

Applicazioni e tecniche di NLP

Christopher Buratti

c.buratti@pm.univpm.it

Corso di «Data Science»

A.A. 2025/2026

Applicazioni e tecniche di NLP

TEXT CLASSIFICATION

DEFINIZIONE

Text Classification = assegnare una o più categorie (classi) a un testo.

Alcuni esempi:



- Email (spam/no spam);
- Sentiment analysis (triste, felice, arrabbiato, etc.);
- Argomento principale (sport, musica, etc.).

Tre tipologie:

- Classificazione binaria (2 classi);
- Classificazione multi-classe (> 2 classi);
- Classificazione multi-label (ogni elemento può avere più di una label associata).



PIPELINE

1. Partire da un dataset di testi etichettato (con label associate ad ogni elemento);
 2. Suddividere il dataset in training + (validation) + test set;
 3. Trasformare ogni testo in un vettore di feature;
 4. Addestrare un classificatore con tali vettori di feature e con le relative label;
 5. Valutare le performance del modello con le metriche di valutazione scelte utilizzando il test set;
 6. Utilizzare il modello per l'inferenza.
- 
- 

NAÏVE BAYES CLASSIFIER

È un classificatore probabilistico basato sul teorema di Bayes.

Stima la probabilità condizionale di ogni feature dato un testo di riferimento per ogni classe e moltiplica le probabilità di tutte le feature del testo per calcolare la probabilità finale per ogni classe.

Infine, sceglie la classe con la massima probabilità di classificazione.

$$P(A|B) = \frac{P(B|A) P(A)}{P(B)}$$

colab

LOGISTIC REGRESSION

Classificatore generativo (Naive Bayes) vs classificatore discriminativo (Logistic Regression):

- Generativo = calcola $P(X|Y)$, ovvero la probabilità di osservare quelle feature data una classe;
- Discriminativo = calcola $P(Y|X)$, ovvero la probabilità di appartenenza a una determinata classe, date le feature.

Obiettivo: apprendere un separatore lineare tra le classi nei dati di addestramento con lo scopo di massimizzare la probabilità delle feature di appartenere alla classe.

Questo "apprendimento" dei pesi delle caratteristiche e della distribuzione delle probabilità su tutte le classi avviene attraverso una funzione chiamata «logistica» e da qui il nome «Regressione Logistica».

colab

SUPPORT VECTOR MACHINE (SVM)

È un classificatore discriminativo (simile alla Regressione Logistica).

Ha come obiettivo quello di trovare il miglior iperpiano separatore per le classi (permettendo divisioni non-lineari tra le classi, a differenza della Regressione Logistica).

In media, richiede più tempo di addestramento.

colab

NEURAL (WORD) EMBEDDING

È possibile addestrare una rete neurale (o utilizzarne una pre-addestrata, come **Word2Vec**) per ottenere l'embedding di ogni parola dei testi del dataset.

Ogni embedding ottenuto rappresenterà la feature della parola di riferimento, e sarà utilizzato per addestrare il classificatore.

Un modello di questo tipo, può essere visto, dunque, come un dizionario che ha come chiavi le parole osservate in fase di training e come valori i rispettivi embedding.

LIMITAZIONE: un approccio di questo tipo funziona soltanto con le parole viste in fase di training (a cui è, quindi, mappato un embedding).

NEURAL (WORD) EMBEDDING

Per effettuare l'embedding di una frase, è necessario aggregare i singoli embedding (il metodo più semplice è tramite la media).

```
# Creating a feature vector by averaging all embeddings for all sentences
def embedding_feats(list_of_lists):
    DIMENSION = 300
    zero_vector = np.zeros(DIMENSION)
    feats = []
    for tokens in list_of_lists:
        feat_for_this = np.zeros(DIMENSION)
        count_for_this = 0
        for token in tokens:
            if token in w2v_model:
                feat_for_this += w2v_model[token]
                count_for_this += 1
        feats.append(feat_for_this/count_for_this)
    return feats

train_vectors = embedding_feats(texts_processed)
print(len(train_vectors))
```

NEURAL (SUBWORD) EMBEDDING

Risolviamo il problema dell'OOV («Out Of Vocabulary»), che si presenta quando abbiamo di fronte una parola non inserita nel «vocabolario», considerando gli n-gram (sottosequenze delle parole).

