma devono essere impostati manualmente dall'utente prima di avviare l'addestramento del modello. Questi parametri influenzano il comportamento del modello e possono includere, ad esempio, la regolarizzazione, il numero di k nel modello KNN, il numero di nodi in un albero, e così via

La fase di Hyperparameter Tuning rappresenta una parte essenziale del processo di sviluppo di modelli di machine learning per la classificazione. Durante questa fase vengono ricercati i valori ottimali o migliori per gli iperparametri del modello al fine di massimizzare le prestazioni del modello stesso.

La ricerca dei parametri ottimali può essere eseguito in diversi modi, tra cui Grid Search e Randomized Search, che saranno approfonditi rispettivamente nei Par. 5.3.1 e Par. 5.3.2

5.3.1 Grid Search

Il processo di Grid Search consiste nell'esplorare sistematicamente un insieme di combinazioni di possibili valori degli iperparametri per determinare quelle che massimizzano le prestazioni del modello [57].

Di seguito sono riportati i passaggi principali per eseguire una Grid Search:

1. Creazione del modello e della metrica di valutazione: bisogna definire il modello di machine learning che si intende utilizzare e la metrica di valutazione sulla base della quale si desidera valutare le prestazioni del modello

2. Definizione della griglia degli iperparametri: consiste nell'identificare gli iperparametri del modello che si desidera ottimizzare e definire un insieme di valori da testare per ciascun iperparametro.
3. Addestramento e valutazione del modello: per ogni combinazione di valori degli iperparametri nella griglia, si addestra il modello utilizzando una specifica combinazione di valori e si valutano le prestazioni in base alla metrica scelta. Per fare ciò, si utilizza la cross-validation (per approfondire vedi [58]) per ottenere una stima più affidabile delle prestazioni del modello.
4. Selezione dei migliori iperparametri: alla fine del processo di Grid Search, si sceglie la combinazione di valori degli iperparametri che ha prodotto le migliori prestazioni del modello in base alla metrica di valutazione.

Lo svolgimento di un Grid Search può richiedere una elaborazione prolungata ed intensiva poiché esegue l'addestramento e la valutazione del modello per ogni combinazione di valori degli iperparametri. Tuttavia, è un metodo efficace per trovare la migliore combinazione di iperparametri per massimizzare le prestazioni del modello e migliorare la sua capacità predittiva.

Per applicare la Grid Search in Python si può usare la classe `GridSearchCV` di `scikit-learn`. Si riporta un esempio in List. 14 che è stato preso direttamente dal sito della libreria `scikit-learn` [59].

5.3.2 Randomized Search

A differenza della Grid Search, che esamina tutte le possibili combinazioni di valori degli iperparametri, la Randomized Search seleziona casualmente un sottoinsieme di combinazioni di valori da testare [60]. I passaggi principali sono gli stessi della Grid Search (vedi Cap. 5.3.1) con delle leggere variazioni. Nel primo passaggio, solitamente al posto di specificare i

```

1  from sklearn import svm, datasets
2  from sklearn.model_selection import GridSearchCV
3  iris = datasets.load_iris()
4  parameters = {'kernel':('linear', 'rbf'), 'C':[1, 10]}
5  svc = svm.SVC()
6  clf = GridSearchCV(svc, parameters)
7  clf.fit(iris.data, iris.target)
8
9
10 sorted(clf.cv_results_.keys())

```

Listing 14: Codice per applicare Grid Search

valori da testare, si definiscono degli intervalli o distribuzioni di probabilità su cui campionare i valori degli iperparametri, come ad esempio una distribuzione loguniforme.

Nel terzo passaggio si definisce il numero totale di combinazioni da testare, e per ogni iperparametro si estrae un valore casuale dalle distribuzioni definite nel primo passaggio in modo da formare una combinazione di iperparametri.

La Randomized Search è utile quando si dispone di un numero elevato di iperparametri o quando non si ha una conoscenza approfondita dell'importanza di ciascun iperparametro. Inoltre, rispetto alla Grid Search, la Randomized Search può essere computazionalmente più veloce, poiché esamina solo un sottoinsieme casuale delle combinazioni di iperparametri.

La Randomized Search potrebbe non esplorare completamente lo spazio degli iperparametri come la Grid Search. È importante notare, però, che la Grid Search, nonostante la sua capacità di esplorare tutto lo spazio dei parametri specificato, richiede un considerevole tempo di esecuzione. Di conseguenza, spesso le persone sono indotte a ridurre lo spazio dei parametri da cercare al fine di risparmiare tempo. In contrasto, la Randomized Search consente di allargare notevolmente lo spazio dei parametri da esplorare, concentrandosi su un sottoinsieme di valori, sebbene in modo non esaustivo. Questo approccio più veloce può portare, in alcune occa-

sioni, a ottenere risultati migliori rispetto alla Grid Search. È pertanto importante considerare entrambe le strategie. Tuttavia, è comunque vero che una ricerca esaustiva su uno spazio di parametri sufficientemente ampio offre maggiori possibilità di individuare la soluzione migliore. Per applicare la Randomized Search in Python, si può utilizzare la classe `RandomizedSearchCV` di `scikit-learn`. Si riporta un esempio di applicazione in List. 15, preso direttamente dal sito di `scikit-learn` [61].

```
1  from sklearn.datasets import load_iris
2  from sklearn.linear_model import LogisticRegression
3  from sklearn.model_selection import RandomizedSearchCV
4  from scipy.stats import uniform
5  iris = load_iris()
6  logistic = LogisticRegression(solver='saga', tol=1e-2, max_iter=200,
7                               random_state=0)
8  distributions = dict(C=uniform(loc=0, scale=4),
9                       penalty=['l2', 'l1'])
10 clf = RandomizedSearchCV(logistic, distributions, random_state=0)
11 search = clf.fit(iris.data, iris.target)
12 search.best_params_
```

Listing 15: Codice per applicare Randomized Search

5.4 Training e scelta del modello

Per quanto riguarda la classificazione delle frasi, ho addestrato tre classificatori: la regressione logistica [5], SVM[3] e Random Forest[4]. Al fine di implementare con successo il processo in Python, ho utilizzato una pipeline che ha consentito di integrare le funzioni di preprocessing direttamente nel flusso di lavoro. Questo approccio ha semplificato l'esecuzione dei modelli di classificazione.

