UNIVERSIDAD NACIONAL DEL ALTIPLANO
FACULTAD DE INGENIERÍA MECÁNICA ELÉCTRICA, ELECTRÓNICA Y SISTEMAS.
ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

Práctica de Laboratorio N°02 - Análisis de Sentimiento con Naive Bayes

PRESENTADO POR:
CHOQQUE LAYME ERLY

DOCENTE:
RUELAS ACERO DONIA ALIZANDRA

CURSO:
TOPICOS AVANZADOS EN INTELIGENCIA COMPUTACIONAL

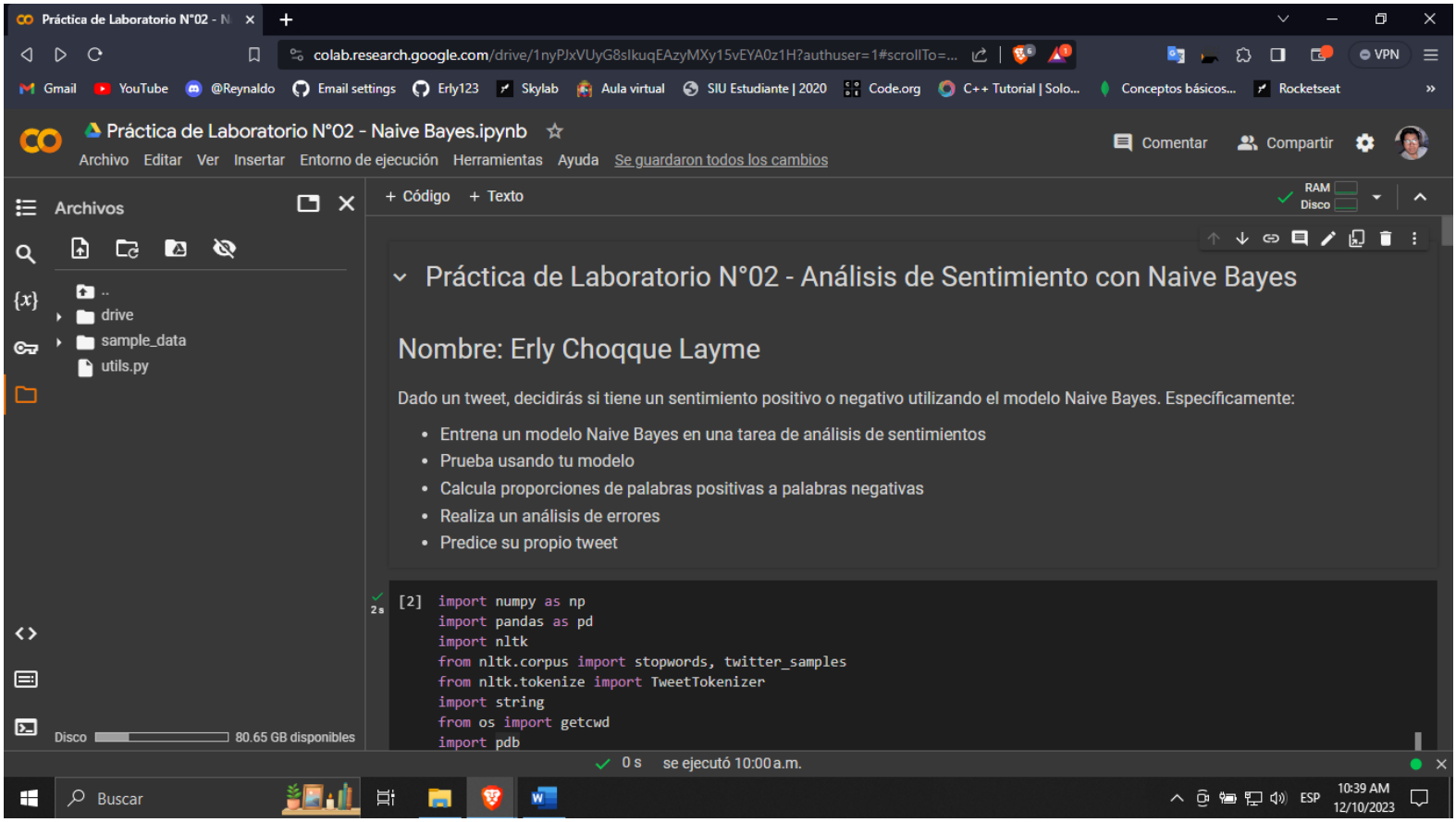
PUNO-PERÚ
2023

Práctica de Laboratorio N°02 - Análisis de Sentimiento con Naive Bayes

Link de google colab:

