Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ingeniería Eléctrica

Tarea 1

Clasificador de Bayes y aproximación Gaussiana

Integrantes: Vincko Fabres
Profesor: Javier Ruiz del Solar
Auxiliar: Patricio Loncomilla Z.
Ayudantes: Juan Pablo Cáceres B.
Pablo Troncoso P.
Rodrigo Salas O.
Rudy García
Sebastian Solanich

Fecha de entrega: 11 de abril de 2021
Santiago, Chile

Índice de Contenidos

Índice de Figuras

Índice de Códigos

1. Introducción

En el campo de machine learning son variadas las técnicas de aprendizaje, esta experiencia busca un acercamiento a 2 de estas; clasificador de Bayes y mediante aproximaciones Gaussianas. Para realizar esta experiencia se utilizará la plataforma colab [1], programando con el lenguaje de programación Python y la utilización de las bibliotecas Pandas, Numpy, Seaborn, Matplotlib y Time. Para entrenar el clasificador y testeo de este se utilizará el conjunto de datos MAGIC Gamma Telescope Data Set, el cual corresponde a un conjunto de simulaciones de air showers generados por rayos gamma primarios v/s hadrones, este conjunto de datos posee 10 características, sin contar su clase; la cual puede ser hadrón o g no-hadrón.
La experiencia cuenta con 4 sub-bloques los cuales son:

- Visualización y análisis de datos
- Clasificación usando naive Bayes e histogramas
- Clasificación usando gaussianas multivariantes
- Comparación de resultados

Las cuales son explicadas en en Desarrollo del informe para posteriormente ser analizadas y dar paso a la sección de Conclusiones.

[1] https://colab.research.google.com/

2. Desarrollo

2.1. Parte 1: Visualización y análisis de datos

La primera parte consta de la manipulación del conjunto de datos **magic04.data**, utilizando la función *pd.read_csv()* con esto se permite subir el archivo al entorno de ejecución.

Código 1: Subida de datos a Collab

```
#subir archivo magic04.data
from google.colab import files
uploaded= files.upload()
```

Debido a que posteriormente se utilizarán las bibliotecas Pandas, Numpy, Seaborn, Matplotlib y Time, se deben importar, como sigue a continuación:

Código 2: Bibliotecas importadas

```
%matplotlib inline
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import time
```

2.1.1.

Seguida la inicialización del código se debe leer el conjunto de datos y nombrar a sus columnas con su respectiva característica, como sigue a continuación:

Código 3: Nombramiento de columnas

```
#Parte 1.1
nombres_columnas=['fLength', 'fWidth', 'fSize', 'fConc', 'fConc1', 'fAsym', 'fM3Long', 'fM3Trans', '
    ↪ fAlpha', 'fDist', 'class']
data = pd.read_csv("magic04.data",names=nombres_columnas)
```

2.1.2.

Una vez nombradas las columnas se debe separar el conjunto de datos en 2 subconjuntos; clase positiva(Hadrón) y clase negativa(No-Hadrón).

Código 4: Separación por subconjunto

```
#Parte 1.2
df= pd.DataFrame(data)
```

```
clase_positiva=df.loc[df['class'] == 'h']
clase_negativa=df.loc[df['class'] == 'g']
```

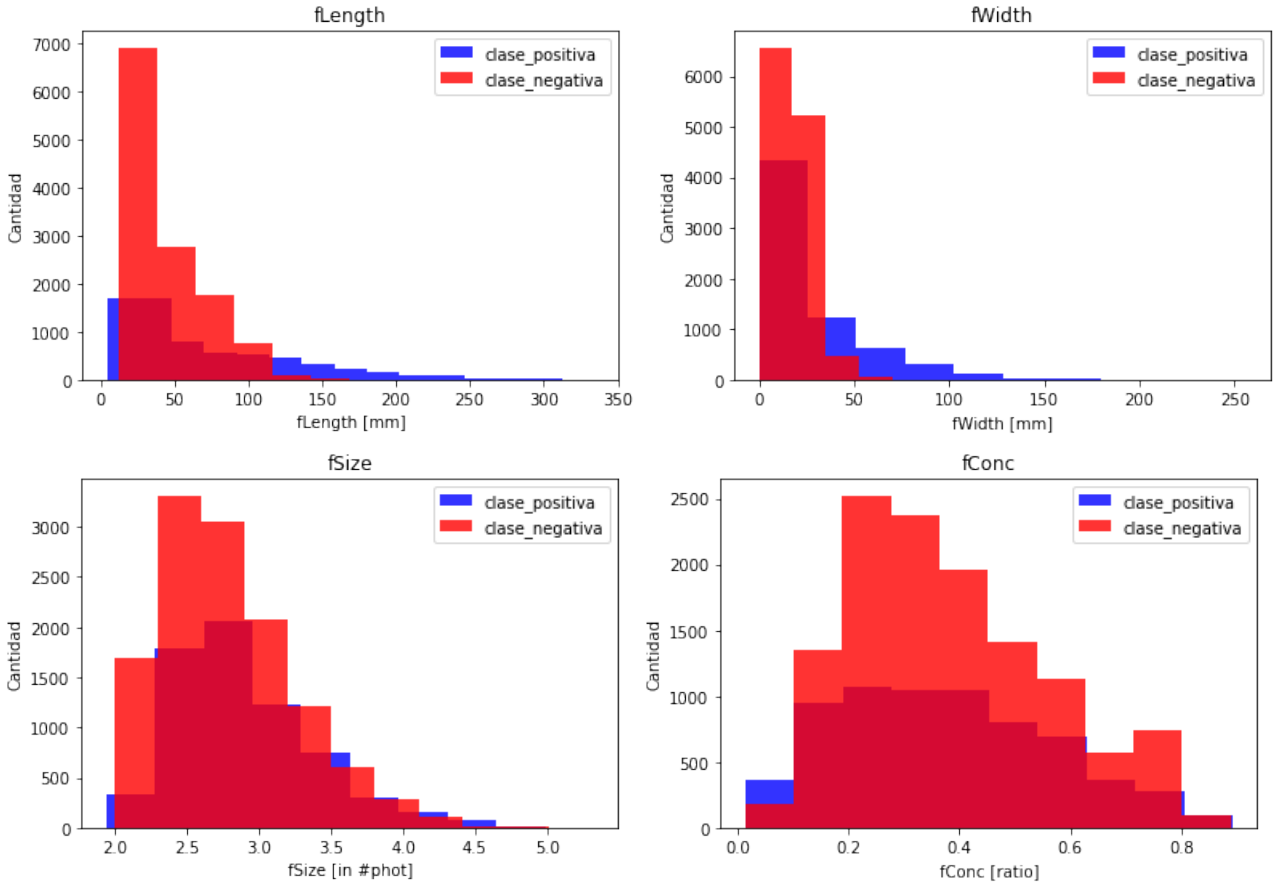
2.1.3.

