Classificazione Testuale con Naive Bayes

Gaetano de Gennaro

6 Agosto 2020

Indice dei contenuti

Il Teorema di Bayes

Il teorema di Bayes proposto da Thomas Bayes, è utilizzato per calcolare la probabilità condizionata di un evento A, sapendo che si è verificato un evento B, a partire dalla conoscenza delle probabilità a priori degli eventi A e B e della probabilità condizionata di B noto A. Il teorema di Bayes è utilizzato in molti campi, come nella diagnosi medica per calcolare la probabilità che un individuo sia affetto da una malattia sapendo che presenti determinati sintomi.

Enunciato del teorema di Bayes

Considerando un insieme di variabili aleatorie indipendenti[1] $A_1, ..., A_n$ che partizionano l'insieme degli eventi Ω, la probabilità condizionata è definita come:

$$P(A_i|B) = \frac{P(B|A_i)P(A_i)}{P(B)} = \frac{P(B|A_i)P(A_i)}{\sum_{j=1}^{n} P(B|A_j)P(A_j)}$$

Dove:

- $P(A_i|B)$: probabilità condizionata di A_i noto B. E' anche conosciuta come probabilità a posteriori visto che dipende dallo specifico valore di B.
- $P(B|A_i)$: probabilità condizionata di B noto A_i
- $P(A_i)$: probabilità a priori di A_i.
- $P(B)$: probabilità a priori di B, e funge da costante di normalizzazione, che permette di ottenre $P(A_i|B) = 1$ al variare di i.

Naive Bayes

L'espressione Naive Bayes ("ingenuo") indica il fatto che si fanno forti assunzioni di indipendenza nel modello. Si assume infatti che data una certa classe, ciascuna delle variabili aleatorie (features) siano indipendenti. Formalmente quindi dato un vettore $B = (B_1, ..., B_n)$ si assume che sia verificata:

$$P(B_1, ..., B_n|A) = \prod_{i=1}^{n} P(B_i|A)$$

Questa assunzione di indipendenza però non sempre è verificata nella realtà, motivo per il quale è detto "Naive Bayes". Nonostante ciò, i modelli che utilizzano Naive Bayes funzionano sorprendentemente bene.

Questa particolarità sta a significare che la probabilità a posteriori:

$$P(A|B_1, ..., B_n) \propto P(A)P(B_1|A)...P(B_n|A)$$

Il teorema può essere rivisto in diverse maniere a seconda della distribuzione di probabilità delle varie variabili aleatorie.

Naive Bayes: distribuzione di Bernoulli

Il torema di Bayes può essere rivisto prendendo in considerazione variabili aleatorie con distribuzione di Bernoulli[2]. In questo caso sia $B = (B_1, ...B_n)$ con $B_1, ...B_n \sim B(p)$ e $p \in [0, 1]$. Si è quindi in presenza di variabili aleatorie binarie. In questo caso la verosomiglianza può essere calcolata con:

$$P(B|A) = \prod_{i=1}^{n} P(B_i|A)^{x_i}(1 - P(B_i|A))^{(1-x_i)}$$

[1] Dato un insieme di eventi A_1, A_2, ..., A_n, si dicono indipendenti qualora il verificarsi di uno di essi non influisce sul calcolo della probabilità di tutti gli altri.

[2] Una variabile aleatoria discreta X ha distribuzione di Bernoulli $B(p)$ con $p \in [0, 1]$ se e solo se $P(X = i) = p^i(i - p)^{1-i}$

Dove xi è un valore booleano che esprime presenza o assenza.

Questo modello è molto popolare nei task di classificazione testuale, dove le occorrenze dei termini nei documenti vengono descritte mediante valori booleani presente/assente.

Naive Bayes: distribuzione Multinomiale

Se si trattano variabili aleatorie con distribuzione Multinomiale[3], la verosomiglianza può essere calcolata come:

$$P(B|A) = \frac{\Sigma_i x_i!}{\Pi_i x_i!} \prod_i^n P(B_i|A)^{x_i}$$

Dove $x = (x_1, ..., x_n)$, e x_i conta il numero di volte in cui l'evento i si verifica in una particolare istanza. Questo modello è usato tipicamente per classificazione di documenti, i cui eventi rappresentano le occorrenze di un termine in un singolo documento.

Nota bene: se una data classe e un dato termine non occorrono mai nei dati a disposizione, allora la probabilità stimata sarà zero. Questo perchè la probabilità stimata è direttamente proporzionale al numero di occorrenze dei valori di una feature. Questo potrebbe rappresentare un problema perchè se un termine non occorre mai nei documenti di una data classe, allora la probabilità che il documento appartenga a quella classe sarà automaticamente zero. Per questo motivo è spesso conveniente incorporare una correzione. Si introduce così una tecnica nota come **Laplace Smoothing**.

Laplace smoothing

Data una osservazione $x = (x_1, ..., x_d)$ da una distribuzione multinomiale con N campioni e il vettore $\theta = (\theta_1, ..., \theta_d)$, una versione "smoothed" dei dati dato lo stimatore:

$$\hat{\theta} = \frac{x_i + \alpha}{N + \alpha d}$$

Dove $\alpha > 0$ è il parametro di "smoothing". Nelle applicazioni di classificazione testuale solitamente si pone $\alpha = 1$, ed in questo caso la tecnica prende il nome di *Laplace Smoothing*.

Vantaggi e svantaggi di Naive Bayes

Vantaggi

- Rapido e richiede poca memoria
- Robusto sulle feature irrilevanti: le feature irrilevanti non influiscono sui risultati
- Molto performante in domini con features equalmente importanti
- Ottimo se utilizzato sotto assunzione di indipendenza delle features

Svantaggi

- L'algoritmo richiede la conoscenza di tutti i dati del problema. In particolar modo delle probabilità semplici e condizionate. Spesso sono informazioni difficili e costose da ottenere.
- L'algoritmo fornisce un'approssimazione "ingenua" (naive) del problema perché non considera la correlazione tra le caratteristiche dell'istanza.

[3]Un insieme di variabili aleatorie indipendenti $B_1, ..., B_n$ hanno distribuzione Multinomiale se e solo se $\frac{\Sigma_i x_i!}{\Pi_i x_i!} \prod_i^n p_i^{x_i}$ dove x_i è il numero di volte in cui l'evento si verifica e p_i è la probabilità con cui esso si verifica.

Classificazione Testuale e Naive Bayes

La classificazione del testo è un'attività che si occupa di classificare testi digitali espressi in lingua naturale, assegnando in maniera automatica collezioni di documenti ad una o più classi appartenenti ad un insieme di classi predefinito.

Per realizzare ciò si utilizzano degli approcci di apprendimento automatico, dove è necessario addestrare il sistema tramite apprendimento di esempi, da cui generare un modello generale per la classificazione automatica. Esistono tuttavia altri approcci, come quello non supervisionato.

In questo progetto si è interessati all'apprendimento supervisionato. Si suppone di disporre di:

- Un set di documenti $D = \{d_1, d_2, ..., d_n\}$, dove ciascun documento d_i è composto da un insieme di termini $T_i = \{t_1, t_2, ..., t_m\}$
- Un set di classi $C = \{c_1, c_2, ..., c_k\}$
- Un insieme di coppie (d_p, c_z) dove $1 \leq p \leq n$ e $1 \leq z \leq k$, dove n è il numero totale di documenti e z è il numero totale di classi.

Il task di classificazione si pone come obiettivo quello di addestrare un classificatore che, dato un generico documento sconosciuto d, predica la classe di appartenenza $c \in C$. Il classificatore si può interpretare come una funzione $\gamma : D \to C$

Step 1: Bag of Words

Il primo step consiste nella creazione di un modello *bag of words*, utilizzato nell'Information Retrieval e nell'elaborazione del linguaggio naturale per rappresentare i documenti ignorando l'ordine delle parole, consentendo una gestione basata su liste. Durante questo step ciascun documento viene processato per creare una rappresentazione maggiormente adeguata per il task di classificazione testuale. Il modello bag of words, solitamente viene costruito dopo i seguenti processi:

- **Tokenizzazione**: segmentazione del testo in parole singole
- **Rimozione delle stopwords**: rimozione dei termini non informativi come gli articoli, congiunzioni, valori numerici, simboli, ecc.
- **Stemming**: si riportano tutti i termini alla propria radice. Ad esempio: studies -> studi, studying -> study
- **Lemmatizzazione**: prende in considerazione l'analisi morfologica della parola. Per fare ciò è necessario un dizionario dettagliato che l'algoritmo può utilizzare per riportare il termine al suo lemma. Il dizionario più comune è **WordNet**.

Di seguito viene riportato un banale esempio di rappresentazione bag of words:

term	frequency
great	2
love	2
recommend	1
laugh	1
happy	1

Step 2: Costruzione della matrice term-doc

I classificatori e gli algoritmi di apprendimento non possono processare direttamente il testo nella sua forma originale, poichè la maggior parte di essi prevede l'utilizzo di vettori numerici con dimensioni fisse anzicchè vettori di testo con lunghezza variabile. Pertanto, durante questa fase, i testi vengono convertiti in una rappresentazione più adeguata per gli obiettivi prefissati. Un approccio comune per l'estrazione di features dal testo è l'uso del modello *bag od words* descritto nello step precedente.

Vi sono due modi per realizzare una matrice termini-documenti. Fissato un documento d_i e un termine t_j, nel primo approccio si pone in posizione (i, j) della matrice il valore 1 se il termine t_j compare nel documento d_i,

0 altrimenti. Questo primo approccio risulta essere il più semplice, ma allo stesso tempo meno performante rispetto al secondo.

Per poter descrivere il secondo approccio di costruzione della matrice termini-documenti, si fa riferimento a due misure denominate **term frequency** e **inverse document frequency**.

