
Instituto Tecnológico y de Estudios Superiores de Monterrey

Escuela de Ingeniería y Ciencias

Ingeniería en Ciencia de Datos y Matemáticas


# Evidencia 1. Sistema en Python que Detecta SPAM

## Análisis de Métodos de Razonamiento e Incertidumbre


*Cantú Rodríguez Pamela* A01285128

*Díaz Seguy Laura Cecilia* A01620523

*Núñez López Daniel I.* A01654137

*Reyes Mancheno Paola Sofía* A00831314

*Torres Alcubilla María Fernanda* A01285041

**Supervisado por**
Dr. Marco Otilio Peña Díaz


Monterrey, Nuevo León. Fecha, 26 de septiembre de 2022

# 1. Problemática

El correo no deseado, o spam lleva más de 40 años atacando a la sociedad y estos no solo ocupan lugar en nuestra bandeja de entrada, sino que también pueden llevar a otras problemáticas como phishing o entrada de malwares a través de la ingeniería social y archivos adjuntos en los correos.

Esta problemática ha llevado a las compañías a implementar detección de spam en sus plataformas de correo electrónico a través de diversos algoritmos y modelos. Actualmente la problemática es tal, que solamente en el mes de abril del 2019 el spam a través del correo electrónico representó el 85 % de todo el correo electrónico a escala global. (Europa Press, 2019)

# 2. Propósito

Para la presente situación se elaboró un sistema para clasificar el mensaje de un correo electrónico a través del Natural Language Processing Algorithm, también conocido como Bag-of-Words (BoW), para el procesamiento de las palabras y el algoritmo de Naive Bayes para clasificar su contenido ya sea como spam o como ham.

Al contar con un algoritmo de clasificación de mensajes, se reduce el riesgo de ser víctima de un ataque en el que nuestra información sensible pueda caer en manos equivocadas, esto es de gran ventaja para personas que no están tan familiarizadas con la tecnología y que aún les cuesta trabajo distinguir entre un correo/mensaje sospechoso y uno 'normal'.

# 3. Información

Para el modelo de representación de palabras usamos Bag-of-Words el cual está conformado de una colección de palabras y es usado para analizar el texto. Este algoritmo toma las palabras de un texto y crea un vector con la frecuencia de cada palabra sin considerar la posición de las palabras en el texto.

También se usó la probabilidad condicional y su inversa del Teorema de Bayes, donde se calcula la probabilidad condicional que un texto sea spam junto con la probabilidad a priori, en este caso la probabilidad de ser spam en la muestra, esto se calculó a través de la siguiente forma del ejemplo:

$$P(\text{'congratulations you have won a car'} \mid SPAM) \cdot P(SPAM) \tag{1}$$

Además, a través de la suposición de Naive Bayes, para calcular la probabilidad condicional inversa se separa cada palabra, lo que resulta en una multiplicación de las probabilidades condicionales de cada palabra dado que provengan de la muestra respectiva al spam.

# 4. Razonamiento

Lo primero que se hizo fue reestructurar el dataset que contenía un mensaje previamente clasificado como spam o ham, aquí, se convirtieron las categorías a valores numéricos (en este caso booleanos), se eliminaron los signos de puntuación, se mezclaron las filas para redistribuir los mensajes y por último se crearon dos datasets, uno para el entrenamiento (con el 70 % de los datos) y otro para la prueba.

Luego de esto, obtuvimos la probabilidad de obtener un mensaje de spam dados los datos que se nos dieron, y esto arrojó un valor de 0.1324. Posteriormente, se crearon ambas BoW, en donde juntamos todas las palabras en una sola lista para ambas categorías. Seguido a esto, obtuvimos cuáles eran las palabras que compartían ambas categorías, ya que estas formarían parte de una lista de palabras comunes, que utilizamos para hacer dos diccionarios, uno que utilizara las palabras comunes en textos de spam y el otro en textos ham. Esto nos permitió tener dos BoWs, para spam y ham respectivamente, en donde la llave hace referencia a la palabra y el valor a su frecuencia en cada clase.

Como se mencionó anteriormente, utilizamos probabilidad condicional, basándonos en el Teorema de Bayes, que si recordamos, está conformado de la siguiente manera:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)} \tag{2}$$

Para predecir si el mensaje es spam, debemos calcular la probabilidad de que provenga de la BoW de spam. Si esta probabilidad es alta, el mensaje es clasificado como spam. Estas probabilidades son calculadas de la siguiente manera:

1. Establecer el objetivo usando probabilidad condicional, ej. obtener la probabilidad de que el mensaje sea spam bajo la condición de que el mensaje es 'you can win 5k cash in no time'
2. Establecer esta probabilidad usando el Teorema de Bayes
3. Usar el algoritmo de suposición de Naive Bayes para obtener la probabilidad de que el mensaje sea spam.
4. Aplicar un logaritmo para evitar inestabilidad numérica.