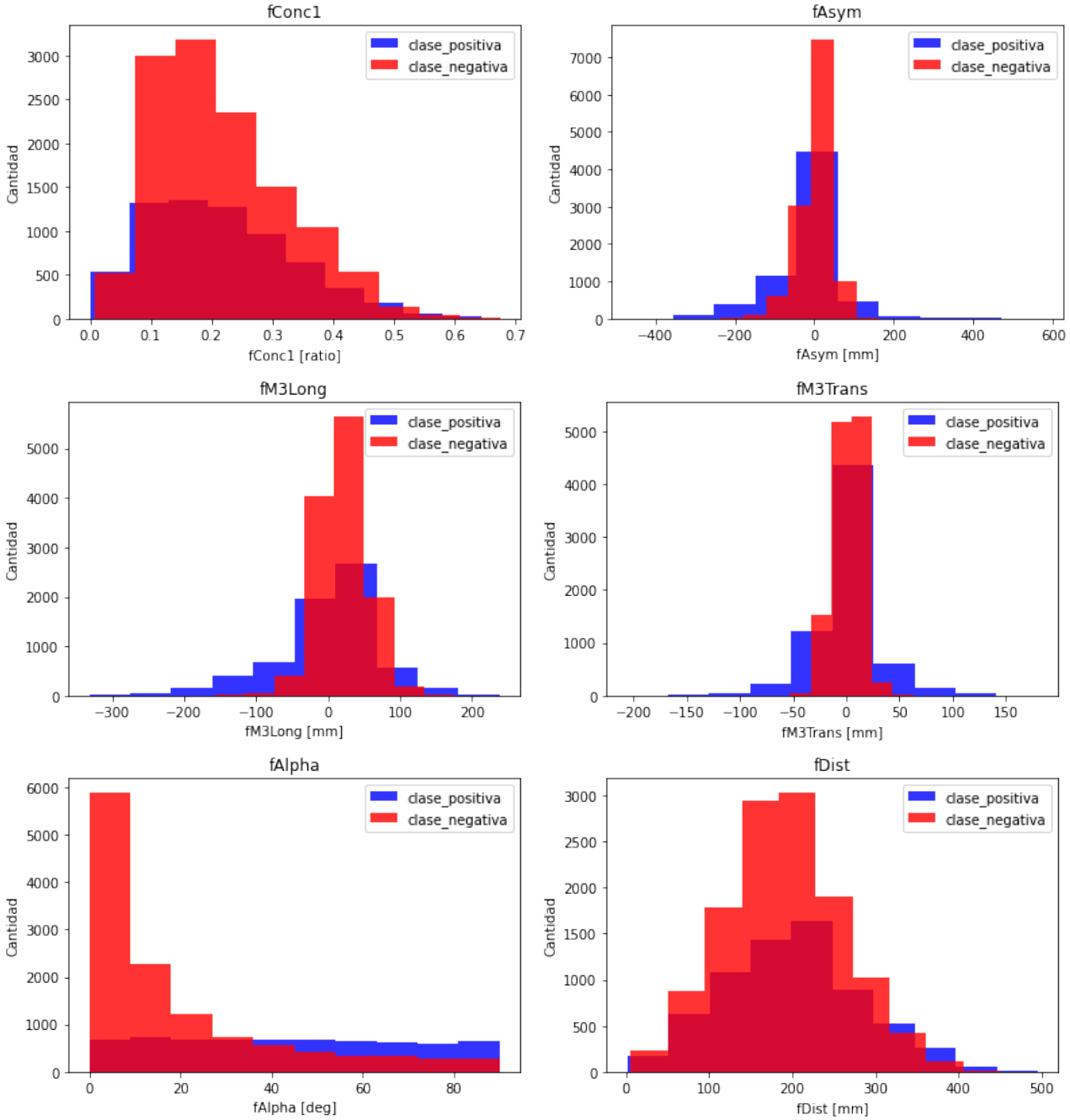
Para cada característica de los subconjuntos se deben generar histogramas, los cuales deben presentarse superpuestos, como en el siguiente ejemplo.

Código 5: Histograma por clase

```
#Parte 1.3
plt.hist(clase_positiva['fLength'],color='b',alpha=0.8,bins=15, label='clase_positiva')
plt.hist(clase_negativa['fLength'],color='r',alpha=0.8, label= 'clase_negativa')
plt.legend()
plt.title("fLength")
plt.ylabel('Cantidad')
plt.xlabel('fLength [mm]')
plt.show()
```

Para crear cada histograma se debe cambiar el *string* que esta dentro de 'clase_positiva[ ]' y 'clase_negativa[ ]' en la función **plt.hit()** al *string* correspondiente al nombre de columna, cambiar los nombres de cada label y su respectiva medida, resultando los siguientes histogramas:





2.1.4.

Una vez obtenidos los histogramas se debe obtener la matriz de correlación de los datos, incluyendo la clase como 0 para no-hadron o 1 para hadrón, utilizando de base la función de correlación de la librería Pandas, lo cual se presenta a continuación:

Código 6: Cálculo de Matriz de Correlación

```
#Parte 1.4
#primero se cambian las etiquetas 'g' y 'h' a 0 y 1 respectivamente
df['class'].replace('g',0, inplace = True)
df['class'].replace('h',1, inplace = True)
clase_positiva=df.loc[df['class'] == 1]
clase_negativa=df.loc[df['class'] == 0]

#####calculo de matriz de correlación################
corr=df.corr()
correlacion=corr.abs()
```

2.1.5.

Se debe indicar en orden las carácteristicas más relacionadas con la clase, lo que es implementado en el siguiente código:

Código 7: Correlación ordenada para cada clase

```

#correlacion.iloc[0] # Primera fila
#correlacion.iloc[:,0] #Primera columna

matriz_corr=[]
for i in range(len(correlacion)):
  matriz_corr.append(list(correlacion.iloc[:,i]))
  #matriz_corr_ordenada.append((correlacion.index[i]))
matriz_corr

matriz_corr_ordenada=[]
for i in range(len(correlacion)):
    matriz_corr_ordenada.append(sorted(matriz_corr[i]))
matriz_corr_ordenada


for i in range(len(correlacion)):
  print ('Para '+ '"'+correlacion.index[i]+'"' + ' las correlaciones en orden ascendente son:')
  for j in range(len(correlacion)):
    print(nombres_columnas[matriz_corr[i].index((matriz_corr_ordenada[i][j]))])
  print('-----------------------------------------------------------------------')
```

El cual retorna para cada clase en orden ascendente sus características más correlacionadas.

2.1.6.

Esta parte pide los 5 pares de características más correlacionadas entre sí. Notando que la matriz es simétrica se trabaja con su parte triangular superior, sin incluir la diagonal, ya que la correlación es igual a 1 debido a que es la correlación con la misma característica. El código para realizar esta

tarea es el siguiente:

Código 8:

```
#Parte 1.6
correlacion_triangular_sup = corr.abs()
pares_mas_corr=[]
for i in range(len(correlacion_triangular_sup)):
  for j in range(len(correlacion_triangular_sup)):
    if j<=i:
      correlacion_triangular_sup.iloc[i][j]=0


triangular_tableada=correlacion_triangular_sup.unstack()
triangular_tableada.sort_values(ascending=False)[:5]
```

El que imprime como resultado:

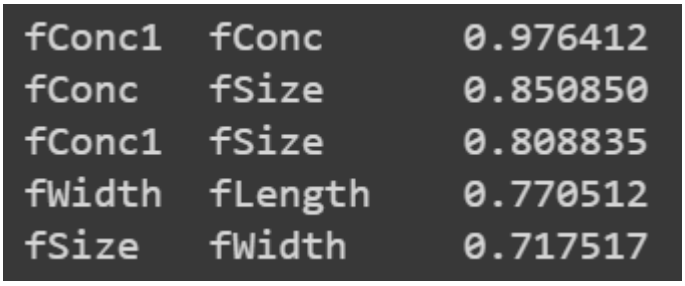


Figura 1: Pares más correlacionados

2.1.7.