È un processo più oneroso ma che porta con sé 2 vantaggi principali:

- Consente di lavorare con parole non viste in training (OOV);
- Facilita e velocizza l'addestramento del classificatore, anche su dataset molto grandi.

fastText è una libreria che si occupa della gestione degli embedding, consentendo di addestrare e testare il modello senza estrarre le feature separatamente.

È stato implementato in modo estremamente efficiente, con una riduzione minima delle performance di classificazione, ricorrendo al «vocabulary pruning» (eliminando, cioè, gli n-gram meno rilevanti) e utilizzando algoritmi di compressione.

DOCUMENT EMBEDDING

Così come per i testi, è possibile addestrare un classificatore per i documenti.

Per ogni documento vanno estratte delle feature (ad es. numero di parole, numero di font utilizzati, numero di paragrafi, etc.), oppure si può scegliere di effettuare l'embedding del documento e di utilizzare tale embedding come feature.

Un esempio di modello che fa ciò è **Doc2vec**.

DEEP LEARNING

È possibile classificare i dati testuali ricorrendo, anche, a reti neurali.

In questo caso, viene richiesto un pre-processing dei dati più sofisticato:

1. Tokenizzare i dati testuali e convertirli in vettori di indici (per ogni parola, sarà mappato il valore, ovvero l'indice, corrispondente);
2. Applicare il padding per portare tutti i vettori a stessa lunghezza;
3. Mappare ogni «indice» al rispettivo vettore di embedding, ottenuto moltiplicando l'indice stesso per la matrice degli embedding (tale matrice può essere ottenuta partendo da modelli pre-addestrati o addestrando un modello sui dati in esame);
4. Utilizzare i vettori di embedding come input alla rete neurale.

A questo punto, si può addestrare un qualsiasi classificatore (ad es. Convolutional Neural Network, Recurrent Neural Network, Large Language Model, etc.).

Applicazioni e tecniche di NLP

INFORMATION EXTRACTION

INFORMATION EXTRACTION

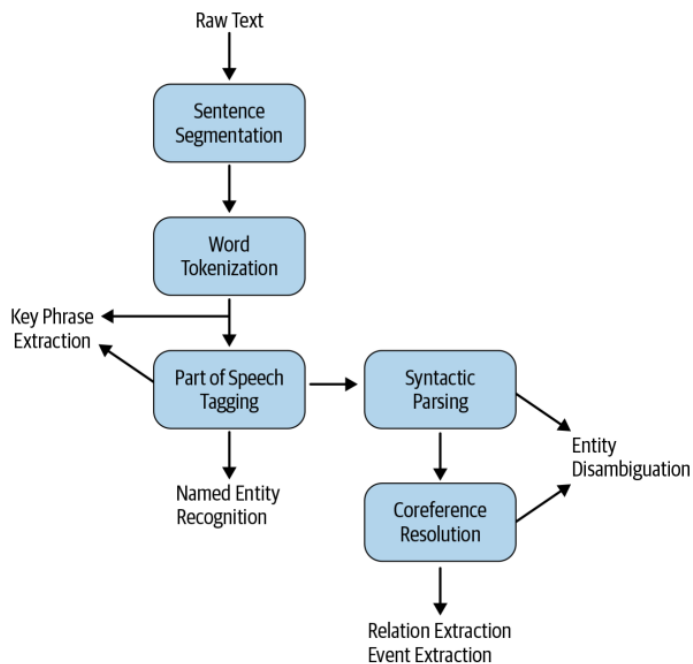
L'information extraction è quel task dell'NLP che riguarda l'estrazione di informazioni rilevanti da testi.

Il problema principale da affrontare riguarda il fatto per cui i dati non sono strutturati, e possono contenere diverse tipologie di informazioni.