- **term frequency**: è la frequenza di un termine all'interno di un documento. Molto spesso però una parola può essere utilizzata troppo spesso, e per questo motivo si potrebbe avere la necessità di ridurre l'influenza di tale parola, e per questo motivo si utilizza il sublinear **term frequency**, definito come il logaritmo del TF:
$$log(tf)$$
- **inverse document frequency**: tutti i task che fanno riferimento al natural language processing soffrono di un problema: tutti i termini sono considerati ugualmente importanti, ma si è sempre in presenza di termini che risultano essere più informativi rispetto ad altri. Ad esempio, tutti gli articoli che parlano di *business* potrebbero presentare molto spesso la parola *"denaro"*, che essendo presente troppe volte, avrebbe poco potere informativo. Per risolvere tale problematica si introduce un meccanismo per attenuare l'effetto dei termini che si verificano troppo spesso nella raccolta di documenti, che si chiama IDF, ed è definito come:
$$IDF = \log(\frac{N}{df_t})$$
Dove:
- **N**: è il numero di documenti nella collezione.
- df_t: è il numero di documenti che contengono il termine t.

Avendo definito **TF** e **IDF** è ora possibile introdurre il concetto di **TF-IDF**, dato da:

$$TFIDF = TF * IDF = \log(tf) * \log(\frac{N}{df_t})$$

Detto ciò può essere realizzata una matrice termini-documenti più sofisticata, dove vengono disposti i documenti sulle righe, e i termini sulle colonne. In questo caso, l'elemento in posizione (i, j) sarà:

- **0**: se il termine non è presente nel documento
- **TF-IDF**: il termine è presente nel documento

Step 3: Naive Bayes per la Classificazione Testuale

In questo step per i vari calcoli di probabilità ci si avvale della matrice termini-documenti descritta durante lo Step 2. Siano definiti un generico documento d ∈ D e una generica classe c ∈ C. La probabilità che la classe c faccia riferimento al documento d è definita dal teorema di Bayes:

$$P(c|d) = \frac{P(d|c)P(c)}{P(d)}$$

Dove:

- $P(d|c)$ è detta probabilità condizionata del documento d, data la classe c. Il valore di questa probabilità dipende dall'assunzione sul tipo di distribuzione che essa possiede.
  - **Bernoulli Naive Bayes**: da utilizzarsi quando la matrice termini-documenti è composta dai soli elementi 0 (termine assente nel documento) e 1 (termine presente nel documento). In questo caso, la probabilità condizionata del documento $d = (t_1, ..., t_m)$ data la classe c risulta seguire:
  $$p(d|c) = \prod_{i=1}^{m} P(t_i|c)^{x_i}(1 - P(t_i|c))^{(1-x_i)}$$
  Dove x_i è un valore che esprime presenza o assenza del termine t_i nel documento d.

– **Multinomial Naive Bayes**: da utilizzarsi quando la matrice termini-documenti contiene la frequenza dei termini nei documenti o alternativamente il TF-IDF. Sia x il vettore per il documento d. L'elemento *i-esimo* del vettore x, corrisponde alla frequenza (o al tf-idf) del termine t_i nel documento d. In questo caso la probabilità condizionata del documento $d = (t_1, ..., t_m)$ data la classe c risulta seguire:

$$P(d|c) = \frac{\Sigma_i x_i!}{\Pi_i x_i!} \prod_i P(t_i|c)^{x_i}$$

Il termine di normalizzazione $\frac{\Sigma_i x_i!}{\Pi_i x_i!}$ in questo caso non sarà utilizzzato perchè non dipende dalla classe. Pertanto si può dire che:

$$p(d|c) \propto \prod_i P(t_i|c)^{x_i}$$

A questo punto si può definire:

$$P(t_i|c) = \frac{count(t_i, c)}{\Sigma_{t \in V} count(t, c)}$$

Quindi, la probabilità che il documento d appartenga alla classe c può essere scritta come definito in precedenza.

- $P(c)$ è la probabilità a priori di c

$$P(c) = \frac{docCount(c)}{N_{doc}}$$

- $P(d)$ è la probabilità a priori di d

Al variare della classe c, si è interessati alla **MAP**, ovvero **"maximum a posteriori"**, cioè al:

$$argmax_{c \in C} P(c|d) = argmax_{c \in C} \frac{P(d|c)P(c)}{P(d)} \propto argmax_{c \in C} P(d|c)P(c) =$$

$$= argmax_{c \in C} P(t_1, t_2, ..., t_m|c)P(c)$$

Attenzione: valgono le seguenti assunzioni:

- Si assume che la posizione dei temrini non è importante
- Si assume che le probabilità $P(t_i|c_j)$ sono indipendenti data la classe c.

$$P(t_1, t_2, ..., t_m|c) = P(t_1|c) * ... * P(t_m|c)$$

Quindi la MAP può essere descritta come:

$$argmax_{c \in C} P(t_1, t_2, ..., t_m|c)P(c) = argmax_{c \in C} P(c) \prod_{t \in d} P(t|c)$$

Metodi di valutazione

Accuracy

L'accuratezza misura l'insieme di etichette predette da un campione rispetto alle etichette reali. Il valore dell'accuratezza varia nell'intervallo $[0, 1]$, dove 0 indica una scarsa accuratezza, ed 1 indica una buona accuratezza. Tale misura rappresenta quindi la percentuale di istanze correttamente classificate, ed è definita come:

$$\frac{\sum_i c_{ii}}{\sum_j \sum_i c_{ij}}$$

Matrice di confusione

Uno dei metodi più semplici e rappresentativi per valutare la bontà di un classificatore è la matrice di confusione. Essa mette in contrapposizione gli output reali con gli output predetti dal classificatore, in modo da poterne studiare la bontà. Un classificatore ideale dovrebbe presentare valori diversi da zero solo sulla diagnola principale della matrice. Nel caso bivariato la matrice di confusione avrebbe dimensione $2x2$, la matrice di confusione può essere interpretata come:

		Predicted Class	
		Normal	Attack
Actual Class	Normal	True Negative (TN)	False Positive (FP)
	Attack	False Negative (FN)	True Positive (TP)

Figure 1: Matrice di confusione 2x2

Dove:

- **TP (true positive)**: istanze classificate come "1" in modo corretto
- **FP (false positive)**: istanze classificate come "1" in modo errato
- **FN (false negative)**: istanze classificate come "0" in modo errato
- **TN (true negative)**: istanze classificate come "0" in modo corretto

Kappa Di Cohen

E' un coefficiente statistico che rappresenta il grado di accuratezza e affidabilità in una classificazione statistica.

$$K = \frac{Pr(a) - Pr(e)}{1 - Pr(e)}$$

Dove:

- $Pr(a)$: è data dal rapporto tra la somma degli elementi posti sulla diagonale principale della matrice di confusione e il numero totale di esempi. Rappresenta la percentuale di giudizio concorde.
- $Pr(e)$: nel caso di matrici di confusione bidimensionali è data da: $(\frac{TP+FP}{N} * \frac{TP+TN}{N}) + (\frac{TN+FN}{N} * \frac{FP+FN}{N})$ e rappresenta la probabilità di raggiungere casualmente un accordo.

I valori di questa statistica sono: $-1 \leq k \leq 1$, dove:

- Se $k < 0$: non c'è concordanza
- Se $k = 1$: concordanza perfetta

Ovviamente si preferiscono valori di k prossimi ad 1.

Cross Validation

La cross validation è una tecnica statistica utilizzabile in presenza di una buona numerosità del campione osservato, quindi, in presenza di un gran numero di documenti. In particolare, la corss validation consiste nella suddivisione dell'insieme di dati totale in k parti di uguale numerosità, e ad ogni passo, la k-esima parte viene utilizzata come insieme di convalida, mentre le restanti parti costituiscono il training set.

Così facendo si allena il modello per ognuna delle k parti, evitando problemi di *overfitting*. In altre parole, si suddivide il campione osservato in gruppi di egual numerosità, si esclude iterativamente un gruppo alla volta e si cerca di predirlo utilizzando i gruppi non esclusi, al fine di verificare la bontà del modello di predizione utilizzato.

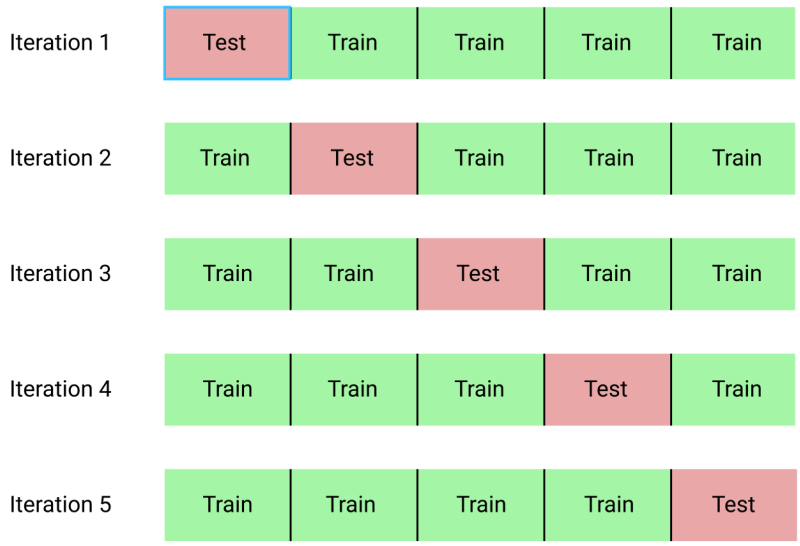


Figure 2: Cross Validation con 5 Folds

Realizzazione

Dopo aver affrontato Naive Bayes e la classificazione testuale dal punto di vista teorico è stato costruito un classificatore Naive Bayes utilizzando il linguaggio Python ed R. Per realizzare un progetto verosimile è stato preso in considerazione un dataset reale, descritto nel paragrafo successivo.

Descrizione del dataset

Il dataset utilizzato per la realizzazione di questo progetto è denominato *BBC News Classification*[4].

