INTELLIGENZA ARTIFICIALE – PROJECT WORK

CLASSIFICAZIONE DI DOCUMENTI TESTUALI

Francesco Patanè - Corso di Laurea Magistrale in Ingegneria Informatica – Big Data

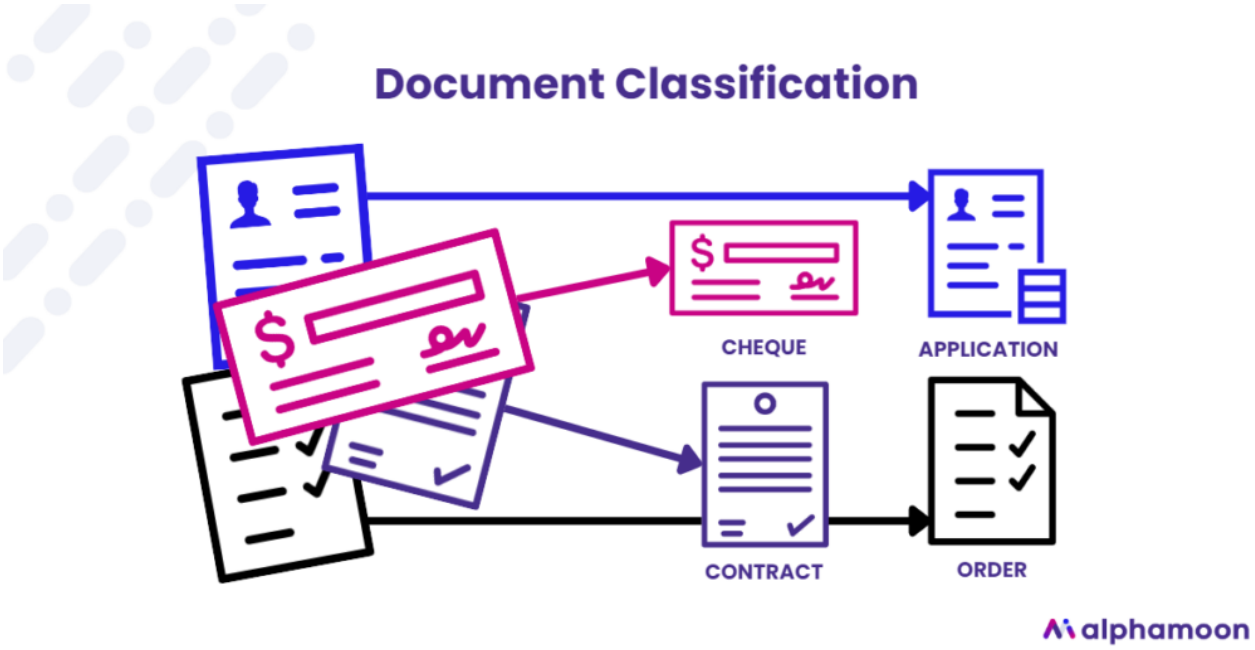
Matricola: *530HHHINGINFOR*

SOMMARIO

INTRODUZIONE

Per classificazione di documenti si intende il processo di assegnazione di un documento a una o più categorie rilevanti, facilitando i processi di gestione e analisi dello stesso.

Le tecniche di classificazione automatica di documenti sono fondamentali nella realizzazione di sistemi di recupero delle informazioni, come ad esempio i motori di ricerca, facilitando gli utenti nel trovare quello che stanno cercando.



Realizzare in modo automatico la classificazione consente un enorme risparmio di tempo, specialmente quando si parla di un numero elevato di documenti, quando le tempistiche di una classificazione manuale mal si adatterebbero alle necessità dell'era digitale.

Lo scopo della classificazione automatica di documenti è quello di creare un modello di classificazione che possa assegnare i singoli documenti alle categorie appropriate con elevata precisione.

In questa relazione verranno prima trattati gli aspetti teorici del problema di classificazione di documenti testuali e dei più comuni approcci alla sua risoluzione, per poi illustrare un'applicazione pratica di questi principi, realizzando un classificatore in linguaggio python.

CLASSIFICAZIONE TESTUALE

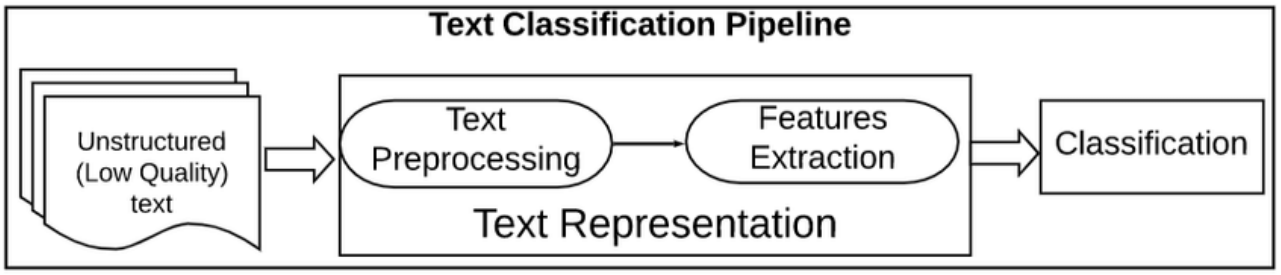
Sebbene un documento possa contenere svariati tipi di dato, di norme la forma preponderante dei dati contenuti è quella testuale. Pertanto in questa trattazione faremo riferimento alla classificazione di documenti composti esclusivamente da testo.