Alcuni esempi applicativi:

- Tagging: associare una o più etichette ad un contenuto;
- Chatbot: comprendere gli elementi della domanda per rispondere correttamente (ad es. per la domanda «What are the best cafes around the Eiffel Tower?», il chatbot deve comprendere che «cafe» e «Eiffel Tower» sono entrambi un «luogo»);
- Estrarre dati da moduli e ricevute: può essere semplice (pattern matching) se abbiamo dati strutturati, altrimenti si deve ricorrere a tecniche più sofisticate (ad es. con l'OCR, ovvero l'Optical Character Recognition, che richiede di estrarre informazioni da un'immagine);
- E molti altri...

PIPELINE



- **Sentence Segmentation:** suddivisione del testo in singole frasi;
- **Word Tokenization:** suddivisione delle frasi in token;
- **Part of Speech Tagging:** ad ogni parola viene assegnata una categoria grammaticale (come verbo, sostantivo, aggettivo);
- **Key Phrase Extraction:** identificare frasi o parole chiave che sono rilevanti per il compito di estrazione;
- **Named Entity Recognition:** è una sorta di classificazione per trovare elementi importanti nel testo;
- **Syntactic Parsing:** si analizza la struttura grammaticale della frase per costruire un albero sintattico, che mostra le relazioni tra le parole;
- **Coreference Resolution:** si individuano le diverse parole o frasi si riferiscono alla stessa entità;
- **Entity Disambiguation:** distinguere tra entità con nomi simili o identici (ad es. «Apple» può essere un frutto o un'azienda);
- **Relation Extraction:** si identificano le relazioni tra entità;
- **Event Extraction:** si individuano eventi descritti nel testo (ad es. un meeting, una conferenza, etc.) e le entità coinvolte.

KEYPHRASE EXTRACTION (KPE)

Keyword and phrase extraction: task dell'information extraction che si occupa di estrarre le parole/frasi più importanti che caratterizzano il testo in esame.

Alcuni esempi di applicazione:

- Ricerca di informazioni (information retrieval);
- Tagging automatico dei documenti;
- Riassunto di testi;
- E molti altri...

Gli algoritmi più efficaci si basano su una rappresentazione a grafo. I nodi del grafo sono le parole e le frasi del testo, mentre il peso indica il livello di importanza del nodo stesso. Gli archi rappresentano le connessioni delle parole/frasi nel testo.

I nodi più importanti sono scelti sulla base della frequenza e del numero di connessioni (ogni algoritmo implementa le proprie «regole»).

NAMED ENTITY RECOGNITION (NER)

Named Entity Recognition: task dell'Information Extraction che consiste nell'identificare le entità in un documento.

Alcuni esempi di entità possono essere luoghi, organizzazioni, persone, date, prodotti, etc.

displaCy Named Entity Visualizer

Justin Pierre James Trudeau PC MP (born December 25, 1971) is a Canadian politician serving as the 23rd prime minister of Canada since 2015 and Leader of the Liberal Party since 2013. Trudeau is the second-youngest Canadian Prime Minister after Joe Clark; he is also the first to be related to a previous holder of the post, as the eldest son of Pierre Trudeau.

Entity labels (select all)

☒ PERSON ☒ NORP ☐ FACILITY ☒ GPE

☒ GPE ☒ LOC ☒ PRODUCT ☐ EVENT

☐ WORK OF ART ☐ LANGUAGE ☒ DATE ☐ TIME

☐ PERCENT ☒ MONEY ☐ QUANTITY ☐ ORDINAL

☐ CARDINAL

Model

English - en_core_web_lg (v2.0.0)

Justin Pierre James Trudeau PERSON PC MP (born December 25, 1971 DATE) is a Canadian NORP politician serving as the 23rd prime minister of Canada GPE since 2015 DATE and Leader of the Liberal Party ORG since 2013 DATE . Trudeau PERSON is the second-youngest Canadian NORP Prime Minister after Joe Clark PERSON ; he is also the first to be related to a previous holder of the post, as the eldest son of Pierre Trudeau PERSON .