Una pipeline in `scikit-learn` è uno strumento conveniente che permette di concatenare più passaggi di elaborazione di dati e algoritmi in un'unica unità [62]. Fornisce un modo sistematico per organizzare ed eseguire una

sequenza di trasformazioni sui dati di input, seguita dall'applicazione di un modello finale per la previsione o la classificazione

La pipeline offre diversi vantaggi. In primo luogo, semplifica il flusso di lavoro, in quanto è possibile avviare l'esecuzione delle trasformazioni e l'addestramento del modello in un unico passaggio. In secondo luogo, aiuta a evitare problemi di data leakage [63], assicurando che le trasformazioni vengano applicate con l'ordine corretto sia ai dati di addestramento che ai dati di test. Infine, consente di ottimizzare e valutare l'intero flusso di lavoro tramite Hyperparameter tuning. Si riporta la pipeline utilizzata per la classificazione delle frasi in Fig. 5.2.



Figura 5.2: *Pipeline utilizzata per la classificazione delle frasi*

Di seguito si descrive l'obiettivo di ogni classe utilizzata nella pipeline:

- **HTMLAccentsReplacer**: sostituisce le rappresentazioni html delle lettere speciali con il corrispondente carattere unicode.
- **Tokenizer**: converte un singolo documento in una lista contenente le parole all'interno del documento.

- `LowerCaseReducer`: prende in input una lista di stringhe, e le converte tutte in minuscolo.
- `EnglishStopWordsRemover`: prende in input una lista di stringhe, e restituisce una lista di stringhe con le stopwords inglesi rimosse.
- `EnglishStemmer`: prende in input una lista di parole, e restituisce una lista delle parole con solo la radice.
- `RemoveNumbers`: prende in input una lista di parole. Restituisce la lista di parole con i numeri rimossi.
- `RemoveEmptyWords`: prende in input una lista di parole. Restituisce la lista di parole dove gli elementi vuoti sono rimossi.
- `TfidfVectorizer`: converte una collezione di documenti di testo in una rappresentazione numerica, utilizzando la tecnica TF-IDF (approfondita nel Cap. 4.3.1).

Durante il processo di ottimizzazione dei modelli, ho impiegato sia il metodo `Grid` che il metodo `Randomized`. È stata osservata una differenza significativa in termini di tempi di elaborazione tra i due approcci. Il tuning dei parametri ha permesso di migliorare notevolmente le prestazioni dei modelli rispetto ai valori predefiniti.

Inoltre, ho utilizzato sia il metodo `Grid` che il metodo `Randomized` per validare i parametri di `TfidfVectorizer` impiegato nella fase di preprocessing. Nel grid search sono stati esplorati i seguenti parametri:

- `ngram_range`: consente di specificare il range di n-grammi¹ da creare. Prende una tupla di 2 valori interi (`min_n` e `max_n`). Il valore `min_n` rappresenta la lunghezza minima degli n-grammi da considerare, mentre `max_n` rappresenta la lunghezza massima.

¹sequenze contigue di n elementi, come parole o lettere, utilizzati per analizzare pattern e modelli di linguaggio all'interno di un testo.

- `max_df`: rappresenta la massima document frequency consentita per un termine. Se la frequenza di un termine supera questa soglia, il termine viene considerato comune e viene ignorato durante l'operazione di Tokenization.
- `min_df`: rappresenta la minima document frequency consentita per un termine. Se non supera la soglia, allora il termine viene considerato "raro" e non si considera durante Tokenization
- `norm`: controlla la normalizzazione dei vettori TF-IDF durante la trasformazione dei testi in rappresentazioni vettoriali. Il parametro accetta una stringa come valore, che può essere "l1", "l2" o "None".
- `use_idf`: controlla l'uso o meno della componente IDF (Inverse Document Frequency) durante la trasformazione dei testi in rappresentazioni vettoriali TF-IDF.

La fase di preprocessing è uguale per tutti e 3 i classificatori, tranne i parametri di `TfidfVectorizer`, che vengono validati insieme agli iperparametri dei classificatori.

Per valutare le prestazioni dei modelli, ho utilizzato la tecnica di Cross Validation [58]. La cross-validation viene utilizzata per stimare quanto bene un modello si generalizzerà su dati non osservato in precedenza. Invece di suddividere il set di dati in un unico set di addestramento e un unico set di test, la cross-validation suddivide il set di dati in diverse parti (chiamate "fold") e addestra e valuta il modello su più combinazioni di queste parti. Il numero di fold utilizzati è stato 5, uguale per tutti i modelli

Nei paragrafi successivi si riportano i risultati ottenuti nella fase di Training.