La classificazione testuale è un problema comunemente affrontato nell'ambito del **Natural Language Processing**.

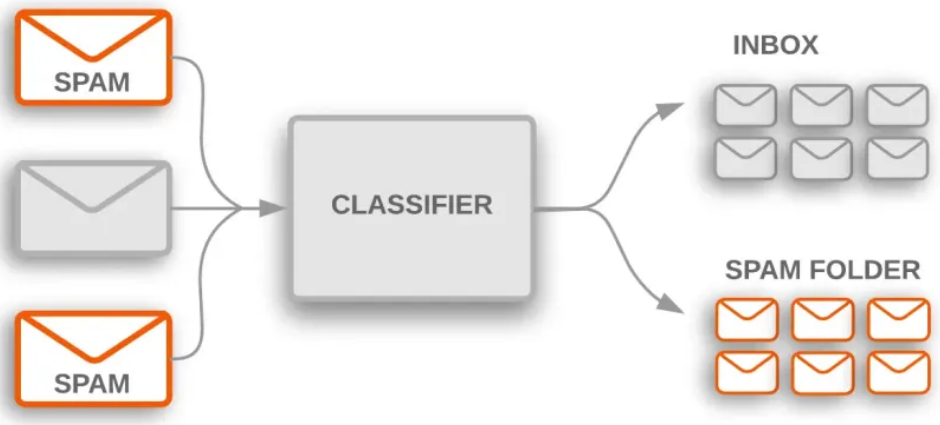
Per **Natural Language Processing,** o elaborazione del linguaggio naturale, si intendono algoritmi di intelligenza artificiale in grado di analizzare, rappresentare e quindi comprendere il linguaggio naturale. Le finalità possono variare dalla comprensione del contenuto, alla traduzione, fino alla produzione di testo in modo autonomo a partire da dati o documenti forniti in input.

Come già accennato, lo scopo della classificazione testuale è assegnare/predire una categoria a un insieme di documenti sconosciuti, spesso con l'aiuto del **machine learning supervisionato**.

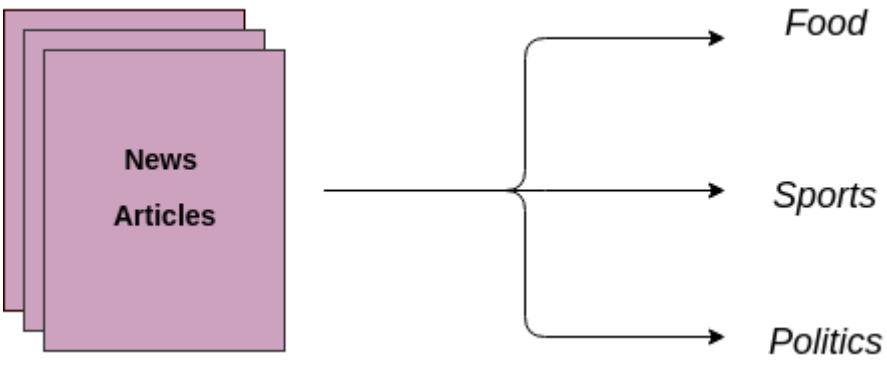
L'applicazione delle tecniche di machine learning prevede l'uso di un algoritmo addestrato su dati forniti appositamente. La particolarità è nel fatto che i dati di addestramento non saranno in forma tabellare, ma in forma puramente testuale.



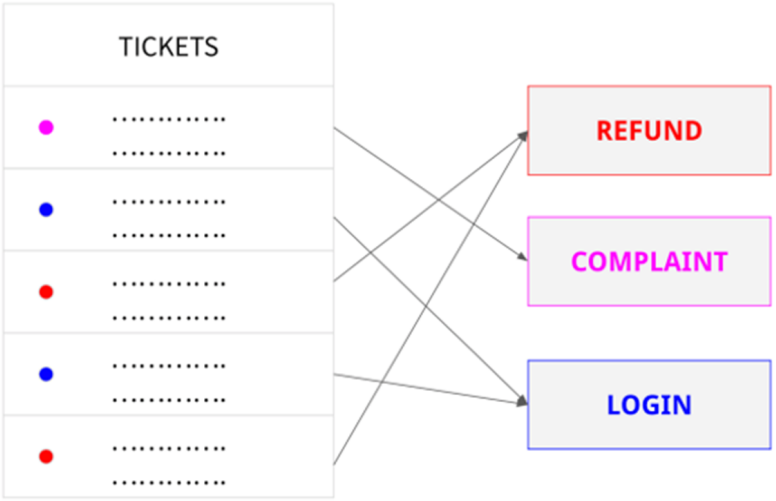
Alcune possibili applicazioni sono, ad esempio, la rilevazione dello spam



