Examen

March 29, 2022

Table of Contents

1 Parte teorica

Si tu modelo tiene un buen performance en el set de datos de entrenamiento pero no generaliza adecuadamente a nuevas observaciones, ¿Qué podría estar pasando? Nombre al menos dos posibles soluciones

R: El modelo estaria cometiendo overfitting * Una posible solución seria reducir la complejidad del modelo, ya sea reduciendo el numero de variables que pueden estar metiendo ruido o directamente en el modelo jugar con los hiperparametros que podrian hacer que se cometa este overfitting, por ejemplo en el arbol de decisión reducir el maximo de profundidad podria ayudar. * La segunda solución sera obtener mas data para entrenar ya que con mas datos al modelo se le hara mas complicado aprenderse todos los casos y asi ayudariamos a que el modelo generalize.

Para un ensamble de modelos ¿Cuál es la diferencia entre votación dura y suave?

R: La votacion dura toma la predicción mas votada por los modelos y la votación suave saca el promedio de las probabilidad que da cada modelo y en base a esa probabilidad promedio se hace la prediccion.

¿Qué hace que los ensambles llamados Extra-trees tengan más elementos aleatorios que un bosque aleatorio? ¿Qué beneficios tienen estos elementos aleatorios extra? ¿Qué modelo es más rápido de ejecutar, un Extra-tree o un bosque aleatorio?

R: Se usa un threshold aleatorio para cada variable que particiona un nodo, es decir en vez de buscar cual seria el mejor corte para dividir el nodo, esto se se hace de manera aleatoria.

Esto hace que se obtiene un mayor sesgo pero una menor varianza.

En terminos de velocidad, el Extra- trees es mas veloz por lo mismo que no estima cual seria la mejor forma de dividir el nodo.

¿Cuál es la idea fundamental detrás de un modelo de máquina de soporte vectorial?

R: Se busca separar el espacio de las variables utilizando hiperplanos y con una vecindad alrededor ,es decir, un margen.

¿A qué llamamos vector de soporte?

R: Es aquel vector que se encuentra en el limite del margen, por lo mismo se llama vector de soporte ya que esta limitando la region decisión

Suponga que está utilizando un modelo de regresión lineal con un set de datos aumentado utilizando un polinomio de grado M. Graficas las curvas de aprendizaje y notas un gran gap entre la curva de error en el set de entrenamiento y la curva de error en el set de validación. ¿Qué está pasando? Menciona al menos dos soluciones

R: El modelo esta sufriendo de overfitting posiblemente porque el grade del polinomio es muy garnde que se esta aprendiendo el set de entrenamiento. * La primera solución seria reducir el grado del polinomio, asi, buscaremos que no se aprenda el set de entrenamiento. * La segunda solución sera utilizar una regresion lineal lasso, asi las variables menos importantes y que podrian estar metiendo ruido sus coeficientes convergerian a 0.

Suponga que está utilizando un modelo de regresión Ridge. Notas que el error en el set de entrenamiento es muy similar al error en el set de validación. Sin embargo, dicho error es elevado. ¿Dirías que el modelo sufre de alto sesgo o alta varianza? ¿Deberías de incrementar el valor del parámetro regularizador alfa o reducirlo?

R: En este caso el modelo estaria sufriendo de underfitting, por lo que el modelo estaria sufiendo de un alto sesgo y baja varianza, ya que se menciono que el error de set de validación es similar al error en el set de entrenamiento.

Se recomendaria reducir el valor regularizador ya que podria ser muy estricto limitando el performance del modelo, se deberia jugar con este parametro para lograr sacarlo del underfitting teniendo cuidado de que no caiga en overfitting.

¿Por qué sería mejor usar? * Una regresión Ridge en lugar de una regresión lineal simple R: Una regresión Ridge seria mejor que una regresion lineal simple porque haria que los pesos W grandes sean penalizados y en cambios los pesos W pequeños aumenten. * Una regresión Lasso en lugar de una regresión Ridge Una regresión Lasso es mas estricta con las variables, ya que hace converger las variables poco importantes a 0, y en cambio la regresión Ridge solo reduce sus pesos w pero no necesariamente a 0 por lo que seria mas facil que la regresión Ridge caiga en overfitting que una regresión Lasso. * Una red elástica en lugar de una regresión Lasso R: Se mencionaron las ventajas de usar ambas regresiones, pero se quisiera buscar de aprovechar lo mejor de ambas, por que la red elástica aprovecha las ventajas de ambas penalizaciones en una combinación, haciendo asi que se pueda jugar con la regularización y poder hacer que generalice mejor.

Si un árbol de decisión presenta sobreajuste de datos, ¿Es una buena idea intentar decrementar el valor del parámetro max_depth (Piensa que estás utilizando el modelo de la librería Sklearn)?

R: Si, decrementar el valor del parametro max_depth ayudaria, tambien ayudaria aumentar el valor de min_samples_split, ya que al ser un valor muy pequeño haria que se crearan mas nodos ocasionando overfitting, o tambien se puede aumentar el parametro min_samples_leaf, que es el minimo de muestra que debe existir en algun nodo, de esta forma podemos hacer que el arbol deje de crecer si no cumple estas condiciones.

2 Parte practica

2.1 Analisis Exploratorio

2.1.1 Carga de paqueterias y del dataset

[1]:
```
import pandas as pd
import os
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pickle
# Nube de palabras
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator


# Vectorizar
import nltk
from unidecode import unidecode
from nltk.tokenize import word_tokenize
from nltk.tokenize import WordPunctTokenizer
from nltk.stem     import WordNetLemmatizer
from nltk.corpus   import stopwords
import spacy
from spacy.lang.es import Spanish
import re
import string
from unidecode import unidecode
from sklearn.feature_extraction.text import TfidfVectorizer
nltk.download('stopwords')
nlp = Spanish()

# Modelos
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn.naive_bayes import GaussianNB
from sklearn.neural_network import MLPClassifier
# Metricas
from sklearn.metrics import roc_auc_score, accuracy_score , recall_score,␣
 ↪accuracy_score,roc_curve,confusion_matrix
from sklearn.metrics import precision_recall_curve, auc,␣
 ↪precision_score,f1_score
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import train_test_split

import warnings
warnings.filterwarnings("ignore")
```

```
[nltk_data] Downloading package stopwords to /home/carlos-
[nltk_data]     vazquez/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

[2]:
```
data = pd.read_csv("data_app_movil.csv")
```

[3]:
```
data.head(5)
```

[3]:

	Unnamed: 0	at	content	reviewCreatedVersion	thumbsUpCount	content.1	topic
0	6	2021-09-22 23:45:00	Que paso con la app, al abrir se cierra despué…	5.62	217	Que paso con la app, al abrir se cierra despué…	transferencias
1	9	2021-03-21 17:26:06	No funciona, después de más de una hora entre …	5.59	201	No funciona, después de más de una hora entre …	transferencias
2	10	2021-08-04 14:39:20	La app ha intentado mejorar desde el 2019, per…	5.60	185	La app ha intentado mejorar desde el 2019, per…	transferencias
3	19	2021-06-14 16:44:16	Poner una estrella es mucho, sólo porque lo pi…	5.59.2	131	Poner una estrella es mucho, sólo porque lo pi…	transferencias
4	24	2021-07-02 14:36:40	Esta aplicación no deja de presentar fallas de…	5.60	126	Esta aplicación no deja de presentar fallas de…	transferencias

```
4  transferencias
```

- Unnamed: 0 : Posible indice, se eliminara.
- at: Fecha en la que se capturo el contenido
- content: llamada pasada a texto
- reviewCreatedVersion: Version
- thumbsUpCount: Pulgares arriba
- content.1: Duplicado de content
- topic: Variable target, si la llamada pertenece al equipo de token o al de transferencias

[4]:
```
# Vemos que esta duplicada la columna
(data["content"] != data["content.1"]).sum()
```

[4]:
```
0
```

[5]:
```
# ELiminamos las columnas que no sirven
data = data.drop(["Unnamed: 0","content.1"],axis = 1)
data.head(5)
```

[5]:

	at	content	reviewCreatedVersion	thumbsUpCount	topic
0	2021-09-22 23:45:00	Que paso con la app, al abrir se cierra despué…	5.62	217	transferencias
1	2021-03-21 17:26:06	No funciona, después de más de una hora entre …	5.59	201	transferencias
2	2021-08-04 14:39:20	La app ha intentado mejorar desde el 2019, per…	5.60	185	transferencias
3	2021-06-14 16:44:16	Poner una estrella es mucho, sólo porque lo pi…	5.59.2	131	transferencias
4	2021-07-02 14:36:40	Esta aplicación no deja de presentar fallas de…	5.60	126	transferencias

2.1.2 Target

Nuestra variable target es la columna *topic* que indica a que equipo pertenece la duda/queja recibida

Definimos la variable target Se etiquetara con 1 si el departamento es el encargado del token y 0 si es el de transferencias

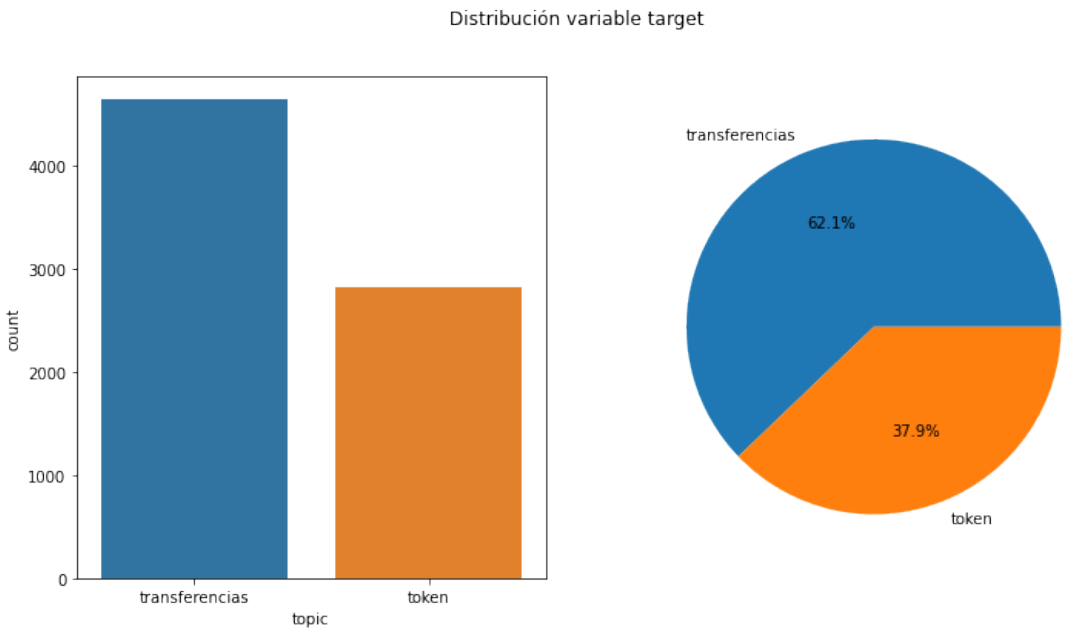
[6]:
```
data['y'] = np.where(data["topic"] == "token",1,0)
```

Se puede observar un poco de desbalance en los datos, la mayoria de las llamadas pertenecen al equipo de transferencias.

[7]:
```
plt.figure(figsize = (12,6))
fig, axs = plt.subplots(1,2,figsize = (12,6))
sns.countplot(data = data,
              x = "topic",ax = axs[0])
axs[1].pie(data["topic"].value_counts(),labels =␣
 ↪["transferencias","token"],autopct = "%1.1f%%")
plt.suptitle("Distribución variable target")
```

[7]:
```
Text(0.5, 0.98, 'Distribución variable target')
```