https://colab.research.google.com/drive/1nyPJxVUyG8sIkuqEAzyMXy15vEYA0z1H?usp=sharing

Captura de pantalla:



Práctica de Laboratorio N°02 - Análisis de Sentimiento con Naive Bayes

Nombre: Erly Choqque Layme

Dado un tweet, decidirás si tiene un sentimiento positivo o negativo utilizando el modelo Naive Bayes. Específicamente:

- Entrena un modelo Naive Bayes en una tarea de análisis de sentimientos
- Prueba usando tu modelo
- Calcula proporciones de palabras positivas a palabras negativas
- Realiza un análisis de errores
- Predice su propio tweet

```
import numpy as np
import pandas as pd
import nltk
from nltk.corpus import stopwords, twitter_samples
from nltk.tokenize import TweetTokenizer
import string
from os import getcwd
import pdb
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Importa tu función *process_tweets* de tu librería *utils* (1 pto)

```
# Importa aquí
nltk.download('stopwords')
# importa tu función de preprocesamiento de tweets process_tweet que realizaste en la Actividad N°01
from utils import process_tweet, build_freqs, lookup
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

```
nltk.download('stopwords')
nltk.download('twitter_samples')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package twitter_samples to /root/nltk_data...
[nltk_data]   Unzipping corpora/twitter_samples.zip.
True
```

Corpus

```
# Obten el conjunto de tweets positivos y negativos
all_positive_tweets = twitter_samples.strings('positive_tweets.json')
all_negative_tweets = twitter_samples.strings('negative_tweets.json')

# Divide el dataset en 2 partes, la primera para el entrenamiento y la segunda para las pruebas.
test_pos = all_positive_tweets[4000:]
train_pos = all_positive_tweets[:4000]
test_neg = all_negative_tweets[4000:]
train_neg = all_negative_tweets[:4000]

train_x = train_pos + train_neg
train_y = np.append(np.ones(len(train_pos)), np.zeros(len(train_neg)))

test_x = test_pos + test_neg
test_y = np.append(np.ones(len(test_pos)), np.zeros(len(test_neg)))
```

1. Procesar el corpus

El primer paso para cualquier proyecto de Machine Learning, una vez haya recopilado los datos, es procesarlos para generar entradas útiles para su modelo.

- **Elimine el ruido:** primero querrá eliminar el ruido de sus datos, es decir, eliminar las palabras que no le dicen mucho sobre el contenido. Estos incluyen todas las palabras comunes como pronombres, preposiciones, y otros que no nos darían suficiente información sobre el sentimiento.
- También eliminaremos los tickers del mercado de valores, los símbolos de retweet, los hipervínculos y los hashtags porque no pueden brindarle mucha información sobre el sentimiento.

También desea eliminar toda la puntuación de un tweet. La razón para hacer esto es que queremos tratar las palabras con o sin puntuación como la misma palabra, en lugar de tratar "feliz", "¿feliz?", "¡feliz!", "feliz" y "feliz". como palabras diferentes. Finalmente, desea usar la lematización para realizar un seguimiento de solo una variación de cada palabra. En otras palabras, trataremos "motivación", "motivado" y "motivar" de manera similar agrupándolos dentro de la misma raíz de "motiv-". Le hemos dado la función process_tweet() que hace esto por usted.

```
ejemplo_tweet = "RT @Twitter @chapagain Hello There! Have a great day. :) #good #morning http://chapagain.com.np"

# print tweet  procesado
print(process_tweet(ejemplo_tweet))
```

```
['hello', 'great', 'day', ':)', 'good', 'morn']
```