la classificazione di articoli

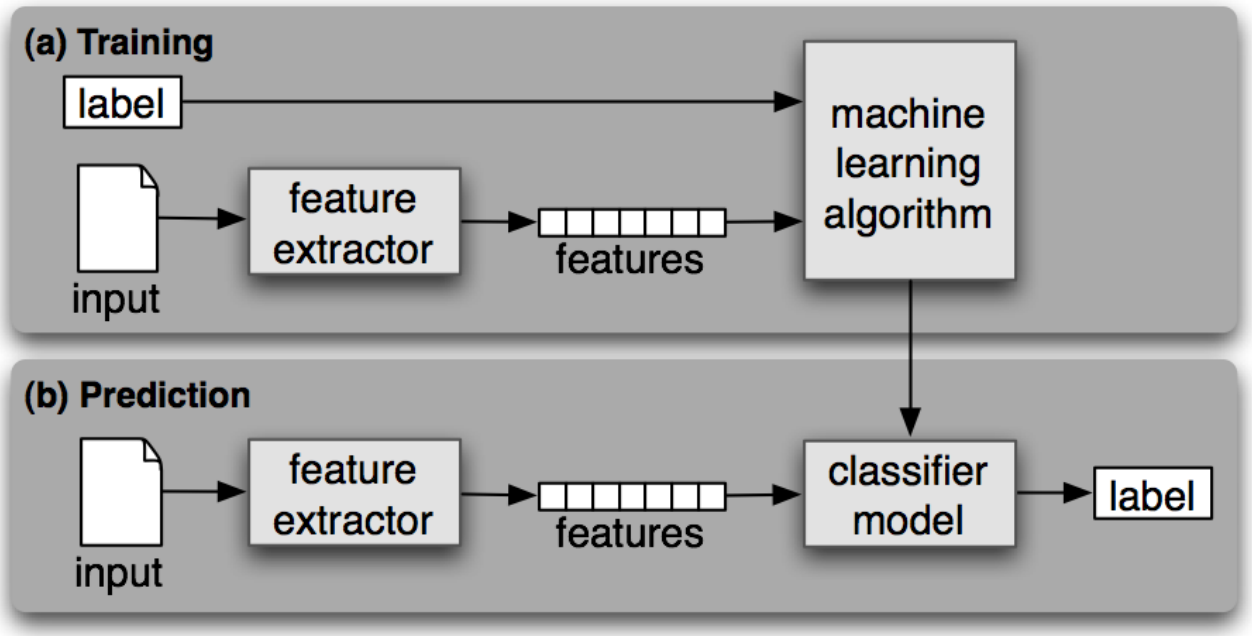


o l'assegnazione di richieste di supporto da parte del cliente.



La classificazione testuale in ambito di Machine Learning è un problema di apprendimento supervisionato. Vengono appresi i dati di input, sotto forma di semplice test, con delle label associate a rappresentare la categoria di appartenenza, definite variabili target.

Come ogni altro problema di learning supervisionato, anche la classificazione testuale prevede due fasi: **addestramento** e **previsione**.



Nella fase di addestramento, o **training**, l'algoritmo è addestrato in base agli input etichettati. al termine di questo processo, otteniamo un modello addestrato che è possibile applicare per ottenere previsioni su nuovi dati (fase di previsione/**prediction**).

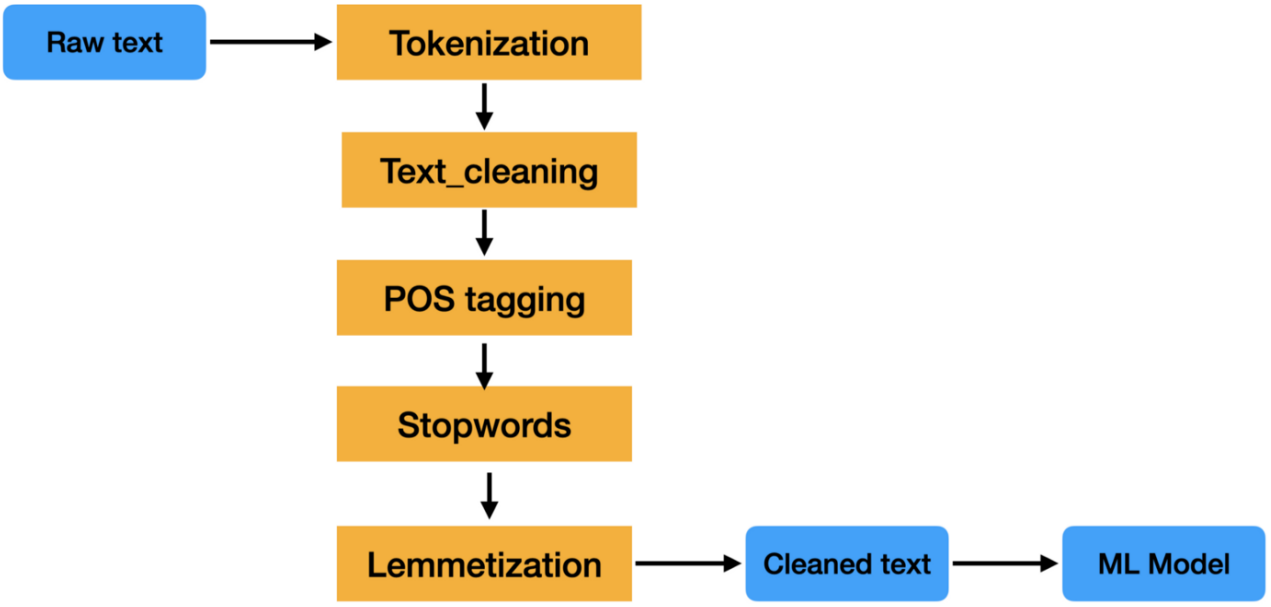
Oltre alle fasi di addestramento e training, è spesso opportuno prevedere dei passaggi a priori dove vengono effettuate una serie di operazioni sui dati, detti **preprocessamento** dei dati.