All'interno dell'archivio sono presenti i seguenti file:

- **BBC News Train.csv**: utilizzato per i task di apprendimento supervisionato. Il dataset è composto da 1490 istanze e 3 features:
  - **ArticleId**: identificativo dell'articolo
  - **Article**: testo dell'articolo
  - **Category**: categoria di appartenenza dell'articolo (*tech*, *business*, *sport*, *entertainment*, *politics*).
- **BBC News Test.csv**: da utilizzarsi per i task di apprendimento non supervisionato. Composto da 736 istanze e 2 features:
  - **ArticleId**: identificativo dell'articolo
  - **Article**: testo dell'articolo
- **BBC News Sample Solutions.csv**: contiene i risultati di classificazione.

Il dataset che più si addice agli scopi di questa analisi risulta essere *BBC News Train.csv*, pertanto tutti gli altri file saranno ignorati.

Python

Import delle librerie

Questo task di classificazione si è deciso di affrontarlo mediante l'utilizzo della libreria **sklearn**, che incorpora al suo interno gli strumenti per la costruzione dei classificatori Naive Bayes necessari. Sono state importate inoltre altre librerie come **pandas** e **numpy** per la creazione di DataFrame e Series e **pyplot** e **seaborn** per il plot di alcuni grafici.

Per pulizia nei successivi chunks di codice si importano tutte le librerie necessarie.

```
import pandas as pd
import numpy as np

from io import StringIO

from matplotlib import pyplot as plt
import seaborn as sns
from IPython.display import display

from sklearn.feature_selection import chi2
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import BernoulliNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import cross_val_score
```

[4]BBC News Classification: https://www.kaggle.com/c/learn-ai-bbc/data

```
from nltk.stem.porter import PorterStemmer
import nltk
import string
```

Caricamento in memoria del dataset

A questo punto si è pronti per caricare in memoria il dataset *BBC News Train.csv* utilizzando un DataFrame.

```
df = pd.read_csv('./BBC News Train.csv', sep=',')
```

Analisi esplorativa del dataset

Si è condotta una fase di analisi esplorativa del dataset al fine di comprenderne la natura e di eseguire una buona successiva fase di pre-processing.

E' stato conteggiato il numero di valori unici per *Category*, mostrata la dimensione dataset, controllato se vi sono valori nulli o duplicati. Di seguito vengono riportate le operazioni eseguite.

Si osservano i valori unici della feature "Category".

```
df.Category.unique()
```

```
## array(['business', 'tech', 'politics', 'sport', 'entertainment'],
##       dtype=object)
```

```
df.head()
```

```
##    ArticleId                                               Text  Category
## 0       1833  worldcom ex-boss launches defence lawyers defe...  business
## 1        154  german business confidence slides german busin...  business
## 2       1101  bbc poll indicates economic gloom citizens in ...  business
## 3       1976  lifestyle  governs mobile choice  faster  bett...      tech
## 4        917  enron bosses in $168m payout eighteen former e...  business
```

Si osserva la dimensione del dataset

```
df.shape
```

```
## (1490, 3)
```

Se il risultato del chunk seguente è *False*, allora non vi sono esempi con valori nulli in corrispondenza di ciascuna feature.

```
df.isnull().any().any()
```

```
## False
```

```
df.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 1490 entries, 0 to 1489
## Data columns (total 3 columns):
##  #   Column     Non-Null Count  Dtype
## ---  ------     --------------  -----
##  0   ArticleId  1490 non-null   int64
##  1   Text       1490 non-null   object
##  2   Category   1490 non-null   object
## dtypes: int64(1), object(2)
## memory usage: 35.0+ KB
```

Si analizzano poi i valori unici per ogni feature:

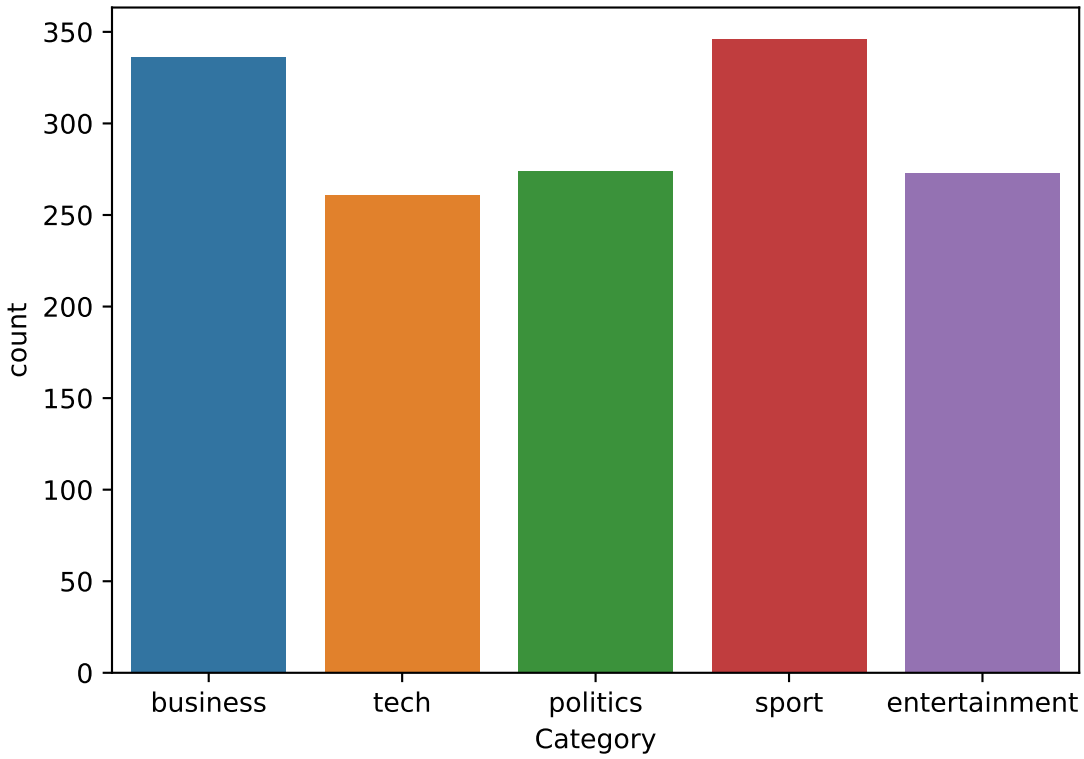
```
for col in df.columns.values:
    print(f"Il numero dei valori unici (non ripetuti) di {col} è: {df[col].nunique()}")
```

```
## Il numero dei valori unici (non ripetuti) di ArticleId è: 1490
## Il numero dei valori unici (non ripetuti) di Text è: 1440
## Il numero dei valori unici (non ripetuti) di Category è: 5
```

Dal precedente chunk si evince che vi sono valori di Text ripetuti più volte. Probabilmente lo stesso articolo è stato pubblicato più volte con un diverso identificativo. Per tale ragione, in fase di pre-processing (in seguito) si è deciso di rimuovere i duplicati per non peggiorare le performance del classificatore.

Si realizza un barplot per osservare graficamente la distribuzione degli esempi all'interno delle 5 diverse categorie.

```
import os
os.environ['QT_QPA_PLATFORM_PLUGIN_PATH'] = 'C:/Users/Gaede/Anaconda3/Library/plugins/platforms'

sns.countplot(data = df, x="Category")
plt.show()
```



Preprocessing

Successivamente è stata condotta una fase di pre-processing, al fine di preparare il dataset per il task di classificazione testuale. Le operazioni svolte durante questa fase sono state:

- Rimozione degli articoli duplicati scoperti durante l'analisi esplorativa

```
df.drop_duplicates(subset ="Text",
                     keep = "first", inplace = True)
df.shape
```

```
## (1440, 3)
```

- Rimozione della feature *ArticleId* poichè ritenuta inutile ai fini del task di classificazione testuale.

```
del df["ArticleId"]
```

- Aggiunta della feature *category_id*, ottenuta a partire dalla feature *Category* mediante l'applicazione di una tecnica conosciuta come *label encoding*.

```
df['category_id'] = df['Category'].factorize()[0]
df.head()
```

```
##                                                 Text  Category  category_id
## 0  worldcom ex-boss launches defence lawyers defe...  business            0
## 1  german business confidence slides german busin...  business            0
## 2  bbc poll indicates economic gloom citizens in ...  business            0
## 3  lifestyle  governs mobile choice  faster  bett...      tech            1
## 4  enron bosses in $168m payout eighteen former e...  business            0
```

Costruzione della matrice termini-documenti

Mediante il meotodo TfidfVectorizer messo a disposizione da Python, è stata realizzata la matrice termini-documenti *termDoc*. Questo metodo prende in input:

- **sublinear_tf**: impostato a *True* se si desidera il calcolo del sublinear TF
- **min_df**: numero minimo di documenti in cui un termine deve essere presente. In questo caso è stato impostato pari a 10.
- **norm**: settata a *l2* per assicurarsi che i vettori abbiano norma Euclidea.
- **encoding**: standard di codifica utilizzato
- **stop_words**: settato a *english* per consentire l'eliminazione delle paroli più comuni come 'a', 'the', 'that', ecc. Si sceglie di rimuovere questi termini perchè possiedono poco potere informativo, quindi risultano essere non discriminanti.
- **token_pattern**: regex che consente di scegliere quali termini prendere in considerazione. Si è scelta come regex, l'espressione "[A-Za-z][A-Za-z]+" per prendere in considerazione i soli termini che contengono caratteri alfebetici e composti da almeno due lettere.
- **tokenizer**: prende in input la funzione *tokenize* che esegue lo stemming dei termini.

```
stemmer = PorterStemmer()
trans_table = {ord(c): None for c in string.punctuation + string.digits}
def tokenize(text):
    tokens = [word for word in nltk.word_tokenize(text.translate(trans_table)) if len(word) > 2]
    stems = [stemmer.stem(item) for item in tokens]
    return stems

tfidf = TfidfVectorizer(sublinear_tf=True, min_df=10, norm='l2', encoding='latin-1',
  stop_words='english', token_pattern='[A-Za-z][A-Za-z]+', tokenizer=tokenize)
termDoc = tfidf.fit_transform(df.Text).toarray()
```

```
## C:\Users\Gaede\ANACON~1\lib\site-packages\sklearn\feature_extraction\text.py:386: UserWarning: Your s
##   'stop_words.' % sorted(inconsistent))
```

```
labels = df.category_id
print("Dimensione della matrice termini-documenti: ", termDoc.shape)
```

```
## Dimensione della matrice termini-documenti:  (1440, 3037)
```

Definizione del metodo executeNB()

Di seguito viene definito il metodo executeNB utilizzato per addestrare e valutare differenti classificatori bayesiani. Il metodo prende in input:

- **termDoc**: matrice termini-documenti
- **labels**: etichette delle classi
- **df**: dataFrame contenente il dataset
- **model**: modello da addestrare

Il metodo divide i dati a disposizione in:

- Train set: 70%
- Test set: 30% Attraverso il test set è possibile predire le classi di appartenenza dei documenti appartenenti a tale insieme, per poi confrontare i risultati ottenuti con quelli reali. In questo modo è possibile valutare la bontà del classificatore appena costruito mediante *Accuracy* e *Matrice di confusione* (descritte in precedenza).

