Documento Análisis de Proyecto 1 Etapa 1

Jhoan Sebastian Sánchez Suarez
Juan Diego Sánchez
Pablo Méndez Morales

Departamento Ing. De Sistemas y Computación
Universidad de los Andes
ISIS3301- Inteligencia de Negocios

22/02/2025

Índice

1. Documentación del proceso de aprendizaje automático.

Se puede encontrar el OWNML Machine Learning Canvas en formato pdf dentro de la wiki del proyecto en el archivo Machine Learning Canvas (v1.2).pdf.

a. Tarea de aprendizaje:

   Es aprendizaje supervisado, más específicamente un modelo de clasificación con datos etiquetados. Los posibles resultados son resultados binarios siendo estos: 0 si es una noticia falsa y 1 si es una noticia verdadera. Los resultados se obtienen en minutos.

b. Decisiones:

   Los datos tienen que estar en español y ser noticias con un título y descripción, estos dos se deben concatenar y deben pasar por un proceso de token y "lemming". Luego estos tokens deben ser vectorizados para que ya los modelos de aprendizaje puedan retornar resultados.

c. Propuesta de valor:

   Los beneficiarios serian la sociedad en general, más en específico hispanohablante. Otro subgrupo serían las instituciones gubernamentales. El problema por resolver es la polarización y manipulación de la opinión pública a nivel internacional y nacional debido a la desinformación y noticias falsas.

   Un posible riesgo para los usuarios es una excesiva seguridad sobre el modelo de clasificación y que se tomen como correctas todas sus respuestas, teniendo en cuenta que el modelo puede no estar preparado para la variabilidad de la desinformación y pueda tener sesgos no detectados.

d. Fuentes de datos:

Un grupo de académicos se encargaron de la recolección y creación de estos datos extraídos de periódicos en línea en español. Los datos originales fueron extraídos de los periódicos 'Público,' 'La Marea,' y 'El Común.' y tienen licencia ATTRIBUTION 4.0 INTERNATIONAL. Los datos son útiles para cumplir el objetivo ya que son periódicos reales que podemos tomar de referencia y los datos fueron extraídos de forma que el procesamiento de texto es fácil y estandarizado.

e. Simulación de impacto:

El costo de las decisiones incorrectas incluye la perpetuación de la desinformación y sus consecuencias sociales y el benefició de las decisiones correctas es el deceso de la desinformación y fortalecimiento de la confianza en el periodismo en línea. Existen restricciones de equidad ya que el modelo no debe presentar sesgos y debe mantenerse objetivo para no apoyar o atacar injustamente a grupos y su postura.

f. Aprendizaje (Uso del modelo):

El uso del modelo es por lotes, pueden ingresar cualquier cantidad de estos y se debe ejecutar cada vez que se quiera identificar si una noticia es real o falsa.

g. Construcción del modelo:

Se solicitaron 3 modelos con algoritmos distintos, para posteriormente ser comparados y elegir el mejor de estos como modelo definitivo. El modelo no aspira a actualizarse por el momento. Y el tiempo de desarrollo del modelo es de 15 días, siendo la liberación de datos el 7 de febrero y la entrega el 22 de febrero.

h. Ingeniería de características:

Las variables de la fuente datos usadas para el modelo son la etiqueta, el titulo y la descripción de la noticia, que se concatenaron en otra columna y posteriormente pasan por un proceso de tokenización y un proceso de lematización. Por último, estos tokens se vectorizaron para generar los distintos modelos.

2. **Entendimiento y preparación de los datos (a nivel de código y análisis).**

a. Variables existentes:

Al cargar el csv se encontro que hay las siguientes 5 columnas:

ID: Una columna que debería tener los identificadores de cada noticia.

Label: Una columna con dos valores posibles (0 = Noticia falsa, 1= Noticia verdadera) las cuales son las etiquetas para la clasificación.

Título: El título de la noticia en formato “string”.

Descripción: La descripción de la noticia en formato “string”.

Fecha: La fecha de la publicación en formato “string”.

b. Completitud:

Se encontro registros vacíos únicamente en la columna Titulo, posteriormente decidiremos que hacer con ellos.

```
df_decision_tree.isna().sum()
```

```
ID             0
Label          0
Titulo        16
Descripcion    0
Fecha          0
```

c. Consistencia:

Todas las columnas son consistentes con sus datos, pero más específicamente la columna “label” es consistente ya que no hay ningún valor distinto de 0 y 1.

```
df_decision_tree["Label"].unique()
```

```
array([1, 0], dtype=int64)
```

d. Perfilamiento de variables:

Se verifico que la columna ID está llena “ID” en formato string y no tiene las verdaderas ID por lo tanto es necesario eliminarla para evitar ruido. Con respecto a la fecha no es útil pues es una variable descriptiva que no se puede adaptar a ningún modelo, ya que lo importantes es el procesamiento de textos. En conclusión, se trabajarán los modelos con las columnas título, descripción y label, ya que el titulo y la descripción se pueden procesar para ser usados en el modelo y este es el objetivo por cumplir y la columna label tiene el etiquetado necesario para el modelo de clasificación.

```
df_decision_tree= df_decision_tree.drop(columns=["ID","Titulo","Descripcion", "Fecha"])
df_decision_tree
```

	Label	d
0	1	,',the,guardian,',ir,con,sanchez,:,',europa,ne...
1	0	,revelan,que,el,gobierno,negocio,el,liberacion...
2	1	,el,',ahora,o,nunca,',de,joan,fuster,sobre,el,...
3	1	,iglesias,alentar,a,yolanda,diaz,,,,erc,y,eh,bi...

Adicionalmente, se observa que la distribución de los datos en cuanto a noticias falsas y no falsas es poco equitativa dejando un total de aproximadamente 58% de noticias falsas y del 42% de noticias no falsas.

```
df['Label'].value_counts()
✓ 0.0s
Label
1    33158
0    23905
Name: count, dtype: int64
```

e. Preparación de los datos:

Considerando que faltan algunos registros en títulos, para no eliminar los registros por completo y hacerlos comparables a los demás registros se decidió crear una nueva columna que concatenara títulos y descripciones para hacer el procesamiento de textos de esta nueva columna y hacer el modelo a partir de estos.

```
# corregimos los valores nulos y unimos las columnas 'Titulo' y 'Descripcion'
df['Titulo']=df['Titulo'].fillna('')
df['preparados']=df['Titulo']+df['Descripcion']
df
```

	ID	Label	Titulo	Descripcion	Fecha	preparados
0	ID	1	'The Guardian' va con Sánchez: 'Europa necesit...	El diario británico publicó este pasado jueves...	02/06/2023	'The Guardian' va con Sánchez: 'Europa necesit...
1	ID	0	REVELAN QUE EL GOBIERNO NEGOCIO LA LIBERACIÓN ...	REVELAN QUE EL GOBIERNO NEGOCIO LA LIBERACIÓN ...	01/10/2023	REVELAN QUE EL GOBIERNO NEGOCIO LA LIBERACIÓN ...
2	ID	1	El 'Ahora o nunca' de Joan Fuster sobre el est...	El valencianismo convoca en Castelló su fiesta...	25/04/2022	El 'Ahora o nunca' de Joan Fuster sobre el est...