Regressione Logistica

Per la regressione logistica, ho allenato il modello inizialmente senza regolarizzazione. Successivamente, ho proceduto con l'ottimizzazione del

parametro C utilizzando la Grid Search. In Python, ho utilizzato la classe GridSearchCV che permette di individuare l'iperparametro C che ottiene le migliori prestazioni in termini di Cross Validation. In List. 16 è possibile vedere la griglia degli iperparametri testati.

```

1  # Setting for each parameter the value space (i.e., the set of values to evaluate)
2  paramSpace = {
3      'classifier__C': [0.001, 0.01, 0.1, 1, 10, 100], # Values for grid search should be enclosed by []
4      'classifier__solver': ['liblinear'],
5      'classifier__class_weight': [None, 'balanced'], # if the classes were imbalanced, we could try this approach
6      'classifier__random_state': [42],
7      'vectorizer__preprocessor': [unityFunction], # since we provided a customized preprocessing pipeline,
8                                                    # we turn off the usual preprocessing pipeline
9      'vectorizer__tokenizer': [unityFunction], # Same as above.
10     'vectorizer__ngram_range': [(1,1), (1,2), (1,3)], #
11     'vectorizer__max_df': [0.7], # If a term is in more of the 70% of documents, it is too frequent to be discriminative
12     'vectorizer__min_df': [2, 4, 7], # Minimum number of documents where the term should appear
13                                     # (otherwise it won't be considered in the Vocabulary)
14     'vectorizer__norm': ["l1", "l2", None],
15     'vectorizer__use_idf': [True, False]
16 }
17 } # a python list [] is mandatory even if only one element is in

```

Listing 16: Griglia degli iperparametri da validare con Grid Search per la Regressione logistica

Ho eseguito lo stesso processo di ottimizzazione utilizzando anche la Randomized Search (con la classe RandomizedSearchCV).

I risultati ottenuti sono riportati in Tab. 5.1.

| Modello | Cross Validation score |
|--|------------------------|
| Regressione logistica baseline | 0.707 |
| Regressione logistica con GridSearchCV | 0.825 |
| Regressione logistica con RandomizedSearchCV | 0.869 |

Tabella 5.1: Cross validation score di Regressione Logistica

Possiamo notare che l'approccio con Randomized Search ha permesso di ottenere un cross validation score più alto. Inoltre entrambe le tec-

niche di tuning dei parametri hanno permesso di migliorare almeno del 10% le performance del modello rispetto a quello baseline.

Support Vector Machine

Per Support Vector Machine, ho seguito un approccio simile. Ho allenato il modello baseline, e successivamente sono andato a validare i parametri C e gamma. In List. 17 è possibile vedere le distribuzioni utilizzate per ogni iperparametro da passare a RandomizedSearchCV.

```

1  # Setting for each parameter the value space (i.e., the set of values to evaluate)
2  paramSpace1 = {
3      'classifier__C': loguniform(1e-4, 1e3),
4      'classifier__gamma': loguniform(1e-4, 1e3),
5      # 'classifier__solver': ['liblinear'],
6      # 'classifier__class_weight': [None, 'balanced'],
7      'vectorizer__preprocessor': [unityFunction],
8      'vectorizer__tokenizer': [unityFunction], # Same as above.
9      'vectorizer__ngram_range': [(1,1), (1,2), (1,3)], #
10     'vectorizer__max_df': uniform.rvs(0.5, 0.4, size = 1000, random_state = 42),
11     'vectorizer__min_df': uniform.rvs(0, 0.3, size = 1000, random_state = 42),
12     'vectorizer__norm': ["l1", "l2", None],
13     'vectorizer__use_idf': [True, False]
14 }
15 # a python list [] is mandatory even if only one element is in

```

Listing 17: Griglia degli iperparametri da validare con Randomized Search per SVM

Riporto in Tab. 5.2 i risultati ottenuti

| Modello | Cross Validation score |
|----------------------------|------------------------|
| SVM baseline | 0.811 |
| SVM con GridSearchCV | 0.862 |
| SVM con RandomizedSearchCV | 0.869 |

Tabella 5.2: Cross validation score di Support Vector Machine

Il modello SVM baseline, a differenza della regressione logistica, parte

con un cross validation score buono. Il tuning dei parametri ha permesso di migliorare leggermente le performance, ottenendo risultati migliori con l'approccio Randomized Search.

Random Forest

Per quanto riguarda il modello Random Forest, ho inizialmente addestrato un modello baseline senza eseguire alcuna ottimizzazione dei parametri. Successivamente, a causa di limiti computazionali, ho usato direttamente la tecnica di Randomized Search per la ricerca dei parametri ottimali. In List. 18 è possibile vedere le distribuzioni utilizzate per ogni iperparametro da passare a RandomizedSearchCV.

```
1 paramSpace1 = {
2     'classifier__n_estimators': [100],
3     'classifier__min_samples_split': uniform.rvs(0, 0.8, size = 1000, random_state = 42),
4     'classifier__random_state': [42],
5     'classifier__max_features': uniform.rvs(0, 1, size = 1000, random_state = 42),
6     '#classifier__solver': ['liblinear'],
7     '#classifier__class_weight': [None, 'balanced'],
8     'vectorizer__preprocessor': [unityFunction],
9     'vectorizer__tokenizer': [unityFunction],
10    'vectorizer__ngram_range': [(1,1), (1,2), (1,3)],
11    'vectorizer__max_df': uniform.rvs(0.5, 0.4, size = 1000, random_state = 42),
12    'vectorizer__min_df': uniform.rvs(0, 0.3, size = 1000, random_state = 42),
13    'vectorizer__norm': ["l1", "l2", None],
14    'vectorizer__use_idf': [True, False]
15 } # a python list [] is mandatory even if only one element is in
16
```

Listing 18: Griglia degli iperparametri da validare con Randomized Search per Random Forest