```
<Figure size 864x432 with 0 Axes>
```



2.1.3 Features

Nuestra principal feature es el texto que se logro obtener de la duda/queja, tambien se vera la frecuencia de las quejas y si hay algun patron de estacionalidad

Valores nulos Unicamente la variables *reviewCreatedVersion* tiene valores nulos, y representan el **5%** de todos los datos.

Como no se usara esta variable como feature no es necesario darle tratamiento

[8]:
```
(data.isnull().sum() / data.shape[0])*100
```

[8]:
```
at                      0.000000
content                 0.000000
```

```
reviewCreatedVersion    5.290651
thumbsUpCount           0.000000
topic                   0.000000
y                       0.000000
dtype: float64
```

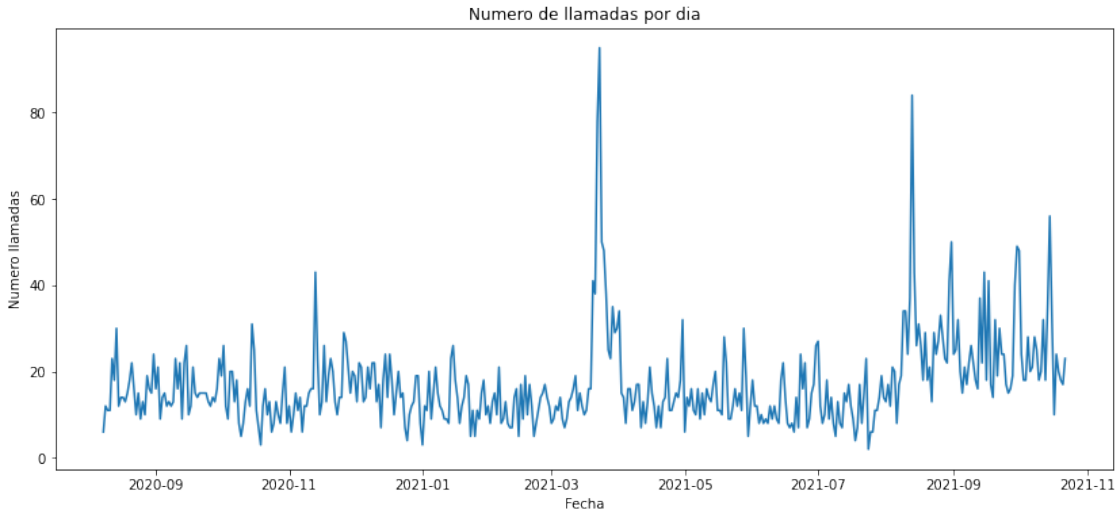
Creamos variables de tiempo

[9]:
```
data["at"] = pd.to_datetime(data["at"])
data["date"] = data["at"].dt.date
data["hora"] = data["at"].dt.hour
data["dia"] = data["at"].dt.day
data["dia_semana"] = data["at"].dt.day_name()
data["tipo_horario"] = np.where(data["hora"].between(8,16),"Oficina","No␣
↪Oficina")
```

Numero de llamadas por dia Podemos ver que el numero de llamadas diarias la mayor parte del tiempo se mantuvo dentro del intervalo 0 y 40, teneindo picos que lograron llegar hasta mas de 80 llamadas en un día, posiblemente un error en la aplicación que ocasiono que se recibiran mas llamadas.

[10]:
```
plt.figure(figsize = (14,6))
plt.plot(data.groupby("date")[["topic"]].count())
plt.title("Numero de llamadas por dia")
plt.xlabel("Fecha")
plt.ylabel("Numero llamadas")
```

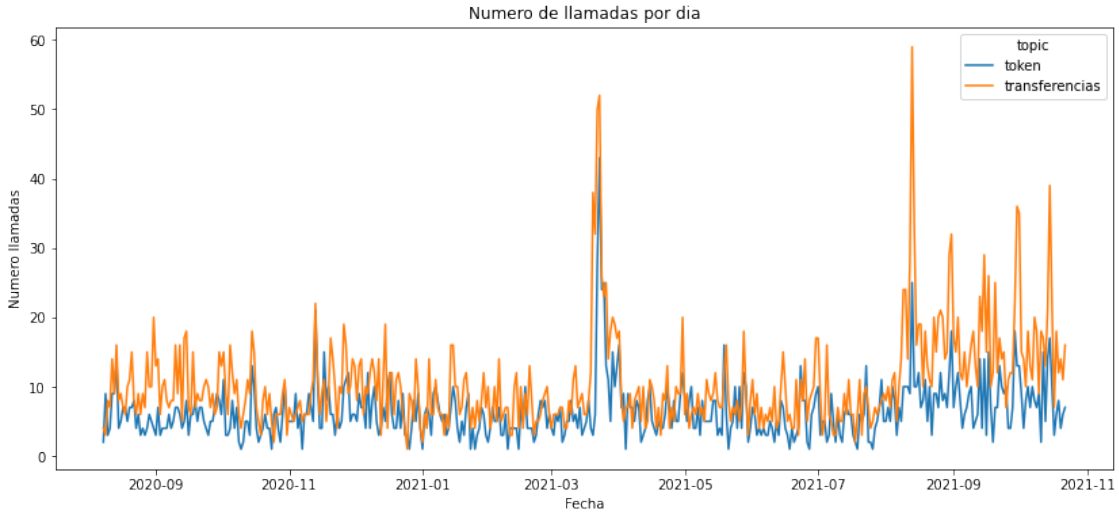
[10]:
```
Text(0, 0.5, 'Numero llamadas')
```



Numero de llamadas por topico Vemos que como se esperaba las llamadas de transferencias es mayor a las llamadas de token, pero ambas siguen la misma tendencia solo que en menor escala para las llamadas de token.

[11]:
```
plt.figure(figsize = (14,6))
sns.lineplot(data = data.groupby(["date","topic"])[["content"]].count().
 ↪reset_index(),
             x = "date",
             y = "content",
             hue = "topic")
plt.title("Numero de llamadas por dia")
plt.xlabel("Fecha")
plt.ylabel("Numero llamadas")
```

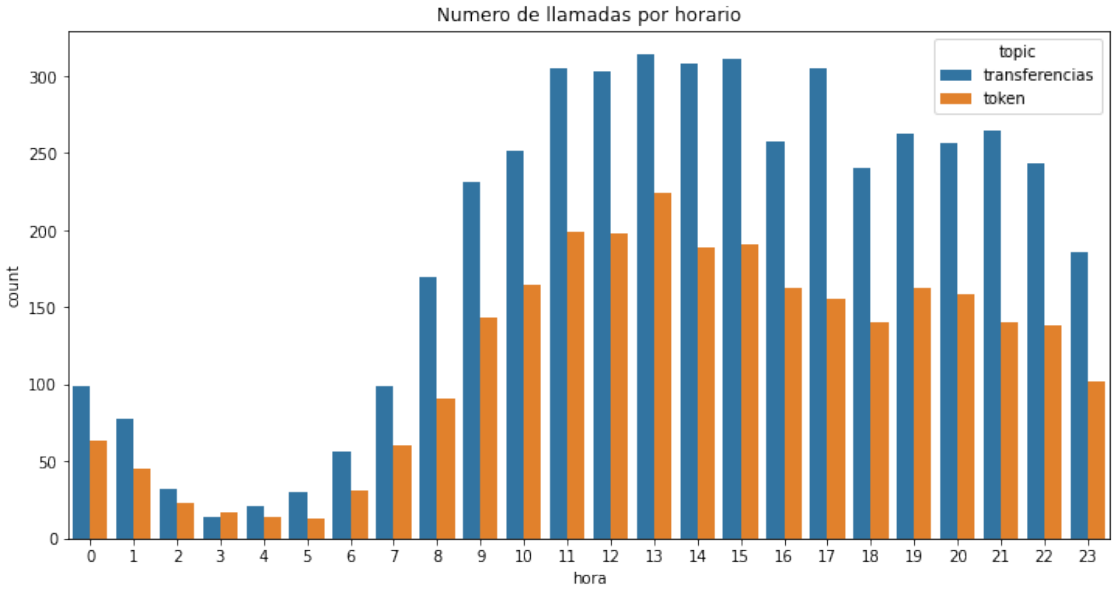
[11]:
```
Text(0, 0.5, 'Numero llamadas')
```



¿A que hora hay mas llamadas? La mayoria de las llamadas se concentra en un horario de 11 a 17 horas, y cuando menos hay es en la madrugada, esto es de esperarse ya que en la madrugada la mayoria de las personas estan durmiendo.

[12]:
```
plt.figure(figsize = (12,6))
sns.countplot(data = data,
              x = "hora",
              hue = "topic")