Se busca obtener la gráfica de las dos características distintas más correlacionadas en un gráfico de dispersión, el cual se implementa como sigue:

Código 9: Código de gráfico de dispersión para mayor correlación

```
#Parte 1.7
plt.scatter(clase_positiva['fConc'],clase_positiva['fConc1'],color='b', s=0.7,label='clase positiva')
plt.scatter(clase_negativa['fConc'],clase_negativa['fConc1'],color='r', s=0.7,label='clase negativa')
plt.title("Mayor correlación")
plt.xlabel('fConc')
plt.ylabel('fConc1')
plt.legend(loc="upper left", title="Clases")
plt.show()
```

El cual imprime el siguiente gráfico de dispersión:

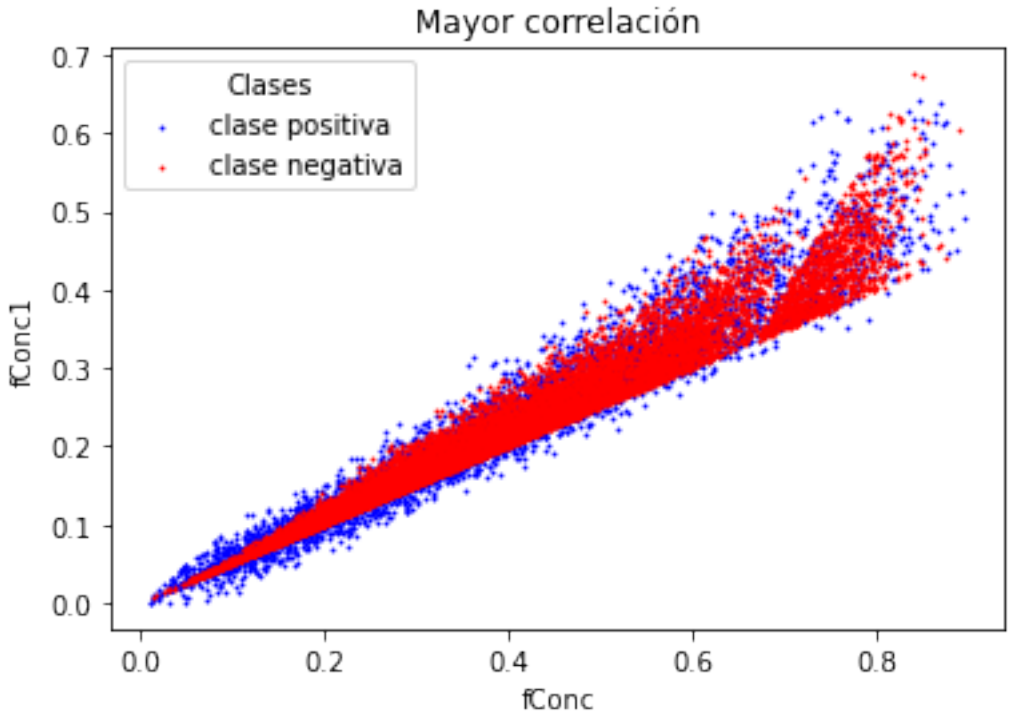


Figura 2: Gráfico de dispersión para mayor correlación

2.1.8.

Esta parte pide las 5 características de menor correlación, lo cual es realizado de forma similar a la tarea de buscar las 5 características de mayor correlación, el código que realiza esta tarea es el siguiente:

Código 10:

```
#Parte 1.8
desc=triangular_tableada.sort_values(ascending=True)
desc[desc>0][:5]
```

El que da como resultado lo siguiente:

```
fM3Trans  fAsym      0.002553
class     fM3Trans   0.003837
fAlpha    fM3Trans   0.004659
class     fConc1     0.004797
fAlpha    fLength    0.008777
dtype: float64
```

Figura 3: Pares menos correlacionados

2.1.9.

Se busca obtener la gráfica de las dos características distintas menos correlacionadas en un gráfico de dispersión, el cual se implementa como sigue:

Código 11: Código de gráfico de dispersión para menor correlación

```
#Parte1.9

plt.scatter(clase_positiva['fAsym'],clase_positiva['fM3Trans'],color='b', s=0.7,label='clase positiva')
plt.scatter(clase_negativa['fAsym'],clase_negativa['fM3Trans'],color='r', s=0.7,label='clase negativa')
plt.title("Menor correlación")
plt.xlabel('fAsym')
plt.ylabel('fM3Trans')
plt.legend(loc="lower right", title="Clases")
plt.show()
```

Esta parte se realizó de forma análoga a la búsqueda de las dos características distintas más correlacionadas, el cual plotea la siguiente gráfica:

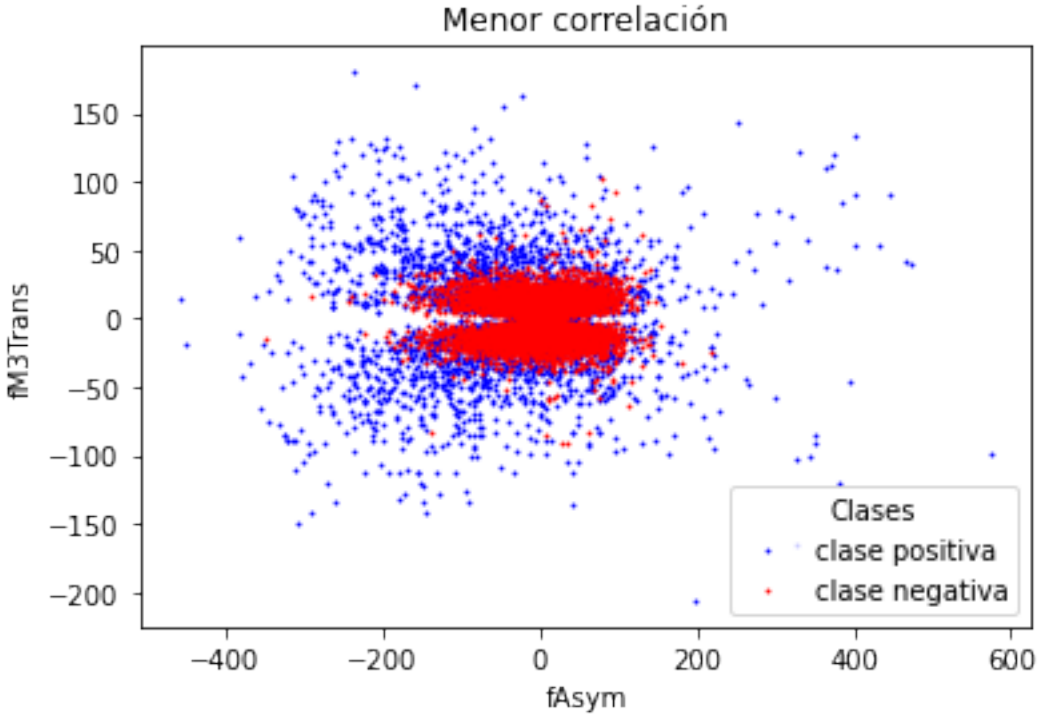


Figura 4: Gráfico de dispersión para menor correlación

2.1.10.

