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

Informe tarea 1

Clasificador de Bayes

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

Fecha de entrega: Domingo 11 de Abril de 2021
Santiago, Chile

Índice de Contenidos

1. Introducción **1**

2. Desarrollo **3**
2.1. Parte 1: Visualización y análisis de datos . . . 3
2.1.1. Actividad 1 . . . 3
2.1.2. Actividad 2 . . . 3
2.1.3. Actividad 3 . . . 3
2.1.4. Actividad 4 . . . 9
2.1.5. Actividad 5 . . . 9
2.1.6. Actividad 6 . . . 10
2.1.7. Actividad 7 . . . 11
2.1.8. Actividad 8 . . . 11
2.1.9. Actividad 9 . . . 11
2.1.10. Actividad 10 . . . 12
2.2. Parte 2: Clasificación usando Naive Bayes e histogramas . . . 12
2.2.1. Actividad 11 . . . 13
2.2.2. Actividad 12 . . . 14
2.2.3. Actividad 13 . . . 16
2.2.4. Actividad 14 . . . 17
2.2.5. Actividad 15 . . . 20
2.3. Parte 3: Clasificación usando gaussianas multivariantes . . . 21
2.3.1. Actividad 16 . . . 21
2.3.2. Actividad 17 . . . 22
2.3.3. Actividad 18 . . . 23
2.3.4. Actividad 19 . . . 23
2.4. Parte 4: Comparación de resultados . . . 24
2.4.1. Actividad 20 . . . 24
2.4.2. Actividad 21 . . . 24
2.4.3. Actividad 22 . . . 25
2.5. Resultados . . . 26

3. Conclusión **28**

Índice de Figuras

1. Código utilizado para subir los archivos. . . . 3
2. Código utilizado para obtener los subconjuntos. . . . 3
3. Código utilizado para el histograma de la característica 'fLength'. . . . 4
4. Histograma de la característica 'fLength'. . . . 4
5. Histograma de la característica 'fWidth'. . . . 5
6. Histograma de la característica 'fSize'. . . . 5
7. Histograma de la característica 'fConc'. . . . 6
8. Histograma de la característica 'fConc1'. . . . 6

9. Histograma de la característica 'fAsym'. 7
10. Histograma de la característica 'fM3Long'. 7
11. Histograma de la característica 'fM3Trans'. 8
12. Histograma de la característica 'fAlpha'. 8
13. Histograma de la característica 'fDist'. 9
14. Código para obtener la matriz de correlación. 9
15. Características más correlacionadas con la clase en orden descendiente. 10
16. Código utilizado para ordenar los 5 pares de características más correlacionados. 10
17. Pares de características más correlacionados entre sí. 10
18. Gráfico de dispersión de las dos características más correlacionadas. 11
19. Pares de características menos correlacionadas. 11
20. Gráfico de dispersión de las dos características menos correlacionadas. 12
21. Matriz de correlación. 12
22. Código previo a la obtención del conjunto de entrenamiento y prueba. 13
23. Obtención de conjuntos y verificación de representatividad. 14
24. Subconjuntos del conjunto de entrenamiento según su clase. 14
25. Obtención de histogramas. 15
26. Histograma para caracteristica 'fLength' del conjunto de Hadrones. 15
27. Configuraciones finales. 16
28. Cálculo de verosimilitudes para el conjunto de hadrones. 17
29. Clasificador de Bayes. 18
30. Cálculo de verdaderos y falsos para cada elemento. 19
31. Cálculo de tasas. 20
32. Curva ROC obtenida mediante los histogramas. 20
33. Curva Precisión-Recall obtenida mediante los histogramas. 21
34. Entrenamiento del clasificador mediante gaussianas multivariantes. 22
35. Cálculo de verosimilitudes mediante gaussianas multivariantes. 23
36. Curva ROC obtenida mediante gaussianas multivariantes. 23
37. Curva de Precisión-Recall obtenida mediante gaussianas multivariantes. 24
38. Curvas ROC. 24
39. Curvas de Precisión-Recall. 25
40. Tiempos de entrenamiento. 25
41. Curvas ROC de ambos clasificadores al considerar 50 bins. 26
42. Curvas Precisión-Recall de ambos clasificadores al considerar 50 bins. 26
43. Tiempos de ejecución al considerar 50 bins. 26
44. Curvas ROC al tener más valores de θ. 27
45. Curvas de Precisión-Recall al considerar más valores de Θ. 27
46. Tiempos de ejecución al considerar más valores de Θ. 27

