In [1]:
```
import time
inicio = time.time()
```

In [2]:
```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from imblearn.over_sampling import RandomOverSampler
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import LeaveOneOut

from sklearn.naive_bayes import GaussianNB

from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.metrics import average_precision_score
from sklearn.metrics import precision_recall_curve
```

Se importan los datos

In [3]:
```
df = pd.read_excel("..//Base_datos_Clasificacion binaria.xlsx", index_col = 0)
```

In [4]:
```
df
```

Out[4]:

	Licitacion reparto	Importe presupuestado	Importe adjudicado	MP	Empresa sancionada	UTE
0	0	1738093.21	1484428.72	6	1	0
1	0	469670.24	272492.00	4	0	0
2	0	1025088.19	707310.85	1	0	0
3	0	999890.00	497621.36	6	0	0
4	0	72598.27	47508.92	1	1	0
...	...	...	...	...	...	...
2211	1	5542028.88	5514320.00	6	1	0
2212	1	6095782.00	5100555.00	6	1	0
2213	1	3752906.00	3744400.00	6	1	0
2214	1	23896564.00	19547338.00	6	0	1
2215	1	33415098.00	33349532.00	6	1	0

2216 rows × 6 columns

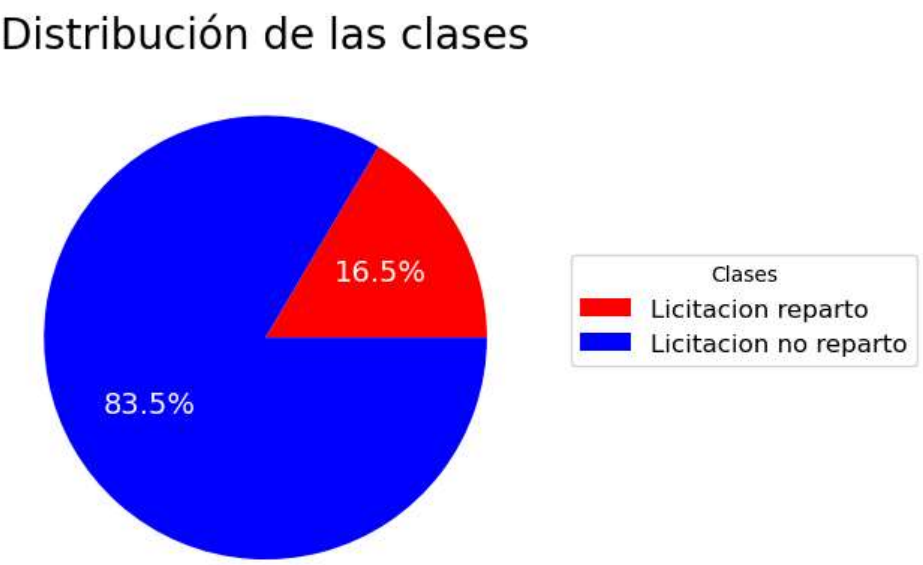
Se trata el desbalanceo de clases mediante una tecnica de Sobre - Muestreo. Hay que

instalar la libreria imbalanced - learn

In [5]:
```
y = df["Licitacion reparto"]
X = df[df.columns[1:]]
```

In [6]:
```
datos = [len(y[y == 1]), len(y[y == 0])]
etiquetas = ['Licitacion reparto', 'Licitacion no reparto']
colores = ['red', 'blue']

plt.pie(datos, labels = etiquetas, colors = colores, autopct = '%1.1f%%', textprop
plt.title('Distribución de las clases', fontsize = 20)
plt.legend(title = "Clases", bbox_to_anchor=(1.7, 0.55), loc='right', fontsize = 1
plt.show()
```



Se utiliza la técnica "RandomOverSampler" que es la generación de nuevos datos de la clase minoritaria mediante muestreo simple con reemplazamiento y de forma aleatoria.