plt.title("Numero de llamadas por horario")
```

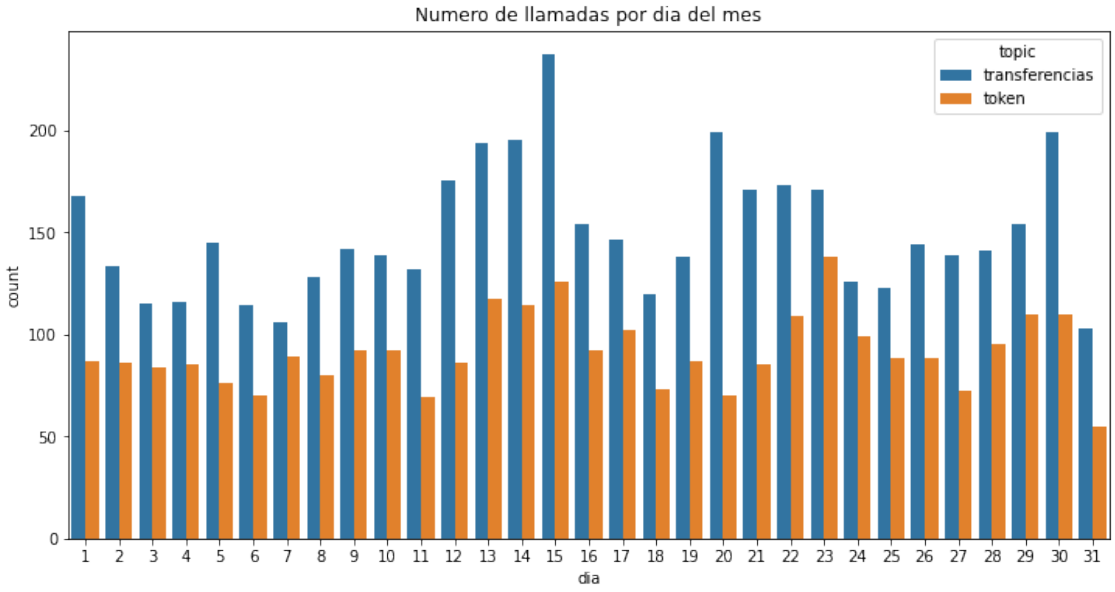
[12]:
```
Text(0.5, 1.0, 'Numero de llamadas por horario')
```



¿Que dia hay mas llamadas? Se puede ver que en los dias mas cercanos a las quincenas es cuando mas llamadas hay, esto se puede deber a que en las quincenas es cuando se suele pagar el sueldo, por lo que la gente estaria mas activa en las aplicaciones.

[13]:
```
plt.figure(figsize = (12,6))
sns.countplot(data = data,
              x = "dia",
              hue = "topic")
plt.title("Numero de llamadas por dia del mes")
```

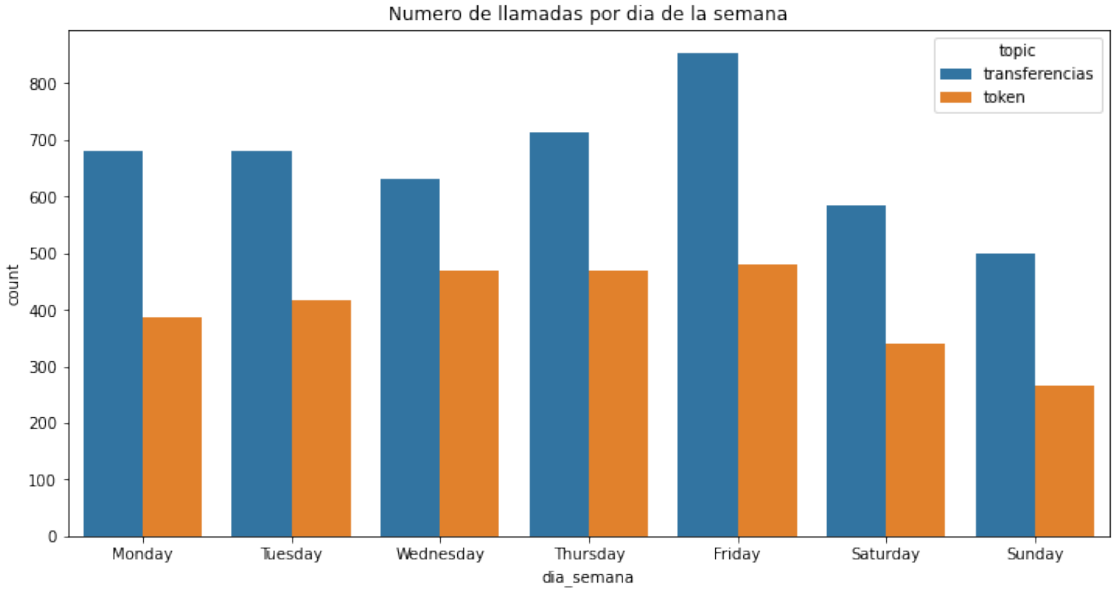
[13]:
```
Text(0.5, 1.0, 'Numero de llamadas por dia del mes')
```



Dia de la semana con mas llamadas No se logra ver una diferencia muy significativa entre los dias de entre semana, salvo el dia viernes que es cuando mas llamadas hay, donde se ve un cambio es en los dias entre semana y los fines de semana.

[14]:
```
plt.figure(figsize = (12,6))
sns.countplot(data = data,
              x = "dia_semana",
              hue = "topic",
              order =␣
 ↪["Monday","Tuesday","Wednesday","Thursday","Friday","Saturday","Sunday"])
plt.title("Numero de llamadas por dia de la semana")
```

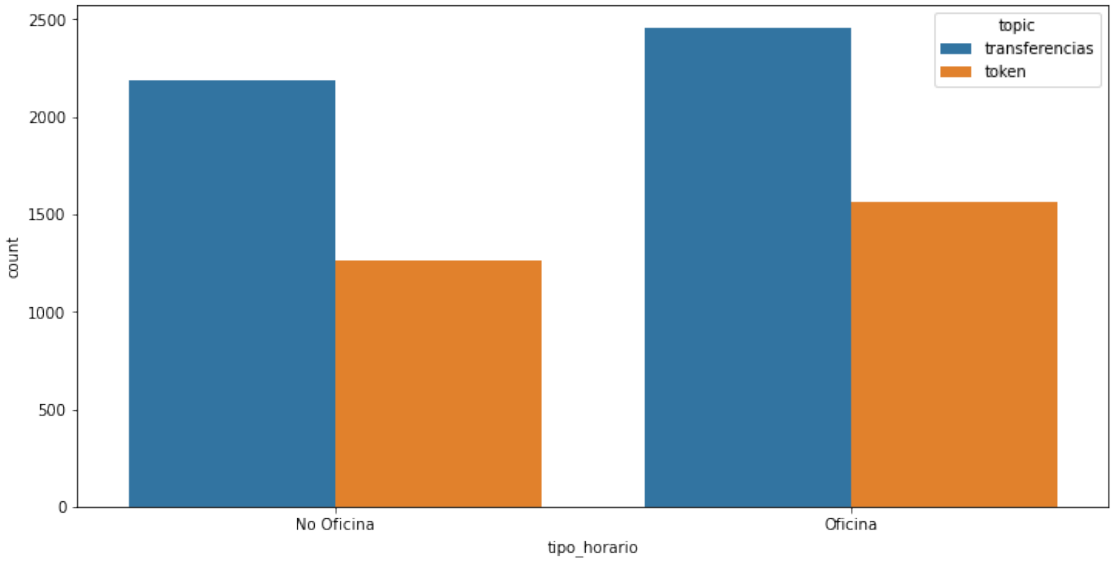
[14]:
```
Text(0.5, 1.0, 'Numero de llamadas por dia de la semana')
```



¿Afecta si es horario de oficina? Podemos ver que suelen hacer mas llamadas en horario de oficina que en horario fuera de oficina.

[15]:
```
plt.figure(figsize = (12,6))
sns.countplot(data = data,
              x = "tipo_horario",
              hue = "topic")
```

[15]: <AxesSubplot:xlabel='tipo_horario', ylabel='count'>

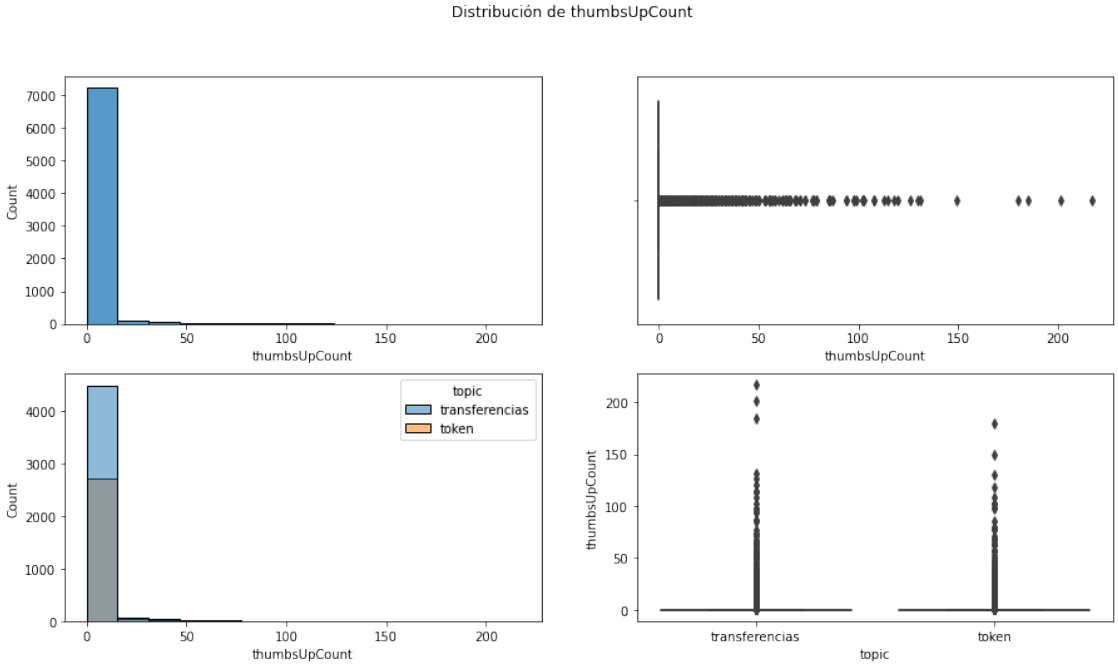


Distribución de thumbsUpCount Se logra ver que existe mucho sesgo en la distribución, la mayoria de las llamadas tienen menos de 50 pulgares.

[16]:
```
fig, axs = plt.subplots(2,2, figsize = (15,8))
sns.histplot(data["thumbsUpCount"],ax = axs[0][0])
sns.boxplot(data["thumbsUpCount"],ax = axs[0][1])

sns.histplot(data = data,x = "thumbsUpCount",hue = "topic",ax = axs[1][0])
sns.boxplot(data = data,y = "thumbsUpCount", x = "topic",ax = axs[1][1])
plt.suptitle("Distribución de thumbsUpCount")
```

[16]:
```
Text(0.5, 0.98, 'Distribución de thumbsUpCount')
```



Llamadas con quejas iguales Hay 589 llamadas que sus quejas o dudas son iguales, esto se puede esperar de quejas muy sencillas y cortas.

[17]:
```
data[data["content"].duplicated()]["content"].head(10)
```

[17]:
```
1096              No puedo hacer transferencias
1337                          No pude transferir
1442    No puedo hacer transferencia bancaria
```