I dati testuali sono spesso ostici da utilizzare così come sono, in quanto non prevedono una struttura precisa e spesso contengono una notevole quantità di rumore, dovuta alla presenza di parole comuni ma non utili ai fini di classificazione, come ad esempio gli articoli, definite **stopwords**.

Il preprocessamento dei dati è quindi uno step importante nelle applicazioni NLP. Il suo scopo è ripulire i dati e prepararli in modo che si trovino in forma ottimizzata per le analisi successive.

Una **pipeline di preprocessamento** è quindi una serie di step di processamento applicati ai dati testuali, in modo da renderli idonei al processo di NLP.

Gli step in una pipeline di processamento non sono fissi, ma variabili in base alle necessità del caso. Tipicamente includono azioni come la tokenizzazione e la rimozione delle stopwords, in modo da ridurre la dimensione dei dati e migliorare la precisione dei processi successivi.



Quesito fondamentale è il come estrarre delle feature significative dal testo, tema intrinsecamente connesso al come rappresentare un il testo nel nostro modello.

I due approcci più comuni sono la **Bag of words** (borsa di parole) e la **tf-idf** (term frequency–inverse document frequency).

Il modello bag of words (BoW) è un approccio semplice che consiste nel rappresentare il testo come un insieme di feature numeriche. Prevede la creazione di un vocabolario di parole note a partire dal corpus documentale, per poi creare un vettore per ogni documento che contenga il numero di occorrenze per ogni parola.

	the	red	dog	cat	eats	food
1. the red dog →	1	1	1	0	0	0
2. cat eats dog →	0	0	1	1	1	0
3. dog eats food →	0	0	1	0	1	1
4. red cat eats →	0	1	0	1	1	0

La sigla tf-idf sta per term frequency–inverse document frequency, ed è un'altra forma di rappresentazione numerica del testo.

Il valore tf-idf per un termine aumenta proporzionalmente al numero di occorrenze del termine nel documento, ma cresce in maniera inversamente proporzionale con la frequenza del termine nella collezione. L'idea alla base di questo comportamento è di dare più importanza ai termini che compaiono nel documento, ma che in generale sono poco frequenti. In questo modo termini specialistici, che sono rari ma legati strettamente ad un ambito, avranno un peso maggiore, il che rende i modelli basati su tf-idf più precisi rispetto a quelli BoW.

Matematicamente, può essere rappresentata come segue.

$$TF(t,d) = \frac{number\ of\ times\ t\ appears\ in\ d}{total\ number\ of\ terms\ in\ d}$$

$$IDF(t) = log\frac{N}{1+df}$$

$$TF - IDF(t,d) = TF(t,d) * IDF(t)$$

Stabilite le principali fasi di un processo di classificazione testuale, nel prossimo capitolo sarà illustrato uno degli algoritmi classici applicati a questo tipo di problema: l'algoritmo Naive Bayes.

NAIVE BAYES

Nel ML, l'algoritmo Naïve Bayes è un algoritmo semplice e potente applicabile ai problemi di classificazione.

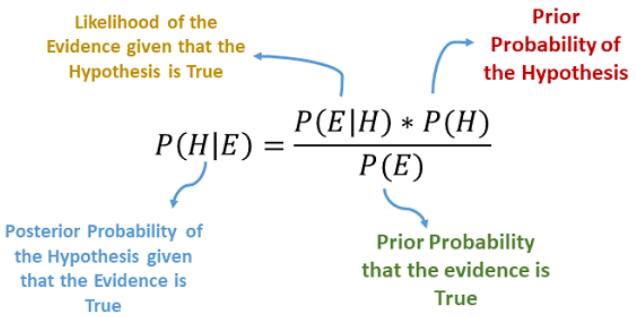
L'algoritmo si basa sull'applicazione del teorema di Bayes, assumendo che vi sia indipendenza tra una particolare feature di una classe rispetto alle altre. Da questa forte assunzione deriva il nominativo Naïve Bayes. Questo approccio è in grado di fornire buoni risultati nel campo dell'analisi di dati testuali nell'ambito dell'elaborazione del linguaggio naturale.

