

CASO REAL IA

GRUPO A-1

```
In [1]: import pandas as pd
import spacy
nlp= spacy.load('es_core_news_lg')
```

2023-11-19 18:51:49.350369: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.

```
In [2]: df = pd.read_csv("spam.csv")
df
```

Out[2]:

| | label | text |
|------|-------|---|
| 0 | ham | Go until jurong point, crazy.. Available only ... |
| 1 | ham | Ok lar... Joking wif u oni... |
| 2 | spam | Free entry in 2 a wkly comp to win FA Cup fina... |
| 3 | ham | U dun say so early hor... U c already then say... |
| 4 | ham | Nah I don't think he goes to usf, he lives aro... |
| ... | ... | ... |
| 5567 | spam | This is the 2nd time we have tried 2 contact u... |
| 5568 | ham | Will Ì_ b going to esplanade fr home? |
| 5569 | ham | Pity, * was in mood for that. So...any other s... |
| 5570 | ham | The guy did some bitching but I acted like i'd... |
| 5571 | ham | Rofl. Its true to its name |

5572 rows x 2 columns

```
In [3]: print(df.duplicated().sum())
```

414

```
In [4]: print(df.isnull().sum())
```

```
label    0
text     0
dtype: int64
```

```
In [5]: df = df.drop_duplicates()
df = df.reset_index(drop=True)
```

Al juntar

2 Preprocesamiento de datos

```
In [6]: strings = df.text
```

```
In [7]: lista_texto = strings.to_list()
```

```
In [8]: import re

def eliminar_emoticonos(comentario):
    # Utiliza una expresión regular para encontrar y eliminar los emoticonos
    comentarios_sin_emoticonos = re.sub(r'^\w\s', '', comentario)
    return comentarios_sin_emoticonos

comentarios_sin_emoticonos = []

for comentario in lista_texto:
    comentario_procesado = eliminar_emoticonos(comentario)
    comentarios_sin_emoticonos.append(comentario_procesado)

## 'comentarios_sin_emoticonos' es una lista que contiene los comentarios procesados
```

```
In [9]: cadena_resultante = ",".join(comentarios_sin_emoticonos)
```

```
In [10]: def limpiar_palabras_vacias(comment):
    # Procesar el texto con Spacy
    doc = nlp(comment)

    # Eliminar palabras vacías y convierte a minúsculas -> Lo pasamos ya toc
    palabras_filtradas = [token.text.lower() for token in doc if not token.is_stop]

    return " ".join(palabras_filtradas)
```

```
In [11]: coments_limpios = limpiar_palabras_vacias(cadena_resultante)
```

```
In [12]: # Dividir el texto en una lista usando la coma como separador
mi_lista = coments_limpios.split(",")
```

```
In [13]: ## elimino los números
def eliminar_numeros(cadena):
    return ''.join(caracter for caracter in cadena if not caracter.isdigit())

# Aplicar la función para eliminar números a cada elemento de la lista
mi_lista = [eliminar_numeros(elemento) for elemento in mi_lista]
```

```
In [14]: comentarios_preprocesados = []
```

```

for comentario in mi_lista:
    # Procesar el texto con spaCy
    doc = nlp(comentario)
    # Obtener lemas de las palabras
    lemmatized_tokens = []
    for token in doc:
        if not token.is_stop:
            lemmatized_tokens.append(token.lemma_)

    texto_procesado = ' '.join(lemmatized_tokens)
    comentarios_preprocesados.append(texto_procesado)

```

```

In [15]: print(len(df))
         print(len(comentarios_preprocesados)) ## chequeamos mismas dimensiones (por

```

```

5158
5158

```

```

In [16]: df= df[['label']]
         df['comentarios']= comentarios_preprocesados

```

3 Modelo TD-idf

```

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

         # Crear un vectorizador TF-IDF
         vectorizer = TfidfVectorizer()

         # Transformar los textos en vectores TF-IDF
         X_tfidf = vectorizer.fit_transform(comentarios_preprocesados)

         # Mostrar el vocabulario TF-IDF generado
         print("Vocabulario TF-IDF:", vectorizer.get_feature_names_out())

         # Mostrarlo en dataframe
         tfidf_df = pd.DataFrame(X_tfidf.toarray(), columns=vectorizer.get_feature_names_out())

```

```

Vocabulario TF-IDF: ['aa' 'aah' 'aaniye' ... 'ûï' 'ûïharry' 'ûò']