```
1518            No puedo hacer transferencias
1727                       No deja transferir
1730           No me deja hacer transferencia
1971            No puedo hacer transferencias
2075            No puedo hacer transferencias
2079       No me permite hacer transferencias
2526            No puedo hacer transferencias
Name: content, dtype: object
```

Palabras mas repetidas Se hizo la nube de palabras antes de la limpieza para ver como estaba antes y despues de a limpieza

General Se puede ver que las palabras que mas se repiten son aplicación, transferencia, token, app, super.

[18]:
```
plt.figure(figsize = (12,8))
# Create and generate a word cloud image:
wordcloud = WordCloud().generate(" ".join(data['content'].values.tolist()))
# Display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.title("Palabras mas frecuentes usadas")
plt.show()
```

Transferencias Al estar reducida unicamente a las llamadas que son de transferencias se puede ver que justamente la palabra mas repetida es transferencia y banco tambien es repetida constantemente.

[19]:
```
plt.figure(figsize = (12,8))
# Create and generate a word cloud image:
wordcloud = WordCloud().generate(" ".join(data[data["y"] == 0]['content'].
 ↪values.tolist()))
# Display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.title("Palabras mas frecuentes usadas")
plt.show()
```

Token AL filtrar por todas las llamadas que son del equipo de token,se puede ver que la plabra token y super con las que mas se repiten.

[20]:
```
plt.figure(figsize = (12,8))
# Create and generate a word cloud image:
wordcloud = WordCloud().generate(" ".join(data[data["y"] == 1]['content'].
 ↪values.tolist()))
# Display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.title("Palabras mas frecuentes usadas")
plt.show()
```

2.2 Procesamiento de datos

Realizaremos una limpieza de nuestra variable content, quitando acentos, pasando a minusculas, quitando stop words, etc. Para despues vectorizar el texto y poder usarlo como input en nuestro modelo.

2.2.1 Limpiar texto

[21]:
```
# Stop words en español
stop_words = list(set(stopwords.words('spanish')))
# Agregamos stopwords
stop_words.append("algo")
stop_words.append('ambos')

def text_processing(text):
    # Quitamos acentos
    text = unidecode(text)
    # Pasamos a minusculas
    text = text.lower()
    # Eliminar caracteres especiales
    text = re.sub(r'[^\w\s]',' ',text)
    # Elimina palabras con menos de tres letras
    text = ' '.join([word for word in text.split() if len(word)>3])
    # Elimina números
    text = re.sub(r'\b\d+(?:\.\d+)?\s+', ' ', text)
```

```
    # Eliminamos dobles espacios
    text = text.replace("  "," ")
    # Quitamos espacios en los laterales
    text = text.strip()
    return text
```

[22]:
```
data['content_clean'] = data['content'].apply(text_processing)
```

2.2.2 Nube de palabras con el texto limpio

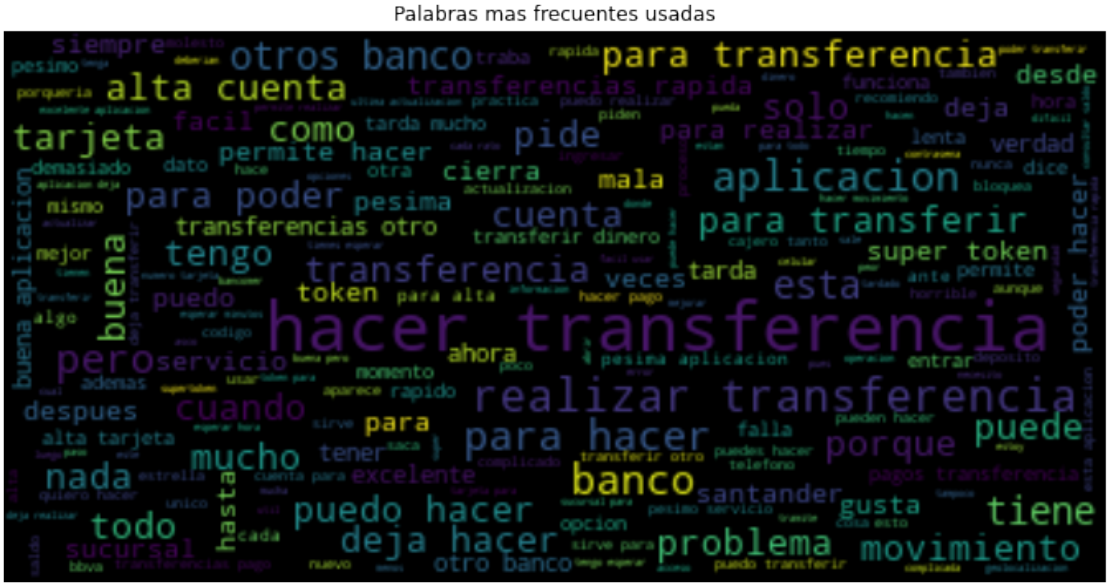
Logramos ver que se redujo el numero de palabras que no aportaba mucho, por ejemplo 'y', 'la', etc. Esto debido a las stopwords

General

[23]:
```
plt.figure(figsize = (12,8))
# Create and generate a word cloud image:
wordcloud = WordCloud().generate(" ".join(data['content_clean'].values.
↪tolist()))
# Display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.title("Palabras mas frecuentes usadas")
plt.show()
```

Transferencias

[24]:
```
plt.figure(figsize = (12,8))
# Create and generate a word cloud image:
wordcloud = WordCloud().generate(" ".join(data[data["y"] == 0]['content_clean'].
 ↪values.tolist()))
# Display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.title("Palabras mas frecuentes usadas")
plt.show()
```



Token

[25]:
```
plt.figure(figsize = (12,8))
# Create and generate a word cloud image:
wordcloud = WordCloud().generate(" ".join(data[data["y"] == 1]['content_clean'].
 ↪values.tolist()))
# Display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.title("Palabras mas frecuentes usadas")
plt.show()
```

Palabras mas frecuentes usadas

2.2.3 Dividimos entre X e y

[26]:
```
X = data["content_clean"]
y = data["y"]
```

2.2.4 Se hace la division entre train test

[27]:
```
xt,xv,yt,yv  = train_test_split(X,y, test_size = 0.3, random_state=42)
```

2.2.5 Se vectoriza el contenido

[28]:
```
vectorizer_TFID = TfidfVectorizer(max_features=100)
vectorizer_TFID.fit(xt)
```

[28]:
```
TfidfVectorizer(max_features=100)
```

Se guarda el objeto vectorizer_TFID para utilizarlo en produccion

[29]:
```
pickle.dump(vectorizer_TFID, open("vectorizer.pickle", "wb"))
```

Se aplica la vectorizacion

[30]:
```
xt_tfid = pd.DataFrame(vectorizer_TFID.transform(xt).toarray(), columns =␣
↪vectorizer_TFID.get_feature_names())
```

```
xv_tfid = pd.DataFrame(vectorizer_TFID.transform(xv).toarray(), columns =␣
↪vectorizer_TFID.get_feature_names())
```

2.3 Modelos

2.3.1 Forma de evaluar modelos

Se usara la metrica F1 para evaluar nuestro modelo, esto debido a que asi podremos buscar maximizar la precisión y el recall a la vez, tambien se calcularan mas metricas para compararlas y ver como se van comportando los modelos, asi tambien veremos sus matrices de confusión y sus curvas ROC y precision recall

Se hara uso de RandomizedSearchCV que nos permitira buscar los mejores hiperparametros de cada modelo aleatoriamente, ademas, este mismo hara cross validation y obtendremos el modelo que mejor metrica obtenga en el cross validation ,despues haremos bootstrap para escoger el mejor modelo.

[31]:
```
def evaluar_modelo(modelo, X_train,y_train, X_test, y_test):
    '''
    Funcion para evaluar los modelos
    '''
    # F1 score
    f1_train = f1_score(y_true = y_train, y_pred= modelo.predict(X_train))
    f1_test = f1_score(y_true = y_test, y_pred= modelo.predict(X_test))
    # Precision score
    precision_train = precision_score(y_true = y_train, y_pred= modelo.
↪predict(X_train))
    precision_test = precision_score(y_true = y_test, y_pred= modelo.
↪predict(X_test))
    # Recall score
    recall_train = recall_score(y_true = y_train, y_pred= modelo.
↪predict(X_train))
    recall_test = recall_score(y_true = y_test, y_pred= modelo.predict(X_test))
    # Accuracy score
    accuracy_train = accuracy_score(y_true = y_train, y_pred= modelo.
↪predict(X_train))
    accuracy_test = accuracy_score(y_true = y_test, y_pred= modelo.
↪predict(X_test))
    # AUC
    auc_train = roc_auc_score(y_true = y_train, y_score= modelo.
↪predict_proba(X_train)[:,1])
    auc_test = roc_auc_score(y_true = y_test, y_score= modelo.
↪predict_proba(X_test)[:,1])
    # ROC
    fpr_train, tpr_train, _ = roc_curve(y_true = y_train, y_score= modelo.
↪predict_proba(X_train)[:,1])
```

```
   fpr_test, tpr_test, _ = roc_curve(y_true = y_test, y_score= modelo.
↪predict_proba(X_test)[:,1])
   # Confusion matrix
   confusion_matrix_train = confusion_matrix(y_true = y_train, y_pred= modelo.
↪predict(X_train))
   confusion_matrix_test = confusion_matrix(y_true = y_test, y_pred= modelo.
↪predict(X_test))
   # Precicion recall curve
   precision_curve_train,recall_curve_train, _ = precision_recall_curve(y_true␣
↪= y_train, probas_pred= modelo.predict_proba(X_train)[:,1])
   precision_curve_test, recall_curve_test, _ =precision_recall_curve(y_true =␣
↪y_test, probas_pred= modelo.predict_proba(X_test)[:,1])
   # Area under curve Precision recall
   auc_p_r_train = auc(recall_curve_train,precision_curve_train)
   auc_p_r_test = auc(recall_curve_test,precision_curve_test)
   # Print Metrics
   metrics = {'train':
↪[f1_train,precision_train,recall_train,accuracy_train,auc_train,auc_p_r_train],
              'test':␣
↪[f1_test,precision_test,recall_test,accuracy_test,auc_test,auc_p_r_test]}
   print(pd.DataFrame(metrics,index= ["F1 score","Precision score" ,"Recall␣
↪score","Accuracy","ROC auc","Precision- Recall auc"]))
   # Plots
   fig, axs = plt.subplots(2,2, figsize = (16,10))
   # Plot confusion matrix
   sns.heatmap(confusion_matrix_train, annot=True, cmap='Blues',ax =␣
↪axs[0][0],fmt = ".10g")
   sns.heatmap(confusion_matrix_test, annot=True, cmap='Blues',ax =␣
↪axs[0][1],fmt = ".10g")
   axs[0][0].title.set_text('Confusion Matrix Train')
   axs[0][1].title.set_text('Confusion Matrix Test')
   # Plot ROC AUC
   sns.lineplot(fpr_train, tpr_train,
                label = "AUC train: {0}".format(round(auc_train,2)),
                color = "r",
                ax = axs[1][0])
   sns.lineplot(fpr_test, tpr_test,
                label = "AUC test: {0}".format(round(auc_test,2)),
                color = "b",
                ax = axs[1][0])
   axs[1][0].legend()
   axs[1][0].title.set_text('ROC curve')
   # Plot precision recall curve
   sns.lineplot(recall_curve_train ,
                precision_curve_train ,
                ax = axs[1][1],
```