Applicazioni e tecniche di NLP

UN ESEMPIO PRATICO: SOCIAL MEDIA

SOCIAL MEDIA

- Diverse piattaforme (Twitter, Facebook, Instagram, WhatsApp, etc.)
- Big Data (Volume, Varietà, Veracità, Velocità, Valore)
- Dati non strutturati

➡ Tecniche di NLP



APPLICAZIONI

**TRENDING TOPIC
DETECTION**

OPINION MINING

**SENTIMENT
DETECTION**

**RUMOR/FAKE NEWS
DETECTION**

**ADULT CONTENT
FILTERING**

**CUSTOMER
SUPPORT**

SFIDE

**ASSENZA DI
GRAMMATICA**

**NONSTANDARD
SPELLING**

MULTILINGUA

CARATTERI SPECIALI

RUMORE

TRASLITTERAZIONE

**LUNGHEZZA DEI
TESTI**

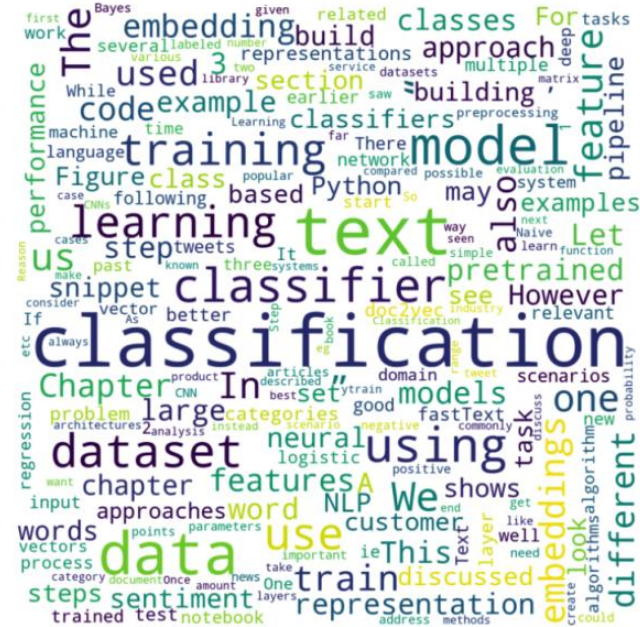
- 





Pipeline:

- Tokenizzazione del testo;
- Rimozione delle stop words;
- Calcolo della frequenza delle parole;
- Plot delle parole più frequenti.



TOKENIZER

Per tokenizzare i tweet meglio utilizzare un tokenizer ad hoc.



- TweetTokenizer
- Twikenizer
- Twokenizer
- twokenize

Esempio di tweet:

"Hey @NLPer! This is a #NLProc tweet :-D"

Tokenizzazione ideale:

['Hey', '@NLPer', '!', 'This', 'is', 'a', '#NLProc', 'tweet', ':-D']

Tokenizzazione tramite un tokenizer classico per la lingua inglese:

['Hey', '@', 'NLPer', '!', 'This', 'is', 'a', '#', 'NLProc', 'tweet', ':', '- D']

TWITTER SENTIMENT

Comprendere il sentimento delle persone in merito ad argomenti è molto più facile con l'avvento dei social.

Un modo per farlo è studiare la polarità e la soggettività di ogni token (ad es. con textblob):

- La polarità è un valore in $[-1, 1]$, dove -1 indica che il sentimento è totalmente negativo, mentre 1 positivo;
- La soggettività è un valore in $[0, 1]$, dove 0 è totalmente oggettivo e 1 soggettivo.

colab

PRE-PROCESSING

Rimozione di eventuali markup (HTML, XML, etc.)

```
from bs4 import BeautifulSoup

markup = '<a href="http://nlp.com/">\nI love <i>nlp</i>\n</a>'
soup = BeautifulSoup(markup)
soup.get_text()
```

Output:

\nI love nlp\n

PRE-PROCESSING

Conversione dati non testuali (emoji, simboli, caratteri speciali, etc.) in caratteri facilmente comprensibili (ad es. nello standard UTF-8)

```
text = 'I love Pizza 🍕! Shall we book a cab 🚗 to gizza?'
Text = text.encode("utf-8")
print(Text)
```

Output:

```
b'I love Pizza \xf0\x9f\x8d\x95!
Shall we book a cab \xf0\x9f\x9a\x95 to get pizza?'
```