También se decidieron normalizar los datos ya que algunos caracteres que no están en Unicode entorpecen el relacionamiento de textos.

```python
def normalizar(texto):
    # todo a minúsculas
    texto = texto.lower()

    # tildes y diacríticas
    texto = unidecode(texto)

    return texto
```

```
0        ' the guardian ' ir con sanchez : ' europa nec...
1       revelan que el gobierno negocio el liberacion ...
2       el ' ahora o nunca ' de joan fuster sobre el e...
3       iglesias alentar a yolanda diaz , erc y eh bil...
4       puigdemont : ' no ser ninguno tragedia uno rep...
```

Para poder trabajar mejor con los textos era necesario tokenizar y lematizar los textos de la columna nueva con los títulos y descripciones. Aquí se tomaron 2 caminos ya que se usaron 2 librerías distintas, el primer camino uso la librería Spacy, la cual se usó para los algoritmos 1 y 2 que se encontraran en el siguiente punto y el segundo uso la librería Stanza, que se usó para el algoritmo 3.

Esto debido a que el uso de la librería Stanza requiere más tiempo para el procesamiento de texto y cada integrante eligio que algoritmo le parecia más eficiente en cuanto a tiempo versus resultados.

a. Preparación Spacy:
   Primero se tokenizaron las palabras con ayuda de cláusulas de lenguaje natural.

```python
def tokenizar(texto):
    # dividir el texto en palabras
    palabras = re.findall(r'\b\w+\b', texto)
    return palabras
```

```
0        [the, guardian, ir, con, sanchez, europa, nece...
1       [revelan, que, el, gobierno, negocio, el, libe...
2       [el, ahora, o, nunca, de, joan, fuster, sobre,...
3       [iglesias, alentar, a, yolanda, diaz, erc, y, ...
4       [puigdemont, no, ser, ninguno, tragedia, uno, ...
```

   A continuación, se cargó el lemmatizer de Spacy y se uso para lematizar las palabras ya tokenizadas.

```
nlp = spacy.load('es_core_news_sm')
```

```
def lemmatizer(text):
    doc = nlp(text)
    return [word.lemma_ for word in doc]
```

Luego se usó la librería Spacy para eliminar las “stop words” que no aportan valor a los textos, ya que normalmente son artículos, conectores y preposiciones.

```
stopwords_espanol = stop_words.STOP_WORDS
#Se usa Spacy porque tiene un "vocabulario" más amplio que NLTK
#Genera un mejor resultado debido a esto

def stopwords_espanol_remove(line):
    new_line=[]
    for word in line:
        if word not in stopwords_espanol:
            new_line.append(word)
    return new_line
```

```
0       [the, guardian, sanchez, europa, necesitar, ap...
1       [revelan, gobierno, negocio, liberacion, mirel...
2       [joan, fuster, estatuto, valenciano, cumplir, ...
3       [iglesias, alentar, yolanda, diaz, erc, eh, bi...
4       [puigdemont, tragedia, repeticion, elecciones,...
```

Despues, se convirtieron las palabras en formato numérico a escrito con el fin de relacionarlas más fácil.

```
def num_to_word(line):
    new_line=[]

    for word in line:
        if re.fullmatch(r'\d+',word):
            word=float(word)
            new_line.append(num2words.num2words(word, lang = 'es_CO'))
        else:
            new_line.append(word)
    return new_line
```

Por último, se creó una nueva variable que midiera el número de palabras tokenizadas que tenía el texto y se transforma la lista de palabras tokenizadas a una cadena de palabras tokenizadas.

```python
def preprocesamiento():
    df['preparados'] = df['preparados'].apply(normalizar)
    df['preparados'] = df['preparados'].apply(tokenizar)
    df['preparados'] = df['preparados'].apply(stopwords_espanol_remove)
    df['preparados'] = df['preparados'].apply(num_to_word)
    df['tamanio']=df['preparados'].apply(len)
    df['preparados'] = df['preparados'].apply(lambda x: ' '.join(x))
    return df
```

b. Preparación Stanza:

Primero se carga el modelo de Stanza y se aplica a la columna que concatena los títulos y descripciones, pero antes de tokenizar a diferencia de los datos preparados con Spacy. Y luego se guarda en un csv para que cada vez que se corra en notebook no se tenga que lematizar de nuevo, ya que este proceso tarda alrededor de una hora. Esto devuelve el texto ya lematizado en una lista de tokens, que luego toca transformar en string para evitar errores en la exportación y carga posterior del csv.

```python
nlp = stanza.Pipeline(lang="es\", processors="tokenize,lemma")
   def tokenizar(linea):
      # dividir el texto en palabras

      texto_procesado = nlp(linea)
      lemmas = [word.lemma for sent in texto_procesado.sentences for word in sent.words]
      return lemmas

df['d']=df['d'].apply(tokenizar)
df.to_csv('dataframe_lematizado.csv',sep=";", index=False)
```

Ya luego se carga de nuevo el csv exportado en el punto anterior y sobre este se hacen las transformaciones a continuación.

Se verificaron registros duplicados completos y se eliminan.

```python
df_decision_tree[df_decision_tree.duplicated()]
```

	ID	Label	
2820	ID	1	La investigación policial por los balazos
2865	ID	1	El Constitucional avala la condena por se
2981	ID	0	Escándalo de corrupción salpica a líderes
3335	ID	1	Más Madrid con Mónica García se proyecta
3473	ID	1	La Fiscalía vincula al emérito con las cu
...	...	...	
56698	ID	1	Un periodista refugiado palestino denunci
56766	ID	1	Los técnicos municipales que denunciaron a
56931	ID	1	La extrema derecha busca sacar rédito po
56962	ID	1	Proponen convertir el pazo de Meirás en u
57018	ID	1	El Supremo confirma la condena a un padre

445 rows × 6 columns

```
df_decision_tree=df_decision_tree.drop_duplicates(keep = "first", inplace= False,ignore_index =True)
df_decision_tree
```

Luego se revisaron los duplicados parciales y se eliminaron otra vez. En realidad, aunque estos fueran duplicados parciales, la comparación se hizo con la columna de datos tokenizados y esta al ser la variable más útil para el modelo implicaría ruido que hay que limpiar.

```
df_decision_tree[df_decision_tree.duplicated(subset = 'd', keep = False)].sort_values('d')
```