Ho applicato RandomizedSearchCV per validare i parametri `n_estimators`, `max_features` e `min_samples_split`. Riporto i risultati ottenuti in Tab. 5.3.

| Modello | Cross Validation score |
|--------------------------------------|------------------------|
| Random Forest baseline | 0.847 |
| Random Forest con RandomizedSearchCV | 0.884 |

Tabella 5.3: Cross validation score di Random Forest

Riporto ora le performance migliori ottenute dai 3 modelli in Tab. 5.4. Il confronto tra i 3 modelli sarà sempre sulla base del Cross Validation.

| Modello | Cross Validation score |
|------------------------|------------------------|
| Regressione logistica | 0.869 |
| Support Vector Machine | 0.869 |
| Random Forest | 0.884 |

Tabella 5.4: Confronto performance dei classificatori

Dalla Tab. 5.4 possiamo vedere che Random Forest ha ottenuto risultati migliori rispetto agli altri 2 modelli. Quindi useremo Random Forest come modello "rappresentante" dell'analisi di machine learning standard.

TPOT

Ho applicato TPOT per scoprire la pipeline ottimale che massimizzasse le prestazioni del modello. TPOT ha esplorato un'ampia gamma di possibili combinazioni di trasformazioni dei dati insieme a diversi modelli di machine learning (per approfondire vedi Cap. 4.5). In Python, ho fissato il numero di generazioni a 100 e la dimensione della popolazione a 50. Il numero di generazioni definisce la quantità di iterazioni che TPOT esegue per generare, valutare e migliorare la pipeline di machine learning. Ogni generazione comprende la selezione delle pipeline migliori, l'incrocio delle pipeline selezionate e la mutazione.

La dimensione della popolazione, invece, indica il numero di pipeline generate e valutate in ogni generazione. Ovviamente una popolazione più ampia offre una maggiore diversità di pipeline, ma richiede risorse com-

putazionali aggiuntive per l'esecuzione del processo di ottimizzazione.

Per poter usare TPOT, ho dovuto prima applicare la tokenization ai dati di input utilizzando TfidfVectorizer e usando i parametri ottimali trovati per il modello Random Forest.

Nel processo di valutazione delle prestazioni della pipeline ottenuta da TPOT, ho adottato un approccio coerente con i modelli precedenti. Ho impostato lo scoring a "accuracy", ossia l'accuratezza, come metrica di valutazione principale. Ho utilizzato la tecnica di Cross Validation con lo stesso numero di fold utilizzato per i 3 modelli.

Finito il training, il cross validation score della pipeline ottimale è risultato essere pari a 0.891. In List. 19 si riporta la pipeline ottimale trovata da TPOT.

```
1  # Average CV score on the training set was: 0.8907407407407408
2  exported_pipeline = make_pipeline(
3      RFE(estimator=ExtraTreesClassifier(criterion="entropy",
4                                          max_features=1.0,
5                                          n_estimators=100), step=0.4),
6      BernoulliNB(alpha=1.0, fit_prior=True)
7  )
8
9
```

Listing 19: Pipeline ottimale di TPOT

5.5 Analisi dei risultati ottenuti

In questo capitolo, valutiamo le performance dei modelli previamente validati utilizzando i dati del Test set.

I modelli includono il modello migliore ottenuto tramite l'analisi di Machine Learning standard, quindi Random Forest, e la pipeline ottimale ottenuta grazie a TPOT. Successivamente andremo a testare le performance del modello BioBert e confronteremo i risultati con i modelli precedenti.

Per rappresentare i risultati, utilizzeremo la funzione “classification_report” di scikit learn [64]. Questo report fornisce una panoramica dettagliata delle metriche di valutazione, come accuracy, precision, recall e F1-score, per ciascuna classe del problema di classificazione.

Random Forest

Il modello Random Forest, dopo la validazione dei parametri effettuata nel capitolo, ha ottenuto un cross validation score pari a 0.884, il migliore di tutti nell’analisi di Machine Learning standard. Analizziamo i risultati ottenuti in Test set in Fig. 5.3

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 0.83 | 0.83 | 0.83 | 30 |
| 1 | 0.83 | 0.83 | 0.83 | 30 |
| accuracy | | | 0.83 | 60 |
| macro avg | 0.83 | 0.83 | 0.83 | 60 |
| weighted avg | 0.83 | 0.83 | 0.83 | 60 |

Figura 5.3: *Classification Report di Random Forest*

In generale, il modello ha ottenuto buone performance nel classificare le istanze del Test set, con un calo dell’accuracy non troppo drastico rispetto al Cross validation score. Tuttavia, bisogna confrontare questi risultati con gli altri modelli.

TPOT

Riporto in Fig. 5.4 i risultati ottenuti in test set utilizzando la pipeline ottimale trovata da TPOT.

Notiamo che la pipeline ottimale di TPOT ha ottenuto prestazioni migliori del Random Forest in tutte le metriche prese in considerazione, quindi accuracy, recall, precision e F1-score. Oltre a ciò è importante notare che TPOT aveva ottenuto risultati migliori anche durante la validazione. Ciò suggerisce che TPOT è stato in grado di individuare

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 0.87 | 0.90 | 0.89 | 30 |
| 1 | 0.90 | 0.87 | 0.88 | 30 |
| accuracy | | | 0.88 | 60 |
| macro avg | 0.88 | 0.88 | 0.88 | 60 |
| weighted avg | 0.88 | 0.88 | 0.88 | 60 |

Figura 5.4: *Classification Report di TPOT*

una combinazione di modelli e iperparametri che offrono una maggiore capacità di discriminazione.