PRE-PROCESSING

Gestione delle emoji (alternativa utilizzando Demoji)

```
tweet = "#startspreadingthenews yankees win great start by 🧑🏻🎅 going 5strong  
innings with 5k's 🔥 🐘 solo homerun 🌋🌋 with 2 solo homeruns  
and 🧛🏿 3run homerun... 🤡 🚣 🧑🏿 with rbi's ... 🔥🔥 🇲🇪 and 🇳🇮  
to close the game 🔥🔥!!!!....WHAT A GAME!! "
```

```
demoji.findall(tweet)
```

Output:

```
{  
  "🔥": "fire",  
  "🌋": "volcano",  
  "🧑🏿": "man judge: medium skin tone",  
  "🧑🏻🎅": "Santa Claus: medium-dark skin tone",  
  "🇲🇪": "flag: Mexico",  
  "🧛🏿": "ogre",  
  "🤡": "clown face",  
  "🇳🇮": "flag: Nicaragua",  
  "🚣": "person rowing boat: medium-light skin tone",  
  "🐘": "ox",  
}
```

PRE-PROCESSING

Rimozione degli url (sia tiny che full)

```
def process_URLs(tweet_text):
    """
    replace all URLs in the tweet text
    """
    UrlStart1 = regex_or('https?://', r'www\.')
    CommonTLDs = regex_or('com', 'co\\.uk', 'org', 'net', 'info', 'ca', 'biz',
                           'info', 'edu', 'in', 'au')
    UrlStart2 = r'[a-z0-9\.-]+?' + r'\.' + CommonTLDs +
                pos_lookahead(r'[/ \W\b]')
    # * not + for case of: "go to bla.com." -- don't want period
    UrlBody = r'^ \t\r\n<>]*?'
    UrlExtraCrapBeforeEnd = '%s+?' % regex_or(PunctChars, Entity)
    UrlEnd = regex_or(r'\.\.+', r'[<>]', r'[s]', '$')
    Url = (optional(r'\b') +
           regex_or(UrlStart1, UrlStart2) +
           UrlBody +
           pos_lookahead(optional(UrlExtraCrapBeforeEnd) + UrlEnd))

    Url_RE = re.compile("(%s)" % Url, re.U|re.I)
    tweet_text = re.sub(Url_RE, " constant_url ", tweet_text)

    # fix to handle unicodes in URL
    URL_regex2 = r'\b(htt[tp:\/]*([\x\\u][a-z0-9]*)'
    tweet_text = re.sub(URL_regex2, " constant_url ", tweet_text)
    return tweet_text
```

- Amazon: [amzn.to](https://www.amazon.to)
- New York Times: [nyti.ms](https://www.nyti.ms)
- BuzzFeed: [bzfd.it](https://www.bzfd.it)
- LinkedIn: [lnkd.in](https://www.lnkcd.in)
- Facebook: [fb.me](https://www.fb.me)

PRE-PROCESSING

Correzione dello spelling

```
from textblob import TextBlob  
  
data = "His sellection is bery antresting"  
output = TextBlob(data).correct()  
print(output)
```

Output:

His selection is very interesting.

```
def prune_multiple_consecutive_same_char(tweet_text):  
    '''  
    yesssssssss is converted to yes  
    sssssssssh is converted to ssh  
    '''  
    tweet_text = re.sub(r'(\.)\1+', r'\1\1', tweet_text)  
    return tweet_text
```

CLASSIFICAZIONE CON EMBEDDING

Utilizziamo un modello di embedding pre-trained: **Word2vec** (addestrato su dati di Wikipedia).

colab

CLASSIFICAZIONE CON EMBEDDING

Utilizziamo un modello di embedding pre-trainato: **Word2vec** (addestrato su dati di Wikipedia).

Possibili miglioramenti:

- Utilizzare modelli addestrati su dati acquisiti dai social network;
- Utilizzare un miglior tokenizer;
- Addestrare da zero il proprio modello (alternativa onerosa; richiede almeno circa 1 milione di tweet per ottenere buone performance).