Il teorema di Bayes è rappresentato dalla seguente formula:

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

dove A e B sono eventi e $P(B) \neq 0$

- $P(A|B)$ è una probabilità condizionata: la probabilità che A avvenga dato B, detta anche probabilità a posteriori di A dato B.
- $P(B|A)$ è anch'essa una probabilità condizionata: la probabilità che B abbia luogo dato A.
- $P(A)$ e $P(B)$ sono le probabilità di osservare A e B rispettivamente senza nessuna condizione data. Sono dette probabilità a priori.



Nell'ambito della classificazione documentale, è di interesse capire la probabilità che un documento faccia parte di una determinata categoria in base alle probabilità delle parole di cui è composto di far parte della categoria.

Esistono vari tipi di algoritmo Naïve Bayes:

1. Gaussian Naïve Bayes Gaussiano
2. Naïve Bayes Multinomiale
3. Naïve Bayes Bernoulli

Oltre alla classificazione di documenti, questa famiglia di algoritmi trova svariate altre applicazioni, come ad esempio il filtraggio antispam, la sentiment analysis e la realizzazione di sistemi di raccomandazione.

Nel prossimo capitolo, illustreremo un'applicazione pratica dell'algoritmo Naïve Bayes Multinomiale nella realizzazione di un classificatore Bayesiano.

REALIZZAZIONE DI UN CLASSIFICATORE BAYESIANO IN PYTHON

Nel prossimo capitolo, illustreremo la creazione di un semplice classificatore Bayesiano mediante il linguaggio python, avvalendosi in particolare delle sue preziose librerie a tema Data Science, come Pandas, Scikitlearn e Nltk. Ci avvarremo inoltre dell'ambiente Google Colab in modo da sfruttare la flessibilità dei Notebook Jupyter.

Come dataset, faremo riferimenti a un corpus di mille documenti, reperibile su kaggle al seguente link:

https://www.kaggle.com/datasets/jensenbaxter/10dataset-text-document-classification

La collezione comprende mille documenti testuali in lingua inglese, divisi equamente in dieci diverse categorie:

1. business
2. entertainment
3. food
4. graphics
5. historical
6. medical
7. politics
8. space
9. sport
10. technologie

Quindi 100 documenti per categoria. Divideremo poi i documenti in insieme di training e test in rapporto 30% a 70%. Useremo 30 documenti di ogni categoria per l'addestramento, e poi useremo il resto della collezione per la fase di test.

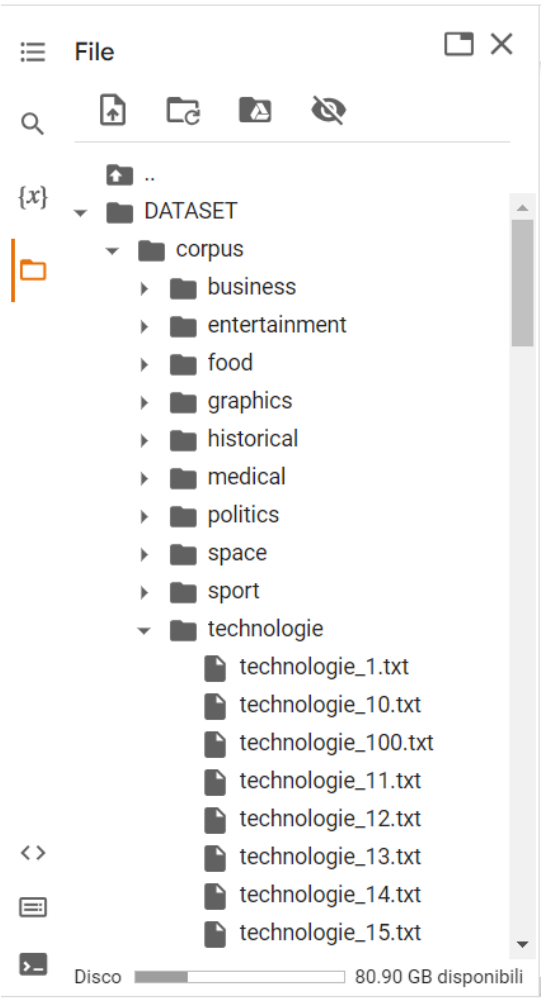
Per prima cosa impostiamo le importazioni delle librerie e funzioni di cui avremo bisogno per la nostra applicazione:

```
[ ]  from pathlib import Path
     import nltk
     from nltk.corpus import stopwords
     from nltk.tokenize import word_tokenize
     import string
     import pandas as pd
     from sklearn.feature_extraction.text import CountVectorizer
     from sklearn.model_selection import train_test_split
     from sklearn.feature_extraction.text import CountVectorizer
     from sklearn.feature_extraction.text import TfidfTransformer
     from sklearn.naive_bayes import MultinomialNB
     from sklearn.metrics import (
         accuracy_score,
         confusion_matrix,
         ConfusionMatrixDisplay,
         f1_score,
         classification_report,
     )
```