2. Implementar Funciones útiles

2.1 Cree una función **count_tweets()** que tome una lista de tweets como entrada, los limpie todos y devuelva un diccionario. (1 pto)

- La clave en el diccionario es una tupla que contiene la palabra y su etiqueta de clase, ejm ("happi", 1).
- El valor del número de veces que aparece esta palabra en la colección dada de tweets (un número entero).

```
def count_tweets(result, tweets, ys):
    '''
    Input:
        result: un diccionario que se utilizará para asignar cada par a su frecuencia
        tweets: lista de tweets
        ys: una lista que corresponde al sentimiento de cada tweets( 0 0 1)
    Output:
        result: un diccionario que mapee cada par a su frecuencia
    '''



    for y, tweet in zip(ys, tweets):
        for word in process_tweet(tweet):
            # define la clave, el cual esta compuesta por la tupla (word,label)
            pair = (word,y)

            # si la clave existe en el diccionario, incrementa el contador
            if pair in result:
                result[pair] += 1

            # caso contrario, si la clave es nueva, agregar al diccionario y configurar el contado a 1
            else:
                result[pair] = 1

    return result
```

2.2 Create freqs dictionary (1 pto)

Dada la función **count_tweets()** , tu puedes calcular un diccionario llamado **freqs** que contiene todas las frecuencias. En ese diccionario **freqs** dictionary, la clave es la tupla (word, label). El valor es el número de veces que aparece.

```
freqs =count_tweets({}, train_x, train_y)
```

Pruebe su función

Salida esperada: {('happi', 1): 1, ('trick', 0): 1, ('sad', 0): 1, ('tire', 0): 2}

```
result = {}
tweets = ['i am happy', 'i am tricked', 'i am sad', 'i am tired', 'i am tired']
ys = [1, 0, 0, 0, 0]
count_tweets(result, tweets, ys)
```

```
{('happi', 1): 1, ('trick', 0): 1, ('sad', 0): 1, ('tire', 0): 2}
```

3. Entrenar su modelo usando Naive Bayes (3 ptos)

Probabilidad de cada clase

- Identificamos el número de clases que se tiene.
- Crear la probabilidad de cada clase. $P(D_{pos})$ es la probabilidad de que el documento sea positivo. $P(D_{neg})$ es la probabilidad de que el documento sea negativo.

Use las siguientes fórmulas y almanece los valores en un diccionario:

$$P(D_{pos}) = \frac{D_{pos}}{D} \tag{1}$$

$$P(D_{neg}) = \frac{D_{neg}}{D} \tag{2}$$

Donde D es el número total de documentos, o tweets en este caso, D_{pos} es el número total de tweets positivos y D_{neg} es el númerto total de tweets negativos.

Prior and Logprior La probabilidad **Prior** representa la probabilidad subyacente en la población objetivo de que un tweet sea positivo frente a negativo. En otras palabras, si no tuviéramos información específica y seleccionamos a ciegas un tweet del conjunto de población, ¿cuál es la probabilidad de que sea positivo frente a que sea negativo? Ese es el "Prior".

El prior es la razon de las probabilidades $\frac{P(D_{pos})}{P(D_{neg})}$. Podemos tomar prior para reescalarlo, y lo llamaremos el **los prior**

$$\text{logprior} = log\left(\frac{P(D_{pos})}{P(D_{neg})}\right) = log\left(\frac{D_{pos}}{D_{neg}}\right)$$

.

Note que $log(\frac{A}{B})$ es lo mismo que $log(A) - log(B)$. Asi que logprior puede ser calculado como la diferencia de dos logaritmos:

$$\text{logprior} = \log(P(D_{pos})) - \log(P(D_{neg})) = \log(D_{pos}) - \log(D_{neg}) \tag{3}$$

Positive and Negative Probability of a Word Para calcular la probabilidad positiva y la probabilidad negativa de una palabra específica del diccionario, Usaremos las siguientes entradas:

- $freq_{pos}$ and $freq_{neg}$ son las frecuencias de esa palabra específica en la clase positiva o negativa. En otras palabras, la frecuencia positiva de una palabra es el número de veces que la palabra se cuenta con la etiqueta de 1.
- N_{pos} and N_{neg} son el número total de palabras positivas y negativas para todos los documentos (para todos los tweets), respectivamente.
- V es el número de palabras únicas en todo el conjunto de documentos, para todas las clases, ya sean positivas o negativas.