BioBert

Per la valutazione del modello BioBert, ho utilizzato la configurazione preaddestrata di BioBert fornitami da Alex Alborghetti con il fine-tuning effettuato utilizzando i miei dati di Training. Riporto in Fig. 5.5 i risultati ottenuti:

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 0.97 | 0.97 | 0.97 | 30 |
| 1 | 0.97 | 0.97 | 0.97 | 30 |
| accuracy | | | 0.97 | 60 |
| macro avg | 0.97 | 0.97 | 0.97 | 60 |
| weighted avg | 0.97 | 0.97 | 0.97 | 60 |

Figura 5.5: *Classification Report di BioBert*

Un punteggio del 0.97 in tutte le metriche, inclusa l'accuracy, la precision, la recall e l'F1-score, suggerisce che il modello BioBert ha raggiunto un livello di precisione molto elevato nella classificazione di entrambe le classi, positive e negative.

BioBert ha superato Random Forest e TPOT grazie al suo addestramento preesistente sul dominio biomedico, e al successivo fine-tuning con i dati di Training effettuato da Alex.

L'approccio di transfer learning (per approfondire vedi[65]), supporta-

to dallo stato dell'arte nell'elaborazione del linguaggio naturale (NLP), è dimostrato essere particolarmente efficace nella classificazione di testo biomedico, e nel nostro caso si è dimostrato essere cruciale per ottenere prestazioni superiori per la classificazione delle frasi.

Capitolo 6

Conclusioni

Il presente lavoro si proponeva di affrontare le sfide legate all’analisi e alla sintesi di un vasto corpus di studi nella ricerca clinica. Uno degli obiettivi principali era lo sviluppo di una web application in grado di semplificare il processo di ricerca dei paper appropriati per la meta-analisi clinica.

Per raggiungere questo obiettivo, ho utilizzato il framework Dash, che mi ha consentito di creare un’interfaccia intuitiva e user-friendly per gli utenti. Ho sviluppato un modulo Python in collaborazione con Alex Alborghetti, che integra le API PubMed [66] e fornisce un motore di ricerca PubMed all’interno della web app. Questo modulo permette agli utenti di effettuare ricerche mirate e di ottenere facilmente le informazioni dei paper, come abstract e full text, da processare per la meta-analisi. Ho poi sviluppato un modulo per l’integrazione dei modelli di classificazione del framework PICO, sviluppati da Alex Alborghetti [31]. Questi modelli, basati sull’intelligenza artificiale, consentono di analizzare le frasi nei paper e di identificare le frasi rilevanti per le diverse componenti del framework PICO. Ciò facilita notevolmente la selezione dei documenti pertinenti e accelera il processo di meta-analisi.

Durante lo sviluppo della web app, il prof. Cesarini ha potuto consultare esperti di meta-analisi e ottenere feedback preziosi per migliorare l’esperienza utente.

Questi passi hanno permesso di sviluppare un’interfaccia intuitiva e fun-

zionalità avanzate che agevolano la ricerca e l'accesso ai paper pertinenti, consentendo ai ricercatori di risparmiare tempo prezioso nella selezione dei documenti da includere nella meta-analisi.

Il secondo obiettivo del mio studio era quello di esplorare l'efficacia dei modelli tradizionali di machine learning nella classificazione delle frasi relative agli studi clinici randomizzati e confrontarli con l'approccio rappresentato da BioBert, uno dei modelli di punta nel dominio medico. Per ottenere risultati ottimali, ho applicato tecniche di preprocessing e ho eseguito il tuning degli iperparametri dei modelli tradizionali al fine di migliorare le loro performance. Nonostante ciò, i risultati ottenuti hanno confermato la superiorità di BioBert, evidenziando un miglioramento delle prestazioni del 10% in tutte le metriche considerate rispetto ai classificatori tradizionali. Anche utilizzando un approccio come TPOT, un metodo automatizzato per l'ottimizzazione delle pipeline di machine learning, BioBert ha dimostrato una netta superiorità. In questo modo, ho raggiunto il mio obiettivo di valutare e confrontare le diverse metodologie di classificazione, offrendo un'importante prospettiva sullo stato dell'arte nella classificazione delle frasi nel contesto degli studi clinici randomizzati.

Un limite della web app sviluppata riguarda il tempo di elaborazione richiesto per processare i paper. Durante la fase di valutazione, si è osservato che la web app impiega un tempo significativo per elaborare un numero limitato di paper. Ad esempio, per analizzare due paper, la web app richiede approssimativamente 10 minuti. Questo può rappresentare una limitazione in termini di efficienza e tempi di lavoro per i ricercatori che utilizzano l'applicazione. Tuttavia, è importante sottolineare che il tempo di elaborazione può variare in base a diversi fattori, come la dimensione dei paper e la complessità delle frasi da analizzare. Pertanto, l'effettivo tempo di elaborazione potrebbe differire a seconda delle speci-

fiche del caso di studio.

Il lavoro sulla web app ha aperto interessanti spunti per future direzioni di sviluppo e miglioramento nell'ambito della ricerca clinica e dell'analisi dei paper. Una delle sfide emerse è stata la necessità di ottimizzare l'efficienza della web app. Nonostante sia stata sviluppata con successo, si potrebbero dedicare ulteriori sforzi per ridurre il tempo di elaborazione dei paper per esempio distribuendo i task di elaborazione sui nodi di un sistema distribuito. Sarebbe utile inoltre esplorare tecniche avanzate di elaborazione del linguaggio naturale e l'implementazione di algoritmi più efficienti, in modo da accelerare il processo complessivo. Un altro spunto interessante potrebbe essere l'introduzione di un'interfaccia utente interattiva per visualizzare l'output dei risultati ottenuti. Al posto di presentare i risultati in un foglio di calcolo o un file Excel, l'interfaccia potrebbe fornire un'esperienza più intuitiva e interattiva per gli utenti.