	ID	Label	Titulo	Descripcion	Fecha	
20325	ID	0	El primer ministro australiano, Scott Morrison...	En un discurso en el Parlamento, Morrison ha r...	08/02/2022	,el,primero,mir
46410	ID	1	El primer ministro australiano, Scott Morrison...	En un discurso en el Parlamento, Morrison ha r...	08/02/2022	,el,primero,mir
1743	ID	0	Escándalo de corrupción salpica a líderes sind...	Un escándalo de corrupción ha salpicado a líde...	06/05/2023	,escanda,el,de,
44567	ID	0	Escándalo de corrupción salpica a líderes sind...	Un escándalo de corrupción ha salpicado a líde...	27/05/2023	,escanda,el,de,
38804	ID	0	Escándalo de malversación de fondos sacude a u...	Un escándalo de malversación de fondos ha sacu...	25/02/2023	,escanda,el,de,ma
45303	ID	0	Escándalo de malversación de fondos sacude a u...	Un escándalo de malversación de fondos ha sacu...	03/02/2023	,escanda,el,de,ma
8788	ID	0	Polémica por la reforma de la ley de protecció...	La propuesta de reforma de la ley de protecció...	28/01/2022	,polemica,por,el
48496	ID	0	Polémica por la reforma de la ley de protecció...	La propuesta de reforma de la ley de protecció...	27/01/2022	,polemica,por,el
9999	ID	0	Ruptura inesperada de coalición provoca crisis...	La inesperada ruptura de una coalición de gobi...	12/10/2022	,ruptura,inespera
24359	ID	0	Ruptura inesperada de coalición provoca crisis...	La inesperada ruptura de una coalición de gobi...	14/10/2022	,ruptura,inespera

Teniendo en cuenta que las variables necesarias para construir el modelo solo son la etiqueta y los tokens se decidió eliminar el resto de las columnas.

```
df_decision_tree= df_decision_tree.drop(columns=["ID","Titulo","Descripcion", "Fecha"])
df_decision_tree
```

	Label	d
0	1	,',the,guardian,',ir,con,sanchez,:,',europa,ne...
1	0	,revelan,que,el,gobierno,negocio,el,liberacion...
2	1	,el,',ahora,o,nunca,',de,joan,fuster,sobre,el,...
3	1	,iglesias,alentar,a,yolanda,diaz,,,erc,y,eh,bi...
4	0	,puigdemont,:,',no,ser,ninguno,tragedia,uno,re...
...	...	...
56608	1	,el,defensor,de,el,pueblo,reclamar,a,el,comuni...
56609	0	,el,equo,plantear,ceder,el,presidencia,de,el,c...
56610	1	,alberto,garzon,:,',que,el,borbones,ser,uno,la...
56611	1	,vox,exigir,entrar,en,el,gobierno,de,castilla,...
56612	1	,uno,300,persona,protestar,contra,el,visita,de...

Luego para normalizar los datos se pasaron de string a listas de string, a partir de estas listas se eliminaron strings de tamaño 1 ya que eran strings que habian quedado por error como signos de puntuación o letras solas.

```
def list_string(line: str)-> list:
    new_list= line
    if isinstance(line, str):
        new_list=line.split(sep=",")
    return new_list

df_decision_tree["d"] = df_decision_tree["d"].apply(list_string)
df_decision_tree
```

	Label	d
0	1	[, ', the, guardian, ', ir, con, sanchez, :, '...
1	0	[, revelan, que, el, gobierno, negocio, el, li...

```
def chars(line: list)-> list:
    new_list = []

    for word in line:
        if len(word) > 1:
            new_list.append(word)

    return new_list

df_decision_tree["d"] = df_decision_tree["d"].apply(chars)
df_decision_tree
```

	Label	d
0	1	[the, guardian, ir, con, sanchez, europa, nece...
1	0	[revelan, que, el, gobierno, negocio, el, libe...

Luego se usó la libreria Spacy para eliminar las stop words que no aportan valor, esto debido a que Stanza no maneja las stop words y que usando Spacy los datos serian más parecidos a los otros que usaron los otros algoritmos.

```
stopwords_espanol = stop_words.STOP_WORDS

def stopwords_espanol_remove(line):
    new_line=[]
    for word in line:
        if word not in stopwords_espanol:
            new_line.append(word)
    return new_line


df_decision_tree['d']=df_decision_tree['d'].apply(stopwords_espanol_remove)
df_decision_tree
```

	Label	d
0	1	[the, guardian, sanchez, europa, necesitar, ap...
1	0	[revelan, gobierno, negocio, liberacion, mirel...

Por último, igual que en la otra preparación de datos se convirtieron los números a letras si es posible.

```
def num_to_word(line):
    new_line=[]

    for word in line:
        if re.fullmatch(r'\d+',word):
            word=float(word)
            new_line.append(num2words.num2words(word, lang = 'es_CO'))
        else:
            new_line.append(word)
    return new_line

df_decision_tree['d']=df_decision_tree['d'].apply(num_to_word)
df_decision_tree
```

	Label	d
0	1	[the, guardian, sanchez, europa, necesitar, ap...
1	0	[revelan, gobierno, negocio, liberacion, mirel...
2	1	[joan, fuster, estatuto, valenciano, cumplir, ...
3	1	[iglesias, alentar, yolanda, diaz, erc, eh, bi...
4	0	[puigdemont, tragedia, repeticion, eleccion, e...

3. Modelado y evaluación:

a. Algoritmo 1 (Random Forest):

El algoritmo de Random Forest lo implementó Juan Diego Sánchez. La decisión de utilizar este algoritmo se produjo debido a que es una evolución al visto en clase (árboles de decisión) y observé que era usado en varias ocaciones para detectar noticias falsas en proyectos más avanzados, por lo que me pareció interesante medir su desempeño en un proyecto más contenido.

De esta forma, la idea de este algoritmo es instanciar varios 'decision trees' y lograr usar todas sus predicciones para tomar una decisión. En primer lugar, usa (de manera aleatoria) distintos subconjuntos de las características de los datos de entrenamiento para generar los diferentes árboles y buscar relaciones. En principio, se dice que al crear varios árboles de manera adecuada se evita el overfitting, bias en los datos y otros problemas para las predicciones. No obstante, el resultado obtenido fue diferente.

En primer lugar, se realizó una partición de los datos de los textos procesados, de forma que dejamos el test con el 20% de los datos.

```
Training examples: 45650, testing examples 11413
```

Luego se revisó la proporción de datos, en la que se puede observar que el porcentaje de datos de notcias falsas no es tán diferente a un 50-50.

```
train_labels.value_counts()
```

```
[17]  ✓ 0.0s
```

```
...  Label
     1    26526
     0    19124
     Name: count, dtype: int64
```

```
test_labels.value_counts()
```

```
[18]  ✓ 0.0s
```

```
...  Label
     1    6632
     0    4781
     Name: count, dtype: int64
```

Luego se necesitaba crear la matriz de terminos. Para lo que se utilizaron diferentes Vectorizers, en la primera iteración un CountVectorizer y en la segunda un TfidfVectorizer. En esta decisión en un primer momento se observó que no hubo diferencia en las métricas pero sí en cuanto al tiempo que le llevó al fit del modelo.