Viene anche realizzato un *countplot* per osservare le differenze di classi predette e reali, e vengono mostrati a video anche gli errori commessi

```
def executeNB(termDoc, labels, df, model):
    # DataFrame che mantiene una corrispondenza 1:1 tra Category e category_id (senza duplicati)
    category_id_df = df[['Category', 'category_id']].drop_duplicates().sort_values('category_id')
    # Dizionario composto da coppie (Category, category_id)
    category_to_id = dict(category_id_df.values)
    # Dizionario composto da coppie (category_id, Category)
    id_to_category = dict(category_id_df[['category_id', 'Category']].values)

    model = model(alpha=1) #alpha = 1 per realizzare Laplace Smoothing
    X_train, X_test, y_train, y_test, indices_train, indices_test = train_test_split(
      termDoc, labels, df.index, test_size=0.30, random_state=1, shuffle=True)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    conf_mat = confusion_matrix(y_test, y_pred)
    fig, ax = plt.subplots(figsize=(10,10))
    sns.heatmap(conf_mat, annot=True, fmt='d', xticklabels=category_id_df.Category.values,
      yticklabels=category_id_df.Category.values)
    plt.ylabel('Actual')
    plt.xlabel('Predicted')
    plt.show()

    #Stampa del valore di accuracy
    from sklearn.metrics import accuracy_score
    print("Accuracy: ", accuracy_score(y_test,y_pred))

    #Creazione del countplot
    pred_plt = list()
    for n in y_pred:
        pred_plt.append(id_to_category[n])

    df_plot = pd.DataFrame({"value": pred_plt,
                            "Class": ["Predicted" for x in pred_plt]})
```

```
    test_plt = list()
    for n in y_test:
        test_plt.append(id_to_category[n])

    df_plot = df_plot.append(pd.DataFrame({"value": test_plt,
                            "Class": ["Real" for x in test_plt]}))

    sns.countplot(x="value", hue="Class", data = df_plot)
    plt.title("Predicted vs Real")
    plt.show()

    # Stampa a video degli errori di classificazione commessi
    for predicted in category_id_df.category_id:
        for actual in category_id_df.category_id:
            if predicted != actual and conf_mat[actual, predicted] >= 1:
                print("'{}' predicted as '{}' : {} examples.".format(id_to_category[actual],
                  id_to_category[predicted], conf_mat[actual, predicted]))
                display(df.loc[indices_test[(y_test == actual) &
                  (y_pred == predicted)]][['Category', 'Text']])
                print('')
```

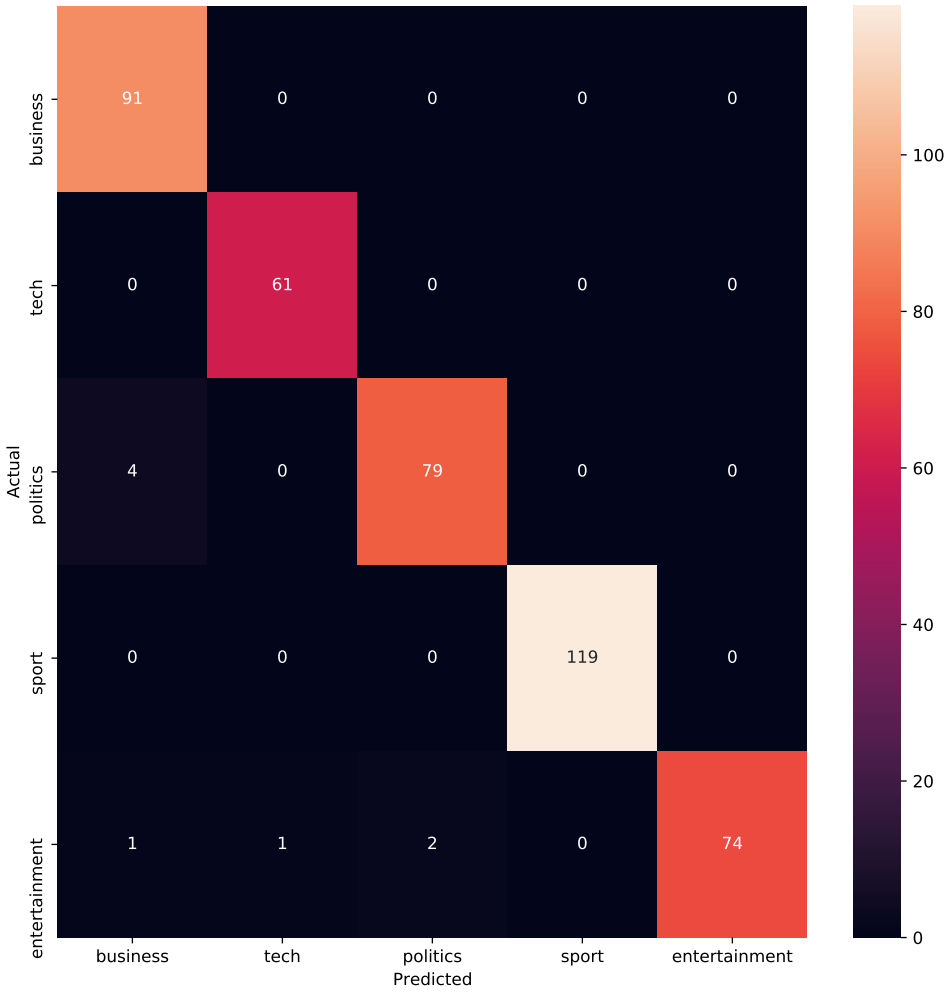
Costruzione del modello Multinomial Naive Bayes

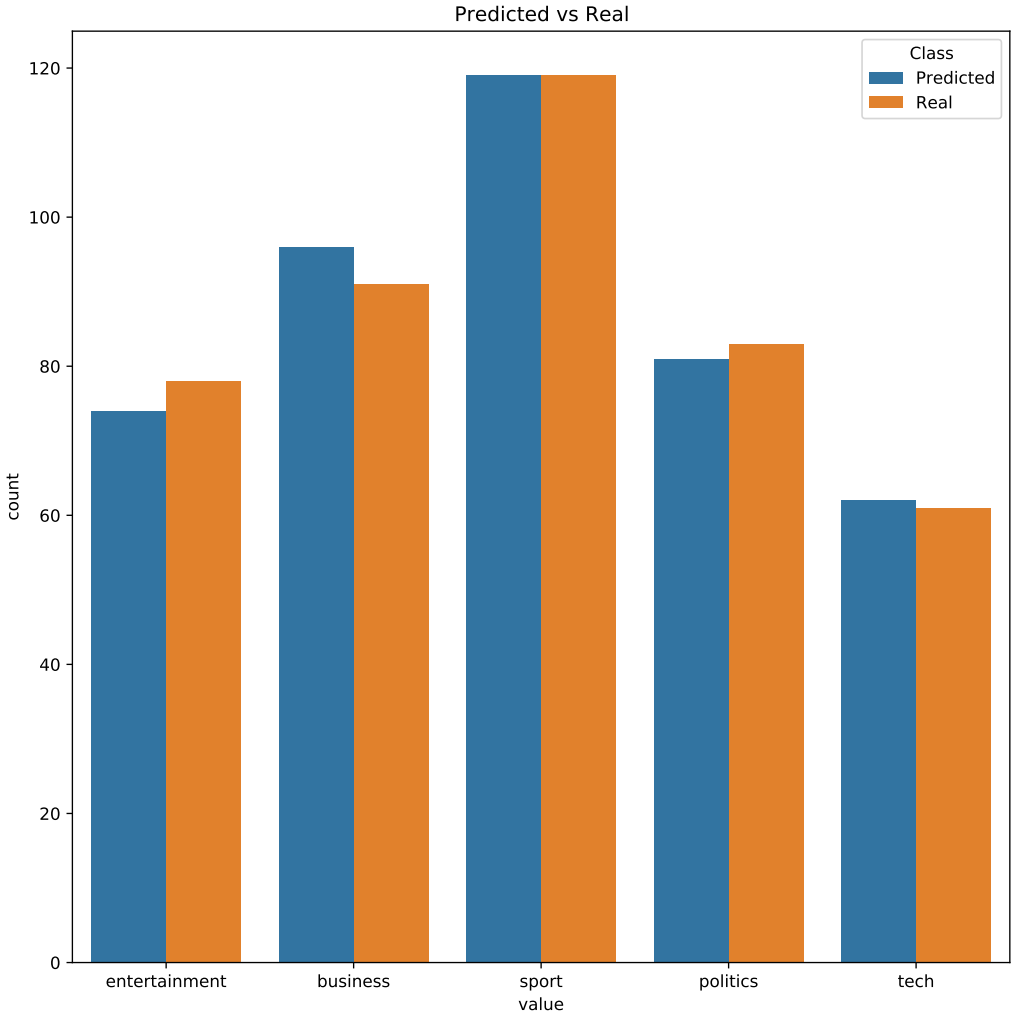
Disponendo di una matrice termini-documenti costruita secondo il criterio del TF-IDF, si ritiene che il modello più appropriato per questa applicazione sia il Multinomial Naive Bayes perchè le pesature TF-IDF seguono l'andamento di una distribuzione Multinomiale.