```
                color = "r",
               label = "Precision-Recall area train: {0}".
↪format(round(auc_p_r_train,2)))
   sns.lineplot(recall_curve_test ,
                precision_curve_test ,
                ax = axs[1][1],
                color = "b",
               label = "Precision-Recall area test: {0}".
↪format(round(auc_p_r_test,2)))
   axs[1][1].title.set_text('Precision-Recall curve')

   plt.show()
```

2.3.2 Logistic Regression

[32]:
```
hiperparametros_logistica = {'C': np.arange(0.1,5,0.1),
                             'class_weight': [{0:x,1:1-x} for x in np.arange(0.
↪3,0.4,0.01)],
                             'l1_ratio': np.arange(0.8,1,0.01)}

grid_logistica = RandomizedSearchCV(estimator= LogisticRegression(penalty =␣
↪"elasticnet",solver = "saga"),
                                    param_distributions =␣
↪hiperparametros_logistica,
                                    scoring = "f1",n_iter=50,
                                    cv = 5,
                                    return_train_score = True,
                                    random_state=42)

grid_logistica.fit(xt_tfid,yt)
```

[32]:
```
RandomizedSearchCV(cv=5,
                   estimator=LogisticRegression(penalty='elasticnet',
                                                solver='saga'),
                   n_iter=50,
                   param_distributions={'C': array([0.1, 0.2, 0.3, 0.4, 0.5,
0.6, 0.7, 0.8, 0.9, 1. , 1.1, 1.2, 1.3,
       1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2. , 2.1, 2.2, 2.3, 2.4, 2.5, 2.6,
       2.7, 2.8, 2.9, 3. , 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8, 3.9,
       4. , 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 4.9]),
                                        'class_weight': [{0: 0.3, 1:…
                                                         {0:
0.37000000000000005,
                                                          1:
0.6299999999999999},
```

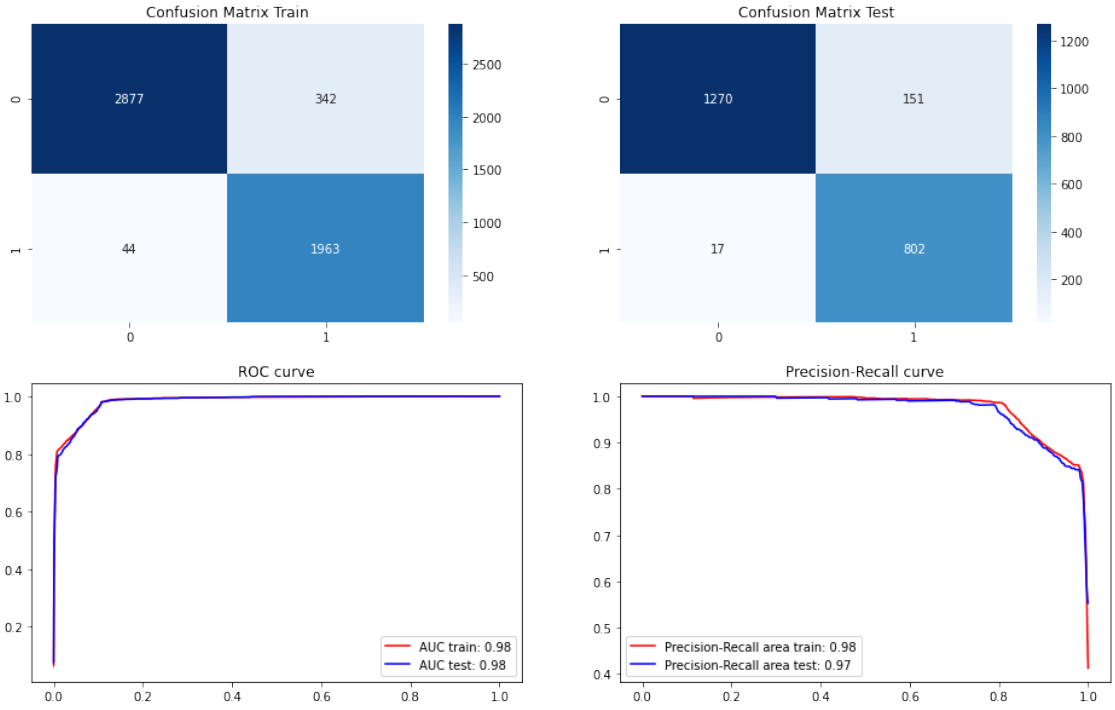
```
                                                                {0:
0.38000000000000006,
                                                                 1:
0.6199999999999999},
                                                                {0:
0.39000000000000007,
                                                                 1:
0.6099999999999999},
                                                                {0: 0.4000000000000001,
                                                                 1:
0.5999999999999999}],
                                          'l1_ratio': array([0.8 , 0.81, 0.82,
0.83, 0.84, 0.85, 0.86, 0.87, 0.88, 0.89, 0.9 ,
       0.91, 0.92, 0.93, 0.94, 0.95, 0.96, 0.97, 0.98, 0.99])},
                   random_state=42, return_train_score=True, scoring='f1')
```

[33]:
```
best_logistic_model = grid_logistica.best_estimator_
best_logistic_model.fit(xt_tfid, yt)
```

[33]:
```
LogisticRegression(C=0.9,
                   class_weight={0: 0.36000000000000004, 1: 0.6399999999999999},
                   l1_ratio=0.9900000000000002, penalty='elasticnet',
                   solver='saga')
```

[34]:
```
evaluar_modelo(modelo = best_logistic_model,
               X_train = xt_tfid,
               y_train = yt,
               X_test = xv_tfid,
               y_test = yv)
```

```
                          train      test
F1 score                0.910482  0.905192
Precision score         0.851627  0.841553
Recall score            0.978077  0.979243
Accuracy                0.926139  0.925000
ROC auc                 0.983522  0.982964
Precision- Recall auc   0.975733  0.972007
```



Se guarda modelo por si es que resulta ganador y pasarlo a producción

[35]:
```
pickle.dump(best_logistic_model, open("logistic_model.pickle", "wb"))
```

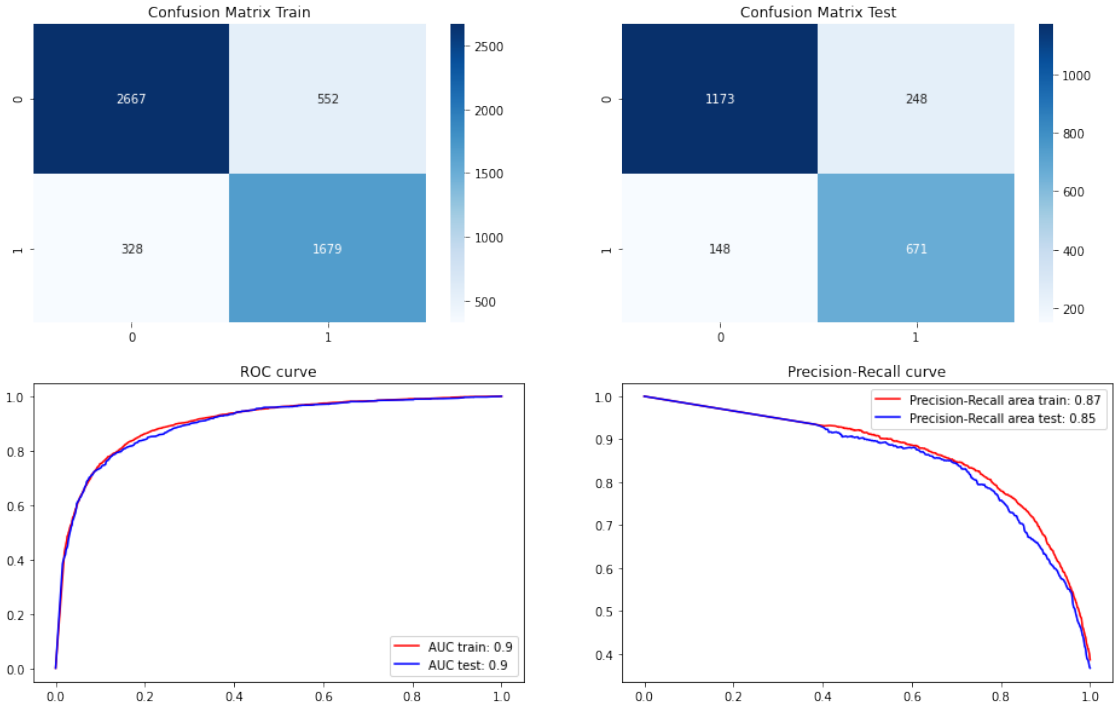
2.3.3 Naive Bayes

[36]:
```
naive_bayes = GaussianNB()
naive_bayes.fit(xt_tfid,yt)
```

[36]:
```
GaussianNB()
```

[37]:
```
evaluar_modelo(modelo = naive_bayes,
               X_train = xt_tfid,
               y_train = yt,
               X_test = xv_tfid,
               y_test = yv)
```