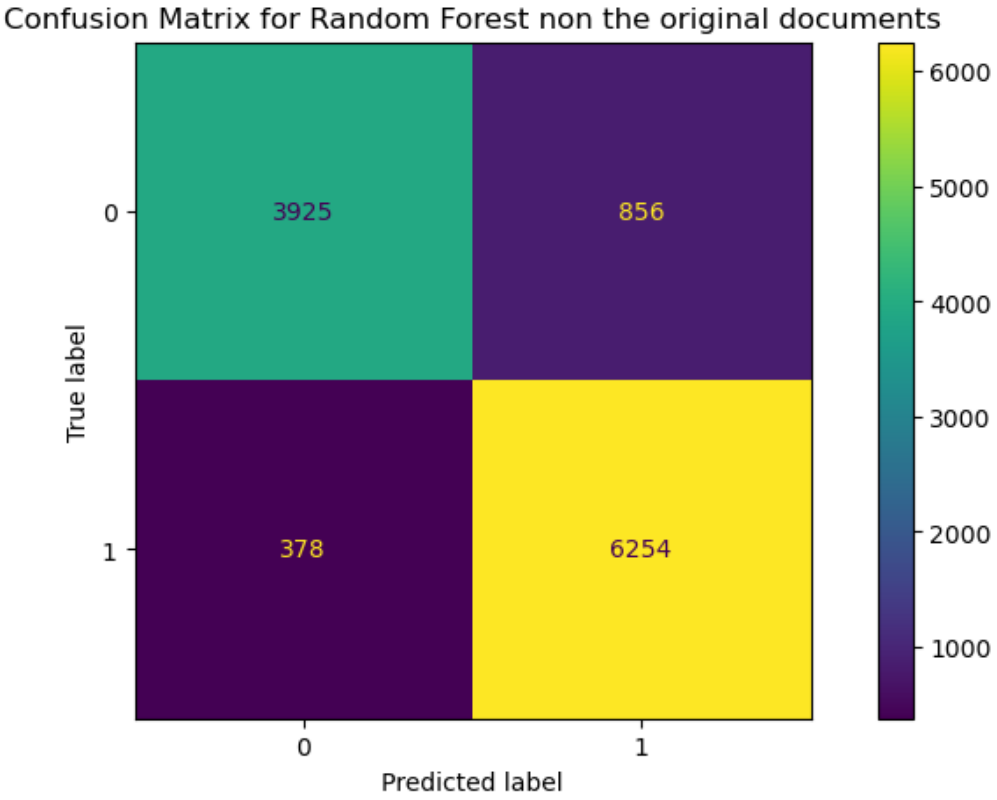
Así, el modelo 1 tuvo estas métricas:

```
# Probemos ahora este modelo sobre test.
y_pred_train = forest.predict(train_x)
y_pred_test = forest.predict(test_x)
print('Exactitud sobre entrenamiento: %.2f' % accuracy_score(train_labels, y_pred_train))
print('Exactitud sobre test: %.2f' % accuracy_score(test_labels, y_pred_test))
```

```
✓ 4.7s
Exactitud sobre entrenamiento: 1.00
Exactitud sobre test: 0.89
```

En esta métrica se observa que la exactitud del modelo con los datos de entrenamiento es 1 y con el conjunto de datos de test es 0.89, por lo que se pordría sospechar de overfitting en el modelo.

Adicionalmente, cuando observamos la matriz de confusión y algunas métricas asociadas:



	precision	recall	f1-score	support
0	0.91	0.82	0.86	4781
1	0.88	0.94	0.91	6632
accuracy			0.89	11413
macro avg	0.90	0.88	0.89	11413
weighted avg	0.89	0.89	0.89	11413

Se puede observar que en proporción se clasifican peor los con el label 0, es decir las noticias que no son falsas fueron clasificadas como falsas en mayor proporción que las noticias que son falsas en noticias falsas como se puede deducir de observar el recall y en el f1 score. Esto se puede explicar debido al ligero desbalanceo que hay en los datos de entrenamiento. Y al overfitting que tiene el modelo.

De esta forma, el objetivo en la siguiente iteración fue buscar el mejor modelo de forma que tenga un menor grado de overfitting.

Para este fin, en un primer momento se intentó realizar la busqueda con validación cruzada de varios hiperparámetros como el max_features, criterion y otros n_estimators. No obstante, debido a limitaciones en poder de computo, esta ejecución tomó mucho tiempo. Más de 2 horas. Por lo que no se puedo continuar con este intento. De esta forma, se modificó el número de estimadores en el parámetro n_estimators del RandomForestClassifier pasandolo de 100 a 200, ya que se tuvo la hipótesis de que al tener mayor número de arboles se podría combatir el overfitting.

```
# Probemos ahora este modelo sobre test.
y_pred_train = forest.predict(train_x)
y_pred_test = forest.predict(test_x)
print('Exactitud sobre entrenamiento: %.2f' % accuracy_score(train_labels, y_pred_train))
print('Exactitud sobre test: %.2f' % accuracy_score(test_labels, y_pred_test))
```

```
26]  ✓ 14.4s

Exactitud sobre entrenamiento: 1.00
Exactitud sobre test: 0.90
```

```
print(classification_report(test_labels, y_pred_test))
```

```
✓ 0.0s

              precision    recall  f1-score   support

           0       0.92      0.84      0.88      4781
           1       0.89      0.95      0.92      6632

    accuracy                           0.90     11413
   macro avg       0.90      0.89      0.90     11413
weighted avg       0.90      0.90      0.90     11413
```

No obstante, al realizar estos calculos se siguen presentando el overfitting y observamos que en este caso se extiende al conjunto de test, se siguen teniendo los mismos problemas que en el anterior modelo.

Con base en lo anterior y considerando las limitaciones de poder de computo y tiempo. Se buscaron otros resultados con otros parámetros como criterion o max_features. Y no se encontraron muchas diferencias en cuanto a overfitting y tiempos más extensos de ejecución. Por lo que se desistió de continuar buscando otros modelos. Y se concluye que se necesitan realizar

otro tipo de transformaciones a los datos para poder utilizar este algoritmo con un mejor rendimiento.

b. Algoritmo 2 (Regresión Logística):

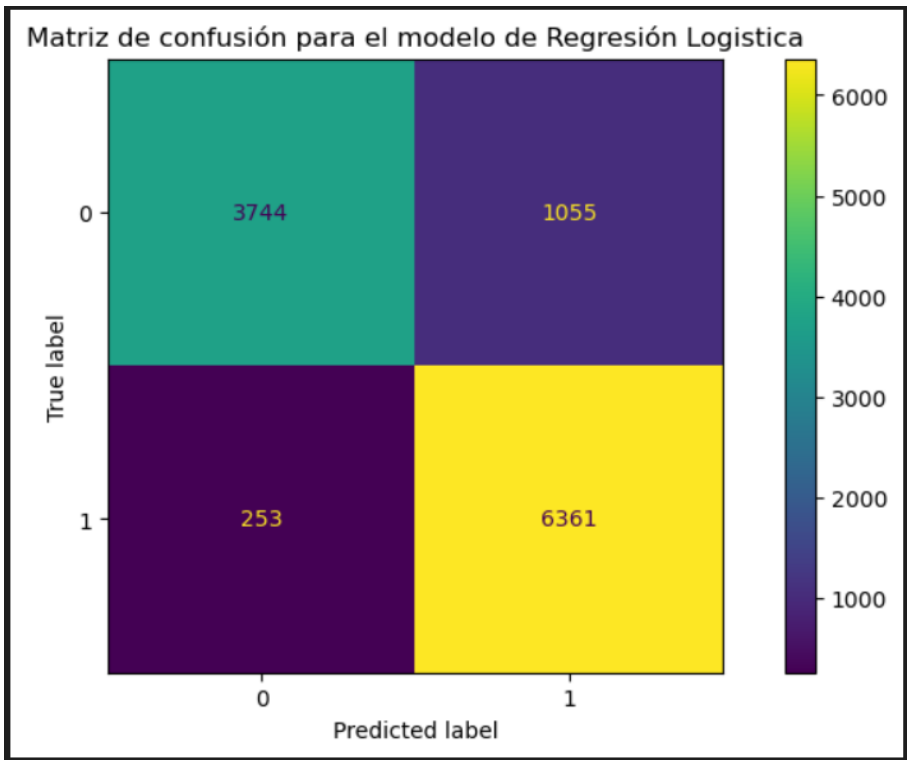
El algoritmo de Regresión Logística fue implementado por Pablo Méndez. La decisión de producir un modelo con este algoritmo fue debido a su similitud con los modelos de regresión lineal que ya se habían visto en clase y practicado en el laboratorio.