In [7]:
```
# conda install -c conda-forge imbalanced-learn
ros = RandomOverSampler(random_state = 123)
X_res, y_res = ros.fit_resample(X, y)
df = pd.concat([y_res, X_res], axis = 1)
df
```

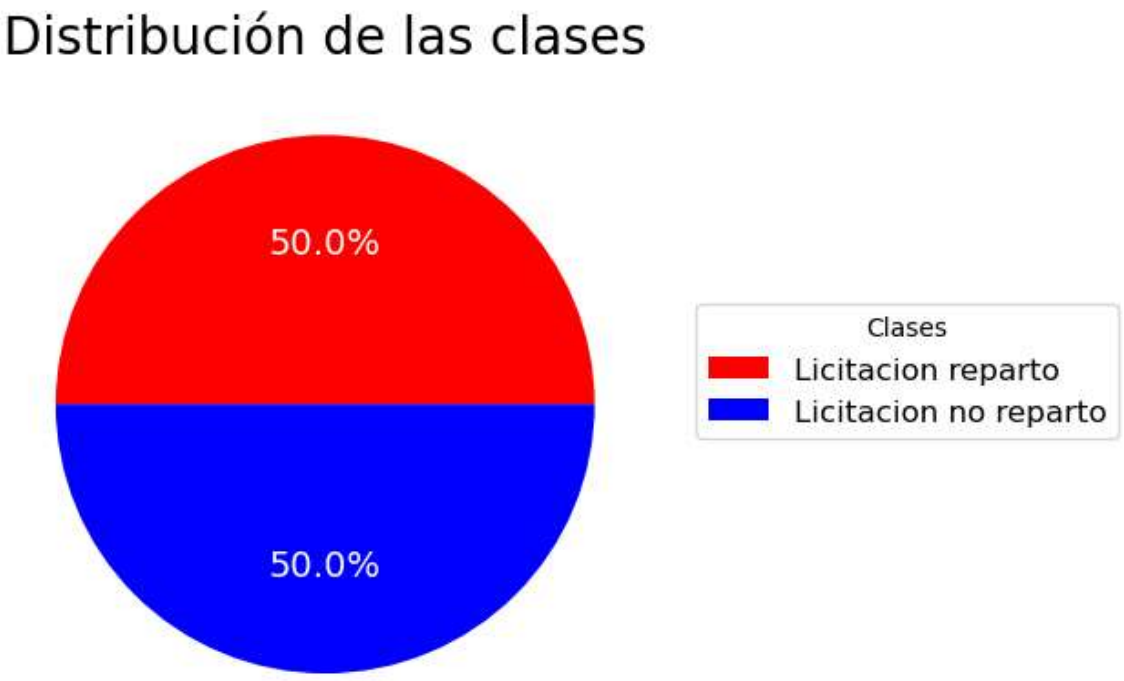
Out[7]:

	Licitacion reparto	Importe presupuestado	Importe adjudicado	MP	Empresa sancionada	UTE
0	0	1738093.21	1484428.72	6	1	0
1	0	469670.24	272492.00	4	0	0
2	0	1025088.19	707310.85	1	0	0
3	0	999890.00	497621.36	6	0	0
4	0	72598.27	47508.92	1	1	0
...	...	...	...	...	...	...
3697	1	25652311.00	23974592.00	6	0	0
3698	1	312588.00	342566.00	1	0	0
3699	1	140176.00	139475.00	1	0	0
3700	1	1573000.00	1339724.00	1	1	0
3701	1	2393596.00	2276722.00	6	0	0

3702 rows × 6 columns

In [8]:
```
datos = [len(y_res[y_res == 1]), len(y_res[y_res == 0])]
etiquetas = ['Licitacion reparto', 'Licitacion no reparto']
colores = ['red', 'blue']

plt.pie(datos, labels = etiquetas, colors = colores, autopct = '%1.1f%%', textprop
plt.title('Distribución de las clases', fontsize = 20)
plt.legend(title = "Clases", bbox_to_anchor=(1.7, 0.55), loc='right', fontsize = 1
plt.show()
```



Se estandarizan y normalizan los predictores numericos

In [9]:
```
predictores_numericos = df[["Importe presupuestado", "Importe adjudicado"]]
```

In [10]:
```
scaler = StandardScaler()
stand = scaler.fit_transform(predictores_numericos)
```