Los usaremos para calcular la probabilidad positiva y negativa de una palabra específica usando esta fórmula:

$$P(W_{pos}) = \frac{freq_{pos} + 1}{N_{pos} + V} \tag{4}$$

$$P(W_{neg}) = \frac{freq_{neg} + 1}{N_{neg} + V} \tag{5}$$

Log likelihood

Para calcular loglikelihood de esa misma palabra, podemos implementar las siguiente ecuación:

$$\text{loglikelihood} = \log\left(\frac{P(W_{pos})}{P(W_{neg})}\right) \tag{6}$$

```
def train_naive_bayes(freqs, train_x, train_y):

    loglikelihood = {}
    logprior = 0


    # calculate V, El número de palabras únicas del vocabulario
    vocab = set([pair[0] for pair in freqs.keys()])
    V = len(vocab)

    # calculate N_pos, N_neg, V_pos, V_neg
```

```
    # calculate N_pos, N_neg, V_pos, V_neg
    N_pos = N_neg = V_pos = V_neg = 0

    for pair in freqs.keys():
        # si el label es positivo (>0)
        if pair[1] > 0:
            # incrementa el contador de palabras únicas positivas en 1
            V_pos += 1

            # incremente el número de palabras positivas por el conteo para este par (word, label)
            N_pos += freqs[pair]

        # caso contrario, el label es negativo
        else:
            # incrementa el contador de palabras únicas negativas en 1
            V_neg +=1

            # incremente el número de palabras negativas por el conteo para este par (word, label)
            N_neg += freqs[pair]

    # Calculate D, el numero de documentos
    D = len(train_y)

    # Calculate D_pos, el número de documentos positivos
    D_pos = len(list(filter(lambda x: x > 0, train_y)))

    # Calculate D_neg, el número de documentos negativos
    D_neg = len(list(filter(lambda x: x <= 0, train_y)))

    # Calculate logprior
    logprior = np.log(D_pos) - np.log(D_neg)

    # Para cada palabra en el vocabulario
    for word in vocab:
        # obtenga la frecuencia positiva y negativa de la plabra
        freq_pos = lookup(freqs, word, 1)
        freq_neg = lookup(freqs, word, 0)

        # Calcula la probabilidad de que cada palabra sea positiva y negativa
        p_w_pos = (freq_pos + 1) / (N_pos + V)
        p_w_neg = (freq_neg + 1) / (N_neg + V)

        # calculate the log likelihood of the word
        loglikelihood[word] = np.log(p_w_pos / p_w_neg)



    return logprior, loglikelihood
```

Prueba de la función train_naive_bayes

Salida Esperada: 0.0 9089

```
logprior, loglikelihood = train_naive_bayes(freqs, train_x, train_y)
print(logprior)
print(len(loglikelihood))
```

```
0.0
9161
```

4. Test el modelo Naive Bayes

4.1. Implementa la función Predicción (3 ptos)

```
def naive_bayes_predict(tweet, logprior, loglikelihood):

    # procesa el tweet para obtener una lista de palabras
    word_l = process_tweet(tweet)

    # inicializa la probabilidad a 0
    p = 0

    # agrega el logprior
    p += logprior
```

```
    for word in word_l:

        # Verifica si la palabra existe en el diccionario loglikelihood
        if word in loglikelihood:
            # Agregar log likelihood de cada palabra a la probabilidad
            p += loglikelihood[word]


    return p
```

Prueba de la función naive_bayes_predict

Salida Esperada: 1.5740278623499175

```
# Experimenta con tu propio tweet
my_tweet = 'She smiled.'
p = naive_bayes_predict(my_tweet, logprior, loglikelihood)
print('Salida esperada:', p)
```

```
Salida esperada: 1.557492820301094
```