Se busca graficar la matriz de correlación mediante mapa de colores, lo cual se implementa como sigue:

Código 12: Heatmap matriz de correlación

```
1
2 #Parte 1.10
3 sns.heatmap(correlacion)
```

Dando como resultado la matriz:

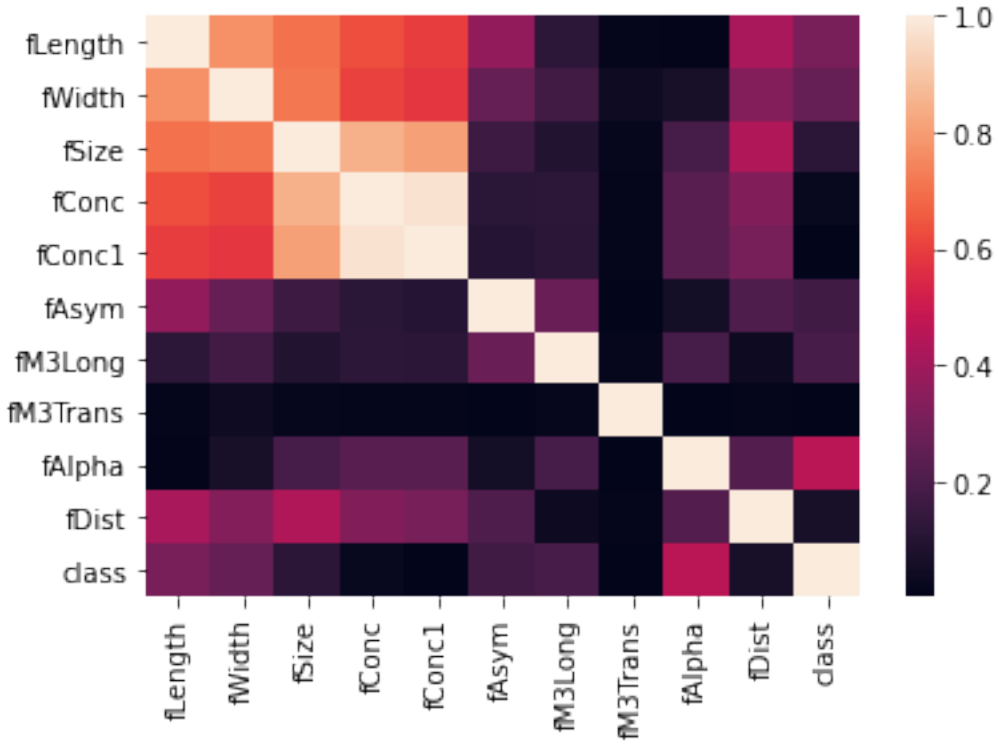


Figura 5: Heatmap

2.2. Parte 2: Clasificación usando naive Bayes e histogramas

Se debe programar, entrenar y calibrar un clasificador naive Bayes para determinar en forma probabilística si un candidato corresponde a un pulsar. La regla de decisión del mencionado clasificador está dada por:

$$\frac{P(\text{características}|\text{hadrón})}{P(\text{características}|\text{no\_hadrón})} \geq \Theta$$

Figura 6: regla de decisión

Donde el umbral Θ depende de los costos asociados a cada decisión y de las probabilidades a priori:

Debido a que no se poseen las distribuciones de probabilidad de los conjuntos, se deben estimar mediante histogramas y un modelo de aproximación Gaussiano

2.2.1.

Lo primero que se debe hacer es dividir la base de datos en 2 conjuntos de interés; entrenamiento y prueba, los cuales deben ser representativos. Para estimar la representatividad se debe verificar una proporción entre clases similar a la base de datos.
Lo primero es permutar al azar la base de datos, la cual estaba ordenada con respecto a las clases.

Código 13: Generación de subconjuntos Entrenamiento y Prueba

```
1
2  #Parte 2.11
3  reorden = df
4  reorden = reorden.sample(frac=1).reset_index(drop=True)
5  num_entr = int(len(reorden)*0.8) # num_test = len(reorden)- num_entr
6  entrenamiento= reorden.iloc[0:num_entr]
7  test= reorden.iloc[num_entr:]
8  test= test.reset_index(drop=True)
9
10 #proporcion conjunto completo
11 ratio_df= len(clase_negativa)/len(clase_positiva) #0/1
12 #poporcion conj entrenamiento
13 ratio_reorden=len(entrenamiento.loc[entrenamiento['class']==0])/len(entrenamiento.loc[
       ↪ entrenamiento['class']==1])
14 print((abs(ratio_df-ratio_reorden)/ratio_df)*100 ,'% de diferencia en representatividad') #formula
       ↪ de error
```

El código en su parte final imprime el error porcentual entre la base datos y el conjunto de entrenamiento.

2.2.2.

Utilizando Naive Bayes, es decir, regla de Bayes con el supuesto de que cada característica es independiente, se deben encontrar los histogramas de cada clase a partir de las muestras del conjunto de entrenamiento, debido a que el mayor coste computacional lo tiene el gráfico de histogramas, solo sus datos fueron almacenados, los cuales son las verosimilitudes de las clases.

Código 14: Entrenamiento Naive Bayes

```
#Parte 2.12

tiempo_bayes_inicio = time.time()

clase_positiva_entrenamiento=entrenamiento.loc[entrenamiento['class'] == 1]
clase_negativa_entrenamiento=entrenamiento.loc[entrenamiento['class'] == 0]

Matriz_verosimilitudes_pos=[]
Matriz_verosimilitudes_neg=[]

medida_caracteristica_pos=[]
medida_caracteristica_neg=[]

for i in range(len(nombres_columnas)):
  Count_pos, bin_pos = np.histogram(clase_positiva_entrenamiento[nombres_columnas[i]], density
    ↪ =True, bins=25)
  Count_neg, bin_neg = np.histogram(clase_negativa_entrenamiento[nombres_columnas[i]], density
    ↪ =True, bins=25)

  medida_caracteristica_pos.append(bin_pos)#rangos
  medida_caracteristica_neg.append(bin_neg)

  L_neg=[]
  L_pos=[]
  for j in range(len(Count_pos)):
    if j == 0 :
      L_pos.append(bin_pos[j])
    else:
      L_pos.append(bin_pos[j]-bin_pos[j-1])

  for k in range(len(Count_pos)):
    L_pos[k]=(L_pos[k]*Count_pos[k])

  for l in range(len(Count_neg)):
    if l == 0 :
      L_neg.append(bin_neg[l])
    else:
      L_neg.append(bin_neg[l]-bin_neg[l-1])

  for m in range(len(Count_neg)):
    L_neg[m]=(L_neg[m]*Count_neg[m])

```

```
  Matriz_verosimilitudes_pos.append(L_pos)
  Matriz_verosimilitudes_neg.append(L_neg)
  L_neg=[]
  L_pos=[]

tiempo_bayes_final = time.time()
```

Adicionalmente se utiliza la función **time.time()**, la cual servirá más adelante para comparar los tiempos de ejecución entre los 2 métodos a utilizar.