Proseguiamo poi con l'impostazione del dataset. Carichiamo prima una versione archiviata nel notebook, per poi estrarla.

```
[1]  !unzip /content/DATASET.zip
```

Possiamo verificare il caricamento dei file nel notebook jupyter



Oltre ai file e al testo/feature al loro interno, dobbiamo predisporre una struttura atta a rappresentare le categorie/label. Una enumerazione fa al caso nostro.

```
[ ]  from enum import Enum

     class Categories(Enum):
       business=0
       entertainment=1
       food=2
       graphics=3
       historical=4
       medical=5
       politics=6
       space=7
       sport=8
       technologie=9
```

Mappare le categorie in valori numerici agevolerà l'esecuzione dell'algoritmo.

Possiamo ora iniziare a definire i dataset di training e test. Come anticipato, il dataset di training sarà composto da 30 documenti per ogni categoria, mentre i documenti restanti verranno adoperati ai fini di test.

Al fine di addestrare il classificatore su un campione omogeneo, per ogni categoria, i documenti con numerazione *3, *5, *7 verranno assegnati all'insieme di training, gli altri a quello di test.

```
[ ]  train_corpus = []
     test_corpus = []

     for child in Path('/content/DATASET/corpus').glob('**/*.txt'):
         if child.is_file():
           splitted_filename = child.name.split("_")
           doc_name = splitted_filename[0]
           doc_number = splitted_filename[1].split(".")[0]
           if doc_number[len(doc_number)-1] == "3" or doc_number[len(doc_number)-1] == "5" or doc_number[len(doc_number)-1] == "7":
             train_corpus.append((child.name, child.read_text().replace("\n", " "), Categories[child.name.split("_")[0]].value))
           else:
             test_corpus.append((child.name, child.read_text().replace("\n", " "), Categories[child.name.split("_")[0]].value))
```

```
[ ]  print("Dimensione insieme di training: ", len(train_corpus))
     print("Dimensione insieme di test: ", len(test_corpus))

     Dimensione insieme di training:  300
     Dimensione insieme di test:  700
```

In questo modo, abbiamo ottenuto due vettori composti da triple (filename, text, category).

Per comodità, trasformiamo i vettori di documenti in dataset pandas, più facili da manipolare.

```
[7]  df_train = pd.DataFrame(train_corpus, columns=['document', 'text', 'category'])
     df_test = pd.DataFrame(test_corpus, columns=['document', 'text', 'category'])
```

Training set

	document	text	category
0	technologie_65.txt	Yahoo celebrates a decade online Yahoo, one o...	9
1	technologie_47.txt	Millions buy MP3 players in US One in 10 adul...	9
2	technologie_7.txt	Microsoft releases bumper patches Microsoft h...	9
3	technologie_3.txt	Microsoft seeking spyware trojan Microsoft is...	9
4	technologie_45.txt	Sony PSP tipped as a 'must-have' Sony's Plays...	9
...	...	...	...
295	food_25.txt	1 lg sweet red pepper 1 ...	2
296	food_67.txt	Increasing the amount of ultra-processed foods...	2
297	food_57.txt	Eating Meat Comes with a Higher Risk for Devel...	2
298	food_53.txt	CAN USING OLIVE OIL REDUCE THE RISK OF CANCER ...	2
299	food_97.txt	Cheese is a great source of protein and calciu...	2

300 rows × 3 columns

Test set

	document	text	category
0	technologie_10.txt	Google's toolbar sparks concern Search engine...	9
1	technologie_82.txt	Games firms 'face tough future' UK video game...	9
2	technologie_18.txt	PlayStation 3 chip to be unveiled Details of ...	9
3	technologie_20.txt	Security scares spark browser fix Microsoft i...	9
4	technologie_89.txt	Latest Opera browser gets vocal Net browser O...	9
...	...	...	...
695	food_66.txt	The conversations around health-focused food s...	2
696	food_32.txt	1 pita pocket 1/4 ...	2
697	food_100.txt	eafy greens that contain beta-carotene, such a...	2
698	food_89.txt	Magnesium is the fourth most abundant mineral ...	2
699	food_16.txt	2 cups all-purpose flour -- more i...	2

700 rows × 3 columns

A questo punto il prossimo step è quello di trasformare le singole parole in feature numeriche. Utilizzeremo la classe **CountVectorizer**, che converte una collezione di documenti in una matrice document-term. Applichiamo, in sostanza, l'approccio Bag of Words.