4.2 Implementa la función test_naive_bayes (2 ptos)

- Implementa **test_naive_bayes** para verificar el accuracy de la predicción.
- La función recibe de entrada **test_x**, **test_y**, log_prior, y loglikelihood
- La función retorna el accuracy del modelo.
- Primero debes usar la función **naive_bayes_predict** para hacer predicciones de cada tweets en text_x.

```
def test_naive_bayes(test_x, test_y, logprior, loglikelihood):

    accuracy = 0


    y_hats = []
    for tweet in test_x:
        # Si la predicción es > 0
        if naive_bayes_predict(tweet, logprior, loglikelihood) > 0:
            # la clase predecida es 1
            y_hat_i = 1
        else:
            # caso contrario la clase predecida es 0
            y_hat_i = 0

        # agregamos la clase predecida y_hats
        y_hats.append(y_hat_i)

    # error es el promedio del valor de la diferencia entre y_hats y test_y
    error = np.mean(np.absolute(y_hats-test_y))

    # Accuracy es 1 menos el error
    accuracy =  1 - error

    return accuracy
```

Prueba de la función test_naive_bayes

Salida Esperada: 0.9940

```
print("Naive Bayes accuracy = %0.4f" %
      (test_naive_bayes(test_x, test_y, logprior, loglikelihood)))
```

```
Naive Bayes accuracy = 0.9955
```

5. Filtra las 5 primeros palabras más positivas y las 5 palabras más negativas

- Algunas palabras tienen más recuentos positivos que otras y pueden considerarse **más positivas**. Asimismo, algunas palabras pueden considerarse **más negativas** que otras.
- Una forma de definir el nivel de positividad o negatividad, sin calcular el log likelihood, es comparar la frecuencia positiva con la negativa de la palabra.Tenga en cuenta que también podemos usar los cálculos de log likelihood para comparar la positividad o negatividad relativa de las palabras
- Una vez que podamos calcular estas proporciones, también podemos filtrar un subconjunto de palabras que tengan una proporción mínima de positividad/negatividad o superior.
- Del mismo modo, también podemos filtrar un subconjunto de palabras que tengan una relación máxima de positividad/negatividad o inferior (palabras que sean al menos tan negativas, o incluso más negativas que un umbral dado).

5.1 Implementa **get_ratio()** (2 ptos)

- Dado el diccionario **freqs** de palabras, usa **lookup(freqs,word,1)** para obtener el recuento positivo de la palabra.
- Similarmente, usa la función **lookup()** para obtener el recuento negativo de esa palabra.
- Calcula el ratio de recuentos positivo dividido de recuentos negativos.

$$ratio = \frac{\text{pos\_words} + 1}{\text{neg\_words} + 1}$$

Donde pos_words y neg_words corresponden a la frecuencia de las palabras en sus respectivas clases.

Words	Positive word count	Negative Word Count
glad	41	2
arriv	57	4
:(	1	3663
:-(	0	378

```
def get_ratio(freqs, word):
    '''
    Input:
        freqs: diccinario que contiene words

    Output: un diccionario con claves 'positive', 'negative', and 'ratio'.
        Example: {'positive': 10, 'negative': 20, 'ratio': 0.5}
    '''
    pos_neg_ratio = {'positive': 0, 'negative': 0, 'ratio': 0.0}

    # use lookup() para encontrar el recuento positivo de la palabra (denotado by the integer 1)
    pos_neg_ratio['positive'] = lookup(freqs, word, 1)

    # use lookup() para encontrar el recuento negativo de la palabra (denotado by integer 0)
    pos_neg_ratio['negative'] = lookup(freqs, word, 0)

    # calcula el ratio
    pos_neg_ratio['ratio'] = (pos_neg_ratio['positive'] + 1) / (pos_neg_ratio['negative'] + 1)

    return pos_neg_ratio
```

Probamos la función *get_ratio()*

```
get_ratio(freqs, 'happi')
```

```
{'positive': 162, 'negative': 18, 'ratio': 8.578947368421053}
```