Una vez realizado esto, ya contamos con un método de clasificación, por lo que con nuestro dataset de prueba realizamos las predicciones y añadimos una columna con el resultado de cada predicción para cada mensaje. Por último, comparamos ambas columnas (la original, que contenía la categoría real del mensaje, y la columna creada por nuestro modelo de predicción) utilizando una matriz de confusión y obtuvimos que el algoritmo tuvo una exactitud del 100 %, ya que no se tuvieron falsos positivos ni falsos negativos.

## 5. Conclusiones

Al haber realizado este clasificador de spam, pudimos ver que con los datos que se nos facilitaron, logramos clasificar satisfactoriamente 1668 mensajes, obteniendo una exactitud del 100 % en nuestra matriz de confusión. Con esto, logramos cumplir nuestro propósito y aplicamos nuestros conocimientos de probabilidad para intentar resolver un problema tan molesto y peligroso como lo puede llegar a ser el spam.

Una posible mejora sería reducir los textos al eliminar palabras que su uso es para la estructura del texto como los artículos, preposiciones, nexos, pronombres, etcétera y observar no solo si se obtuvo un cambio en la matriz de confusión, también comparar el tiempo de corrida ya que al usar el sistema para bases de datos más grandes se podría tener un problema de tiempo de espera.

## 6. Referencias

EUROPA PRESS. (2019, 19 junio). El ≪spam≫ supone el 85 % de todo el correo electrónico en el mundo. ABC Tecnología. Recuperado 25 de septiembre de 2022, de "https://www.abc.es/tecnologia/redes/abci-spam-supone-85-por-ciento-todo-correo-electronico-mundo-201906190114_noticia.html"