In [11]:
```
df[["Importe presupuestado", "Importe adjudicado"]] = stand
predictores_numericos = df[["Importe presupuestado", "Importe adjudicado"]]
predictores_numericos
```

Out[11]:

	Importe presupuestado	Importe adjudicado
0	-0.243817	-0.245165
1	-0.268501	-0.270829
2	-0.257692	-0.261621
3	-0.258182	-0.266062
4	-0.276228	-0.275593
...	...	...
3697	0.221564	0.231083
3698	-0.271557	-0.269345
3699	-0.274913	-0.273646
3700	-0.247029	-0.248230
3701	-0.231060	-0.228388

3702 rows × 2 columns

In [12]:
```
norm = (predictores_numericos - predictores_numericos.min()) / (predictores_numeri
```

In [13]:
```
df[["Importe presupuestado", "Importe adjudicado"]] = norm
predictores_numericos = df[["Importe presupuestado", "Importe adjudicado"]]
predictores_numericos
```

Out[13]:

	Importe presupuestado	Importe adjudicado
0	0.002679	0.002861
1	0.000699	0.000494
2	0.001566	0.001343
3	0.001526	0.000934
4	0.000079	0.000054
...	...	...
3697	0.039999	0.046796
3698	0.000454	0.000631
3699	0.000185	0.000234
3700	0.002421	0.002579
3701	0.003701	0.004409

3702 rows × 2 columns

In [14]:
```
df
```

Out[14]:

	Licitacion reparto	Importe presupuestado	Importe adjudicado	MP	Empresa sancionada	UTE
0	0	0.002679	0.002861	6	1	0
1	0	0.000699	0.000494	4	0	0
2	0	0.001566	0.001343	1	0	0
3	0	0.001526	0.000934	6	0	0
4	0	0.000079	0.000054	1	1	0
...	...	...	...	...	...	...
3697	1	0.039999	0.046796	6	0	0
3698	1	0.000454	0.000631	1	0	0
3699	1	0.000185	0.000234	1	0	0
3700	1	0.002421	0.002579	1	1	0
3701	1	0.003701	0.004409	6	0	0

3702 rows × 6 columns

Codificacion OneHotEncoder

In [15]:
```
codificador = OneHotEncoder()
```

In [16]:
```
df[["MP", "Empresa sancionada", "UTE"]] = df[["MP", "Empresa sancionada", "UTE"]].
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3702 entries, 0 to 3701
Data columns (total 6 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Licitacion reparto     3702 non-null   int64
 1   Importe presupuestado  3702 non-null   float64
 2   Importe adjudicado     3702 non-null   float64
 3   MP                     3702 non-null   category
 4   Empresa sancionada     3702 non-null   category
 5   UTE                    3702 non-null   category
dtypes: category(3), float64(2), int64(1)
memory usage: 98.1 KB
```

In [17]:
```
codificacion = codificador.fit_transform(df[["MP"]])

mercados = pd.DataFrame(codificacion.toarray(),
                        columns = ["MP1", "MP4", "MP6"])
df = pd.concat([df, mercados], axis = 1)
df.drop("MP", axis = 1, inplace = True)
df
```

Out[17]:

	Licitacion reparto	Importe presupuestado	Importe adjudicado	Empresa sancionada	UTE	MP1	MP4	MP6
0	0	0.002679	0.002861	1	0	0.0	0.0	1.0
1	0	0.000699	0.000494	0	0	0.0	1.0	0.0
2	0	0.001566	0.001343	0	0	1.0	0.0	0.0
3	0	0.001526	0.000934	0	0	0.0	0.0	1.0
4	0	0.000079	0.000054	1	0	1.0	0.0	0.0
...	...	...	...	...	...	...	...	...
3697	1	0.039999	0.046796	0	0	0.0	0.0	1.0
3698	1	0.000454	0.000631	0	0	1.0	0.0	0.0
3699	1	0.000185	0.000234	0	0	1.0	0.0	0.0
3700	1	0.002421	0.002579	1	0	1.0	0.0	0.0
3701	1	0.003701	0.004409	0	0	0.0	0.0	1.0

3702 rows × 8 columns

In [18]:
```
nombre_clases = ["Licitacion no reparto", "Licitacion reparto"]
nombre_predicciones = list(map(lambda x : x + " predicho", nombre_clases))
```