In definitiva, il presente elaborato ha contribuito a fornire un'importante soluzione per la ricerca clinica, semplificando il processo di ricerca e selezione dei paper pertinenti. Gli sviluppi futuri suggeriti offrono la possibilità di migliorare ulteriormente la web app e di arricchire il campo della ricerca clinica, consentendo ai ricercatori di ottenere una valutazione globale e oggettiva dell'efficacia dei trattamenti medici.

Elenco delle figure

| | | |
|-----|---|----|
| 3.1 | <i>Layout riferito a List. 1</i> | 14 |
| 3.2 | <i>Schermata iniziale con query</i> | 21 |
| 3.3 | <i>Schermata iniziale per upload</i> | 21 |
| 3.4 | <i>Schermata finale con Keywords da inserire</i> | 22 |
| 4.1 | <i>Esempi di iperpiano in SVM [46]</i> | 35 |
| 4.2 | <i>Esempio di Random Forest [47]</i> | 37 |
| 4.3 | <i>Esempio di albero in TPOT [6]</i> | 38 |
| 4.4 | <i>Esempio do Output di TPOT</i> | 40 |
| 4.5 | <i>Confusion Matrix [51]</i> | 41 |
| 4.6 | <i>Rappresentazione di Recall, Precision e Accuracy [52]</i> | 42 |
| 5.1 | <i>Head del dataset contenente le frasi etichettate</i> | 46 |
| 5.2 | <i>Pipeline utilizzata per la classificazione delle frasi</i> | 54 |
| 5.3 | <i>Classification Report di Random Forest</i> | 62 |
| 5.4 | <i>Classification Report di TPOT</i> | 63 |
| 5.5 | <i>Classification Report di BioBert</i> | 63 |

Elenco dei codici

| | | |
|----|---|----|
| 1 | Codice per strutturare Dash Layout [27] | 15 |
| 2 | Esempio Web App con Input | 17 |
| 3 | Esempio di Web App con Dash HTML components [29] . | 18 |
| 4 | Codice HTML riferito a List. 3 | 18 |
| 5 | Callback in Dash | 19 |
| 6 | Codice per effettuare Tokenizer | 28 |
| 7 | Codice per effettuare la rimozione delle Stopwords | 30 |
| 8 | Codice per effettuare lo Stemming | 31 |
| 9 | Codice per applicare Bag of Words | 32 |
| 10 | Codice per effettuare lo Stemming | 33 |
| 11 | Codice per applicare TPOT | 39 |
| 12 | Codice per dividere il testo in una lista di frasi | 46 |
| 13 | Codice per applicare train_test_split | 48 |
| 14 | Codice per applicare Grid Search | 52 |
| 15 | Codice per applicare Randomized Search | 53 |
| 16 | Griglia degli iperparametri da validare con Grid Search per la Regressione logistica | 57 |
| 17 | Griglia degli iperparametri da validare con Randomized Search per SVM | 58 |
| 18 | Griglia degli iperparametri da validare con Randomized Search per Random Forest | 59 |
| 19 | Pipeline ottimale di TPOT | 61 |

Elenco delle tabelle

| | | |
|-----|---|----|
| 5.1 | Cross validation score di Regressione Logistica | 57 |
| 5.2 | Cross validation score di Support Vector Machine | 58 |
| 5.3 | Cross validation score di Random Forest | 60 |
| 5.4 | Confronto performance dei classificatori | 60 |

Bibliografia

- [1] C. Schardt, M. B. Adams, T. Owens, S. Keitz, and P. Fontelo, “Utilization of the pico framework to improve searching pubmed for clinical questions,” BMC medical informatics and decision making, vol. 7, pp. 1–6, 2007.
- [2] M. Ikonomakis, S. Kotsiantis, and V. Tampakas, “Text classification using machine learning techniques,” WSEAS transactions on computers, vol. 4, no. 8, pp. 966–974, 2005.
- [3] W. S. Noble, “What is a support vector machine?,” Nature biotechnology, vol. 24, no. 12, pp. 1565–1567, 2006.
- [4] G. Biau and E. Scornet, “A random forest guided tour,” Test, vol. 25, pp. 197–227, 2016.
- [5] D. G. Kleinbaum, K. Dietz, M. Gail, M. Klein, and M. Klein, Logistic regression. Springer, 2002.
- [6] R. S. Olson and J. H. Moore, “Tpot: A tree-based pipeline optimization tool for automating machine learning,” in Workshop on automatic machine learning, pp. 66–74, PMLR, 2016.
- [7] J. Lee, W. Yoon, S. Kim, D. Kim, S. Kim, C. H. So, and J. Kang, “Biobert: a pre-trained biomedical language representation model for biomedical text mining,” Bioinformatics, vol. 36, no. 4, pp. 1234–1240, 2020.
- [8] M. Lutz, Programming python. ” O’Reilly Media, Inc.”, 2001.