# 7. Apéndice: Código en Python

```
#### Para realizar un sistema que detecte si un mensaje es considerado spam,
se hará uso de un Natural Language Processing Algorithm conocido como
Bag-of-Words (BoW).

#### Este analiza la frecuencia de las palabras en un texto para después determinar
la relevancia de cada palabra en el mismo. Posteriormente, se hará uso del algoritmo
de Naive Bayes para clasificar el mensaje como spam o ham.

#### El siguiente código realizado por Andrew Han ayuda a entender el algoritmo de
Naive Bayes sin la necesidad de utilizar librerías de aprendizaje automático:

Importar librerias:
"""

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import string
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay

"""Leer el dataset proporcionado:"""

spam_df = pd.read_csv('spam.csv', encoding="ISO-8859-1")
spam_df

"""Cambiar el nombre de la primera columna a 'label':"""

spam_df.rename(columns={'type': 'label'}, inplace=True)

"""Convertir 'spam' y 'ham' a variables binarias:"""

spam_df['label'] = spam_df['label'].apply(lambda a: True if a == 'spam' else False)

"""Convertir todo el texto a minúsculas y usar el método str.maketrans() para crear
una tabla de traducciones que remueva los signos de puntuación:"""

table = str.maketrans('','',string.punctuation)
spam_df['text'] = spam_df['text'].apply(lambda t: t.lower().translate(table))
spam_df

"""Mezclar filas aleatoriamente para que los registros de 'spam' y 'ham' esten
distribuidos equitativamente en los datasets de entrenamiento y prueba: """

spam_df = spam_df.sample(frac=1)
spam_df

"""Crear los datasets para entrenamiento (70%) y prueba (30%):"""

train_spam_df = spam_df.iloc[:int(len(spam_df)*0.7),:]
test_spam_df = spam_df.iloc[int(len(spam_df)*0.7):,:]

"""Obtener la probabilidad de ser un mensaje de spam ya que su valor será
utilizado para el algoritmo BoW:"""
```

```
FRAC_SPAM_TEXTS = train_spam_df['label'].mean()
print(FRAC_SPAM_TEXTS)

"""Tomar cada palabra en los textos de spam y ham y ponerlos en su respectivo
Bag-of-Words. Tendremos 2 BoWs:
* BoW para spam: todas las palabras en los textos de spam con su respectiva frecuencia.
* BoW para ham: todas las palabras en los textos de ham con su respectiva frecuencia.

Primero, concatenamos todos los textos de spam y ham respectivamente:
"""

train_spam_words_str = ' '.join(train_spam_df.loc[train_spam_df['label'] == True, 'text'])
train_non_spam_words_str = ' '.join(train_spam_df.loc[train_spam_df['label'] == False, 'text'])

"""Poner las palabras en una lista:"""

train_spam_words_list = train_spam_words_str.split(' ')
train_non_spam_words_list = train_non_spam_words_str.split(' ')

"""Encontrar palabras comunes que aparezcan en ambas BoW para poder compararlas.
No se pueden comparar las palabras que solo aparecen en una BoW:"""

common_words = set(train_spam_words_list).intersection(set(train_non_spam_words_list))

"""Construir un diccionario usando solo las palabras comunes en textos spam:"""

train_spam_bow = dict()
for w in common_words:
    train_spam_bow[w] = train_spam_words_list.count(w) / len(train_spam_words_list)

"""Construir un diccionario usando solo las palabras comunes en textos ham:"""

train_non_spam_bow = dict()
for w in common_words:
    train_non_spam_bow[w] = train_non_spam_words_list.count(w) / len(train_non_spam_words_list)

"""Ahora tenemos dos BoWs para spam y ham respectivamente en donde la llave
hace referencia a la palabra y el valor a su frecuencia en cada clase.

Para predecir si el mensaje es spam, debemos calcular la probabilidad de que
provenga de la BoW de spam. Si esta probabilidad es alta, el mensaje es clasificado como spam.

Estas probabilidades son calculadas de la siguiente manera:
1. Establecer el objetivo usando probabilidad condicional, ej. obtener la
probabilidad de que el mensaje sea spam bajo la condición de que el mensaje
es 'you can win 5k cash in no time'
2. Establecer esta probabilidad usando el Teorema de Bayes
3. Usar el algoritmo de suposición de Naive Bayes para obtener la
probabilidad de que el mensaje sea spam.
4. Aplicar un logaritmo para evitar inestabilidad numérica.
"""

def predict_text(text, verbose=False):
    """
    # Summary: Calculate spam/non-spam scores for the given text, \
               returning the prediction based on the scores.
    # Input: a text message comprised of different words
    # output: classification of the input message (spam or non-spam)
```

```
    """

    # Create a Python list of valid words that are present in both BoWs
    words_list = text.lower().split()
    valid_words = [word for word in words_list if word in train_spam_bow]

    # get the probabilities of each valid word showing up in respective BoWs
    spam_probs = [train_spam_bow[w] for w in valid_words]
    non_spam_probs = [train_non_spam_bow[w] for w in valid_words]

    # If verbose, print probabilities of seeing the word in respective classes
    if verbose:
        data_df = pd.DataFrame()
        data_df['word'] = valid_words
        data_df['spam_prob'] = spam_probs
        data_df['non_spam_prob'] = non_spam_probs
        data_df['ratio'] = [sp/nsp if nsp > 0 else np.inf \
                            for sp, nsp in zip(spam_probs, non_spam_probs)]
        print(data_df)


    # calculate spam score as sum of logs for all probabilities
    # log(P(word_1|SPAM)) + log(P(word_2|SPAM)) + ... \
    #      + log(P(word_n|SPAM)) + log(P(SPAM))
    spam_score = sum([np.log(p) for p in spam_probs]) + np.log(FRAC_SPAM_TEXTS)

    #calculate non-spam score as sum of logs for all probabilities
    # log(P(word_1|non-SPAM)) + log(P(word_2|non-SPAM)) + ... \
    #      + log(P(word_n|non-SPAM)) + log(P(non-SPAM))
    non_spam_score = sum([np.log(p) for p in non_spam_probs]) + np.log(1-FRAC_SPAM_TEXTS)

    #if verbose, report the two scores
    if verbose:
        print('Spam Score: %s'%spam_score)
        print('Non-Spam Score: %s'%non_spam_score)

    #if spam score is higher, mark the input text as spam
    if spam_score >= non_spam_score:
        return(True)
    else:
        return(False)

predict_text('congratulations you have won a car', verbose=True)

"""El texto 'congratulations you have won a car' es clasificado
correctamente como spam.

Ahora, usando los textos en el dataset de prueba, hacemos un pandas
Series que guarde los valores predichos por el algoritmo:
"""

predicted_values = [predict_text(text) for text in test_spam_df['text']]
predicted_values_series = pd.Series(predicted_values, index=test_spam_df.index)
predicted_values_series

"""Concatenamos los valores predichos al DataFrame con los valores reales:"""

test_spam_df_predicted = pd.concat([test_spam_df, predicted_values_series], axis=1)
```

```
test_spam_df_predicted

"""Obtenemos la matriz de confusión del modelo:"""

confusion_matrix_predicted =
confusion_matrix(test_spam_df_predicted['label'],test_spam_df_predicted.iloc[:,0])

ConfusionMatrixDisplay(confusion_matrix_predicted).plot()

"""Segun la información obtenida a partir de la matriz de confusión,
el algoritmo propuesto por Andrew Han usando Bag-of-Words y Naive Bayes es 100% exacto. """
```