5.2 Implementa get_words_by_threshold(freqs,label,threshold) (2 ptos)

- If we set the label to 1, then we'll look for all words whose threshold of positive/negative is at least as high as that threshold, or higher.
- If we set the label to 0, then we'll look for all words whose threshold of positive/negative is at most as low as the given threshold, or lower.
- Use the `get_ratio()` function to get a dictionary containing the positive count, negative count, and the ratio of positive to negative counts.
- Append a dictionary to a list, where the key is the word, and the dictionary is the dictionary `pos_neg_ratio` that is returned by the `get_ratio()` function. An example key-value pair would have this structure:

```
        {'happi':
          {'positive': 10, 'negative': 20, 'ratio': 0.5}
        }


def get_words_by_threshold(freqs, label, threshold):
    '''
    Input:
        freqs: dictionary of words
        pos_neg_ratio: diccionario de conteos positivos, conteos negativos y relación de conteos positivos/negativos.
        label: 1 para positivo, 0 para negativo
        threshold: ratio que se usará como límite para incluir una palabra en el diccionario devuelto
    Output:
        word_set: dicctionario que contiene
        {'happi':
            {'positive': 10, 'negative': 20, 'ratio': 0.5}
        }
    '''
    word_list = {}

    for key in freqs.keys():
        word, _ = key

        # obtener el ratio orción positiva/negativa de una palabra (usa la función get_ratio())
        pos_neg_ratio = get_ratio(freqs, word)

        # si la etiqueta es 1 y la relación es mayor o igual al umbral...
        if label == 1 and pos_neg_ratio['ratio'] >= threshold :

            # Añadir la pos_neg_ratio al diccionario
            word_list[word] = pos_neg_ratio

        # Si la etiqueta es 0 y pos_neg_ratio es menor o igual al umbral...
        elif label == 0 and pos_neg_ratio['ratio'] <= threshold:
            # Añadir la pos_neg_ratio al diccionario
            word_list[word] = pos_neg_ratio

        # de lo contrario, no incluya esta palabra en la lista (no haga nada)

    return word_list
```

probando la función **get_words_by_threshold(freqs, label=0, threshold=0.05)**

```
# Encuentra palabras negativas en o por debajo de un umbral
get_words_by_threshold(freqs, label=0, threshold=0.05)

     {':(': {'positive': 1, 'negative': 3675, 'ratio': 0.0005440696409140037},
      ':-(': {'positive': 0, 'negative': 386, 'ratio': 0.002583979328165375},
      'zayniscomingbackonjuli': {'positive': 0, 'negative': 19, 'ratio': 0.05},
      '26': {'positive': 0, 'negative': 20, 'ratio': 0.047619047619047616},
      '>:(': {'positive': 0, 'negative': 43, 'ratio': 0.022727272727272728},
      'lost': {'positive': 0, 'negative': 19, 'ratio': 0.05},
      '♕': {'positive': 0, 'negative': 210, 'ratio': 0.004739336492890996},
      '》 ': {'positive': 0, 'negative': 210, 'ratio': 0.004739336492890996},
      'believ': {'positive': 0, 'negative': 35, 'ratio': 0.02777777777777776},
      'will': {'positive': 0, 'negative': 35, 'ratio': 0.02777777777777776},
      'justin': {'positive': 0, 'negative': 35, 'ratio': 0.02777777777777776},
      ' s e e': {'positive': 0, 'negative': 35, 'ratio': 0.02777777777777776},
      'm e': {'positive': 0, 'negative': 35, 'ratio': 0.02777777777777776}}
```