Si costruisce e valuta quindi il classificatore Naive Bayes Multinomiale con la matrice term-doc con pesatura TF-IDF:

```
executeNB(termDoc, labels, df, MultinomialNB)
```

```
## Accuracy:  0.9814814814814815
## 'politics' predicted as 'business' : 4 examples.
##       Category                                              Text
## 713   politics  straw praises kashmir moves the uk has welcome...
## 542   politics  hewitt decries  career sexism  plans to extend...
## 230   politics  brown calls for £5.5bn aids fund gordon brown ...
## 1339  politics  super union  merger plan touted two of britain...
##
## 'entertainment' predicted as 'business' : 1 examples.
##           Category                                          Text
## 450  entertainment  uk s national gallery in the pink the national...
##
## 'entertainment' predicted as 'tech' : 1 examples.
##           Category                                          Text
## 159  entertainment  uk tv channel rapped for csi ad tv channel fiv...
##
## 'entertainment' predicted as 'politics' : 2 examples.
##            Category                                           Text
## 1071  entertainment  bbc  should allow more scrutiny  mps have urge...
## 1292  entertainment  uganda bans vagina monologues uganda s authori...
```


91 0 0 0 0
0 61 0 0 0
4 0 79 0 0
0 0 0 119 0
1 1 2 0 74
business
tech
politics
sport
entertainment
Actual
business
tech
politics
sport
entertainment
Predicted
100
80
60
40
20
0



Costruzione del modello Bernoulli Naive Bayes

Oltre che il modello Naive Bayes Multinomiale, si è costruito un modello Naive Bayes Bernoulliano. Per fare questo però si è convertita la matrice termini-documenti con pesi TF-IDF in una normale matrice contenente fissati i e j:

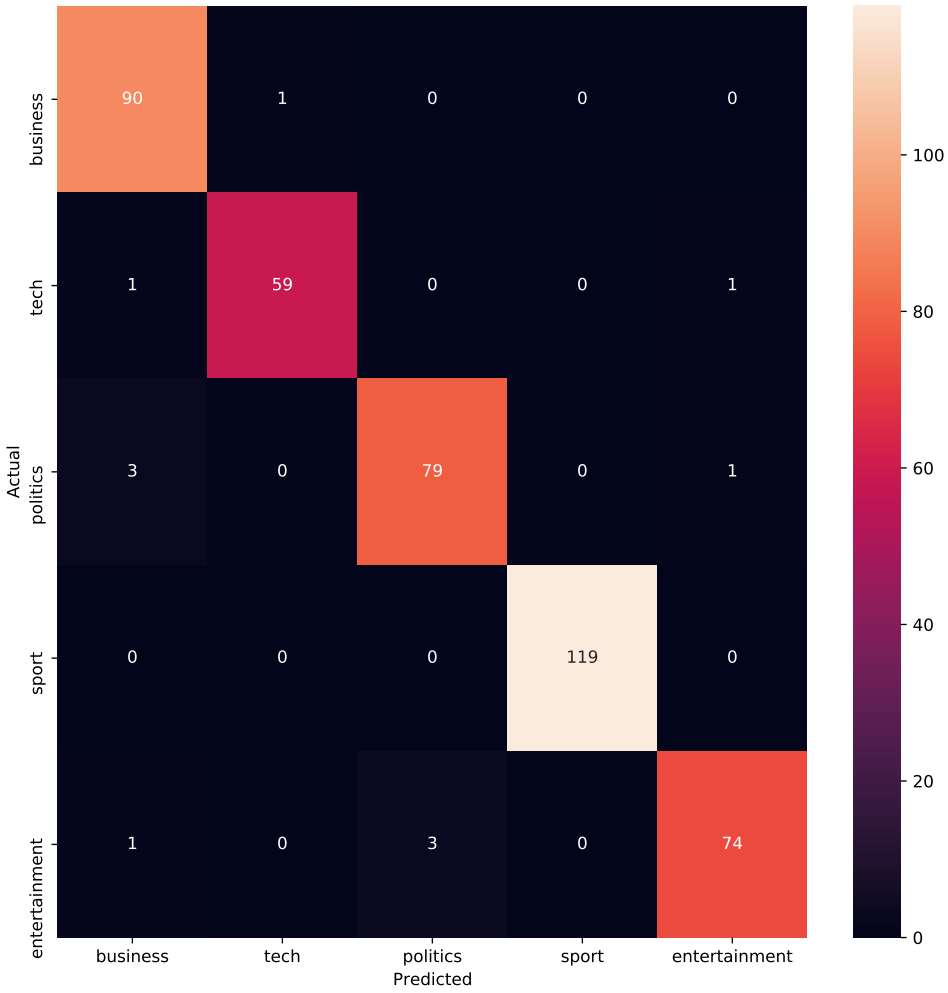
- 1 se il termine i-esimo è presente nel documento j-esimo
- 0 se il termine i-esimo nonè presente nel documento j-esimo

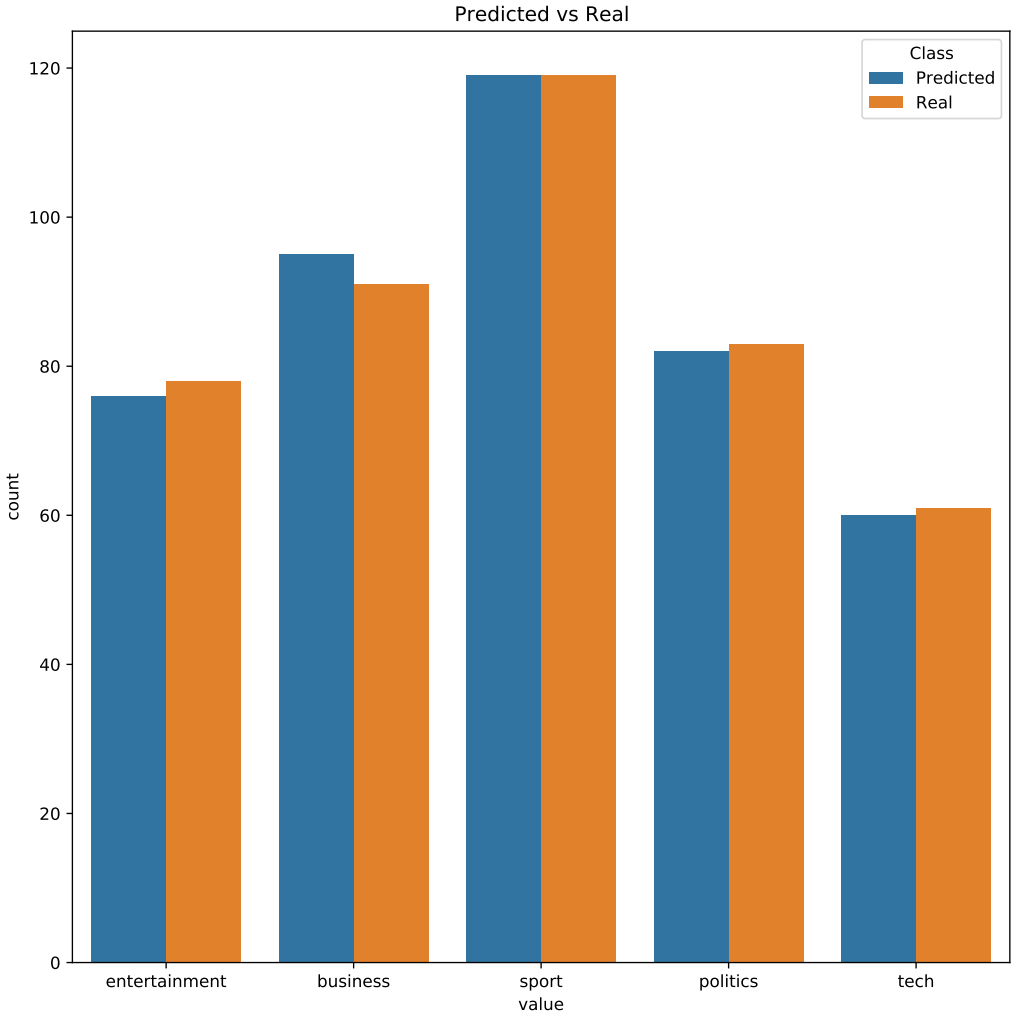
Questa matrice è versione meno precisa della precedente, ma risulterà utile per comparare i vari modelli.

```
# Si binarizza la matrice termDoc con pesature Tf-IDF
termDoc_bin = [np.where(x > 0, 1, 0) for x in termDoc]
executeNB(termDoc_bin, labels, df, BernoulliNB)
```

```
## Accuracy:  0.9745370370370371
## 'tech' predicted as 'business' : 1 examples.
```

```
##     Category                                                Text
## 339     tech  us duo in first spam conviction a brother and ...
##
## 'politics' predicted as 'business' : 3 examples.
##       Category                                                  Text
## 713   politics  straw praises kashmir moves the uk has welcome...
## 1339  politics  super union  merger plan touted two of britain...
## 1096  politics  crisis  ahead in social sciences  a national b...
##
## 'entertainment' predicted as 'business' : 1 examples.
##           Category                                                 Text
## 450  entertainment  uk s national gallery in the pink the national...
##
## 'business' predicted as 'tech' : 1 examples.
##      Category                                                  Text
## 131  business  call centre users  lose patience  customers tr...
##
## 'entertainment' predicted as 'politics' : 3 examples.
##            Category                                                 Text
## 1071  entertainment  bbc  should allow more scrutiny  mps have urge...
## 1292  entertainment  uganda bans vagina monologues uganda s authori...
## 264   entertainment  franz man seeks government help franz ferdinan...
##
## 'tech' predicted as 'entertainment' : 1 examples.
##    Category                                               Text
## 91     tech  2d metal slug offers retro fun like some drill...
##
## 'politics' predicted as 'entertainment' : 1 examples.
##       Category                                                  Text
## 1233  politics  bnp leader nick griffin arrested the leader of...
```


90
1
0
0
0
1
59
0
0
1
3
0
79
0
1
0
0
0
119
0
1
0
3
0
74
business
tech
politics
sport
entertainment
Actual
Predicted
100
80
60
40
20
0



Estensioni

Comparazione dei modelli

Di seguito viene definito un metodo atto al confronto di diversi modelli, ovvero:

- Bernoulli Naive Bayes
- Multinomial Naive Bayes
- Gaussian Naive Bayes

Quest'ultimo è stato introdotto per mostrare come potrebbero peggiorare le prestazioni del classificatore assumendo (erroneamente) che i dati abbiano una distribuzione Gaussiana. In questo caso infatti, ci si aspetta di ottenere prestazioni peggiori per il classificatore Gaussiano. Questo perchè non è detto che la distribuzione dei dati assunta dalla matrice term-doc pesata con tf-idf, segua un andamento Gaussiano.

