In [1]:
```
# Estos dos comandos evitan que haya que hacer reload cada vez que se modifica un paquete
%load_ext autoreload
%autoreload 2
```

In [2]:
```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
import pandas as pd
import matplotlib as plt
from matplotlib import pyplot as plot
from scipy import stats
import numpy as np
import numpy.ma as ma
import csv
from scipy.stats import norm
from sklearn.naive_bayes import GaussianNB
from scipy import stats
```

Ejercicio de clasificación de texto

Naive Bayes es una técnica estadística que consiste en repetir el método anterior en problemas cuyos sucesos no son independientes, pero suponiendo independencia. A lo largo de este trabajo desarrollarán un modelo de Naive Bayes para el problema de clasificación de artículos periodístios.En este caso podemos estimar la probabilidad de ocurrencia de cada palabra según la categoría a la que pertenece el artículo.

Dataset

El primer paso es obtener el dataset que vamos a utilizar. El dataset a utilizar es el de TwentyNewsGroup(TNG) que está disponible en sklearn.

Se puede encontrar más información del dataset en la documentación de scikit-learn.

In [3]:
```
#Loading the data set - training data.
from sklearn.datasets import fetch_20newsgroups
twenty_train = fetch_20newsgroups(subset='train', shuffle=True)
twenty_test = fetch_20newsgroups(subset='test', shuffle=True)
```

El siguiente paso es analizar el contenido del dataset, como por ejemplo la cantidad de artículos, la cantidad de clases, etc.

Preguntas:

1) Cuántos articulos tiene el dataset?

2) Cuántas clases tiene el dataset?

3) Es un dataset balanceado?

4) Cuál es la probabilidad a priori de la clase 5? A que corresponde esta clase?

5) Cuál es la clase con mayor probabilidad a priori?

1) Cuántos articulos tiene el dataset?

In [4]:
```
print('Cantidad de articulos')
numArticulos=len(twenty_train['data'])
print(numArticulos)
```

```
Cantidad de articulos
11314
```

2) Cuántas clases tiene el dataset?

In [5]:
```
print('Cantidad de clases')
numClases=len(twenty_train['target_names'])
print(numClases)
```

```
Cantidad de clases
20
```

3) Es un dataset balanceado?

In [6]:
```
cantidadPorClases=np.zeros(numClases)
for i in range(numClases):
    cantidadPorClases[i]=len(twenty_train['target'][twenty_train['target']==i
])
```

In [7]:
```
indexMin=np.argmin(cantidadPorClases)
indexMax=np.argmax(cantidadPorClases)

if(cantidadPorClases[indexMin]==cantidadPorClases[indexMax]):
    print('si')
else:
    print('no')
```

```
no
```

Cuál es la probabilidad a priori de la clase 5? A que corresponde esta clase?

In [8]:
```
probApriori5=cantidadPorClases[5]/numArticulos
print('La clase correspondiente a 5 es:')
print(twenty_train['target_names'][5])
print('Su probavilidad a priori es '+str(probApriori5))
```

```
La clase correspondiente a 5 es:
comp.windows.x
Su probavilidad a priori es 0.05241293972070002
```

5) Cuál es la clase con mayor probabilidad a priori?

In [22]:
```
print('La maxima proba a priori correponde a la clase:' + twenty_train['target
_names'][indexMax]+' '+str(indexMax))
```

```
La maxima proba a priori correponde a la clase:rec.sport.hockey 10
```

Preprocesamiento

Para facilitar la comprensión de los algoritmos de preprocesamiento, se aplican primero a un solo artículo.

Mas info en: http://text-processing.com/demo/stem/ (http://text-processing.com/demo/stem/)

- **Tokenization (nltk):**

Dada una secuencia de caracteres, la tokenización consiste en dividir esta subpartes denominadas tokens. Un token puede ser palabras individuales, sílabas, frases o cualquier combinación de ellos.