Particolarmente utile è la possibilità di indicare al Vettorizzatore di ignorare le stopwords.

```
[ ]  vectorizer = CountVectorizer(stop_words="english")
```

```
[ ]  X_train = vectorizer.fit_transform(df_train.text, df_train.category).toarray()
     y_train = df_train.category
     X_train.shape

     (300, 14774)
```

```
[ ]  X_test = vectorizer.transform(df_test.text).toarray()
     y_test = df_test.category
     X_test.shape

     (700, 14774)
```

Definiamo X_train e X_test gli insiemi di feature di training e test. Y rappresenterà le label, uguali in entrambi i casi, essendo i valori numerici relativi alle categorie come descritto nella enumerazione precedentemente definita.

A questo punto, passiamo alla realizzazione del classificatore propriamente detto. Adotteremo un classificatore Naive Bayes di tipo multinomiale, addestrato sui dati di training.

```
[ ]  clf = MultinomialNB().fit(X_train, y_train)
```

Provando a sottoporre al classificatore delle frasi inventate sul momento, noteremo che le previsioni del presentano una buona accuratezza.

```
[ ] print(Categories(clf.predict(vectorizer.transform(["the risk of cancer increases after 60 years of age"]))).name)

    medical
```

```
[ ] print(Categories(clf.predict(vectorizer.transform(["Policy of media was shaped by Italy pm Silvio Berlusconi, great friend of Tony Blair."]))).name)

    politics
```

Sottoponiamo ora al classificatore un documento scelto a caso dall'insieme di test, e verifichiamo che sia assegnato alla categoria corretta.

```
[ ] filename = '/content/DATASET/corpus/space/space_36.txt'
    file_text = ""
    with open(filename, 'r') as reader:
        file_text = reader.read()
    print(file_text)
```

```
[ ] print(Categories(clf.predict(vectorizer.transform([file_text]))).name)

    space
```

Applichiamo ora il classificatore all'intero insieme di test.

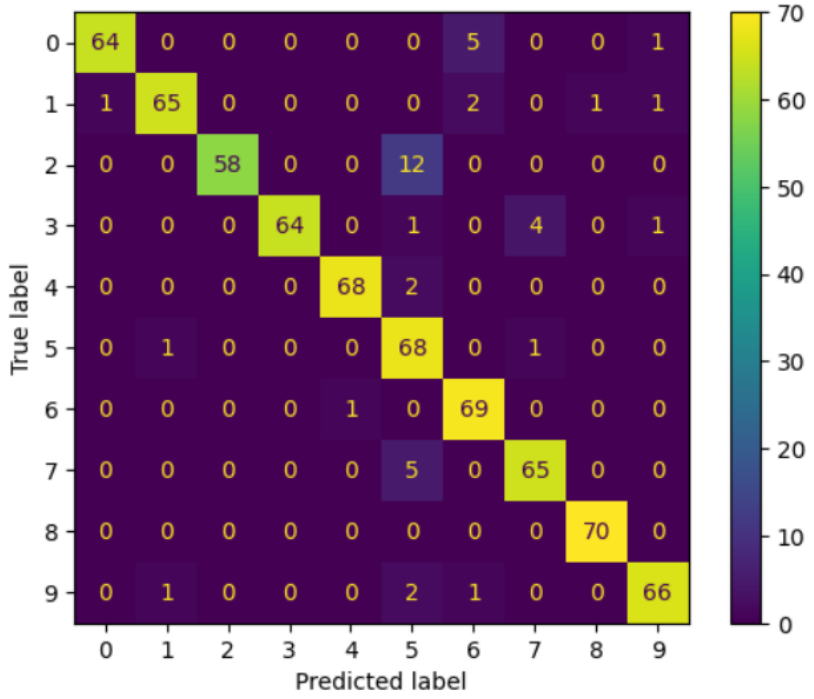
```
[ ] y_pred = clf.predict(X_test)
    accuracy = accuracy_score(y_pred, y_test)
    print(accuracy)

    0.9385714285714286
```

Il classificatore mostra una precisione di circa il 94%.

Una confusion matrix può aiutarci a visualizzare meglio il risultato ottenuto.

```
[ ]  cm = confusion_matrix(y_test, y_pred)
     disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=None)
     disp.plot();
```



Notiamo come il risultato migliore si sia riscontrato nella categoria 8, ovvero sport, con 70 documenti su 70 correttamente assegnati. Al contrario, il peggior risultato è di gran lunga quello della categoria 2, food. con solo 58 documenti correttamente classificati. E' interessante notare come tutti i documenti relativi a food classificati in modo errato sono tutti stati indicati dal classificatore come di categoria 5, ovvero medical. Abbiamo un'ulteriore prova di quanto una sana alimentazione sia fondamentale per mantenersi in buona salute!

In totale, sono stati correttamente classificati 657 documenti su 700 correttamente. Proviamo ad ottenere un risultato migliore adottando la metrica tf-idf al posto dell'approccio Bag of Words.