Ogni classificatore viene valutato utilizzando la tecnica nota come *cross validation*, con un numero di *fold* pari a 5.

```python
def compareMethods(termDoc, labels):
    models = [
        BernoulliNB(),
        MultinomialNB(),
        GaussianNB(),
    ]

    CV = 5
    entries = []
    for model in models:
        model_name = model.__class__.__name__
        accuracies = cross_val_score(model, termDoc, labels, scoring='accuracy', cv=CV)
        for fold_idx, accuracy in enumerate(accuracies):
            entries.append((model_name, fold_idx, accuracy))
    cv_df = pd.DataFrame(entries, columns=['model_name', 'fold_idx', 'accuracy'])
    print(cv_df.groupby('model_name').accuracy.mean())
```

```python
print("Accuracy dei modelli con la matrice termDoc pesata con TF-IDF")
```

```
## Accuracy dei modelli con la matrice termDoc pesata con TF-IDF
```

```python
compareMethods(termDoc, labels)
```

```
## model_name
## BernoulliNB      0.961111
## GaussianNB       0.891667
## MultinomialNB    0.968056
## Name: accuracy, dtype: float64
```

```python
print("Accuracy dei modelli con la matrice termDoc binarizzata")
```

```
## Accuracy dei modelli con la matrice termDoc binarizzata
```

```python
compareMethods(termDoc_bin, labels)
```

```
## model_name
## BernoulliNB      0.961111
## GaussianNB       0.922917
## MultinomialNB    0.965278
## Name: accuracy, dtype: float64
```

Test χ^2

E' stato utilizzato un test χ^2 per trovare i termini che sono più correlati ad ogni categoria di documento. Nel seguente chunk di codice vengono calcoalte le statistiche chi-quadro tra ciascuna feature e la classe. Questo score può essere utilizzato per selezionare le n_features con i valori più alti per il test chi-quadro.

Il test proposto misura la dipendenza tra le variabili, quindi l'uso di questa funzione "elimina" le features che hanno maggiori probabilità di essere indipendenti dalla classe, e quindi irrilevanti per la classificazione.

Si ricorda che:

$$\chi^2 = \sum_{i=1}^{n} \frac{o_i e_i}{e_i}$$

Dove, fissato un indice i:

- o_i: valori osservati per l'i-esimo esempio
- e_i: valori attesi per l'i-esimo esempio

Il metodo prende in input la matrice *termDoc* e un vettore target y contenente l'id della classe per la quale si vuole eseguire il test.

Se le frequenze osservate coincidono con quelle teoriche, allora $\chi^2 = 0$, mentre se differiscono, $\chi^2 > 0$. Ovviamente più grande è il valore di χ^2, più grande è la discrepanza tra le frequenze osservate e quelle teoriche.

```
# DataFrame che mantiene una corrispondenza 1:1 tra Category e category_id (senza duplicati)
category_id_df = df[['Category', 'category_id']].drop_duplicates().sort_values('category_id')
# Dizionario composto da coppie (Category, category_id)
category_to_id = dict(category_id_df.values)
# Dizionario composto da coppie (category_id, Category)
id_to_category = dict(category_id_df[['category_id', 'Category']].values)

nTerms = 5
for category, category_id in sorted(category_to_id.items()):
    termDoc_chi2 = chi2(termDoc, labels == category_id)
    indices = np.argsort(termDoc_chi2[0])
    feature_names = np.array(tfidf.get_feature_names())[indices]
    print(f"Class '{category}': {', '.join(feature_names[-nTerms:])}")
```

```
## Class 'business': economi, oil, growth, profit, bank
## Class 'entertainment': album, award, actor, star, film
## Class 'politics': parti, blair, elect, tori, labour
## Class 'sport': injuri, coach, cup, match, champion
## Class 'tech': mobil, technolog, softwar, comput, user
```

Dai risultati sopra mostrati è possibile osservare quali sono i termini più correlati rispetto a ciascuna classe.

Classificazione Testuale non supervisionata

In precedenza si è trattato un caso di classificazione testuale supervisionato perchè si disponeva delle etichette di classe. Come ci si dovrebbe comportare se non si disponesse di queste informazioni? La risposta ovviamente, risiede nell'utilizzo di algoritmi di classificazione differenti, orientati all'apprendimento non supervisionato.

Successivamente viene rapidamente trattato un esempio di classificazione non supervisionata utilizzando l'algoritmo K-Means.

Si suppone di conoscere a priori il numero delle etichette di classe, pertanto si è lanciato l'algoritmo con n_clusters=5 (#classi = 5).

```
#Si addestra il classificatore KMeans
from sklearn.cluster import KMeans
from sklearn.metrics import accuracy_score

true_k = 5
labels = labels.values
X_train, X_test, y_train, y_test= train_test_split(
      termDoc, labels, test_size=0.30, random_state=0, shuffle=True)
model = KMeans(n_clusters=5, init='k-means++', max_iter=100, n_init=1)
model.fit(X_train)
```

```
## KMeans(max_iter=100, n_clusters=5, n_init=1)
```

```
y_pred = model.predict(X_test)
print('Accuracy: {}'.format(accuracy_score(y_test, y_pred)))
```

```
## Accuracy: 0.5763888888888888
```

```
order_centroids = model.cluster_centers_.argsort()[:, ::-1] #in ordine decrescente
terms = tfidf.get_feature_names()

#Si stampano a video i centroidi per ciascun cluster
for i in range(true_k):
    print("Cluster %d:" % i)
    for ind in order_centroids[i, :10]:
        print(" %s" % terms[ind], end=" ") #mostro i termini con tf-idf maggiore
    print("")
```

```
## Cluster 0:
##  said  ha  compani  market  year  firm  wa  share  sale  price
## Cluster 1:
##  hi  game  win  play  player  wa  champion  england  team  cup
## Cluster 2:
##  elect  labour  parti  blair  tori  govern  minist  said  say  plan
## Cluster 3:
##  use  peopl  technolog  user  servic  softwar  said  digit  phone  mobil
## Cluster 4:
##  film  award  star  wa  hi  music  actor  best  year  includ
```

Dai valori ottenuti, nonostante lo scarso valore di accuracy rispetto agli altri modelli visti, è possibile intuire abbastanza facilmente a quale classe ciascun cluster fa riferimento. E' probabile (dipende dall'esecuzione di K-Means) che i termini stampati a video, che corrispondo ad alcuni elementi del centroide, corrispondano a quelli generati dal test χ^2. Questo perchè χ^2 restituisce i termini più rilevanti per ciascuna classe, ed è molto probabile che tali termini possano appartenere al centroide, ovvero il miglior rappresentante per quel cluster.

Ovviamente l'algoritmo K-Means non è l'unico implementabile per questi scopi. In letteratura sono noti molti altri algoritmi, come E-M.

Risultati

Dai risultati ottenuti con l'esecuzione del metodo compareMethods si evince che:

- Il modello **Naive Bayes Multinomiale** risulta funzionare piuttosto bene con la matrice term-doc con pesatura tf-idf. Le sue prestazioni peggiorano nell'ordine dei millesimi con la matrice binarizzata, ma anche in questo caso si ritiene che i risultati siano convincenti, sebbene sarebbe pratica più corretta utilizzare un classificatore Naive Bayes Bernoulliano.
- Il modello **Naive Bayes Bernoulliano** risulta funzionare allo stesso modo con ambe le matrici. Questo perchè nel caso della matrice con pesatura TF-IDF, il classificatore binarizza automaticamente le feature se riceve in input dati non binari. Tuttavia in entrambi i casi le sue prestazioni risultano essere inferiori rispetto al classificatore Multinomiale.
- Il modello **Naive Bayes Gaussiano** risulta essere il peggiore tra quelli testati, anche se la sua accuratezza non è un risultato da disprezzare. Tale peggioramento è dovuto al fatto che in entrambi i casi, i dati non seguono una distribuzione di tipo Gaussiano.

In conclusione, si può affermare l'ottenimento di buoni risultati di accuracy per ciascuno dei modelli testati.

Il seguente progetto è stato esteso mediante la realizzazione di un test χ^2 per mostrare i termini più rilevanti per ogni categoria di documento. Il test scarta i termini che hanno maggiori probabilità di essere indipendenti dalla classe di appartenenza, e che risultano quindi poco informativi per il task di classificazione.

Successivamente è stato mostrato un esempio di algoritmo non supervisionato con lo stesso dataset privato dalle etichette di classe. L'algoritmo utilizzato è stato il K-Means, i cui risultati sono decisamente deludenti a causa dell'attività di apprendimento non supervisionata. Esistono tuttavia algoritmi probabilmente più performanti per l'estrazione di topic.

R

In questo paragrafo sono stati costruiti gli stessi classificatori visti in Python, utilizzando il linguaggio R.