- [9] E. Dabbas, Interactive Dashboards and Data Apps with Plotly and Dash: Harness the power of a fully fledged frontend web framework in Python—no JavaScript required. Packt Publishing Ltd, 2021.
- [10] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. E. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. B. Hamrick, J. Grout, S. Corlay, et al., Jupyter Notebooks—a publishing format for reproducible computational workflows., vol. 2016. 2016.
- [11] S. Bird, E. Klein, and E. Loper, Natural language processing with Python: analyzing text with the natural language toolkit. ” O’Reilly Media, Inc.”, 2009.
- [12] W. McKinney et al., “pandas: a foundational python library for data analysis and statistics,” Python for high performance and scientific computing, vol. 14, no. 9, pp. 1–9, 2011.
- [13] S. Van Der Walt, S. C. Colbert, and G. Varoquaux, “The numpy array: a structure for efficient numerical computation,” Computing in science & engineering, vol. 13, no. 2, pp. 22–30, 2011.
- [14] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” Journal of Machine Learning Research, vol. 12, pp. 2825–2830, 2011.
- [15] J. White, “Pubmed 2.0,” Medical reference services quarterly, vol. 39, no. 4, pp. 382–387, 2020.
- [16] P. H. Swain and H. Hauska, “The decision tree classifier: Design and potential,” IEEE Transactions on Geoscience Electronics, vol. 15, no. 3, pp. 142–147, 1977.
- [17] L. E. Peterson, “K-nearest neighbor,” Scholarpedia, vol. 4, no. 2, p. 1883, 2009.

- [18] W. Samek, G. Montavon, S. Lapuschkin, C. J. Anders, and K.-R. Müller, “Explaining deep neural networks and beyond: A review of methods and applications,” Proceedings of the IEEE, vol. 109, no. 3, pp. 247–278, 2021.
- [19] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” arXiv preprint arXiv:1810.04805, 2018.
- [20] G. Rossi, O. Pastor, D. Schwabe, and L. Olsina, Web engineering: modelling and implementing web applications. Springer Science & Business Media, 2007.
- [21] T. Berners-Lee and D. Connolly, “Hypertext markup language-2.0,” tech. rep., 1995.
- [22] H. W. Lie and B. Bos, Cascading style sheets: Designing for the web, Portable Documents. Addison-Wesley Professional, 2005.
- [23] A. Guha, C. Saftoiu, and S. Krishnamurthi, “The essence of javascript,” in ECOOP 2010–Object-Oriented Programming: 24th European Conference, Maribor, Slovenia, June 21-25, 2010. Proceedings 24, pp. 126–150, Springer, 2010.
- [24] T. Dresher, A. Zuker, and S. Friedman, Hands-On Full-Stack Web Development with ASP. NET Core: Learn end-to-end web development with leading frontend frameworks, such as Angular, React, and Vue. Packt Publishing Ltd, 2018.
- [25] “Dash python user guide.” <https://dash.plotly.com/>. Accessed: 2023-06-12.
- [26] J. Spurlock, Bootstrap: responsive web development. ” O’Reilly Media, Inc.”, 2013.

- [27] “Layout - dbc docs.” <https://dash-bootstrap-components.opensource.faculty.ai/docs/components/layout/>. Accessed: 2023-06-12.
- [28] “Dash core components.” <https://dash.plotly.com/dash-core-components>. Accessed: 2023-06-15.
- [29] “Dash html components.” <https://dash.plotly.com/dash-html-components>. Accessed: 2023-06-12.
- [30] O. Khattab and M. Zaharia, “Colbert: Efficient and effective passage search via contextualized late interaction over bert,” 2020.
- [31] A. Alborghetti, “e-pico: un’intelligenza artificiale a supporto delle meta-analisi,” 2023.
- [32] C. C. Shilakes and J. Tylman, “Enterprise information portals,” nov 1998.
- [33] A. Gandomi and M. Haider, “Beyond the hype: Big data concepts, methods, and analytics,” International journal of information management, vol. 35, no. 2, pp. 137–144, 2015.
- [34] “Structured vs unstructured data.” <https://www.datamation.com/big-data/structured-vs-unstructured-data/>. Retrieved: 2018-10-02.
- [35] T. Fredriksson, D. I. Mattos, J. Bosch, and H. H. Olsson, “Data labeling: An empirical investigation into industrial challenges and mitigation strategies,” in Product-Focused Software Process Improvement: 21st International Conference, PROFES 2020, Turin, Italy, November 25–27, 2020, Proceedings 21, pp. 202–216, Springer, 2020.
- [36] S. B. Kotsiantis, D. Kanellopoulos, and P. E. Pintelas, “Data pre-processing for supervised learning,” International journal of computer science, vol. 1, no. 2, pp. 111–117, 2006.

- [37] S. Vijayarani, M. J. Ilamathi, M. Nithya, et al., “Preprocessing techniques for text mining-an overview,” International Journal of Computer Science & Communication Networks, vol. 5, no. 1, pp. 7–16, 2015.
- [38] J. J. Webster and C. Kit, “Tokenization as the initial phase in nlp,” in COLING 1992 volume 4: The 14th international conference on computational linguistics, 1992.
- [39] “nltk.tokenize package.” <https://www.nltk.org/api/nltk.tokenize.html>. Accessed: 2023-06-04.
- [40] C. Silva and B. Ribeiro, “The importance of stop word removal on recall values in text categorization,” in Proceedings of the International Joint Conference on Neural Networks, 2003., vol. 3, pp. 1661–1666, IEEE, 2003.
- [41] J. B. Lovins, “Development of a stemming algorithm,” Mech. Transl. Comput. Linguistics, vol. 11, no. 1-2, pp. 22–31, 1968.
- [42] “Sample usage for stem.” <https://www.nltk.org/howto/stem.html#stemmers>. Accessed: 2023-06-03.
- [43] Y. Zhang, R. Jin, and Z.-H. Zhou, “Understanding bag-of-words model: a statistical framework,” International journal of machine learning and cybernetics, vol. 1, pp. 43–52, 2010.
- [44] X.-Y. Liu, J. Wu, and Z.-H. Zhou, “Exploratory undersampling for class-imbalance learning,” IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), vol. 39, no. 2, pp. 539–550, 2008.
- [45] S. B. Kotsiantis, I. Zaharakis, P. Pintelas, et al., “Supervised machine learning: A review of classification techniques,” Emerging artificial intelligence applications in computer engineering, vol. 160, no. 1, pp. 3–24, 2007.