Ancora una volta scikitlearn ci fornisce una facile soluzione attraverso la classe **TfidfTransformer**, che trasforma la matrice di conteggi ottenuta con CountVectorizer in una rappresentazione tf-idf.

```
[ ] vectorizer = CountVectorizer(stop_words="english")
    tfidf_transformer = TfidfTransformer()

    X_tf_train = vectorizer.fit_transform(df_train.text, df_train.category).toarray()
    y_train = df_train.category
    X_idf_train = tfidf_transformer.fit_transform(X_tf_train)
    X_idf_train.shape
```

```
(300, 14774)
```

```
[ ] clf_idf = MultinomialNB().fit(X_idf_train, y_train)
```

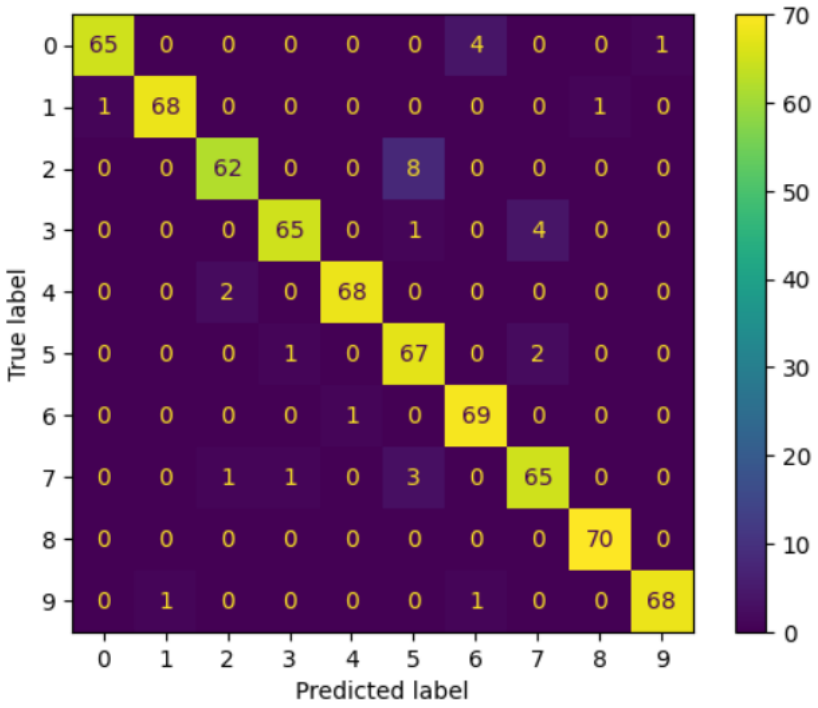
```
[ ] X_tf_test = vectorizer.transform(df_test.text).toarray()
    y_test = df_test.category
    X_idf_test = tfidf_transformer.transform(X_tf_test)
    X_idf_test.shape
```

```
(700, 14774)
```

```
[ ] y_idf_pred = clf.predict(X_idf_test)
    accuracy = accuracy_score(y_idf_pred, y_test)
    accuracy
```

```
0.9528571428571428
```

Osserviamo un aumento della precisione di circa 1,4%, con 667 documenti correttamente classificati. Un incremento di 10 documenti corrispondente al 95% di precisione.



Notiamo in particolare un miglioramento nella classificazione della categoria food, con 4 documenti addizionali classificati correttamente. Rimane invariata la categoria sport con 70 previsioni corrette su 70.

CONCLUSIONI

In questo documento abbiamo illustrato gli aspetti teorici del problema di classificazione di documenti testuali nell'ambito del Natural Language Processing e delle sue possibili applicazioni pratiche.

Abbiamo discusso le varie fasi di processamento dei dati necessarie per trattare dati testuali, abbiamo analizzato i più diffusi approcci per la definizione di un documento testuale in termini di feature, analizzando una delle famiglie di algoritmi più efficaci nella costruzione di classificatori testuali.

Infine abbiamo illustrato a livello pratico come costruire un semplice classificatore Bayesiano adoperando il linguaggio python e le librerie da esso offerte, a partire da un corpo di 1000 documenti, adoperati in parte per l'addestramento e in parte per il test.

Abbiamo mostrato i risultati ottenuti in termini di precisione tramite un approccio basato su Bag of Words, per poi migliorare la precisione adottando l'approccio Tf-IDF.

Nel complesso, è stata sviluppata una semplice applicazione che tuttavia mostra i tratti principali di un modello per la classificazione documentale e ne mostra le potenzialità.

BIBLIOGRAFIA

https://daxg39y63pxwu.cloudfront.net/images/blog/machine-learning-nlp-text-classification-algorithms-and-models/Text_Classification_Machine_Learning_NLP.webp

https://www.researchgate.net/publication/344945166/figure/fig1/AS:951814711148546@1603941719820/Text-Classification-Pipeline.ppm

https://www.datacamp.com/tutorial/text-classification-python

*https://miro.medium.com/v2/resize:fit:1400/format:webp/1*1pTLnoOPJKKcKIcRi3q0WA.jpeg*

https://www.kaggle.com/code/prashant111/naive-bayes-classifier-in-python