L'analisi esplorativa non è stata ripetuta perchè ci si aspetta gli stessi risultati ottenuti in Python e per non rendere prolissa la documentazione.

Import delle librerie

```
library(tidyverse)
library(magrittr)
library(tm)
library(naivebayes)
library(gmodels)
require(caret)
```

Caricamento in memoria del dataset

Si è caricato in memoria il dataset *BBC News Train.csv* utilizzando un data.frame.

```
df<- read.csv('./BBC News Train.csv')
```

Preprocessing

Esattamente come nel paragrafo precedente è stata condotta la fase di pre-processing, al fine di preparare il dataset per il task di classificazione testuale. Anche in questo caso le operazioni svolte durante questa fase sono state:

- Rimozione degli articoli duplicati scoperti durante l'analisi esplorativa

```
df %>% distinct(Text, .keep_all = TRUE)
```

- Rimozione della feature *ArticleId* poichè ritenuta inutile ai fini del task di classificazione testuale.

```
df <- df[c("Text", "Category")]
```

- Aggiunta della feature *categoryId*, mediante l'applicazione di una tecnica conosciuta come *label encoding*, che trasforma i valori categorici di *Category* in vlori numerici mediante una codifica arbitraria.

```
df["categoryId"] <- as.integer(factor(df[["Category"]]))
```

Costruzione della matrice termini-documenti

Mediante la libreria **tm**, che implementa la classe **VCorpus** è stato realizzato un corpus testuale, oggetto di operazioni di pre-processing testuale per costruire in seguito la matrice term-doc. Le operazioni eseguite sul corpus sono state:

- Rimozione dei segni di punteggiatura
- Rimozione degli spazi in eccesso tra le varie parole
- Rimozione dei numeri
- Trasformazione di tutti i termini in minuscolo
- Rimozione delle stopwords e dei termini che rispettano le seguenti espressioni regolari:
  - $[0-9]+[A-Za-z]*$: rimozione dei termini composti da cifre e caratteri
  - $[A-Za-z]*[0-9]+$: rimozione dei termini composti da caratteri e cifre
  - $[A-Za-z]$: rimozione dei termini composti da un solo carattere
  - $[A-Za-z][A-Za-z]$: rimozione dei termini composti da due soli caratteri
- Stemming

Ecco la funzione realizzata in R che esegue quanto appena descritto:

```
create_corpus <- function(df)
{
  library(magrittr)
  library(tm)
  corpus = VCorpus(VectorSource(df$Text))
  corpus_clean = corpus %>%
    tm_map(removePunctuation) %>%
    tm_map(stripWhitespace) %>%
    tm_map(removeNumbers) %>%
    tm_map(content_transformer(tolower)) %>%
    tm_map(removeWords, c(stopwords(), '[0-9]+[A-Za-z]*', '[A-Za-z]*[0-9]+', '[A-Za-z]',
                          '[A-Za-z][A-Za-z]')) %>%
    tm_map(stemDocument)

  return(corpus_clean)
}

corpus_clean <- create_corpus(df)
```

Successivamente si è costruita la *DocumentTermMatrix* utilizzando il corpus appena processato e una pesatura *TF-IDF*. Inoltre sono stati rimossi i termini che compaiono in *meno di 10 documenti*.

```
dtm_tfidf <- DocumentTermMatrix(corpus_clean,
                          control = list(weighting = weightTfIdf, minDocFreq=10))
```

Per scopi di confronto dei due diversi classificatori viene realizzata anche la DocumentTerm Matrix utilizzando una pesatura *binarizzata*.

```
dtm_bin <- DocumentTermMatrix(corpus_clean,
                          control = list(weighting = weightBin, minDocFreq=10))
```

Definizione del metodo executeNB()

Di seguito viene definito il metodo executeNB utilizzato per addestrare e valutare differenti classificatori bayesiani. Il metodo prende in input:

- **dtMatrix**: matrice document-terms
- **df**: dataFrame contenente il dataset
- **model**: modello da addestrare

Il metodo splitta la dtMatrix in train e test set, prendendo in considerazione circa il 70% dei documenti per il train, e il restante 30% per il test.

Per la costruzione del classificatore è stato inoltre impostato il parametro *laplace=1* per utilizzare la tecnica nota come *Laplace Smoothing*.

Infine il metodo comprende una parte di valutazione dei risultati ottenuti dalla predizione, mostrando la matrice di confusione e alcune metriche come l'*Accuracy*.

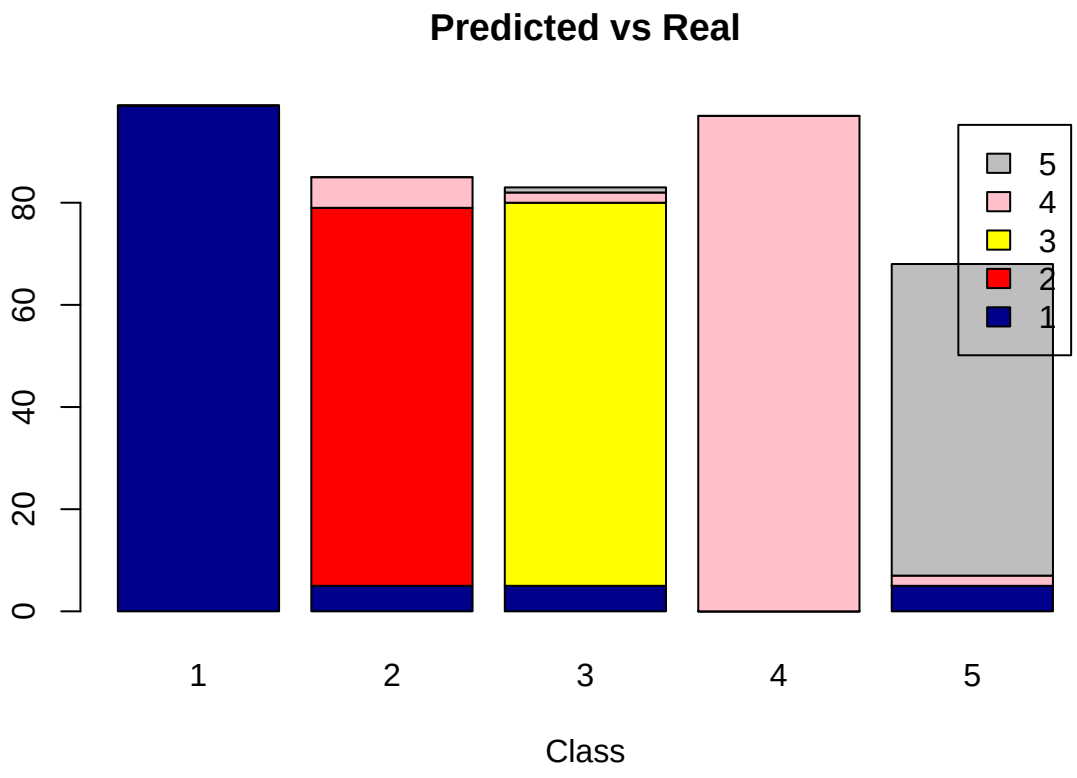
```
executeNB <- function(dtMatrix, df, model)
{
  #creazione train e test set
  dtm_train <- dtMatrix[1:1008, ]
  dtm_test <- dtMatrix[1009:1440, ]
  train_labels <- df[1:1008, ]$categoryId
  test_labels <- df[1009:1440, ]$categoryId
```

```
  library(naivebayes)
  classifier <- model(as.matrix(dtm_train), as.factor(train_labels), laplace = 1)
  pred <- predict(classifier, as.matrix(dtm_test))

  #evaluation
  counts <- table(pred, test_labels)
  barplot(counts, main="Predicted vs Real", xlab="Class", col=c("darkblue","red", "yellow", "pink", "gr

  library(gmodels)
  CrossTable(pred, test_labels, prop.chisq = FALSE, chisq = FALSE,
             prop.t = FALSE,
             dnn = c("Predicted", "Actual"))

  require(caret)
  confusionMatrix(table(pred, test_labels))
}
```

Costruzione del modello Multinomial Naive Bayes

A questo punto è possibile lanciare la funzione appena definita per costruire e valutare il modello Naive Bayes Multinomiale avvalendosi della *DocumentTermMatrix* con pesatura *TF-IDF*.

```
executeNB(dtMatrix = dtm_tfidf, df = df, model = multinomial_naive_bayes)
```




```
##
##
##    Cell Contents
```