In [76]:
```
import nltk
from nltk.tokenize import word_tokenize
nltk.download('punkt')
tok = word_tokenize(twenty_train['data'][0])
print(tok)
```

```
['From', ':', 'lerxst', '@', 'wam.umd.edu', '(', 'where', "'s", 'my', 'thin
g', ')', 'Subject', ':', 'WHAT', 'car', 'is', 'this', '!', '?', 'Nntp-Posting
-Host', ':', 'rac3.wam.umd.edu', 'Organization', ':', 'University', 'of', 'Ma
ryland', ',', 'College', 'Park', 'Lines', ':', '15', 'I', 'was', 'wondering',
'if', 'anyone', 'out', 'there', 'could', 'enlighten', 'me', 'on', 'this', 'ca
r', 'I', 'saw', 'the', 'other', 'day', '.', 'It', 'was', 'a', '2-door', 'spor
ts', 'car', ',', 'looked', 'to', 'be', 'from', 'the', 'late', '60s/', 'earl
y', '70s', '.', 'It', 'was', 'called', 'a', 'Bricklin', '.', 'The', 'doors',
'were', 'really', 'small', '.', 'In', 'addition', ',', 'the', 'front', 'bumpe
r', 'was', 'separate', 'from', 'the', 'rest', 'of', 'the', 'body', '.', 'Thi
s', 'is', 'all', 'I', 'know', '.', 'If', 'anyone', 'can', 'tellme', 'a', 'mod
el', 'name', ',', 'engine', 'specs', ',', 'years', 'of', 'production', ',',
'where', 'this', 'car', 'is', 'made', ',', 'history', ',', 'or', 'whatever',
'info', 'you', 'have', 'on', 'this', 'funky', 'looking', 'car', ',', 'pleas
e', 'e-mail', '.', 'Thanks', ',', '-', 'IL', '--', '--', 'brought', 'to', 'yo
u', 'by', 'your', 'neighborhood', 'Lerxst', '--', '--']

[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\mpier\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

- **Lemmatization (nltk):**

Consiste en llevar a las distintas conjugaciones de una palabra a su origen.

In [77]:
```
from nltk.stem import WordNetLemmatizer
nltk.download('wordnet')
lemmatizer = WordNetLemmatizer()
lem=[lemmatizer.lemmatize(x,pos='v') for x in tok]
print(lem)
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\mpier\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!

['From', ':', 'lerxst', '@', 'wam.umd.edu', '(', 'where', "'s", 'my', 'thin
g', ')', 'Subject', ':', 'WHAT', 'car', 'be', 'this', '!', '?', 'Nntp-Posting
-Host', ':', 'rac3.wam.umd.edu', 'Organization', ':', 'University', 'of', 'Ma
ryland', ',', 'College', 'Park', 'Lines', ':', '15', 'I', 'be', 'wonder', 'i
f', 'anyone', 'out', 'there', 'could', 'enlighten', 'me', 'on', 'this', 'ca
r', 'I', 'saw', 'the', 'other', 'day', '.', 'It', 'be', 'a', '2-door', 'spor
t', 'car', ',', 'look', 'to', 'be', 'from', 'the', 'late', '60s/', 'early',
'70s', '.', 'It', 'be', 'call', 'a', 'Bricklin', '.', 'The', 'doors', 'be',
'really', 'small', '.', 'In', 'addition', ',', 'the', 'front', 'bumper', 'b
e', 'separate', 'from', 'the', 'rest', 'of', 'the', 'body', '.', 'This', 'b
e', 'all', 'I', 'know', '.', 'If', 'anyone', 'can', 'tellme', 'a', 'model',
'name', ',', 'engine', 'specs', ',', 'years', 'of', 'production', ',', 'wher
e', 'this', 'car', 'be', 'make', ',', 'history', ',', 'or', 'whatever', 'inf
o', 'you', 'have', 'on', 'this', 'funky', 'look', 'car', ',', 'please', 'e-ma
il', '.', 'Thanks', ',', '-', 'IL', '--', '--', 'bring', 'to', 'you', 'by',
'your', 'neighborhood', 'Lerxst', '--', '--']
```

- **Stop Words (nltk):**

Uno de los mayores objetivos del pre procesamiento es remover los datos que no aportan información. Las palabras que no aportan información suelen llamarse Stop Words. Estos pueden ser palabras como un, una, la, etc.

In [78]:
```
from nltk. corpus import stopwords
nltk.download('stopwords')
stop=[x for x in lem if x not in stopwords.words('english')]
print(stop)
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\mpier\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!

['From', ':', 'lerxst', '@', 'wam.umd.edu', '(', "'s", 'thing', ')', 'Subjec
t', ':', 'WHAT', 'car', '!', '?', 'Nntp-Posting-Host', ':', 'rac3.wam.umd.ed
u', 'Organization', ':', 'University', 'Maryland', ',', 'College', 'Park', 'L
ines', ':', '15', 'I', 'wonder', 'anyone', 'could', 'enlighten', 'car', 'I',
'saw', 'day', '.', 'It', '2-door', 'sport', 'car', ',', 'look', 'late', '60
s/', 'early', '70s', '.', 'It', 'call', 'Bricklin', '.', 'The', 'doors', 'rea
lly', 'small', '.', 'In', 'addition', ',', 'front', 'bumper', 'separate', 're
st', 'body', '.', 'This', 'I', 'know', '.', 'If', 'anyone', 'tellme', 'mode
l', 'name', ',', 'engine', 'specs', ',', 'years', 'production', ',', 'car',
'make', ',', 'history', ',', 'whatever', 'info', 'funky', 'look', 'car', ',',
'please', 'e-mail', '.', 'Thanks', ',', '-', 'IL', '--', '--', 'bring', 'neig
hborhood', 'Lerxst', '--', '--']
```

- **Stemming (nltk):**

Es un método para reducir la palabra a su raíz. El algoritmo más usado es el algoritmo de Porter.

In [79]:
```
from nltk.stem import PorterStemmer