2.2.3.

Se deben hallar las verosimilitudes en ambas clases para cada muestra del conjunto de prueba, usando los histogramas, lo cual se realiza mediante el siguiente código.

Código 15: Creación LUT Naive Bayes

```

#Parte 2.13 y 2.14
def producto(L):
  return np.prod(L)

def ubicar(carac_elemento,medida): #buscar la posición del bin correspondiente en la matriz de verosimilitudes
  k=0
  while k < len(medida):
    if carac_elemento < medida[0]:
      return 0
    elif carac_elemento >= medida[k] and carac_elemento < medida[k+1]:
        return k
    elif carac_elemento >= medida[len(medida)-2]: #se resta 1 ya que el ultimo indice es len(medida) -1 y como la ubicacion es con resp a LUT otro mas
        return (len(medida)-2)
    k+=1

def verosimilitudelemento(elemento,medidacarac,matriz):
  L=[]
  for j in range(10):#recorre las 10 caracteristicas
    k=ubicar(elemento[j],medidacarac[j])
    if float(matriz[j][k]) == 0.0:
      L.append(10**-10)
    else:
      L.append(matriz[j][k])
  return producto(L)

def verosimilitud_positiva(df):
  #verosimilitud=[] #guarda la verosimilitud de las 10 caracteristicas
  verosimilitudpos=[] #guarda la verosimilitud de cada elemento del conjunto(df)
  for i in range(len(df)):#recorre todos los elementos del df
```

```
    verosimilitudpos.append(verosimilitudelemento(df.iloc[i],medida__caracteristica__pos,
      ↪ Matriz__verosimilitudes__pos))
  return verosimilitudpos

def verosimilitud__negativa(df):
  #verosimilitud=[] #guarda la verosimilitud de las 10 caracteristicas
  verosimilitudneg=[] #guarda la verosimilitud de cada elemento del conjunto(df)
  for i in range(len(df)):#recorre todos los elementos del df
    verosimilitudneg.append(verosimilitudelemento(df.iloc[i],medida__caracteristica__neg,
      ↪ Matriz__verosimilitudes__neg))
  return verosimilitudneg


a=verosimilitud__positiva(test)
b=verosimilitud__negativa(test)


### test de theta <=> P(x|hadron=1=positiva)/P(x|no hadron=0=negativa)>=theta
TP=[]
FP=[]
FN=[]
TN=[]


for theta in np.logspace(-30,10,num=150,base=10,dtype='float'):
  tp=0
  fp=0
  fn=0
  tn=0
  clase=[]
  for i in range(len(a)):
    if a[i]/b[i] >= theta:
      clase.append(1)
    else:
      clase.append(0)
    if clase[i] == 1:
      if test['class'][i]== 1:
        tp += 1
      else:
        fp += 1
    elif clase[i] == 0:
      if test['class'][i]== 0:
        tn += 1
      else:
        fn += 1
  TP.append(tp)
  FP.append(fp)
  FN.append(fn)
  TN.append(tn)

Tasa__TP=[]
```

```
Tasa_FP=[]
for i in range(len(TP)):
  Tasa_TP.append(TP[i]/(TP[i]+FN[i]))
  Tasa_FP.append(FP[i]/(FP[i]+TN[i]))
```

Como se puede apreciar debido a la diferencia de dimensiones entre la LUT que posee las verosimilitudes obtenidas por el conjunto de entrenamiento y el vector que posee la medida tope (rango) de cada bin se debe modificar el recorrido realizado, otro punto a destacar es la utilización del supuesto de Naive Bayes, que simplifica la tarea del clasificador debido a que se utiliza como verosimilitud la pitatoria de los elementos que componen al vector de verosimilitudes.

2.2.4.

Se debe variar el valor θ, clasificar el conjunto de pruebas y calcular su tasa de Verdaderos Positivos y Tasa de Falsos Positivos para luego generar la curva ROC. Lo que en parte fue ejecutado en el código anterior, para generar la curva ROC se debe implementar el siguiente código:

Código 16: Código plot curva ROC Naive Bayes

```

plt.plot(Tasa_FP, Tasa_TP, '-',color='b',label='Curva ROC')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.legend()
plt.title("Curva ROC")
plt.ylabel('Tasa de detección')
plt.xlabel('Tasa falsa alarma')
plt.show()
```

Dando como resultado el siguiente gráfico:

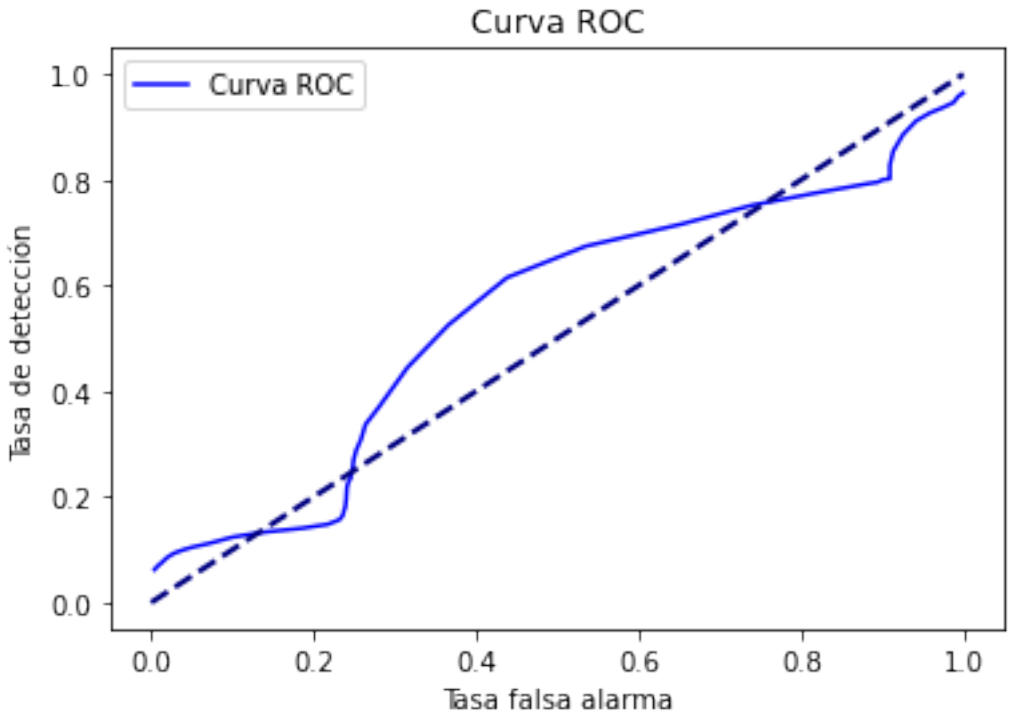


Figura 7: Curva ROC Bayes

2.2.5.