Se hace la estimacion por NaiveBayes

In [19]:
```
X = df[df.columns[1:]]
y = df["Licitacion reparto"]

loo = LeaveOneOut()
gnb = GaussianNB()
predicciones = []
prob_predicciones = []

for train_index, test_index in loo.split(X):
```

```
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y[train_index], y[test_index]

    gnb.fit(X_train, y_train)
    y_pred = gnb.predict(X_test) #son 0 o 1 dependiendo de si la probabilidad es m
    y_pred_prob = gnb.predict_proba(X_test)
    predicciones.append(y_pred[0])
    prob_predicciones.append(y_pred_prob[0][1])
```

In [20]:
```
gnb.get_params()
```

Out[20]:
```
{'priors': None, 'var_smoothing': 1e-09}
```

In [21]:
```
predicciones = pd.Series(predicciones, name = "Prediccion licitacion reparto")
prob_predicciones = pd.Series(prob_predicciones, name = "Prob prediccion")
cp = pd.concat([y, predicciones, prob_predicciones], axis = 1)
cp
# Este dataframe no contiene las predicciones calculadas en el punto optimo de la c
# Contiene las predicciones calculadas segun el punto por defecto para determinar l
```

Out[21]:

	Licitacion reparto	Prediccion licitacion reparto	Prob prediccion
0	0	0	0.013187
1	0	1	0.999650
2	0	0	0.459507
3	0	0	0.005597
4	0	1	0.670565
...	...	...	...
3697	1	0	0.008386
3698	1	0	0.457502
3699	1	0	0.457793
3700	1	1	0.666102
3701	1	0	0.005513

3702 rows × 3 columns

CURVA ROC. El punto optimo es = (fpr = 0, tpr = 1)

In [22]:
```
auc_score = roc_auc_score(y, prob_predicciones)
auc_score
```

Out[22]:
```
0.7768043964729447
```

In [23]:
```
fpr, tpr, thresholds = roc_curve(y, prob_predicciones)
#fpr es 1 - especificidad = 1 - (TN / (FP + TN)) = FP / (FP + TN)
#tpr es la sensibilidad = TP / (TP + FN)
```

In [24]:
```
distances = np.linalg.norm(np.column_stack((fpr, tpr)) - np.array([0, 1]), axis=1)
optimal_threshold_index = np.argmin(distances)
```

```
optimal_threshold = thresholds[optimal_threshold_index]
optimal_threshold
```