```
                        train      test
F1 score             0.792355  0.772152
Precision score      0.752577  0.730141
Recall score         0.836572  0.819292
Accuracy             0.831611  0.823214
ROC auc              0.904387  0.900052
Precision- Recall auc  0.866889  0.854886
```



Se guarda modelo por si es que resulta ganador y pasarlo a producción

[38]:
```
pickle.dump(naive_bayes, open("naive_bayes_model.pickle", "wb"))
```

2.3.4 Neural Network

[39]:
```
hiperparametros_nn = {'hidden_layer_sizes':[(a,b,c) for a in range(5,100,10)␣
↪for b in range(5,100,10) for c in range(5,100,5)],
                      'activation': ["relu",'identity',"logistic","tanh"],
                      'learning_rate_init': [0.001,0.01,0.005]}

grid_nn = RandomizedSearchCV(estimator= MLPClassifier(solver = "adam",
                                                      learning_rate =␣
↪"invscaling",
                                                     random_state = 42,
                                                     early_stopping = True),
                                  param_distributions = hiperparametros_nn,
                                  scoring = "f1",n_iter=20,
                                  cv = 5,
                                  return_train_score = True,
                                  random_state=42)

grid_nn.fit(xt_tfid,yt)
```

[39]:
```
RandomizedSearchCV(cv=5,
                   estimator=MLPClassifier(early_stopping=True,
                                           learning_rate='invscaling',
                                           random_state=42),
                   n_iter=20,
                   param_distributions={'activation': ['relu', 'identity',
                                                      'logistic', 'tanh'],
                                       'hidden_layer_sizes': [(5, 5, 5),
                                                              (5, 5, 10),
                                                              (5, 5, 15),
                                                              (5, 5, 20),
                                                              (5, 5, 25),
                                                              (5, 5, 30),
                                                              (5, 5, 35),
                                                              (5, 5, 40),
                                                              (5, 5, 45),
                                                              (5, 5, 50),
                                                              (5, 5, 55),
                                                              (5, 5, 60),
                                                              (5, 5, 65),
                                                              (5, 5, 70),
                                                              (5, 5, 75),
                                                              (5, 5, 80),
                                                              (5, 5, 85),
                                                              (5, 5, 90),
                                                              (5, 5, 95),
                                                              (5, 15, 5),
                                                              (5, 15, 10),
                                                              (5, 15, 15),
                                                              (5, 15, 20),
                                                              (5, 15, 25),
                                                              (5, 15, 30),
                                                              (5, 15, 35),
                                                              (5, 15, 40),
                                                              (5, 15, 45),
                                                              (5, 15, 50),
                                                              (5, 15, 55),
    …],
                                       'learning_rate_init': [0.001, 0.01,
                                                              0.005]},
                   random_state=42, return_train_score=True, scoring='f1')
```

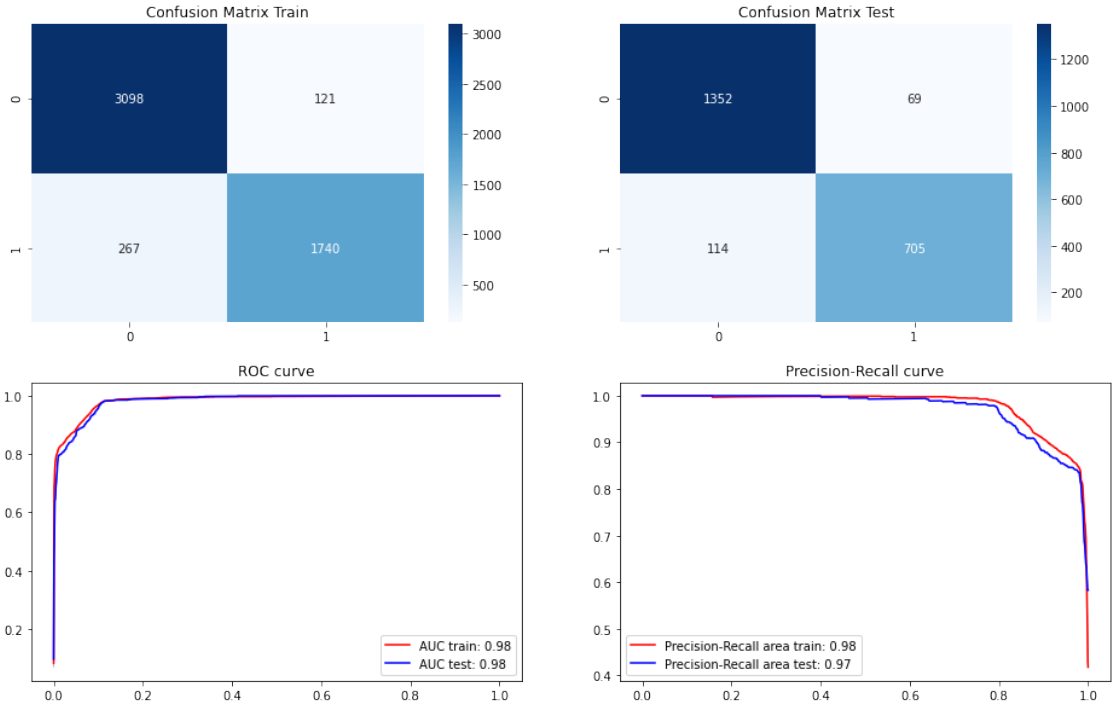
[40]:
```
best_nn_model = grid_nn.best_estimator_
best_nn_model.fit(xt_tfid,yt)
```

[40]:
```
MLPClassifier(activation='identity', early_stopping=True,
              hidden_layer_sizes=(95, 95, 35), learning_rate='invscaling',
```

```
                    learning_rate_init=0.005, random_state=42)
```

[41]:
```
evaluar_modelo(modelo = best_nn_model,
               X_train = xt_tfid,
               y_train = yt,
               X_test = xv_tfid,
               y_test = yv)
```

```
                          train      test
F1 score                0.899690  0.885122
Precision score         0.934981  0.910853
Recall score            0.866966  0.860806
Accuracy                0.925756  0.918304
ROC auc                 0.984732  0.982312
Precision- Recall auc   0.978036  0.971328
```



Se guarda modelo por si es que resulta ganador y pasarlo a producción

[42]:
```
pickle.dump(best_nn_model, open("nn_model.pickle", "wb"))
```

2.3.5 Decision tree

[43]:
```
hiperparametros_tree = {'max_depth': [1,2,3,4],
                        'class_weight': [{0:x,1:1-x} for x in np.arange(0.1,1,0.
 ↪01)],
                        'criterion':["gini","entropy"],
                        'min_samples_split': [2,4,6,8],
                        'min_samples_leaf': [2,3,4,5],
                        'max_features':['auto',"sqrt","log2",None] +␣
 ↪[round(xt_tfid.shape[1] * i/100)  for i in range(10,101)]
                        }

grid_tree = RandomizedSearchCV(estimator=␣
 ↪DecisionTreeClassifier(random_state=42,),
                                param_distributions = hiperparametros_tree,
                                scoring = "f1",n_iter=100,
                                cv = 5,
                                return_train_score = True,
                                random_state=42)

grid_tree.fit(xt_tfid,yt)
```

[43]:
```
RandomizedSearchCV(cv=5, estimator=DecisionTreeClassifier(random_state=42),
                   n_iter=100,
                   param_distributions={'class_weight': [{0: 0.1, 1: 0.9},
                                                         {0: 0.11, 1: 0.89},
                                                         {0: 0.12, 1: 0.88},
                                                         {0: 0.13, 1: 0.87},
                                                         {0:
0.13999999999999999,
                                                          1: 0.86},
                                                         {0:
0.14999999999999997,
                                                          1:
0.8500000000000001},
                                                         {0:
0.15999999999999998,
                                                          1:
0.8400000000000001},
                                                         {0:
0.16999999999999998,
                                                          1:
0.8300000000000001},…
                                                          1:
0.6200000000000001},
                                                         {0: 0.3899999999999999,
                                                          1:
```

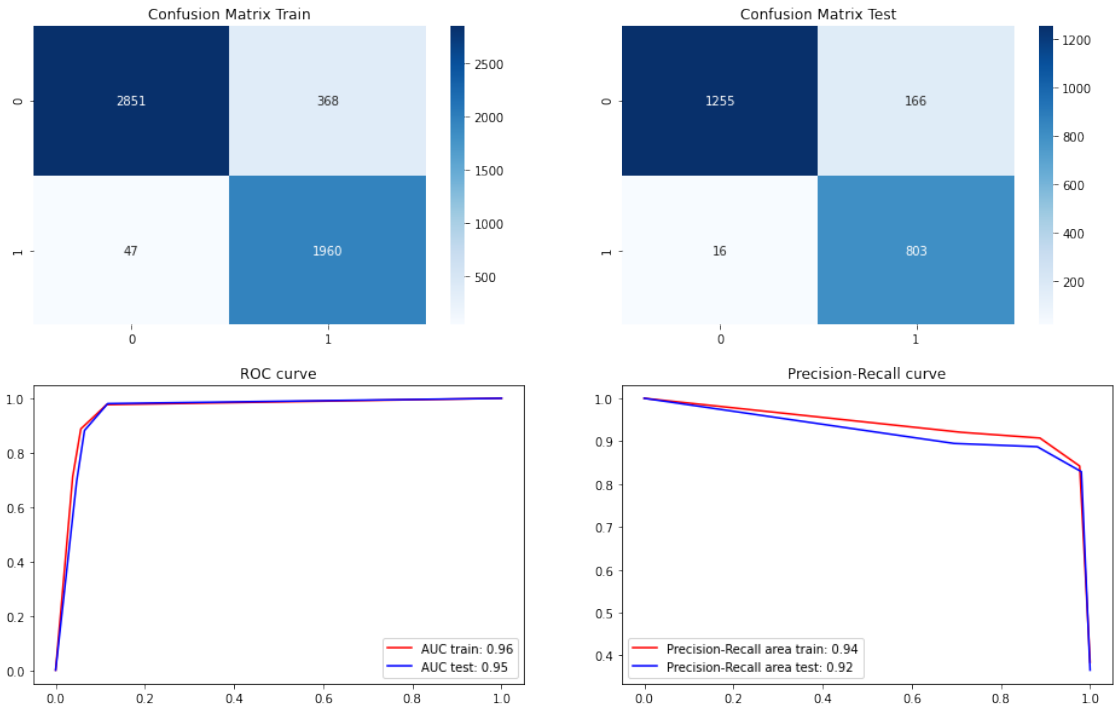
```
0.6100000000000001}, …],
                                        'criterion': ['gini', 'entropy'],
                                        'max_depth': [1, 2, 3, 4],
                                        'max_features': ['auto', 'sqrt', 'log2',
                                                         None, 10, 11, 12, 13,
                                                         14, 15, 16, 17, 18, 19,
                                                         20, 21, 22, 23, 24, 25,
                                                         26, 27, 28, 29, 30, 31,
                                                         32, 33, 34, 35, …],
                                        'min_samples_leaf': [2, 3, 4, 5],
                                        'min_samples_split': [2, 4, 6, 8]},
                    random_state=42, return_train_score=True, scoring='f1')
```

[44]:
```
best_tree_model = grid_tree.best_estimator_
best_tree_model.fit(xt_tfid,yt)
```

[44]:
```
DecisionTreeClassifier(class_weight={0: 0.3599999999999999,
                                     1: 0.6400000000000001},
                       max_depth=2, max_features=90, min_samples_leaf=3,
                       min_samples_split=6, random_state=42)
```

[45]:
```
evaluar_modelo(modelo = best_tree_model,
               X_train = xt_tfid,
               y_train = yt,
               X_test = xv_tfid,
               y_test = yv)
```