La extracción de características se hizo por medio del método TF-IDF, en el que el valor numérico asociado a una palabra es la multiplicación de la frecuencia de la palabra en un documento (TF) y el inverso de la frecuencia a través de múltiples documentos (IDF). De esta manera se disminuye el riesgo de que una palabra tenga un peso mucho más significativo que las otras o que los *outliers* afecten de manera significativa el modelo.

Se uso una proporción de 80-20 para entrenar y probar el modelo. Se escogieron estos valores siguiendo las buenas prácticas que nos han enseñado en clase, estos valores dieron como resultado la siguiente distribución:

```
Training examples: 45650, testing examples 11413
```

Después de entrenar y correr el modelo se obtuvo una exactitud sobre los datos de entrenamiento de 0.91 y una exactitud sobre los datos de entrenamiento de 0.89. Estos valores por si solos no dan mucha información, para poder usar una métrica más concreta se utiliza una matriz de confusión:



	precision	recall	f1-score	support
0	0.94	0.78	0.85	4799
1	0.86	0.96	0.91	6614
accuracy			0.89	11413
macro avg	0.90	0.87	0.88	11413
weighted avg	0.89	0.89	0.88	11413

Como se puede apreciar para las mayoria de las métricas se obtienen valores de alrededor del 85%. Se concluyo que fue un resultado razonable y de calidad.

c. Algoritmo 3 (Arboles de decisión):

El algoritmo de árboles de decisión lo implementó Jhoan Sebastian Sánchez Suarez. La decisión de producir un modelo con este algoritmo fue producto de que en clase este fue uno de los algoritmos que se usó para la clasificación binaria y se tenía un ejemplo con todo el procedimiento para tomar de referencia.

Antes de explicar la construcción del modelo hay que tener en cuenta que los datos usados en este modelo son los que fueron procesados con Stanza.

Antes de entrenar el modelo se decidió separar los datos de entrenamiento y de prueba en una proporción de 80%-20% y mantener las proporciones de los datos etiquetados como 0 y 1, la cual fue de 57.83% de la etiqueta 1 y el 42.17% restante para la etiqueta 0, la cual se considera una proporción equilibrada y válida para arboles de decisión. Si se hubiera hecho uso de shuffle, la proporción hubiera cambiado y hubiera sido riesgoso para el modelo.

```
df_decision_tree["Label"].value_counts()
```

```
Label
1    32737
0    23876
Name: count, dtype: int64
```

```
from sklearn.model_selection import train_test_split

df_decision_tree['textos'] = df_decision_tree['d'].apply(lambda x: ' '.join(x))

X_data, y_data = df_decision_tree['textos'],df_decision_tree['Label']


train_text, test_text, train_labels, test_labels = train_test_split(X_data, y_data, stratify=y_data, test_size=0.2)
print(f'Training examples: {len(train_text)}, testing examples {len(test_text)}')
```

```
Training examples: 45290, testing examples 11323
```

Los árboles de decisión solo pueden trabajar con variables numéricas, por lo que para poder trabajar con la lista de palabras tokenizadas se hizo uso de Vectorizer. El Vectorizer transforma las listas en múltiples columnas de frecuencia, que tienen como valor el número de veces que se repitió el valor y ese valor se convierte en la columna.

```
from sklearn.feature_extraction.text import CountVectorizer

vectoryzer= CountVectorizer()
train_x= vectoryzer.fit_transform(train_text)
test_x= vectoryzer.transform(test_text)
```

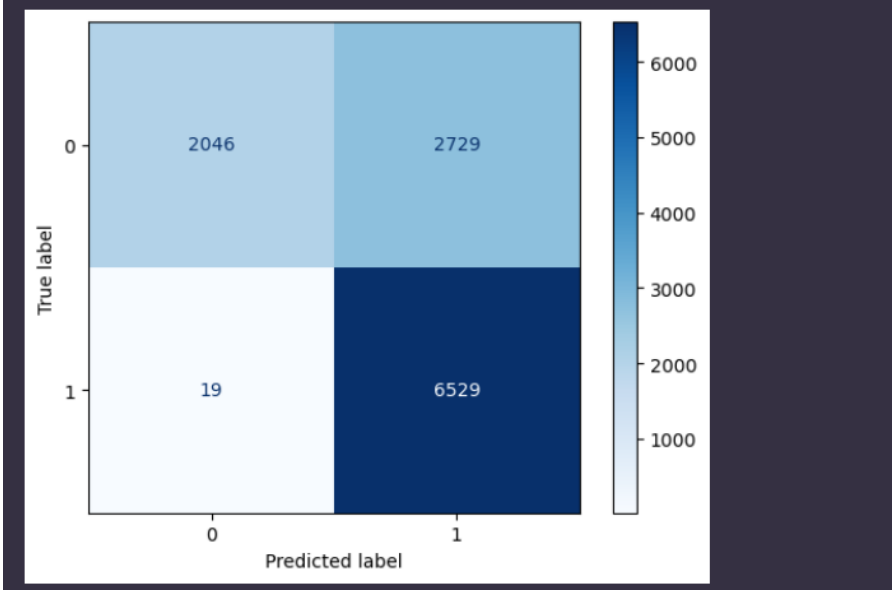
El primer modelo para acercarse a los datos se hizo con un árbol de decisión de altura máxima 4 y con el criterio de entropía, para entender la distribución de los datos.

```
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
arbol = DecisionTreeClassifier(criterion='entropy', max_depth = 4, random_state = 0)
arbol
```

```
DecisionTreeClassifier
ecisionTreeClassifier(criterion='entropy', max_depth=4, random_state=0)
```

El modelo arrojo la siguiente matriz de confusión y métricas de la matriz:

```
from sklearn.metrics import ConfusionMatrixDisplay
import matplotlib.pyplot as plt
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=arbol.classes_)
disp.plot(cmap=plt.cm.Blues)

plt.show()
```



Los resultados del primer acercamiento aparentan ser satisfactorios ya que las distintas métricas están por encima del 0.7, leyendo la gráfica de matriz de confusión, se puede decir que la meta actual es reducir el número de falsos positivos (Predicted 1, True label 0)

```
print('Exactitud: %.2f' % accuracy_score(test_labels, y_pred))
print("Recall: {}".format(recall_score(test_labels,y_pred)))
print("Precisión: {}".format(precision_score(test_labels,y_pred)))
print("Puntuación F1: {}".format(f1_score(test_labels,y_pred)))
```

```
Exactitud: 0.76
Recall: 0.9970983506414173
Precisión: 0.7052279109958954
Puntuación F1: 0.8261419714032646
```

```
print(classification_report(test_labels, y_pred))
```