Out[24]:
```
0.09155009466032639
```

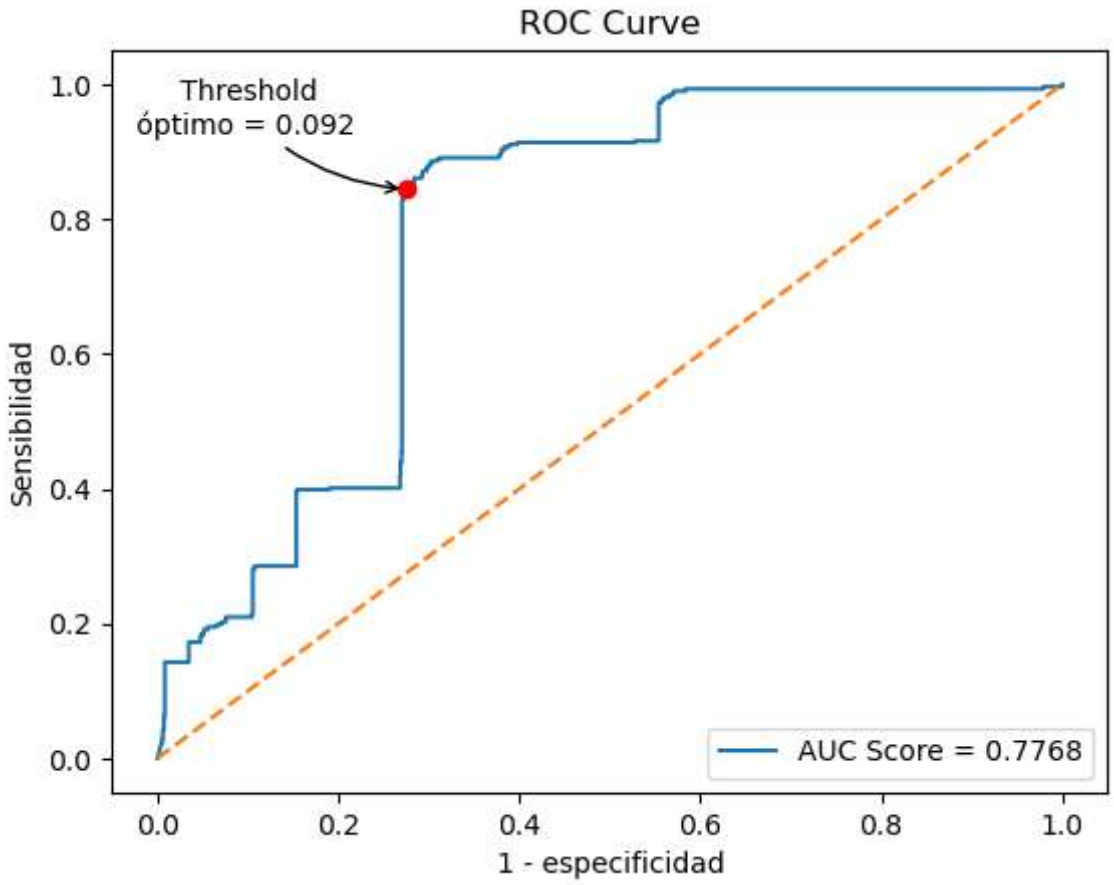
In [25]:
```
optimal_fpr = fpr[optimal_threshold_index]
optimal_fpr
```

Out[25]:
```
0.2744462452728255
```

In [26]:
```
optimal_tpr = tpr[optimal_threshold_index]
optimal_tpr
```

Out[26]:
```
0.8444084278768234
```

In [27]:
```
plt.plot(fpr, tpr, label = f'AUC Score = {auc_score:.4}')
plt.plot([0,1], linestyle="--")
plt.plot(optimal_fpr, optimal_tpr, 'ro')
plt.annotate(f'      Threshold\n óptimo = {optimal_threshold:.2}',
             xy=(optimal_fpr, optimal_tpr), xycoords='data',
             xytext=(-100, +20), textcoords='offset points', fontsize=10,
             arrowprops=dict(arrowstyle="->", connectionstyle = "arc3, rad = .2"))
plt.ylabel('Sensibilidad')
plt.xlabel('1 - especificidad')
plt.title('ROC Curve')
plt.legend(loc = 'lower right')
plt.show()
```



In [28]:
```
cp["Prediccion licitacion reparto en el punto optimo"] = 0
prediccion_final = cp["Prediccion licitacion reparto en el punto optimo"]
prediccion_final = cp.apply(lambda x: 1 if x["Prob prediccion"] >= optimal_thresho
prediccion_final
```

Out[28]:
```
0       0
1       1
2       1
3       0
4       1
       ..
3697    0
3698    1
3699    1
3700    1
3701    0
Length: 3702, dtype: int64
```

In [29]:
```
cp["Prediccion licitacion reparto en el punto optimo"] = prediccion_final
pd.concat([cp, prediccion_final], axis = 1)
cp.drop(columns = "Prediccion licitacion reparto", axis = 1, inplace = True)
cp
```

Out[29]:

	Licitacion reparto	Prob prediccion	Prediccion licitacion reparto en el punto optimo
0	0	0.013187	0
1	0	0.999650	1
2	0	0.459507	1
3	0	0.005597	0
4	0	0.670565	1
...	...	...	...
3697	1	0.008386	0
3698	1	0.457502	1
3699	1	0.457793	1
3700	1	0.666102	1
3701	1	0.005513	0

3702 rows × 3 columns

In [30]:
```
cm = confusion_matrix(y, cp["Prediccion licitacion reparto en el punto optimo"]) #
cm = pd.DataFrame(cm, columns = nombre_clases, index = nombre_predicciones)
cm # esta matriz de confusion esta calculada en el punto optimo de la curva
```