stemmer=PorterStemmer()
stem=[stemmer.stem(x) for x in stop]
print(stem)
```

```
['from', ':', 'lerxst', '@', 'wam.umd.edu', '(', "'s", 'thing', ')', 'subjec
t', ':', 'what', 'car', '!', '?', 'nntp-posting-host', ':', 'rac3.wam.umd.ed
u', 'organ', ':', 'univers', 'maryland', ',', 'colleg', 'park', 'line', ':',
'15', 'I', 'wonder', 'anyon', 'could', 'enlighten', 'car', 'I', 'saw', 'day',
'.', 'It', '2-door', 'sport', 'car', ',', 'look', 'late', '60s/', 'earli', '7
0', '.', 'It', 'call', 'bricklin', '.', 'the', 'door', 'realli', 'small',
'.', 'In', 'addit', ',', 'front', 'bumper', 'separ', 'rest', 'bodi', '.', 'th
i', 'I', 'know', '.', 'If', 'anyon', 'tellm', 'model', 'name', ',', 'engin',
'spec', ',', 'year', 'product', ',', 'car', 'make', ',', 'histori', ',', 'wha
tev', 'info', 'funki', 'look', 'car', ',', 'pleas', 'e-mail', '.', 'thank',
',', '-', 'IL', '--', '--', 'bring', 'neighborhood', 'lerxst', '--', '--']
```

- **Filtrado de palabras:**

Removemos todo lo que no sean palabras.

In [80]:
```
alpha=[x for x in stem if x.isalpha()]
print(alpha)
```

```
['from', 'lerxst', 'thing', 'subject', 'what', 'car', 'organ', 'univers', 'ma
ryland', 'colleg', 'park', 'line', 'I', 'wonder', 'anyon', 'could', 'enlighte
n', 'car', 'I', 'saw', 'day', 'It', 'sport', 'car', 'look', 'late', 'earli',
'It', 'call', 'bricklin', 'the', 'door', 'realli', 'small', 'In', 'addit', 'f
ront', 'bumper', 'separ', 'rest', 'bodi', 'thi', 'I', 'know', 'If', 'anyon',
'tellm', 'model', 'name', 'engin', 'spec', 'year', 'product', 'car', 'make',
'histori', 'whatev', 'info', 'funki', 'look', 'car', 'pleas', 'thank', 'IL',
'bring', 'neighborhood', 'lerxst']
```

Preprocesamiento completo

Utilizar o no cada uno de los métodos vistos es una decisión que dependerá del caso particular de aplicación. Para este ejercicio vamos a considerar las siguientes combinaciones:

- Tokenización
- Tokenización, Lematización, Stemming.
- Tokenización, Stop Words.
- Tokenización, Lematización, Stop Words, Stemming.
- Tokenización, Lematización, Stop Words, Stemming, Filtrado.

Tokenización

Armo un vector con cada articulo tokenizado

In [18]:
```
import nltk
from nltk.tokenize import word_tokenize

nltk.download('punkt')
token =[word_tokenize(x) for x in twenty_train['data']]
test_token =[word_tokenize(x) for x in twenty_test['data']]
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\mpier\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

In [82]:
```

```

Tokenización, Lematización, Stemming.

In [19]:
```
from nltk.stem import WordNetLemmatizer
nltk.download('wordnet')
lemmatizer = WordNetLemmatizer()
tokenLem=[[lemmatizer.lemmatize(x,pos='v') for x in y] for y in token ]
test_tokenLem=[[lemmatizer.lemmatize(x,pos='v') for x in y] for y in test_toke
n ]
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\mpier\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

In [20]:
```
from nltk.stem import PorterStemmer

stemmer=PorterStemmer()
tokenLemStem=[[stemmer.stem(x) for x in y] for y in tokenLem]
test_tokenLemStem=[[stemmer.stem(x) for x in y] for y in test_tokenLem]
```

Tokenización, Stop Words.

In [21]:
```
from nltk. corpus import stopwords
nltk.download('stopwords')
tokenStop=[[x for x in y if x not in stopwords.words('english')] for y in toke
n]
test_tokenStop=[[x for x in y if x not in stopwords.words('english')] for y in
test_token]
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\mpier\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Tokenización, Lematización, Stop Words, Stemming.