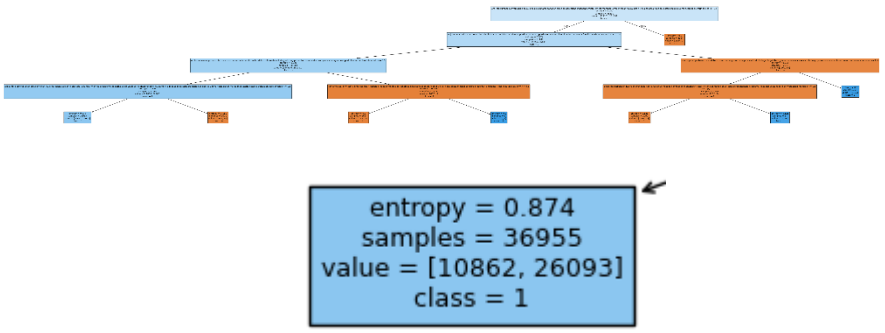
```
              precision    recall  f1-score   support

           0       0.99      0.43      0.60      4775
           1       0.71      1.00      0.83      6548

    accuracy                           0.76     11323
   macro avg       0.85      0.71      0.71     11323
weighted avg       0.83      0.76      0.73     11323
```

Antes de continuar se verifica la posibilidad de que haya overfitting probando el modelo sobre sus mismos datos de entrenamiento y prueba. Los resultados son positivos y no es probable que haya overfitting ya que no dio 1 el accuracy score del modelo sobre los datos de entrenamiento. Además se encuentran resultados muy parecidos para el accuracy score de los datos de prueba. Teniendo en cuenta que se mantuvieron las proporciones de las etiquetas y que no hay overfitting significa que es un modelo satisfactorio.

```
y_pred_train = arbol.predict(train_x)
y_pred_test = arbol.predict(test_x)
print('Exactitud sobre entrenamiento: %.4f' % accuracy_score(train_labels, y_pred_train))
print('Exactitud sobre test: %.4f' % accuracy_score(test_labels, y_pred_test))
```

```
Exactitud sobre entrenamiento: 0.7602
Exactitud sobre test: 0.7573
```

Revisando la entropía en el árbol de decisión se evidencia que dicho modelo tiene valores de entropía muy altos en uno de los nodos finales, que además tiene muchos registros, por lo tanto, se debe mejorar este aspecto en los siguientes modelos.



Una de las mejores técnicas para mejorar un árbol de decisión es la modificación de hiper-parametros y la cross-validation. Para trabajar con ello se hizo uso de KFold y GridSearchCV. Se decidió usar 10 particiones, ya que es un estándar conocido como 10-Fold-Cross-Validation.

```
from sklearn.model_selection import KFold
particiones = KFold(n_splits=10, shuffle=True, random_state = 0)
```

En el primer acercamiento se usó entropy porque es una métrica que facilita ver que tan dispersos están los datos y una altura de 4 para evitar el overfitting, ahora se probara con gini y unas alturas de mayor valor para evaluar los resultados y buscar los mejores hiper-parametros.

```
param_grid = {'criterion':['gini', 'entropy'],'max_depth':[4,6,8,10,12]}
```

```
from sklearn.model_selection import GridSearchCV
# Ahora utilizamos GridSearch sobre el grid definido y con 10 part
mejor_modelo = GridSearchCV(arbol, param_grid, cv=particiones)
# Ajuste del modelo
mejor_modelo.fit(train_x, train_labels)
```

El mejor algoritmo de acuerdo con el GridSearchCV usa gini y es de altura máxima 12.

```
mejor_modelo.best_params_
```

```
{'criterion': 'gini', 'max_depth': 12}
```

Se vio una mejora sustancial en el accuracy score.

```
# Obtener el mejor modelo.
arbol_final = mejor_modelo.best_estimator_
# Probemos ahora este modelo sobre test.
y_pred_train = arbol_final.predict(train_x)
y_pred_test = arbol_final.predict(test_x)
print('Exactitud sobre entrenamiento: %.4f' % accuracy_score(train_labels, y_pred_train))
print('Exactitud sobre test: %.4f' % accuracy_score(test_labels, y_pred_test))
```

```
Exactitud sobre entrenamiento: 0.8496
Exactitud sobre test: 0.8414
```

Debido a que esto fue resultado de usar gini y aumentar la altura máxima del árbol. Se busco repetir el GridSearchCV con mayores alturas y se encontró que el mejor modelo es con gini y altura máxima de 20.

```
param_grid = {'criterion':['gini', 'entropy'],'max_depth':[14,16,18,20]}
```

```
mejor_modelo2.best_params_
```

```
{'criterion': 'gini', 'max_depth': 20}
```

Comparando el accuracy score del modelo con altura máxima de 20 y el de altura máxima 12, la diferencia es nula y la complejidad aumenta drásticamente.

```
# Obtener el mejor modelo.
arbol_final2 = mejor_modelo2.best_estimator_
# Probemos ahora este modelo sobre test.
y_pred_train2 = arbol_final.predict(train_x)
y_pred_test2 = arbol_final.predict(test_x)
print('Exactitud sobre entrenamiento: %.4f' % accuracy_score(train_labels, y_pred_train2))
print('Exactitud sobre test: %.4f' % accuracy_score(test_labels, y_pred_test2))
```

```
Exactitud sobre entrenamiento: 0.8496
Exactitud sobre test: 0.8414
```

Ahora ningún gini supera el 0.459 lo cual se considera una métrica aceptable, pero si se mira que el accuracy de ambos modelos es igual y la única mejora es el gini en los últimos nodos, siendo el de altura 12 aun con resultados satisfactorios y el de altura 20 mejor pero no tan considerable. Por lo que el mejor modelo hasta el momento es el de altura máxima 12.

El anterior experimento abrió la posibilidad de que el GridSearchCV haya elegido el árbol de altura máxima 12 no fuera el más eficiente costo/beneficio. Por lo que se miraron las métricas del árbol usando gini a distintas alturas máximas menores a 12.