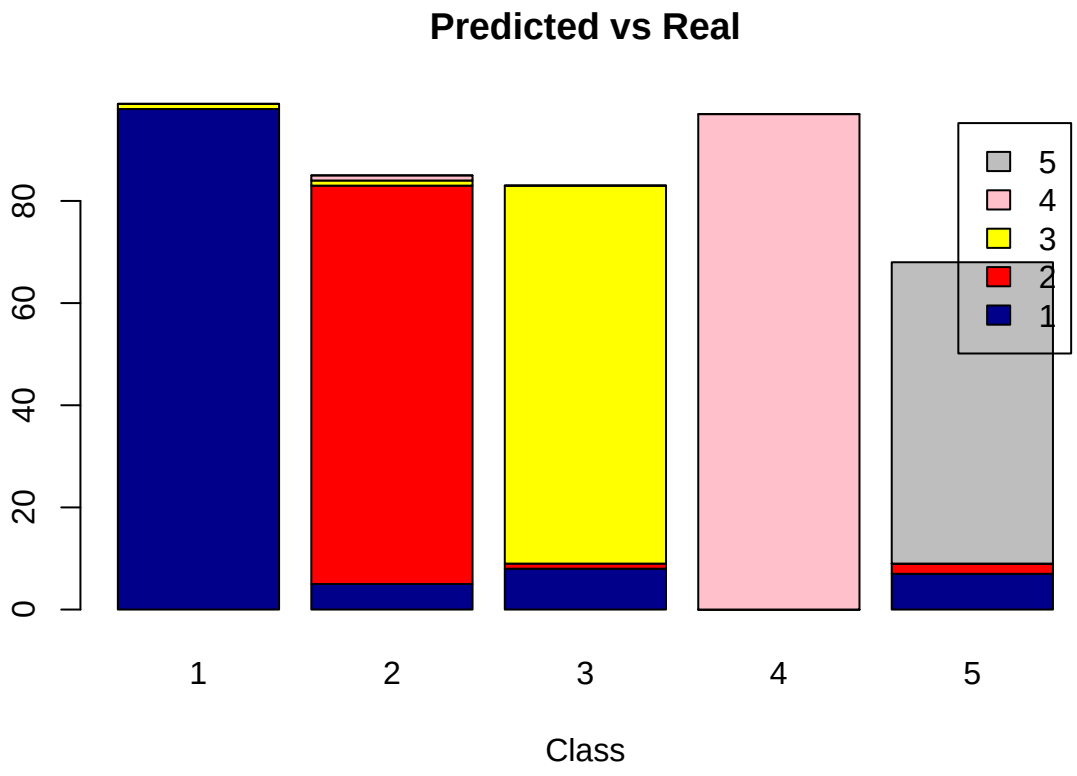
```
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |-------------------------|
##
##
## Total Observations in Table:  432
##
##
##              | Actual
##    Predicted |         1 |         2 |         3 |         4 |         5 | Row Total |
## -------------|-----------|-----------|-----------|-----------|-----------|-----------|
##            1 |        99 |         5 |         5 |         0 |         5 |       114 |
##              |     0.868 |     0.044 |     0.044 |     0.000 |     0.044 |     0.264 |
##              |     1.000 |     0.059 |     0.060 |     0.000 |     0.074 |           |
## -------------|-----------|-----------|-----------|-----------|-----------|-----------|
##            2 |         0 |        74 |         0 |         0 |         0 |        74 |
##              |     0.000 |     1.000 |     0.000 |     0.000 |     0.000 |     0.171 |
##              |     0.000 |     0.871 |     0.000 |     0.000 |     0.000 |           |
## -------------|-----------|-----------|-----------|-----------|-----------|-----------|
##            3 |         0 |         0 |        75 |         0 |         0 |        75 |
##              |     0.000 |     0.000 |     1.000 |     0.000 |     0.000 |     0.174 |
##              |     0.000 |     0.000 |     0.904 |     0.000 |     0.000 |           |
## -------------|-----------|-----------|-----------|-----------|-----------|-----------|
##            4 |         0 |         6 |         2 |        97 |         2 |       107 |
##              |     0.000 |     0.056 |     0.019 |     0.907 |     0.019 |     0.248 |
##              |     0.000 |     0.071 |     0.024 |     1.000 |     0.029 |           |
## -------------|-----------|-----------|-----------|-----------|-----------|-----------|
##            5 |         0 |         0 |         1 |         0 |        61 |        62 |
##              |     0.000 |     0.000 |     0.016 |     0.000 |     0.984 |     0.144 |
##              |     0.000 |     0.000 |     0.012 |     0.000 |     0.897 |           |
## -------------|-----------|-----------|-----------|-----------|-----------|-----------|
## Column Total |        99 |        85 |        83 |        97 |        68 |       432 |
##              |     0.229 |     0.197 |     0.192 |     0.225 |     0.157 |           |
## -------------|-----------|-----------|-----------|-----------|-----------|-----------|
##
##

## Confusion Matrix and Statistics
##
##     test_labels
## pred  1  2  3  4  5
##    1 99  5  5  0  5
##    2  0 74  0  0  0
##    3  0  0 75  0  0
##    4  0  6  2 97  2
##    5  0  0  1  0 61
##
## Overall Statistics
##
##                Accuracy : 0.9398
##                  95% CI : (0.9131, 0.9603)
##     No Information Rate : 0.2292
```

```
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9242
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            1.0000   0.8706   0.9036   1.0000   0.8971
## Specificity            0.9550   1.0000   1.0000   0.9701   0.9973
## Pos Pred Value         0.8684   1.0000   1.0000   0.9065   0.9839
## Neg Pred Value         1.0000   0.9693   0.9776   1.0000   0.9811
## Prevalence             0.2292   0.1968   0.1921   0.2245   0.1574
## Detection Rate         0.2292   0.1713   0.1736   0.2245   0.1412
## Detection Prevalence   0.2639   0.1713   0.1736   0.2477   0.1435
## Balanced Accuracy      0.9775   0.9353   0.9518   0.9851   0.9472
```

Costruzione del modello Bernoulli Naive Bayes

Viene lanciata la stessa funzione per costruire e valutare il modello Naive Bayes Bernoulliano, questa volta utilizzando la *DocumentTermMatrix* binarizzata.

```
executeNB(dtMatrix = dtm_bin, df = df, model = bernoulli_naive_bayes)
```




```
##
##
```

```
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |-------------------------|
##
##
## Total Observations in Table:  432
##
##
##              | Actual
##    Predicted |         1 |         2 |         3 |         4 |         5 | Row Total |
## -------------|-----------|-----------|-----------|-----------|-----------|-----------|
##            1 |        98 |         5 |         8 |         0 |         7 |       118 |
##              |     0.831 |     0.042 |     0.068 |     0.000 |     0.059 |     0.273 |
##              |     0.990 |     0.059 |     0.096 |     0.000 |     0.103 |           |
## -------------|-----------|-----------|-----------|-----------|-----------|-----------|
##            2 |         0 |        78 |         1 |         0 |         2 |        81 |
##              |     0.000 |     0.963 |     0.012 |     0.000 |     0.025 |     0.188 |
##              |     0.000 |     0.918 |     0.012 |     0.000 |     0.029 |           |
## -------------|-----------|-----------|-----------|-----------|-----------|-----------|
##            3 |         1 |         1 |        74 |         0 |         0 |        76 |
##              |     0.013 |     0.013 |     0.974 |     0.000 |     0.000 |     0.176 |
##              |     0.010 |     0.012 |     0.892 |     0.000 |     0.000 |           |
## -------------|-----------|-----------|-----------|-----------|-----------|-----------|
##            4 |         0 |         1 |         0 |        97 |         0 |        98 |
##              |     0.000 |     0.010 |     0.000 |     0.990 |     0.000 |     0.227 |
##              |     0.000 |     0.012 |     0.000 |     1.000 |     0.000 |           |
## -------------|-----------|-----------|-----------|-----------|-----------|-----------|
##            5 |         0 |         0 |         0 |         0 |        59 |        59 |
##              |     0.000 |     0.000 |     0.000 |     0.000 |     1.000 |     0.137 |
##              |     0.000 |     0.000 |     0.000 |     0.000 |     0.868 |           |
## -------------|-----------|-----------|-----------|-----------|-----------|-----------|
## Column Total |        99 |        85 |        83 |        97 |        68 |       432 |
##              |     0.229 |     0.197 |     0.192 |     0.225 |     0.157 |           |
## -------------|-----------|-----------|-----------|-----------|-----------|-----------|
##
##

## Confusion Matrix and Statistics
##
##     test_labels
## pred  1  2  3  4  5
##    1 98  5  8  0  7
##    2  0 78  1  0  2
##    3  1  1 74  0  0
##    4  0  1  0 97  0
##    5  0  0  0  0 59
##
## Overall Statistics
##
##                Accuracy : 0.9398
##                  95% CI : (0.9131, 0.9603)
```

```
##     No Information Rate : 0.2292
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9242
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.9899   0.9176   0.8916   1.0000   0.8676
## Specificity            0.9399   0.9914   0.9943   0.9970   1.0000
## Pos Pred Value         0.8305   0.9630   0.9737   0.9898   1.0000
## Neg Pred Value         0.9968   0.9801   0.9747   1.0000   0.9759
## Prevalence             0.2292   0.1968   0.1921   0.2245   0.1574
## Detection Rate         0.2269   0.1806   0.1713   0.2245   0.1366
## Detection Prevalence   0.2731   0.1875   0.1759   0.2269   0.1366
## Balanced Accuracy      0.9649   0.9545   0.9429   0.9985   0.9338
```

Per una migliore interpretazione della matrice di confusione mostrata precedentemente, si tenga a mente la corrispondenza tra le variabili categoriche e le variabili numeriche:

```
print(paste0("1: ", unique(df$Category[df$categoryId == 1])))
```

```
## [1] "1: business"
```

```
print(paste0("2: ", unique(df$Category[df$categoryId == 2])))
```

```
## [1] "2: entertainment"
```

```
print(paste0("3: ", unique(df$Category[df$categoryId == 3])))
```

```
## [1] "3: politics"
```

```
print(paste0("4: ", unique(df$Category[df$categoryId == 4])))
```

```
## [1] "4: sport"
```

```
print(paste0("5: ", unique(df$Category[df$categoryId == 5])))
```

```
## [1] "5: tech"
```

Risultati

Dopo la valutazione dei due modelli, si evince che si ottengono dei valori di accuracy equivalenti per il classificatore Bernoulliano con matrice binarizzata e il classificatore multinomiale con la matrice pesata mediante tf-idf. Attraverso le matrici di confusione è possibile osservare gli errori di predizione commessi, che risultano essere differenti tra i due classificatori, anche se gli score di accuracy si equivalgono. Tuttavia si può affermare che i risultati ottenuti sono del tutto soddisfacenti e in entrambi i casi si è costruito un modello eccellente per la classificazione testuale.

Python vs R

Sia in Python che in R si sono costruiti dei modelli piuttosto accurati. In entrambi i casi si è cercato di eseguire le stesse operazioni di pre-processing testuale, in modo da partire da una stessa matrice termini-documenti, con la speranza di giungere agli stessi risultati. In entrambi i casi si sono costruiti classificatori Bayesiani a seconda della pesatura utilizzata nella matrice termini-documenti. Tuttavia l'accuracy ottenuta in R risulta essere inferiore di qualche decimo, fenomeno causato probabilmente dal diverso pre-processing testuale eseguito dai due linguaggi o dalla differente implementazione dei classificatori all'interno delle librerie.