Para obtener otra evaluación de rendimiento se pide graficar la curva 'Precision-recall', la cual es implementada como sigue:

Código 17: Código plot Precision-Recall Naive Bayes

```
#Parte 2.15
Tasa_precision=[]
for i in range(len(TP)):
  Tasa_precision.append(TP[i]/(TP[i]+FP[i]))

plt.plot(Tasa_TP, Tasa_precision, '-',color='b',label='Curva precision-recall')
plt.plot([1, 0], [0, 1], color='navy', lw=2, linestyle='--')
plt.legend()
plt.title('Curva precision-recall')
plt.ylabel('Precision')
plt.xlabel('Recall')
plt.show()
```

Dando como resultado la curva:

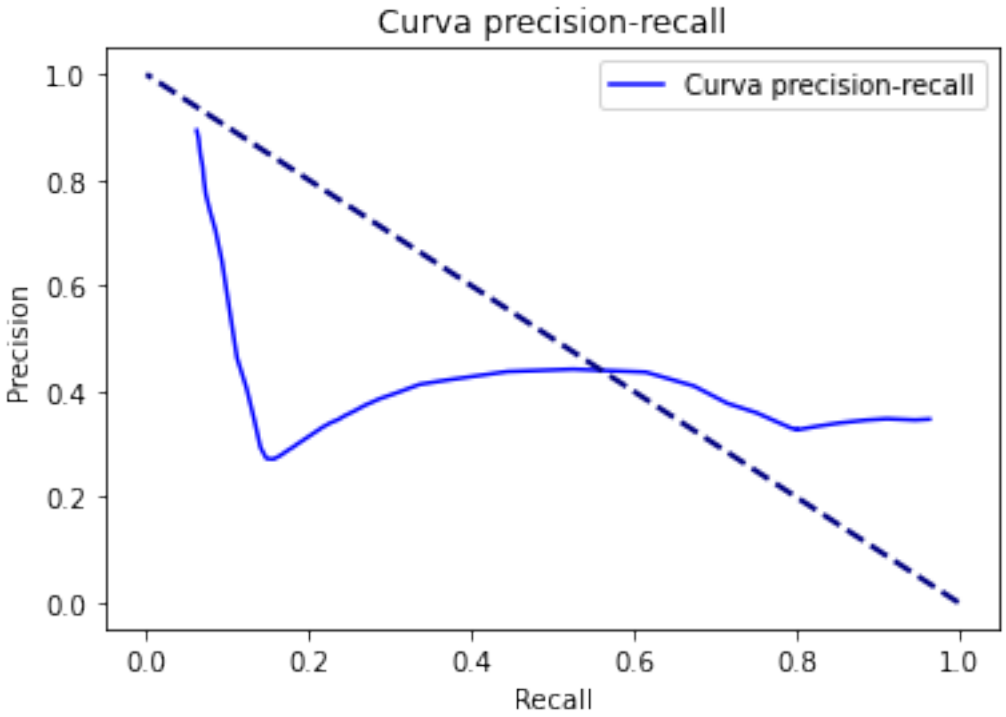


Figura 8: Curva Precision-recall

2.3. Parte 3: Clasificación usando gaussianas multivariantes

2.3.1.

La primera tarea a realizar es entrenar el modelo Gaussiano, es decir, encontrar la covarianza y esperanza de los datos de entrenamiento, siendo esta última el promedio de los datos.Para esto se utilizarán funciones de la biblioteca de numpy. El código a continuación muestra su implementación:

Código 18: Entrenamiento Gaussiana multidimensional

```
1
2  #Parte 3.16
3  clase_positiva_entrenamiento
4  clase_negativa_entrenamiento
5  tiempo_gauss_inicio = time.time()
6
7  esperanza_neg=[]
8  esperanza_pos=[]
9  for i in range(len(nombres_columnas)-1):
10   esperanza_neg.append(np.mean(clase_negativa_entrenamiento[nombres_columnas[i]]))
11   esperanza_pos.append(np.mean(clase_positiva_entrenamiento[nombres_columnas[i]]))
12
13
14 c_pos_snclass=clase_positiva_entrenamiento.drop(columns=['class'])
15 covarianza_pos = np.cov(c_pos_snclass,rowvar=False)
16
17 c_neg_snclass=clase_negativa_entrenamiento.drop(columns=['class'])
18 covarianza_neg = np.cov(c_neg_snclass,rowvar=False)
19
20 tiempo_gauss_final = time.time()
```

Las clases positiva y negativa de entrenamiento son las mismas utilizadas para el entrenamiento del clasificador de Bayes, razón por lo cual no es necesario definirlas nuevamente, el entrenamiento de este clasificador a diferencia de Bayes no requiere de una LUT, sino que solo 2 parámetros, en esta parte del código igualmente se utilizó la función **time.time()** para posteriormente comparar los tiempos de ejecución.

2.3.2.

Se deben hallar las verosimilitudes en ambas clases para cada muestra del conjunto de prueba, razón por la cual se utilizan las Gaussianas. El código a continuación presenta como se implementa el hallazgo de las verosimilitudes.

Código 19: Verosimilitudes Gaussiana conjunto de Prueba

```
1
2  #Parte 3.17
3  def verosimilitud_gaussiana_pos(df):
```

```
 ver_pos=[]
 for i in range(len(df)):
   xmenosu=[]
   for j in range(10):#recorre las 10 caracteristicas
     xmenosu.append((df.iloc[i][j])-esperanza_pos[j])
   xmenosutr= np.transpose(xmenosu)
   inversa = np.linalg.inv(covarianza_pos)
   prod_matr= np.dot(np.dot(xmenosutr,inversa),xmenosu)
   dividendo= np.exp((-1/2)*prod_matr)
   divisor= np.sqrt((2*np.pi)**10*np.linalg.det(covarianza_pos))
   valor = dividendo/divisor
   ver_pos.append(valor)
 return ver_pos

ver_ga_pos=verosimilitud_gaussiana_pos(test)

def verosimilitud_gaussiana_neg(df):
 ver_neg=[]
 for i in range(len(df)):
   xmenosu=[]
   for j in range(10):#recorre las 10 caracteristicas
     xmenosu.append((df.iloc[i][j])-esperanza_neg[j])
   xmenosutr= np.transpose(xmenosu)
   inversa = np.linalg.inv(covarianza_neg)
   prod_matr= np.dot(np.dot(xmenosutr,inversa),xmenosu)
   dividendo= np.exp((-1/2)*prod_matr)
   divisor= np.sqrt((2*np.pi)**10*np.linalg.det(covarianza_neg))
   valor = dividendo/divisor
   ver_neg.append(valor)
 return ver_neg

ver_ga_neg=verosimilitud_gaussiana_neg(test)
```

En la implementación de este código se implementa la función de distribución de una Gaussiana multidimensional, la cual tiene la forma:

$$f_{\mathbf{X}}(x_1, \ldots, x_k) = \frac{\exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right)}{\sqrt{(2\pi)^k|\boldsymbol{\Sigma}|}}$$

Figura 9: Función de distribución

De igual forma se utilizó la biblioteca de numpy, la cual posee funciones para el producto matricial y cálculo de determinantes.

2.4.