Después de comparar los accuracy score se consideró que la altura máxima que es más eficiente en cuestión de costo valor es la de altura 8, es una mejora considerable de la altura 4 y no está muy alejado su accuracy score de la altura 12 que es donde los accuracy score se quedan estancados.

```
arbol3 = DecisionTreeClassifier(criterion='gini', max_depth = 10, random_state = 0)
arbol3 = arbol3.fit(train_x,train_labels)
y_pred_train3 = arbol3.predict(train_x)
y_pred_test3 = arbol3.predict(test_x)
print('Exactitud sobre entrenamiento: %.4f' % accuracy_score(train_labels, y_pred_train3))
print('Exactitud sobre test: %.4f' % accuracy_score(test_labels, y_pred_test3))
```

```
Exactitud sobre entrenamiento: 0.8373
Exactitud sobre test: 0.8296
```

```
arbol3 = DecisionTreeClassifier(criterion='gini', max_depth = 8, random_state = 0)
arbol3 = arbol3.fit(train_x,train_labels)
y_pred_train3 = arbol3.predict(train_x)
y_pred_test3 = arbol3.predict(test_x)
print('Exactitud sobre entrenamiento: %.4f' % accuracy_score(train_labels, y_pred_train3))
print('Exactitud sobre test: %.4f' % accuracy_score(test_labels, y_pred_test3))
```

```
Exactitud sobre entrenamiento: 0.8223
Exactitud sobre test: 0.8170
```

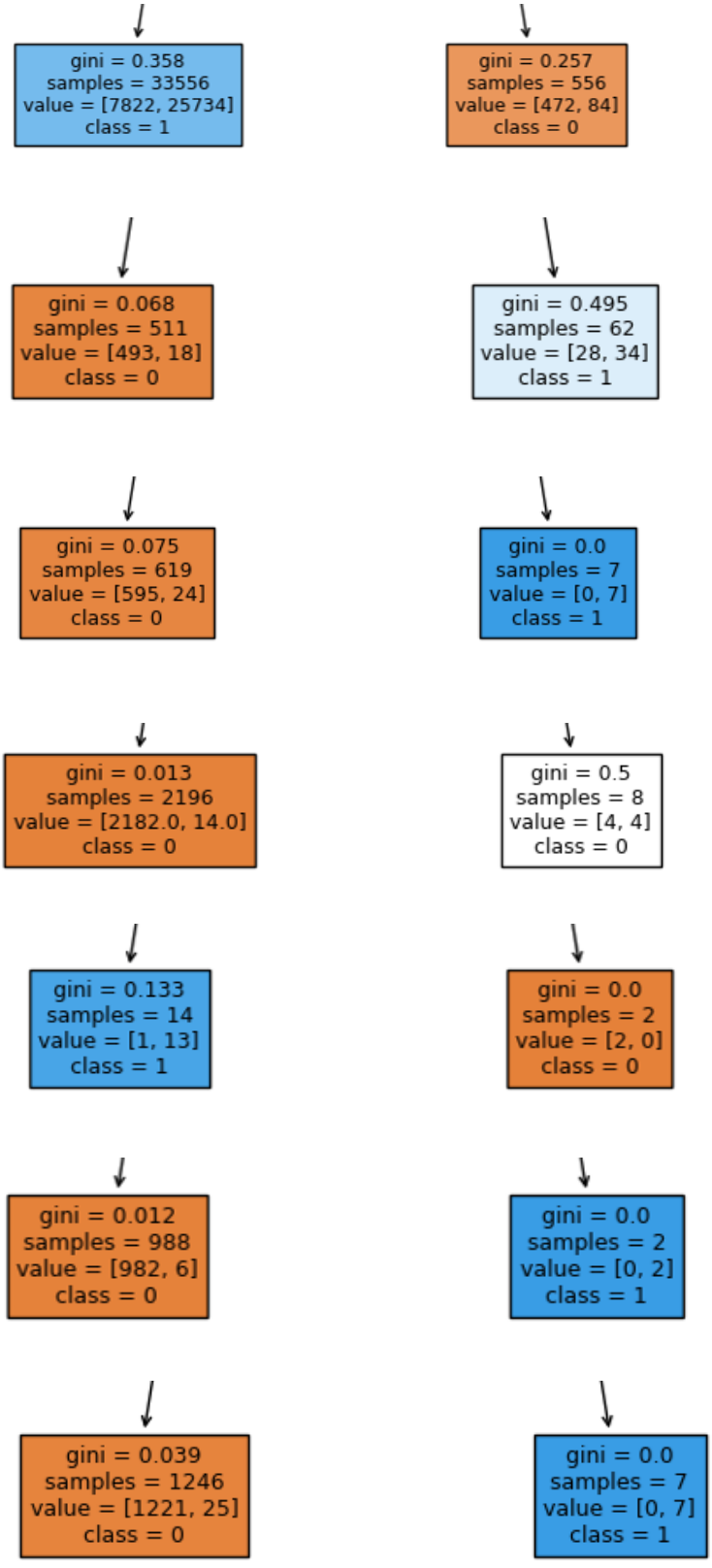
```
arbol3 = DecisionTreeClassifier(criterion='gini', max_depth = 6, random_state = 0)
arbol3 = arbol3.fit(train_x,train_labels)
y_pred_train3 = arbol3.predict(train_x)
y_pred_test3 = arbol3.predict(test_x)
print('Exactitud sobre entrenamiento: %.4f' % accuracy_score(train_labels, y_pred_train3))
print('Exactitud sobre test: %.4f' % accuracy_score(test_labels, y_pred_test3))
```

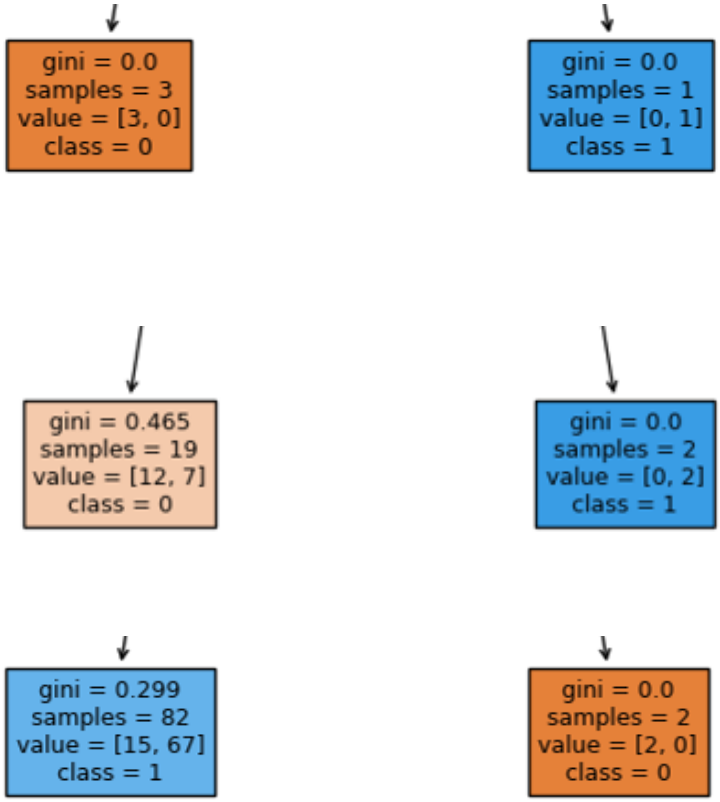
```
Exactitud sobre entrenamiento: 0.8020
Exactitud sobre test: 0.7961
```

```
arbol3 = DecisionTreeClassifier(criterion='gini', max_depth = 4, random_state = 0)
arbol3 = arbol3.fit(train_x,train_labels)
y_pred_train3 = arbol3.predict(train_x)
y_pred_test3 = arbol3.predict(test_x)
print('Exactitud sobre entrenamiento: %.4f' % accuracy_score(train_labels, y_pred_train3))
print('Exactitud sobre test: %.4f' % accuracy_score(test_labels, y_pred_test3))
```