- [46] “Support vector machine — introduction to machine learning algorithms.” <https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47>. Accessed: 2023-06-03.
- [47] “Applying random forest (classification) — machine learning algorithm from scratch with real datasets.” <https://medium.com/@ar.ingenious/applying-random-forest-classification-machine-learning-algorithm-from-scratch-with-real-24ff198a1c57>. Accessed: 2023-06-03.
- [48] W. Fu, R. Olson, Nathan, G. Jena, PGijsbers, T. Augspurger, J. Romano, P. Saha, S. Shah, S. Raschka, sohn, DanKoretzky, kdarakos, Jaimecclin, bartdp1, G. Bradway, J. Ortiz, J. J. Smit, J.-H. Menke, M. Ficek, A. Varik, A. Chaves, J. Myatt, Ted, A. G. Badaracco, C. Kastner, C. Jerônimo, Hristo, M. Rocklin, and R. Carnevale, “Epistasislab/tpot: v0.11.5,” June 2020.
- [49] M. Junker, R. Hoch, and A. Dengel, “On the evaluation of document analysis components by recall, precision, and accuracy,” in Proceedings of the Fifth International Conference on Document Analysis and Recognition. ICDAR’99 (Cat. No. PR00318), pp. 713–716, IEEE, 1999.
- [50] Z. C. Lipton, C. Elkan, and B. Narayanaswamy, “Thresholding classifiers to maximize f1 score,” arXiv preprint arXiv:1402.1892, 2014.
- [51] K. Horak, J. Klecka, O. Bostik, and D. Davidek, “Classification of surf image features by selected machine learning algorithms,” in 2017 40th International Conference on Telecommunications and Signal Processing (TSP), pp. 636–641, IEEE, 2017.
- [52] F. Maleki, K. Ovens, K. Najafian, B. Forghani, C. Reinhold, and R. Forghani, “Overview of machine learning part 1: fundamen-

- tals and classic approaches,” Neuroimaging Clinics, vol. 30, no. 4, pp. e17–e32, 2020.
- [53] X. J. Zhu, “Semi-supervised learning literature survey,” 2005.
- [54] H. He and E. A. Garcia, “Learning from imbalanced data,” IEEE Transactions on knowledge and data engineering, vol. 21, no. 9, pp. 1263–1284, 2009.
- [55] S. Raschka, “Model evaluation, model selection, and algorithm selection in machine learning,” arXiv preprint arXiv:1811.12808, 2018.
- [56] L. Yang and A. Shami, “On hyperparameter optimization of machine learning algorithms: Theory and practice,” Neurocomputing, vol. 415, pp. 295–316, 2020.
- [57] I. Syarif, A. Prugel-Bennett, and G. Wills, “Svm parameter optimization using grid search and genetic algorithm to improve classification performance,” TELKOMNIKA (Telecommunication Computing Electronics and Control), vol. 14, no. 4, pp. 1502–1509, 2016.
- [58] C. Schaffer, “Selecting a classification method by cross-validation,” Machine learning, vol. 13, pp. 135–143, 1993.
- [59] “sklearn.model_selection.gridsearchcv.” https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html. Accessed: 2023-06-15.
- [60] J. Bergstra and Y. Bengio, “Random search for hyper-parameter optimization,” Journal of machine learning research, vol. 13, no. 2, 2012.
- [61] “sklearn.model_selection.randomizedsearchcv.” https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html. Accessed: 2023-06-15.

- [62] “sklearn.pipeline.pipeline.” <https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html>. Accessed: 2023-06-10.
- [63] I. E. Tampu, A. Eklund, and N. Haj-Hosseini, “Inflation of test accuracy due to data leakage in deep learning-based classification of oct images,” Scientific Data, vol. 9, no. 1, p. 580, 2022.
- [64] “sklearn.metrics.classification_report.” https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html. Accessed: 2023-06-15.
- [65] K. Weiss, T. M. Khoshgoftaar, and D. Wang, “A survey of transfer learning,” Journal of Big data, vol. 3, no. 1, pp. 1–40, 2016.
- [66] “Entrez programming utilities help.” <https://www.ncbi.nlm.nih.gov/books/NBK25501/>. Accessed: 2023-06-16”.

Ringraziamenti

Alla fine di questo elaborato, vorrei ringraziare le persone che mi sono state vicine nel mio percorso universitario.

Ringrazio la mia famiglia per avermi sempre sostenuto e incoraggiato in ogni fase del mio percorso universitario. Grazie per aver fatto il tifo per me e per aver celebrato ogni esame passato con successo.

Ringrazio il Prof. Cesarini per il suo sostegno durante lo stage interno, per avermi guidato nella stesura di questo elaborato, l'enorme disponibilità e per i suoi consigli.

Ringrazio i miei amici di università, in particolare Giovanni e Giacomo, per aver reso il mio percorso accademico piacevole e memorabile. Le conversazioni interessanti, i dibattiti stimolanti e le partite di basket al campetto hanno reso i momenti trascorsi insieme davvero speciali.

Infine ringrazio la mia ragazza, Nicole, per il suo amore, il suo sostegno incondizionato e la sua pazienza. Grazie per essere sempre stata al mio fianco, per avermi sostenuto durante i momenti di stress e per aver reso il mio percorso universitario più gioioso e significativo.