```

4.1 TD-idf (primeras 10 rows)

```

In [19]: tfidf_df.head(10)

```

Out [19]:

| | and | are | but | call | for | have | im | in |
|--|----------|-----|-----|----------|----------|----------|-----|----------|
| go until jurong point crazy available only in bugis n great world buffet cine there got amore wat | 0.000000 | 0.0 | 0.0 | 0.000000 | 0.000000 | 0.000000 | 0.0 | 1.000000 |
| ok lar joking wif oni | 0.000000 | 0.0 | 0.0 | 0.000000 | 0.000000 | 0.000000 | 0.0 | 0.000000 |
| free entry in wkly comp to win fa cup tkts st may text fa to to receive entry questionstd txt ratetcs apply overs | 0.000000 | 0.0 | 0.0 | 0.000000 | 0.000000 | 0.000000 | 0.0 | 0.407535 |
| dun say so early hor c already then say | 0.000000 | 0.0 | 0.0 | 0.000000 | 0.000000 | 0.000000 | 0.0 | 0.000000 |
| nah i dont think goes to usf lives around here though | 0.000000 | 0.0 | 0.0 | 0.000000 | 0.000000 | 0.000000 | 0.0 | 0.000000 |
| freemsg hey there darling its been weeks now and word back id like some fun you up for it still tb ok xxx std chgs to send å to rcv | 0.357415 | 0.0 | 0.0 | 0.000000 | 0.388644 | 0.000000 | 0.0 | 0.000000 |
| even my brother is not like to speak with they treat like aids patent | 0.000000 | 0.0 | 0.0 | 0.000000 | 0.000000 | 0.000000 | 0.0 | 0.000000 |
| as per your request melle melle oru minnaminunginte nurungu vettam has been set as your callertune for all callers press to copy your friends callertune | 0.000000 | 0.0 | 0.0 | 0.000000 | 0.302471 | 0.000000 | 0.0 | 0.000000 |
| winner as valued network customer you | 0.000000 | 0.0 | 0.0 | 0.499392 | 0.000000 | 0.485429 | 0.0 | 0.000000 |

and are but call for have im in

have been
selected to
receive a prize
reward to claim
call claim code kl
valid hours only

had your mobile
months or more r
entitled to
update to the
latest colour
mobiles with
camera for free
call the mobile
update co free
on

0.000000 0.0 0.0 0.357933 0.330411 0.000000 0.0 0.000000

4 Dividir en train y test

```
In [20]: from sklearn.model_selection import train_test_split

X = df["comentarios"]
y = df["label"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Crear un vectorizador TF-IDF para convertir el texto en características numéricas
vectorizer = TfidfVectorizer()
X_train = vectorizer.fit_transform(X_train)
X_test = vectorizer.transform(X_test)
```

5 Modelo bayesiano

```
In [ ]: from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix, classification_report

# Entrenamos un modelo de bayes ingenuo
model = MultinomialNB()
model.fit(X_train, y_train)

# Realizamos predicciones en el conjunto de prueba
y_pred = model.predict(X_test)

# Calcular el puntaje F1
f1_score = classification_report(y_test, y_pred)
print(f1_score)
```

Precision: En el contexto de la precisión mide la proporción de predicciones correctas de la categoría de spam con respecto al

total de predicciones de spam. En otras palabras, ¿cuántas de las predicciones de spam fueron correctas? Una precisión de 1.0 es que el modelo es muy bueno para predecir spam.

Por otro lado para ham, es decir, comentarios que son escritos por personas, hay una precisión de 0.94 que es muy buen score.

Sabiendo que la fórmula es: Verdaderos positivos/(Verdaderos positivos + falsos positivos), podemos determinar que no hubo ningún falso positivo para spam

Recall: El recall en este contexto mide la proporción de casos reales de spam que fueron correctamente identificados por el modelo. Es decir, ¿cuántos de los casos reales de spam el modelo logró capturar en sus predicciones? En este caso el score es de 0.63, lo cual indica que el modelo identificó correctamente como spam el 63% de los mensajes en comparación al número total de mensajes de spam.

Por otro lado para el ham, tenemos un score de 1.00, es decir, de los mensajes reales de ham, el modelo identificó bien todos ellos. Es decir el modelo siempre identifica bien los mensajes de ham, pero falla más con los mensajes de spam.

La fórmula de recall es: Verdaderos positivos/(Verdaderos positivos + falsos negativos).

Accuracy: el score de accuracy indica el porcentaje de predicciones que el modelo acertó. Su fórmula es: N° de predicciones correctas/ Total de predicciones.

Para este modelo obtenemos un 0.95, lo cual es muy buen resultado

Puntaje F1 (F1-score): El puntaje F1 combina tanto la precisión como el recall en una sola métrica. Es útil cuando queremos equilibrar la precisión y el recall. Un alto score F1 indica un equilibrio entre la capacidad del modelo para predecir "spam" y su capacidad para identificar correctamente los casos de "spam". Viceversa para ham

Para spam, el modelo tiene un F1 de 0.77 que no es un mal resultado, y para ham tiene un score de 0.97, es decir predice e

identifica muy bien los mensajes de ham

```
In [22]: # Matriz de confusión
conf_matrix = confusion_matrix(y_test, y_pred)
conf_matrix
```

```
Out[22]: array([[892,  0],
               [ 52,  88]])
```

El 892 representa los verdaderos positivos, que son los casos positivos (spam) que el modelo clasificó correctamente como positivos.

El 0 representa los falsos negativos, que son los casos positivos reales que el modelo clasificó incorrectamente como negativos.