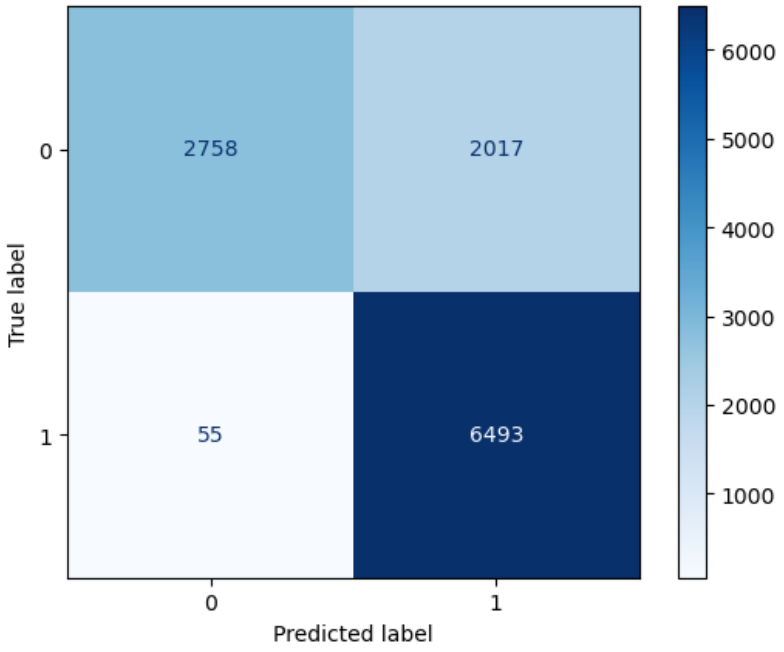
```
Exactitud sobre entrenamiento: 0.7665
Exactitud sobre test: 0.7626
```

El gini más alto es de 0.499 que es considerado aceptable y en el que solo se encuentran 44 samples, en cambio el nodo final donde hay más samples (33556) el gini es de solo 0.358, los demás ginis con una cantidad de muestras considerables (500) no tienen ginis más altos que los del nodo final donde hay más samples, por lo tanto, se considera que en cuestiones de indice gini es un modelo funcional.


gini = 0.358
samples = 33556
value = [7822, 25734]
class = 1
gini = 0.257
samples = 556
value = [472, 84]
class = 0
gini = 0.068
samples = 511
value = [493, 18]
class = 0
gini = 0.495
samples = 62
value = [28, 34]
class = 1
gini = 0.075
samples = 619
value = [595, 24]
class = 0
gini = 0.0
samples = 7
value = [0, 7]
class = 1
gini = 0.013
samples = 2196
value = [2182.0, 14.0]
class = 0
gini = 0.5
samples = 8
value = [4, 4]
class = 0
gini = 0.133
samples = 14
value = [1, 13]
class = 1
gini = 0.0
samples = 2
value = [2, 0]
class = 0
gini = 0.012
samples = 988
value = [982, 6]
class = 0
gini = 0.0
samples = 2
value = [0, 2]
class = 1
gini = 0.039
samples = 1246
value = [1221, 25]
class = 0
gini = 0.0
samples = 7
value = [0, 7]
class = 1



Se ve ahora una mejora considerable en la detección de 0 (Noticias falsas) aunque se sacrificó por una pequeña parte de los 1 detectados correctamente se fueron a falsos 0, en proporción hubo más ganancias que perdidas, comparándolo con el primero modelo de árboles de decisión hecho.



Los resultados ahora son satisfactorios se logró que todas las métricas estuvieran pon encima del 0.76, y si se comparan los resultados de este modelo con el primer modelo hay un aumento en exactitud de 0.06, un aumento de 0.06 en precisión, un aumento de 0.04 de F1 y por último un decremento de 0.006 en recall. Lo cual se considera un éxito y una mejora relevante al modelo.

```
print("Precisión: {}".format(precision_score(test_labels,y_pred_test3)))
print("Puntuación F1: {}".format(f1_score(test_labels,y_pred_test3)))
```

```
Exactitud: 0.82
Recall: 0.9916004886988393
Precisión: 0.7629847238542891
Puntuación F1: 0.8623987249302696
```

```
print(classification_report(test_labels, y_pred_test3))
```

```
              precision    recall  f1-score   support

           0       0.98      0.58      0.73      4775
           1       0.76      0.99      0.86      6548

    accuracy                           0.82     11323
   macro avg       0.87      0.78      0.79     11323
weighted avg       0.85      0.82      0.81     11323
```

Por último, se ve una mejora del 0.06 en accuracy y que los resultados son parecidos tanto sobre el modelo de entrenamiento como sobre las pruebas. Esto otra vez debido a que se mantienen las proporciones de datos en los datos de entrenamiento como los de prueba.

```
y_pred_train = arbol_final.predict(train_x)
y_pred_test = arbol_final.predict(test_x)
print('Exactitud sobre entrenamiento: %.4f' % accuracy_score(train_labels, y_pred_train))
print('Exactitud sobre test: %.4f' % accuracy_score(test_labels, y_pred_test))
```

```
Exactitud sobre entrenamiento: 0.8223
Exactitud sobre test: 0.8170
```

4. Resultados:

d. Modelo elegido:
El modelo elegido fue Regresión Logística ya que fue el que tuvo los mejores resultados en comparación con los otros modelos.

e. Análisis de palabras importantes:

5. Trabajo en equipo:

a. Distribución de roles y tareas:

Líder de proyecto: Jhoan Sebastián Sánchez Suárez - 202215911

Tareas:

Documentación del proceso de aprendizaje automático.

Algoritmo de árboles de decisión.

Video y diapositivas.

Líder de datos: Juan Diego Sánchez - 202214625

Tareas:

Entendimiento y preparación de los datos (a nivel de código y análisis).

Algoritmo de Random Forest.

Líder de analítica: Pablo Méndez Morales – 202210379

Tareas:

Resultados

Algoritmo de Regresión Logística.

b. Distribución de puntos:

Todos nos distribuimos equitativamente 33 puntos, porque cada integrante cumplió con una tarea de 20% y apoyo en las diversas tareas de cada sección. El trabajo entregado por cada uno fue de calidad y en función de horas cada uno invirtió más o menos 12 horas en sus tareas.

c. Reflexión individual:
   a. Jhoan Sebastián Sánchez Suarez:
      El resultado del trabajo fue satisfactorio, en cuanto a la carga considero que fue equitativa y el mayor desafío del trabajo fue el margen de tiempo limitado y los horarios disponibles para reuniones que se dificultaban. El uso de chatGPT fue limitado a asesor de código y entendimiento de las librerías usadas.
   b. Juan Diego Sánchez:

      El desarrollo del trabajo fue satisfactorio, se logró tener un entregabe a tiempo, se realizó una distribución de cargas de trabajo y crear un modelo cada uno. Creo que como equipo tenemos una oportunidad de mejora en cuanto al manejo del tiempo dedicado para el desarrollo de los modelos y el

preprocesamiento de datos. De todas formas, todos participamos activamente en este item y se logró un nivel aceptable. El uso de chatGPT se limitó a las sugerencias de copilot a la hora de escribir código.

c. Pablo Méndez Morales