In [22]:
```
from nltk.stem import WordNetLemmatizer
nltk.download('wordnet')
lemmatizer = WordNetLemmatizer()
tokenStopLem=[[lemmatizer.lemmatize(x,pos='v') for x in y] for y in tokenStop
]
test_tokenStopLem=[[lemmatizer.lemmatize(x,pos='v') for x in y] for y in test_
tokenStop ]
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\mpier\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

In [23]:
```
from nltk.stem import PorterStemmer

stemmer=PorterStemmer()
tokenStopLemStem=[[stemmer.stem(x) for x in y] for y in tokenStopLem]
test_tokenStopLemStem=[[stemmer.stem(x) for x in y] for y in test_tokenStopLem
]
```

Tokenización, Lematización, Stop Words, Stemming, Filtrado.

In [24]:
```
tokenStopLemStemAlpha=[[x for x in y if x.isalpha()] for y in tokenStopLemStem
]
test_tokenStopLemStemAlpha=[[x for x in y if x.isalpha()] for y in test_tokenS
topLemStem ]
```

Vector de datos

In [25]:
```
datosProcesados=[token,tokenLemStem,tokenStop,tokenStopLemStem,tokenStopLemSte
mAlpha]
test_datosProcesados=[test_token,test_tokenLemStem,test_tokenStop,test_tokenSt
opLemStem,test_tokenStopLemStemAlpha]
```

In [26]:
```
tosave=[datosProcesados,test_datosProcesados]
```

Calculo de longitudes promedio

In [15]:
```
def lenPromedio(x):
    #calculo el pormedio de plabaras
    temp=0
    for i in x:
        temp=temp+len(set(i))
    return(temp/len(x))
```

In [16]:
```
for i in range(len(itemlist[0])):
    print(lenPromedio(itemlist[0][i]))
```

```
180.95837016086264
164.87661304578398
149.42805373873077
139.27320134346826
102.65396853455896
```

Preguntas

- Cómo cambia el tamaño del vocabulario al agregar Lematización y Stemming?
- Cómo cambia el tamaño del vocabulario al Stop Words?
- Analice muy brevemente ventajas y desventajas del tamaño del dataset en cada caso.

Guardado de pre procesamiento

Vamos a guardar lo preprocesado usando pickle, que nos permite serializar objetos y guardarlos en disco, es muy importante que sepan hacer esto si no quieren perder tiempo!

In [27]:
```
#Salvado del procesamiento a disco:
'''
import pickle

with open('art_filt.pck', 'wb') as fp:
    pickle.dump(tosave, fp)
'''
```

In [11]:
```
import pickle
with open ('art_filt.pck', 'rb') as fp:
    itemlist = pickle.load(fp)
```

Vectorización de texto

- **Obtención del vocabulario y obtención de la probabilidad**

Como se vió en clase, los vectorizadores cuentan con dos parámetros de ajuste.

- max_df: le asignamos una maxima frecuencia de aparición, eliminando las palabras comunes que no aportan información.
- min_df: le asignamos la minima cantidad de veces que tiene que aparecer una palabra.

Al igual que con las diferentes opciones de preprocesamiento, lo mismo ocurre con la vectorización. Podemos utilizar CountVectorizer o TfidfVectorizer según el caso (con diferentes valores de max_df y min_df). Para este ejercicio deben utilizar ambos métodos.

In [17]:
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [13]:
```
#junto los arreglos de strings a un parrafo para cada combinacion

dataToProces=[[' '.join(y)  for y in x ] for x in itemlist[0]]
test_dataToProces=[[' '.join(y)  for y in x ] for x in itemlist[1]]
```

In [ ]:
```
raw_data.toarray() #Es una sparse matrix, vamos a expandirla
```

In [ ]:
```
raw_data.toarray()[0,:].argmax() #Veamos a qué palabra pertenece la máxima ocu
rrencia en el primer artículo
```

Entrenamiento del modelo

Primero deben separar correctamente el dataset para hacer validación del modelo.

Y luego deben entrenar el modelo de NaiveBayes con el dataset de train.

Deben utilizar un modelo de NaiveBayes Multinomial y de Bernoulli. Ambos modelos estan disponibles en sklearn.

In [20]:
```
from sklearn.naive_bayes import MultinomialNB, BernoulliNB
results=[]
#results.append(['Escala de gris','Alpha','Grupo','Accuraci'])