Out[30]:

	Licitacion no reparto	Licitacion reparto
Licitacion no reparto predicho	1343	508
Licitacion reparto predicho	288	1563

Metricas de la Matriz de Confusion

In [31]:
```
TP = cm.iloc[1,1]
FN = cm.iloc[1,0]
FP = cm.iloc[0,1]
TN = cm.iloc[0,0]
```

In [32]:
```
Tasa_de_aciertos = (TP + TN) / len(df)
Tasa_de_aciertos
```

Out[32]:
```
0.7849810913019989
```

In [33]:
```
Tasa_de_errores = 1 - Tasa_de_aciertos
Tasa_de_errores
```

Out[33]:
```
0.2150189086980011
```

In [34]:
```
Sensibilidad = TP / (TP + FN)
Sensibilidad # se calculo antes. Es optimal_tpr
```

Out[34]:
```
0.8444084278768234
```

In [35]:
```
Tasa_de_falsos_negativos = 1 - Sensibilidad
Tasa_de_falsos_negativos
```

Out[35]:
```
0.15559157212317665
```

In [36]:
```
Especificidad = TN / (FP + TN)
Especificidad # se calculo antes. Es optimal_fpr = 1 - Especificidad
```

Out[36]:
```
0.7255537547271745
```

In [37]:
```
Precision = TP / (TP + FP)
Precision
```

Out[37]:
```
0.754707870593916
```

In [38]:
```
FP / (FP + TP) # 1 - Precision
```

Out[38]:
```
0.24529212940608403
```

In [39]:
```
F1 = (2 * Sensibilidad * Precision) / (Sensibilidad + Precision)
F1
```

Out[39]:
```
0.7970423253442122
```

Métricas de clasificación binaria en el punto óptimo

In [40]:
```
nombre_fichero = "Clasificacion binaria Sobre_Muestreo gnb"
```

In [41]:
```
indice_resultados = ["Tasa de aciertos", "Sensibilidad", "Especificidad", "Precisi
resultados_valores = [Tasa_de_aciertos, Sensibilidad, Especificidad, Precision, F1

resultados = pd.DataFrame(data={f"{nombre_fichero}": resultados_valores},
                          index=indice_resultados)
resultados
```

Out[41]:

	Clasificacion binaria Sobre_Muestreo gnb
Tasa de aciertos	0.784981
Sensibilidad	0.844408
Especificidad	0.725554
Precision	0.754708
F1	0.797042

Se exportan los datos

In [42]:
```
nombre_columna = list(resultados.columns)[0]
resultados.to_excel(f"Resultados de las métricas de {nombre_fichero}.xlsx")
```

In [43]:
```
average_precision = average_precision_score(y, prob_predicciones)
average_precision
```

Out[43]:
```
0.7246651538832807
```

In [44]:
```
precision, recall, thresholds = precision_recall_curve(y, prob_predicciones)
#precision = TP / (TP + FP)
#recall = sensibilidad = TP / (TP + FN)
```

In [45]:
```
distances = np.linalg.norm(np.column_stack((precision, recall)) - np.array([1, 1]),
optimal_threshold_index = np.argmin(distances)
optimal_threshold = thresholds[optimal_threshold_index]
optimal_threshold
```

Out[45]:
```
0.029688669730336868
```

In [46]:
```
optimal_precision = precision[optimal_threshold_index]
optimal_precision
```