4.3 Modifica el código de arriba y obten las 5 primeros palabras más positivas y las 5 palabras más negativas (3 ptos)

```
# 5 palabras más positivas
positive_words = get_words_by_threshold(freqs, label=1, threshold=0.95)
top_5_positive = {k: positive_words[k] for k in list(positive_words)[:5]}
print("5 palabras más positivas:")
print(top_5_positive)

# 5 palabras más negativas
negative_words = get_words_by_threshold(freqs, label=0, threshold=0.05)
top_5_negative = {k: negative_words[k] for k in list(negative_words)[:5]}
print("\n5 palabras más negativas:")
print(top_5_negative)
```

```
5 palabras más positivas:
{'followfriday': {'positive': 23, 'negative': 0, 'ratio': 24.0}, 'top': {'positive': 30, 'negative': 5, 'ratio': 5.166666666666

5 palabras más negativas:
{':(': {'positive': 1, 'negative': 3675, 'ratio': 0.000544069640914037}, ':-(': {'positive': 0, 'negative': 386, 'ratio': 0.002
```

6. Análisis de Errores de predicción (2 ptos)

En esta parte, verá algunos tweets que el modelo clasificó incorrectamente.

```
print('Truth Predicted Tweet')
for x, y in zip(test_x, test_y):
    y_hat = naive_bayes_predict(x, logprior, loglikelihood)
    if y != (np.sign(y_hat) > 0):
        print('%d\t%0.2f\t%s' % (y, np.sign(y_hat) > 0, ' '.join(
            process_tweet(x)).encode('ascii', 'ignore')))
```

```
Truth Predicted Tweet
1       0.00    b'truli later move know queen bee upward bound movingonup'
1       0.00    b'new report talk burn calori cold work harder warm feel better weather :p'
1       0.00    b'harri niall 94 harri born ik stupid wanna chang :d'
1       0.00    b'park get sunlight'
1       0.00    b'uff itna miss karhi thi ap :p'
0       1.00    b'hello info possibl interest jonatha close join beti :( great'
0       1.00    b'u prob fun david'
0       1.00    b'pat jay'
0       1.00    b'sr financi analyst expedia inc bellevu wa financ expediajob job job hire'
```

Responde las siguientes preguntas

1.- ¿Por qué crees que ocurrieron las clasificaciones erróneas?

Las clasificaciones erróneas puede suceder debido a que el modelo Naive Bayes asume independencia entre las características, lo que puede no ser cierto en la realidad de los datos. Esta suposición de independencia puede ser demasiado muy simple para algunos conjuntos de datos, especialmente si hay interdependencia o correlación entre las características. Si el modelo no logra obtener estas relaciones entre las características, generaria predicciones incorrectas, ya que no considera las relaciones reales que podrían existir entre los diferentes elementos que componen los datos.

2.- ¿Hubo algunas suposiciones hechas por el modelo naive bayes?

El modelo Naive Bayes hace suposiciones para simplificar el proceso de aprendizaje debido a la independencia de características, el cual asume que cada característica contribuye de manera independiente a la probabilidad de pertenecer a una clase determinada, lo cual simplifica los cálculos al considerar cada característica por separado, hay suposicion tambien por la distribución condicional el cual estima la probabilidad de cada característica dada la clase, esta suposición ayuda en el cálculo de las probabilidades condicionales.

7. Predice tu propio tweet (1 pto)

Pruebe con su propio tweet; siéntete libre de modificar **my_tweet**

```
my_tweet = 'Thinking of you is the best sensation :)'

p = naive_bayes_predict(my_tweet, logprior, loglikelihood)
print(p)
```

```
7.773056323303992
```

Conclusiones

Escriba 3 conclusiones de la Práctica de Laboratorio 2

1.- El modelo naive bayes realiza una clasificacion probabilistica con una suposicoin de independencia condicional el cual proporciona una mejor precision de clasificacion en conjunto de datos esto previo a un entrenamiento para luego predecir con una alta probabilidad.

2.- A pesar de su eficiencia y simplicidad, el modelo Naive Bayes asume independencia entre características y distribuciones condicionales específicas. Estas suposiciones pueden ser demasiado simplistas para reflejar la complejidad real de los datos, lo que lleva a clasificaciones erróneas en situaciones donde las características están interrelacionadas o no siguen las distribuciones asumidas por el modelo

3.- El algoritmo Naive Bayes hace el analisis de sentimientos en tweets. Sin embargo, el modelo asume independencia entre palabras y distribuciones específicas. A pesar de sus simplificaciones, el modelo procesa una base solida para el análisis de sentimientos en este caso.