El 52 representa los falsos positivos, que son los casos negativos reales que el modelo clasificó incorrectamente como positivos.

El 88 representa los verdaderos negativos, que son los casos negativos (ham) que el modelo clasificó correctamente como negativos.

6 Modelo SVM

```
In [23]: from sklearn.svm import SVC

# Entrenamos el clasificador SVM
svm_classifier = SVC()
svm_classifier.fit(X_train, y_train)
```

```
Out[23]: ▼ SVC
SVC()
```

```
In [24]: # Realizamos predicciones en el conjunto de prueba
y_pred = svm_classifier.predict(X_test)

f1_score = classification_report(y_test, y_pred)
print(f1_score)
```

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| ham | 0.99 | 1.00 | 0.99 | 892 |
| spam | 0.98 | 0.91 | 0.94 | 140 |
| accuracy | | | 0.99 | 1032 |
| macro avg | 0.99 | 0.95 | 0.97 | 1032 |
| weighted avg | 0.99 | 0.99 | 0.99 | 1032 |

Precision: en ambos el modelo predijo muy bien los casos de ham y spam(0.99 y 0.98)

Recall: en ambos casos el modelo identificó muy bien que casos eran de ham y cuáles de spam. Por lo que vemos, a los modelos no les cuesta identificar los casos de ham (puntuación perfecta 1.0), aunque si que tiene algunos fallos con los de spam (0.91). Aún así, este modelo identifica mucho mejor el spam que el modelo bayesiano

El accuracy es casi perfecto (0.99). Esto significa que el 99% de las predicciones del modelo fueron correctas

Siguiendo los comentarios anteriores, el modelo tiene muy buenos scores para ham y spam, es decir predice e identifica muy bien los casos de ham y spam. El modelo SVM muestra ser mucho mejor que el modelo bayesiano.

```
In [25]: # Matriz de confusión
conf_matrix = confusion_matrix(y_test, y_pred)

print("Matriz de Confusión:")
print(conf_matrix)
```

```
Matriz de Confusión:
[[890   2]
 [ 13 127]]
```

El 890 representa los verdaderos positivos, que son los casos positivos (spam) que el modelo clasificó correctamente como positivos.

El 2 representa los falsos negativos, que son los casos positivos reales que el modelo clasificó incorrectamente como negativos.

El 13 representa los falsos positivos, que son los casos negativos reales que el modelo clasificó incorrectamente como positivos.

El 127 representa los verdaderos negativos, que son los casos negativos (ham) que el modelo clasificó correctamente como negativos.

Vemos que este modelo ha acertado mas verdaderos negativos que el modelo bayesiano

7 Modelo Árboles de decisión

```
In [26]: from sklearn.tree import DecisionTreeClassifier
# Entrenamos el clasificador de árbol de decisión
decision_tree_classifier = DecisionTreeClassifier()
decision_tree_classifier.fit(X_train, y_train)
```

```
Out[26]: ▼ DecisionTreeClassifier
DecisionTreeClassifier()
```

```
In [27]: # Realizamos predicciones en el conjunto de prueba
y_pred = decision_tree_classifier.predict(X_test)

f1_score = classification_report(y_test, y_pred)
print(f1_score)
```

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| ham | 0.98 | 0.97 | 0.98 | 892 |
| spam | 0.82 | 0.90 | 0.86 | 140 |
| accuracy | | | 0.96 | 1032 |
| macro avg | 0.90 | 0.93 | 0.92 | 1032 |
| weighted avg | 0.96 | 0.96 | 0.96 | 1032 |

Precision: Este modelo predice muy bien el ham, pero no tanto el spam, en comparación a los modelos anteriores que siempre lo predecían perfectamente.

En cuanto al recall, este modelo identifica bastante bien spam (0.9) y muy bien el ham (0.97).

Tiene muy buen score de accuracy 0.96, aunque no supera al SVM.

Podríamos decir que mejor que el modelo bayesiano, pero sigue sin superar al modelo SVM.

En el F1 score tiene muy buenos resultados.

```
In [28]: # Matriz de confusión
conf_matrix = confusion_matrix(y_test, y_pred)

print("Matriz de Confusión:")
print(conf_matrix)
```

```
Matriz de Confusión:
[[865  27]
 [ 14 126]]
```

El 865 representa los verdaderos positivos, que son los casos positivos (spam) que el modelo clasificó correctamente como positivos.

El 27 representa los falsos negativos, que son los casos positivos reales que el modelo clasificó incorrectamente como negativos.

El 14 representa los falsos positivos, que son los casos negativos reales que el modelo clasificó incorrectamente como positivos.

El 126 representa los verdaderos negativos, que son los casos negativos (ham) que el modelo clasificó correctamente como negativos.

No hay muchas diferencias en cuanto al modelo SVM

Conclusión

Podríamos decir que el modelo que mejor identifica y predice es el modelo SVM, además de tener la mejor accuracy.

Por otro lado el modelo bayesiano predice mejor que el modelo de árboles de decisión, y viceversa para la identificación de spam y ham