```
                          train      test
F1 score                0.904268  0.898210
Precision score         0.841924  0.828689
Recall score            0.976582  0.980464
Accuracy                0.920589  0.918750
ROC auc                 0.957393  0.953282
Precision- Recall auc   0.936367  0.921078
```



Se guarda modelo por si es que resulta ganador y pasarlo a producción

[46]:
```
pickle.dump(best_tree_model, open("tree_model.pickle", "wb"))
```

2.3.6 Random Forest

[47]:
```
hiperparametros_rf = {'max_depth': [1,2,3,4,6,8],
                      'class_weight': [{0:x,1:1-x} for x in np.arange(0.1,1,0.
 ↪01)],
                      'criterion':["gini","entropy"],
                      'min_samples_split': [2,4,6,8],
                      'max_features':['auto',"sqrt","log2",None]+␣
 ↪[round(xt_tfid.shape[1] * i/100)  for i in range(10,101)]
                      }

grid_rf = RandomizedSearchCV(estimator=␣
 ↪RandomForestClassifier(random_state=42,),
                                  param_distributions = hiperparametros_rf,
                                  scoring = "f1",n_iter=10,
                                  cv = 5,
                                  return_train_score = True,
                                  random_state=42)
```

```
grid_rf.fit(xt_tfid,yt)
```

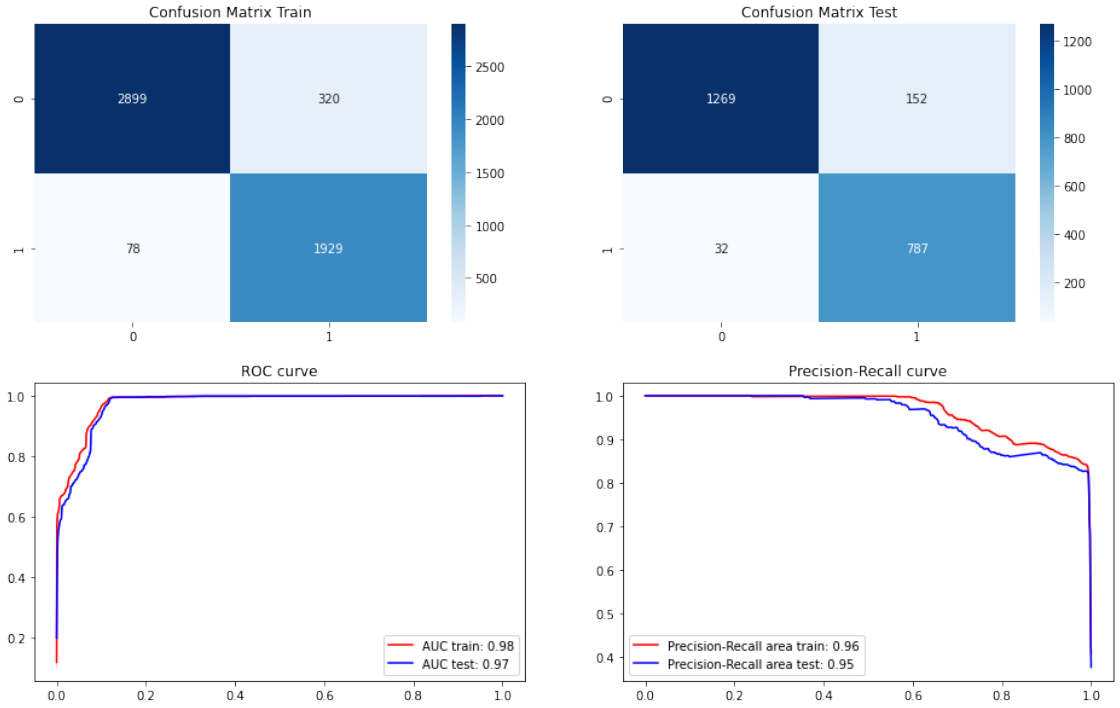
[47]:
```
RandomizedSearchCV(cv=5, estimator=RandomForestClassifier(random_state=42),
                   param_distributions={'class_weight': [{0: 0.1, 1: 0.9},
                                                         {0: 0.11, 1: 0.89},
                                                         {0: 0.12, 1: 0.88},
                                                         {0: 0.13, 1: 0.87},
                                                         {0:
0.13999999999999999,
                                                          1: 0.86},
                                                         {0:
0.14999999999999997,
                                                          1:
0.8500000000000001},
                                                         {0:
0.15999999999999998,
                                                          1:
0.8400000000000001},
                                                         {0:
0.16999999999999998,
                                                          1:
0.8300000000000001},
                                                         {0: 0.179999…
                                                         {0: 0.3799999999999999,
                                                          1:
0.6200000000000001},
                                                         {0: 0.3899999999999999,
                                                          1:
0.6100000000000001}, …],
                                        'criterion': ['gini', 'entropy'],
                                        'max_depth': [1, 2, 3, 4, 6, 8],
                                        'max_features': ['auto', 'sqrt', 'log2',
                                                         None, 10, 11, 12, 13,
                                                         14, 15, 16, 17, 18, 19,
                                                         20, 21, 22, 23, 24, 25,
                                                         26, 27, 28, 29, 30, 31,
                                                         32, 33, 34, 35, …],
                                        'min_samples_split': [2, 4, 6, 8]},
                   random_state=42, return_train_score=True, scoring='f1')
```

[48]:
```
best_rf_model = grid_rf.best_estimator_
best_rf_model.fit(xt_tfid,yt)
```

[48]:
```
RandomForestClassifier(class_weight={0: 0.33999999999999986,
                                     1: 0.6600000000000001},
                       max_depth=3, max_features=23, random_state=42)
```

[49]:
```
evaluar_modelo(modelo = best_rf_model,
               X_train = xt_tfid,
               y_train = yt,
               X_test = xv_tfid,
               y_test = yv)
```

```
                          train      test
F1 score                0.906485  0.895336
Precision score         0.857715  0.838126
Recall score            0.961136  0.960928
Accuracy                0.923842  0.917857
ROC auc                 0.977525  0.972138
Precision- Recall auc   0.964058  0.950907
```



Se guarda modelo por si es que resulta ganador y pasarlo a producción

[50]:
```
pickle.dump(best_rf_model, open("rf_model.pickle", "wb"))
```

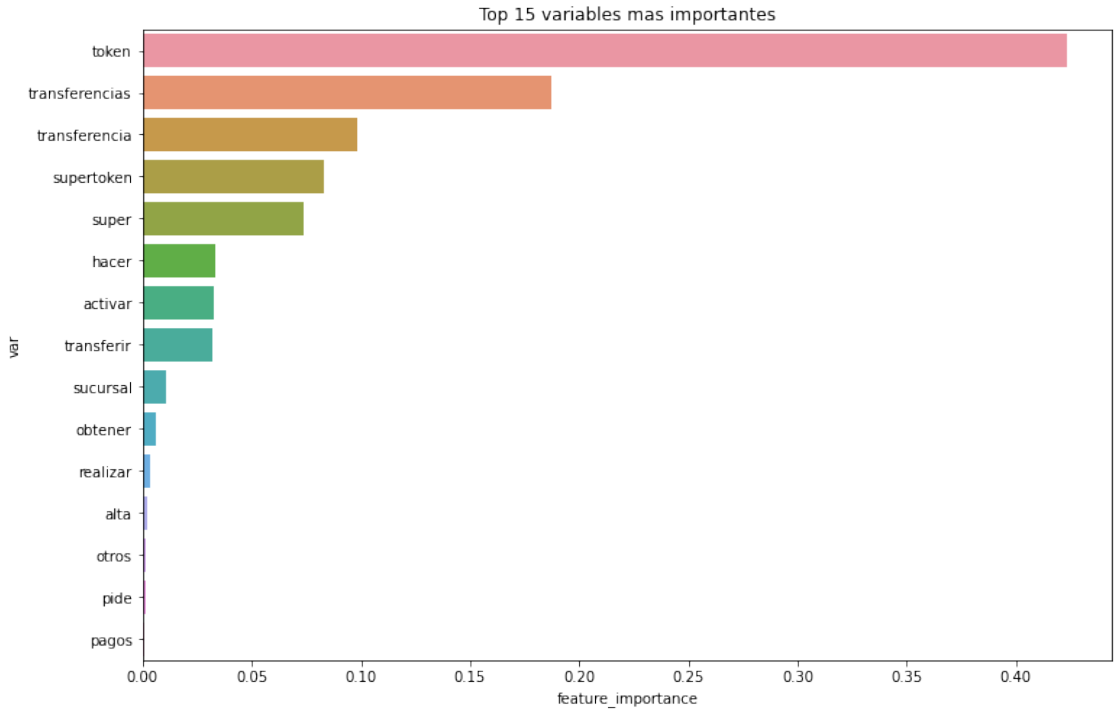
2.3.7 Feature importance

Las variables mas importantes son las que pudimos haber esperado, que son token y transferencias, ya que al solo mencionarlas podemos darnos la idea de a que equipo pertenece.

[51]:
```
feature_importance = pd.DataFrame(zip(xt_tfid.columns,best_rf_model.
↪feature_importances_),
             columns = ["var","feature_importance"]).sort_values(by =␣
↪"feature_importance",ascending = False)
```

[52]:
```
plt.figure(figsize = (12,8))
sns.barplot(data = feature_importance.head(15),
            y = "var",
            x = "feature_importance")
plt.title("Top 15 variables mas importantes")
```

[52]:
```
Text(0.5, 1.0, 'Top 15 variables mas importantes')
```



2.4 Bootstrap para escoger el mejor modelo

El mejor modelo que se obtuvo anteriormente utilizando cross validation fue la regresión logistica, veremos si este se mantiene en el bootstrap

[53]:
```
# Se realizaron solo 100 por temas de poder computacional.
M = 100
X_bootstrap = X.copy()
y_bootstrap = y.copy()
n_samples = X_bootstrap.shape[0]
```