Se debe variar el valor θ, clasificar el conjunto de pruebas y calcular su tasa de Verdaderos Positivos y Tasa de Falsos Positivos para luego generar la curva ROC. Lo cual es implementado como sigue:

Código 20: Código Tasa TP y FP Gaussiana

```
1
2  #Parte 3.18
3  ### test de theta <=> P(x|hadron=1=positiva)/P(x|no hadron=0=negativa)>=theta
4  TPG=[]
5  FPG=[]
6  FNG=[]
7  TNG=[]
8
9
10 for theta in np.logspace(-30,10,num=150,base=10,dtype='float'):
11   tp=0
12   fp=0
13   fn=0
14   tn=0
15   clase=[]
16   for i in range(len(a)):
17     if ver_ga_pos[i]/ver_ga_neg[i] >= theta:
18       clase.append(1)
19     else:
20       clase.append(0)
21     if clase[i] == 1:
22       if test['class'][i]== 1:
23         tp += 1
24       else:
25         fp += 1
26     elif clase[i] == 0:
27       if test['class'][i]== 0:
28         tn += 1
29       else:
30         fn += 1
31   TPG.append(tp)
32   FPG.append(fp)
33   FNG.append(fn)
34   TNG.append(tn)
35
36 Tasa_TPG=[]
37 Tasa_FPG=[]
38 for i in range(len(TPG)):
39   Tasa_TPG.append(TPG[i]/(TPG[i]+FNG[i]))
40   Tasa_FPG.append(FPG[i]/(FPG[i]+TNG[i]))
41
42
43 plt.plot(Tasa_FPG, Tasa_TPG, '-',color='b',label='Curva ROC')
```

```
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.legend()
plt.title("Curva ROC")
plt.ylabel('Tasa de detección')
plt.xlabel('Tasa falsa alarma')
plt.show()
```

Este código da como resultado el siguiente gráfico:

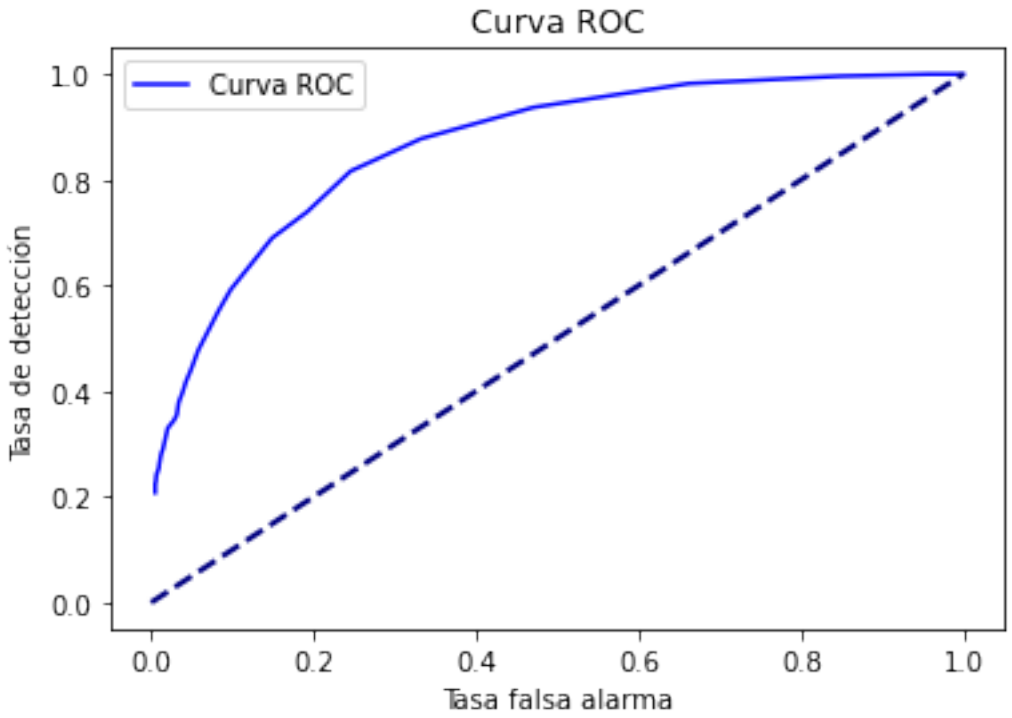


Figura 10: Curva ROC Gaussiana multidimensional

2.4.1.

Igualmente se debe generar la curva Precision-Recall, la cual es implementada mediante el código:

Código 21: Código curva Precision-Recall Gaussaina

```

#Parte 3.19
Tasa_precisionG=[]
for i in range(len(TPG)):
  Tasa_precisionG.append(TPG[i]/(TPG[i]+FPG[i]))

plt.plot(Tasa_TPG, Tasa_precisionG, '-',color='b',label='Curva precision-recall')
plt.plot([1, 0], [0, 1], color='navy', lw=2, linestyle='--')
plt.legend()
```

```
plt.title('Curva precision-recall')
plt.ylabel('Precision')
plt.xlabel('Recall')
plt.show()
```

El que plotea como resultado:

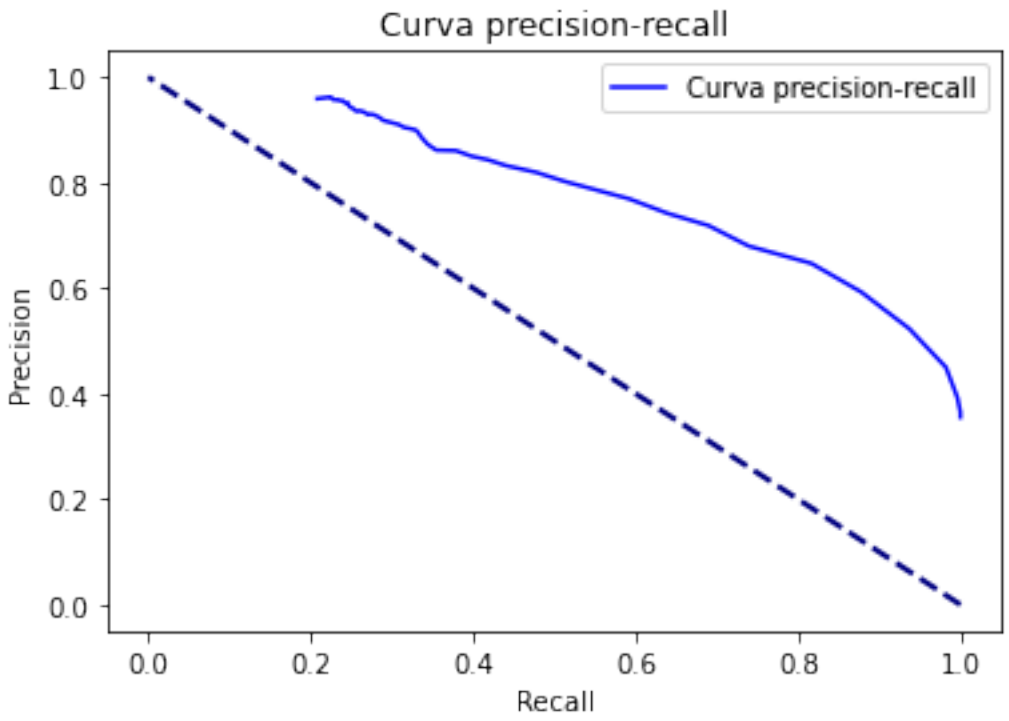


Figura 11: Caption

2.5. Parte 4: Comparación de resultados

Para comparar los resultados se utilizan gráficos de evaluación de rendimiento como son el caso de las curvas ROC y Precision-Recall.