_max_df=[0.9,0.8,0.7,0.6]# max_df establece que si aparece en el 80% de los do
cumentos no aporta información
_min_df=[100,90,80,70]# min_df establece que si no aparece en por lo menos 100
documentos tampoco
clf = [MultinomialNB(),BernoulliNB()]

for i in range(2): #Barro las dos distribuciones
    for j in range(len(dataToProces)):#Barro los distintos tipos de filtrados
        for k in range(len(_max_df)):#Barro los max_df
            for u in range(len(_min_df)):#Barro los min_df
                for m in range(2):
                    if(m==0):
                        count_vect = CountVectorizer(max_df=_max_df[k],min_df=
_min_df[u])
                    else:
                        count_vect = TfidfVectorizer(max_df=_max_df[k],min_df=
_min_df[u])


                    raw_data = count_vect.fit_transform(dataToProces[j])
                    clf[i].fit(raw_data, twenty_train['target'])
                    results.append([m,i,j,_max_df[k],_min_df[u],clf[i].score(r
aw_data, twenty_train['target'])])
```

In [21]:
```
'''
# opening the csv file in 'w' mode
file = open('resultadosTrain.csv', 'w', newline ='')

with file:
    # identifying header
    header = ['CountVectorizer_0','Multinomial_0','Filtros','max_df','min_d
f','Accuraci']
    writer = csv.DictWriter(file, fieldnames = header)

    # writing data row-wise into the csv file
    writer.writeheader()
    for x in results:
        writer.writerow({'CountVectorizer_0':x[0],
                         'Multinomial_0':x[1],
                         'Filtros':x[2],
                         'max_df':x[3],
                         'min_df':x[4],
                         'Accuraci':x[5]})
'''
```

Finalmente comprobar el accuracy en train.

Preguntas

- Con que combinación de preprocesamiento obtuvo los mejores resultados? Explique por qué cree que fue así.
- Con que modelo obtuvo los mejores resultados? Explique por qué cree que fue así.

Performance de los modelos

En el caso anterior, para medir la cantidad de artículos clasiicados correctamente se utilizó el mismo subconjunto del dataset que se utilizó para entrenar.

Esta medida no es una medida del todo útil, ya que lo que interesa de un clasificador es su capacidad de clasificación de datos que no fueron utilizados para entrenar. Es por eso que se pide, para el clasificador entrenado con el subconjunto de training, cual es el porcentaje de artículos del subconjunto de testing clasificados correctamente. Comparar con el porcentaje anterior y explicar las diferencias.

Finalmente deben observar las diferencias y extraer conclusiones en base al accuracy obtenido, el preprocesamiento y vectorización utilizado y el modelo, para cada combinación de posibilidades.

In [23]:
```
from sklearn.naive_bayes import MultinomialNB, BernoulliNB
results=[]
#results.append(['Escala de gris','Alpha','Grupo','Accuraci'])

_max_df=[0.9,0.8,0.7,0.6]# max_df establece que si aparece en el 80% de los do
cumentos no aporta información
_min_df=[100,90,80,70]# min_df establece que si no aparece en por lo menos 100
documentos tampoco
clf = [MultinomialNB(),BernoulliNB()]

for i in range(2): #Barro las dos distribuciones
    for j in range(len(dataToProces)):#Barro los distintos tipos de filtrados
        for k in range(len(_max_df)):#Barro los max_df
            for u in range(len(_min_df)):#Barro los min_df
                count_vect = CountVectorizer(max_df=_max_df[k],min_df=_min_df[
u])
                raw_data =count_vect.fit_transform(dataToProces[j])
                test_raw_data=count_vect.transform(test_dataToProces[j])
                clf[i].fit(raw_data, twenty_train['target'])
                results.append([i,j,_max_df[k],_min_df[u],clf[i].score(test_ra
w_data, twenty_test['target'])])
```

In [24]:
```
'''
# opening the csv file in 'w' mode
file = open('resultadosTest.csv', 'w', newline ='')

with file:
    # identifying header
    header = ['Multinomial_0','Filtros','max_df','min_df','Accuraci']
    writer = csv.DictWriter(file, fieldnames = header)

    # writing data row-wise into the csv file
    writer.writeheader()
    for x in results:
        writer.writerow({'Multinomial_0':x[0],
                         'Filtros':x[1],
                         'max_df':x[2],
                         'min_df':x[3],
                         'Accuraci':x[4]})

'''
```

Preguntas

- El accuracy en el dataset de test es mayor o menor que en train? Explique por qué.