Out[46]:
```
0.7454545454545455
```

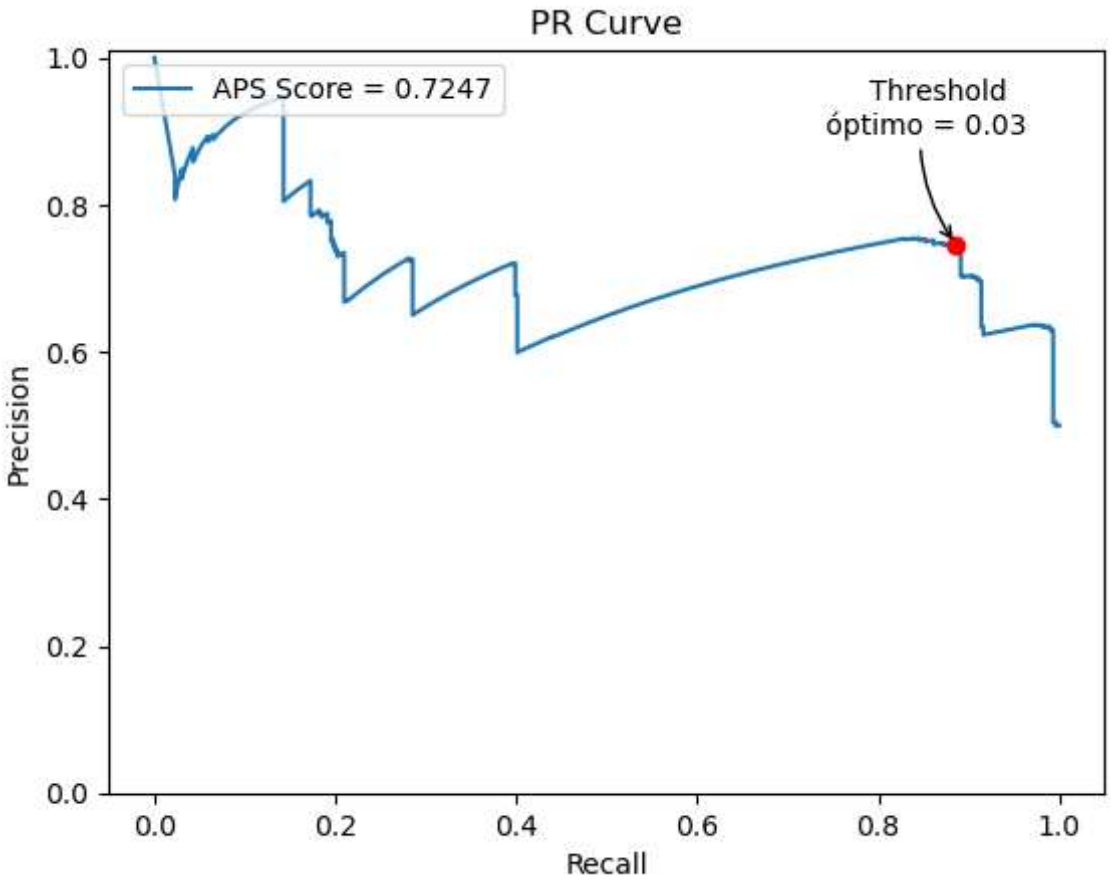
In [47]:
```
optimal_recall = recall[optimal_threshold_index]
optimal_recall
```

Out[47]:
```
0.8860075634792004
```

CURVA PR. El punto óptimo es = (precision = 1, recall = 1)

In [48]:
```
plt.plot(recall, precision, label = f'APS Score = {average_precision:.4}')
plt.plot(optimal_recall, optimal_precision, 'ro')
plt.annotate(f'      Threshold\n óptimo = {optimal_threshold:.2}',
             xy=(optimal_recall, optimal_precision), xycoords='data',
             xytext=(-50, +40) ,textcoords='offset points', fontsize=10,
             arrowprops=dict(arrowstyle="->", connectionstyle = "arc3, rad = .2"))
plt.ylabel('Precision')
plt.xlabel('Recall')
plt.title('PR Curve')
plt.legend(loc = 'upper left')
```

```
plt.ylim(0, 1.01)
plt.show()
```



Se calcula cuanto tiempo ha transcurrido en la ejecucion de todo el notebook y se guarda el tiempo en un fichero de texto para almacenarlo.

In [49]:
```
fin = time.time()
tiempo_transcurrido = fin - inicio
minutos = int((tiempo_transcurrido % 3600) // 60)
segundos = int(tiempo_transcurrido % 60)
with open(f"Tiempo de ejecucion transcurrido en el notebook {nombre_fichero}.txt",
    f.write(f"Tiempo transcurrido: {minutos} minutos, {segundos} segundos")
```

In [50]:
```
print(f"Tiempo transcurrido: {minutos} minutos, {segundos} segundos")
```

```
Tiempo transcurrido: 0 minutos, 20 segundos
```