2.5.1.

Gráfico de ambas curvas ROC:

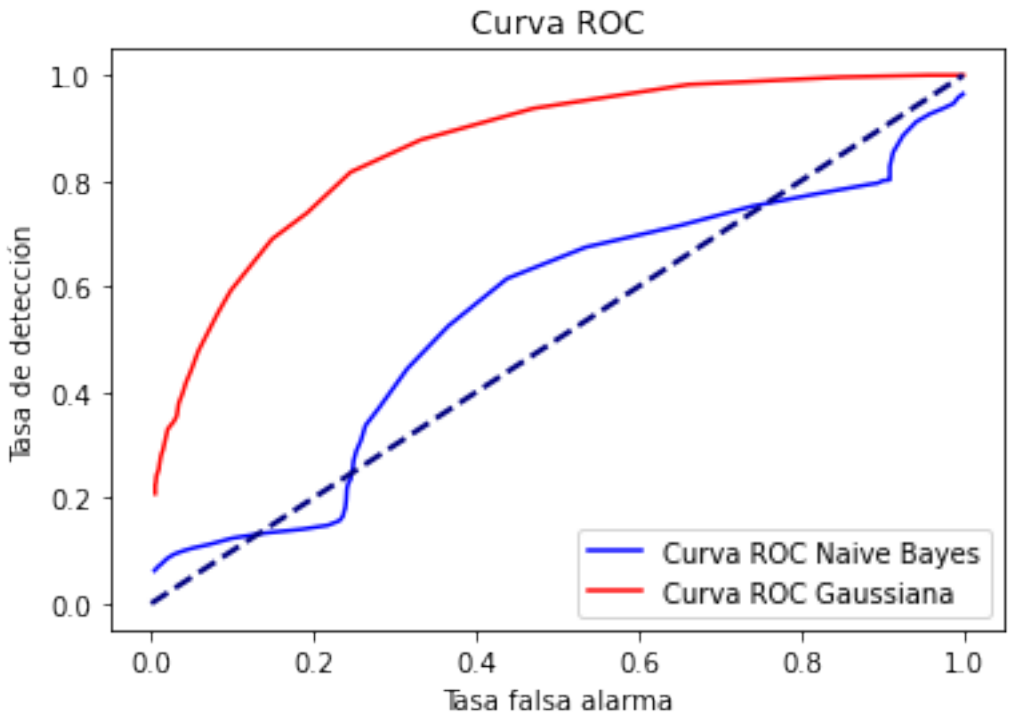


Figura 12: Comparación de curvas ROC

Para implementar ese código solo se deben unir los códigos de ambos gráficos, indicando a cuál corresponde cada uno mediante los labels.

2.5.2.

Gráfico de ambas curvas Precision-Recall:

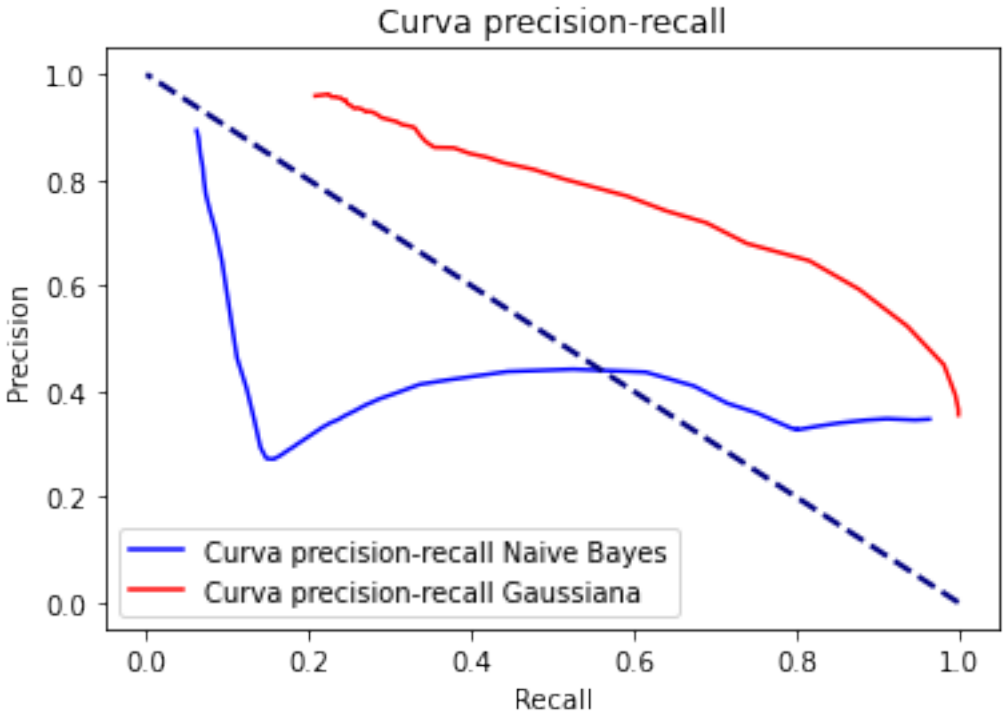


Figura 13: Comparación de curvas Precision-Recall

2.5.3.

El último punto de comparación son los tiempos de respuesta de cada entrenamiento, lo cual es visualizado con la funcion **time.time()**, la cual fue utilizada para medir los tiempos que requieren cada clasificador para ser entrenados. Los tiempos de ejecución son presentados a través del código:

Código 22: Comparación tiempos de ejecución

```
1
2 #Parte 4.22
3 print('Tiempo entrenamiento Naive Bayes = ',tiempo_bayes_final-tiempo_bayes_inicio,'[s]')
4 print('Tiempo entrenamiento Gaussiana = ',tiempo_gauss_final-tiempo_gauss_inicio,'[s]')
```

Dando como respuesta 0.012310266494750977 [s] para el entrenamiento del clasificador Naive Bayes y 0.0073299407958984375 [s] para la aproximación por Gaussiana multivariable.

3. Conclusiones

Los resultados obtenidos por el clasificador de Bayes fueron inesperados, ya que existen zonas de operación donde el umbral de decisión hace que este opere peor que una clasificación azarosa, lo cual puede deberse al bajo número de bines utilizados para las verosimilitudes de la LUT, o el rango de operaciones utilizado para Θ.
La mejor solución tanto por tiempo de ejecución como resultados correctos es la aproximación por Gaussiana multivariable.
Para mejorar el desempeño de los algoritmos puede ser viable analizar las correlaciones de las características, de este modo es posible eliminar características que no aportan de forma significativa en la clasificación final, de esta forma se requiere menor coste computacional haciendo más efectivo del punto de vista temporal los algoritmos.
Como aprendizaje al llevar a cabo la experiencia se logró un acercamiento a las bibliotecas de Pandas, Numpy, Seaborn y Matplotlib, siendo esta la mayor dificultad, el escaso conocimiento de estas.