```
models_ = {"Logistic Regression":best_logistic_model,
           'Naive Bayes': naive_bayes,
           'Neural Network': best_nn_model,
           "Decision tree":best_tree_model,
           "Random Forest":best_rf_model}
metrics_bootstrap = []
for i in range(M):
    indices_train = np.random.choice(X_bootstrap.index,n_samples, replace =␣
 ↪True)
    indices_test = [c for c in X_bootstrap.index if c not in indices_train]
    x_train_bootstrap = X_bootstrap.iloc[indices_train]
    y_train_bootstrap = y_bootstrap.iloc[indices_train]
    x_test_bootstrap = X_bootstrap.iloc[indices_test]
    y_test_bootstrap = y_bootstrap.iloc[indices_test]
    # Vectorizamos
    vectorizer_bootstrap = TfidfVectorizer(max_features=100)
    vectorizer_bootstrap.fit(x_train_bootstrap)
    # Transformamos
    x_train_bootstrap = pd.DataFrame(vectorizer_bootstrap.
 ↪transform(x_train_bootstrap).toarray())
    x_test_bootstrap = pd.DataFrame(vectorizer_bootstrap.
 ↪transform(x_test_bootstrap).toarray())
    for m in models_.keys():
        models_[m].fit(x_train_bootstrap,y_train_bootstrap)
        pred_aux = models_[m].predict(x_test_bootstrap)
        f1_aux = f1_score(y_true = y_test_bootstrap, y_pred= pred_aux)
        metrics_bootstrap.append([i,m,f1_aux])
```

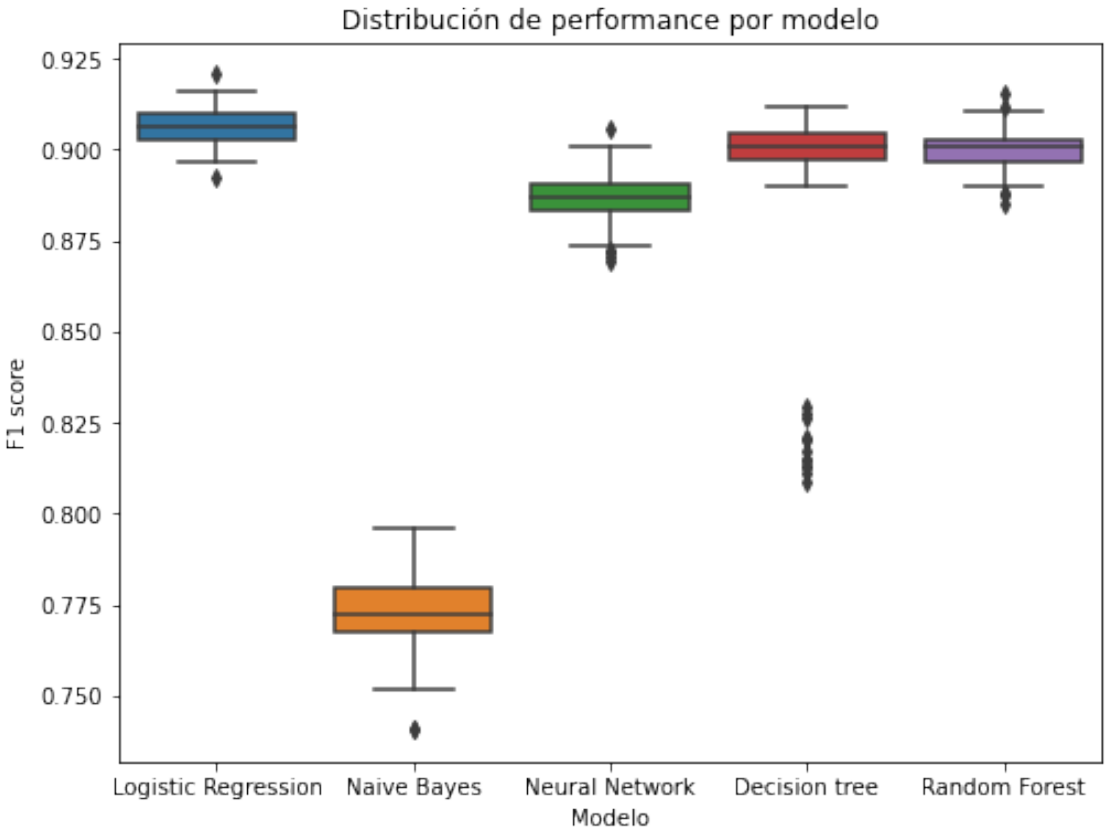
[54]:
```
metrics_bootstrap_df = pd.DataFrame(metrics_bootstrap,columns =␣
 ↪["iteracion","Modelo","F1 score"])
```

[55]:
```
def quantile_975(x):
    return np.quantile(x,0.975)
def quantile_025(x):
    return np.quantile(x,0.025)
def IC_95(x):
    return np.quantile(x,0.975) - np.quantile(x,0.025)
resumen_performance = metrics_bootstrap_df.pivot_table(index = "Modelo",
                                values = "F1 score",
                                aggfunc =␣
 ↪["mean",'median',"std",'max',"min",quantile_975,quantile_025])
resumen_performance.columns = [c[0] for c in resumen_performance.columns]
resumen_performance["IC 95%"] = "[" +resumen_performance["quantile_025"].
 ↪round(2).astype(str) +\
```

```
                                    resumen_performance["quantile_975"].round(2).
↪astype(str) + "]"
resumen_performance = resumen_performance.sort_values(by = "mean",ascending =␣
↪False)
```

[63]:
```
plt.figure(figsize = (8,6))
sns.boxplot(data = metrics_bootstrap_df,
            x = "Modelo",
            y = "F1 score")
plt.title("Distribución de performance por modelo")
```

[63]:
```
Text(0.5, 1.0, 'Distribución de performance por modelo')
```

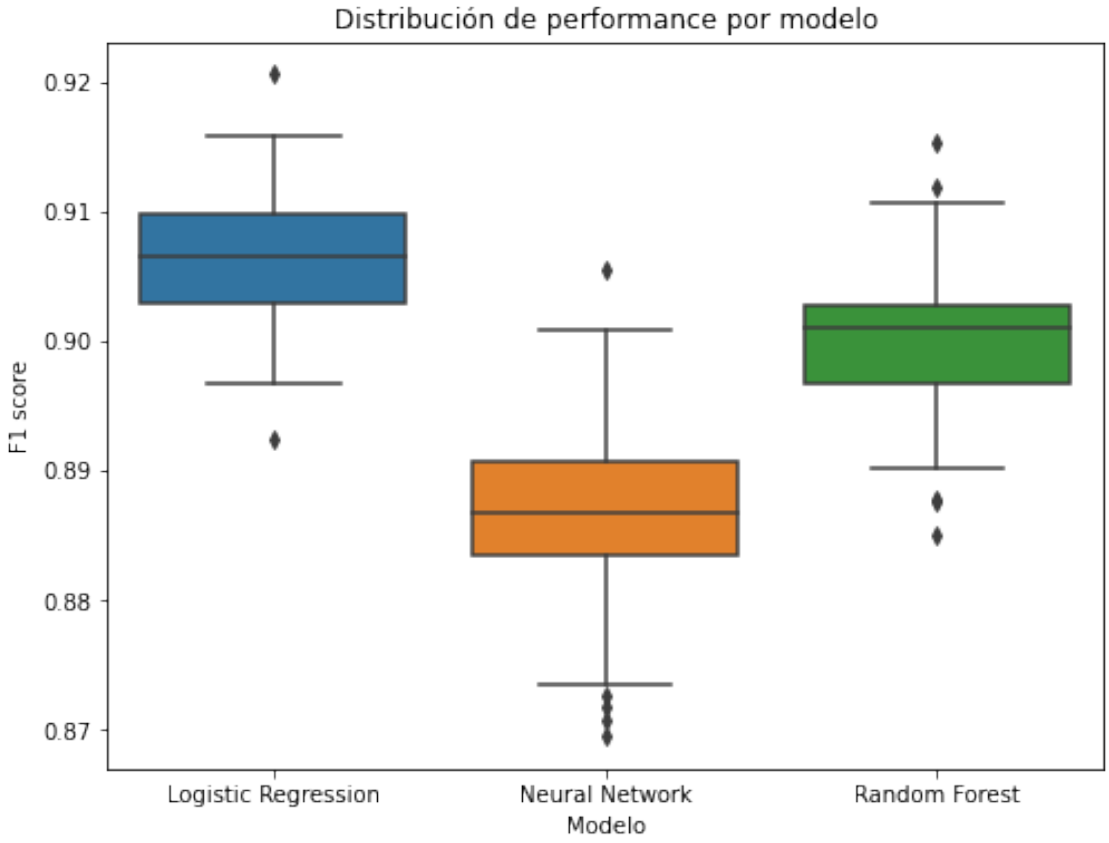


Podemos ver que los modelos Decision tree y Naive Bayes tienen mayor varianza que los demas modelos, entonces quedan descartados

[64]:
```
plt.figure(figsize = (8,6))
sns.boxplot(data = metrics_bootstrap_df[~metrics_bootstrap_df["Modelo"].
↪isin(["Decision tree","Naive Bayes"])],
            x = "Modelo",
```

```
            y = "F1 score")
plt.title("Distribución de performance por modelo")
```

[64]: `Text(0.5, 1.0, 'Distribución de performance por modelo')`



En promedio el modelo de regresion logistica dio mejores resultados y con una varianza mas pequeña, seguido del random forest y por ultimo la red neuronal, por lo que el modelo ganador fue la regresión logistica.

Lugares 1.- Regresión Logistica 2.- Random Forest 3.- Red neuronal 4.- Arbol de Decisión 5.- Naive Bayes

2.5 Modelo Ganador

El modelo ganador fue **Regresion Logistica**, por lo que pasara a produccion

3 Pase modelo a producción

3.1 Planteamiento

Para pasar el modelo a producción se seguira los siguientes pasos: * Crear una plataforma donde permita grabar la voz y lo pase a texto. - Esto lo logramos utlizando streamlit, creamos una pagina web que no necesita mucho conocimiento en backend ni front end y ademas dentro de su comunidad se encontro un código que permitio grabar y transcribir una grabación hecha en el momento. * El siguiente paso es usar ese texto que recibimos como input en nuestro modelo, por lo que se realizo la misma limpieza que cuando se entreno el modelo. * Ahora que ya recibimos el texto y lo limpiamos, procedemos a vectorizarlo, esto lo hacemos ya que guardamos el pickle de TfidfVectorizer y unicamente tuvimos que cargarlo en la pagina y lo aplicamos. * Una vez que ya tenemos todo listo se aplicara el modelo para realizar la prediccion * Dada la predicción se tomara la decisión a que grupo se redirigira la llamada.

3.2 Donde se subio

Como se menciono, se puso el modelo en producción utilizando streamlit, que permite crear una pagina para compartir apartir del codigo de github.

Se anexan los links tanto del repositorio de github, donde se encuentra este mismo notebook, y el código con el que se realizo la página, tambien se anexa el link donde puede ver en producción el modelo.

Nota: Al probar la aplicación revisar que su dispositivo este en español para mejores resultados.

Repositorio: https://github.com/carloschong/examen-final-mod2

Aplicación creada: https://share.streamlit.io/carloschong/examen-final-mod2/main/main.py