1. Introducción

En el presente informe se detallará la tarea número 1 del curso de EL4106 Inteligencia Computacional, tarea que consistía en la implementación de un clasificador de bayes, clasificador que se encargará de discernir si el dato que se le entrega corresponde a un Hadrón o a un Rayo Gamma (No-Hadrón). Con la finalidad de realizar el objetivo previamente mencionado se utilizará un conjunto de datos MAGIC Gamma Telescope Data Set, conjunto que corresponde a un conjunto de simulaciones air showers que son generadas en base a los elementos anteriormente mencionado. La herramienta a utilizar será la plataforma de *Google Colaboratory*, herramienta que es capaz de ejecutar notebooks de *Python* de manera online, además se utilizaran las librerias de *Pandas*, *Matplotlib*, *Numpy* y *Seaborn*, librerías que están disponibles en el lenguaje de programación *Python*.

Las secciones que contendrá el informe así como también las subsecciones de estas se detallan a continuación:

- **Parte 1: Visualización y análisis de datos**:
  - **Actividad 1:** La primera actividad de la tarea consiste en darle lectura al conjunto de datos *'magic04.data'*.
  - **Actividad 2:** En la segunda actividad se tendrá que separar los datos en dos subconjuntos, uno de clase positiva (Hadrón) y otro de clase negativa (No-Hadrón).
  - **Actividad 3:** La siguiente actividad consiste en obtener los histogramas de los subconjuntos previamente mencionados por características y dibujarlos superpuestos.
  - **Actividad 4:** En la cuarta actividad se debe calcular la matriz de correlación de los datos, considerando como 0 a la clase de los elementos que son No-Hadrones y como 1 a los que son Hadrones, una vez calculada la matriz de correlación se deberá aplicar valor absoluto a las componentes de la matriz obtenida.
  - **Actividad 5:** En esta actividad se debe indicar por orden cuáles son las características que están mas correlacionadas con la clase.
  - **Actividad 6:** En la siguiente actividad se debe indicar por orden los 5 pares de características que están más correlacionadas entre sí.
  - **Actividad 7:** En la séptima actividad se deben graficar las dos características distintas que están más correlacionadas entre sí. Esto debe realizarse con un *Scatter Plot*.
  - **Actividad 8:** Luego se deben indicar por orden los 5 pares de características que estén menos correlacionadas entre sí.
  - **Actividad 9:** A continuación se tienen que graficar las dos características que estaban menos correlacionadas. Al igual que en la actividad 7 la gráfica debía obtenerse mediante un *Scatter Plot*.
  - **Actividad 10:** Finalmente se debe graficar la matriz de correlación de datos.
- **Parte 2: Clasificación usando Naive Bayes e histogramas**
  - **Actividad 11:** La primera actividad de la segunda parte de la tarea consiste en dividir la base de datos en dos conjuntos representativos, el 80 % de los datos corresponderían al conjunto de entrenamiento y el 20 % restante correspondería al conjunto de prueba, además

se debía comprobar la representatividad de los conjuntos, verificando si la proporción de ambas clases se mantiene cercana a la proporción del conjunto completo.

- **Actividad 12:** En esta actividad se pide encontrar los histogramas normalizados de cada clase a partir de las muestras del conjunto de entrenamiento.
- **Actividad 13:** La siguiente actividad corresponde a encontrar las verosimilitudes en ambas clases para cada elemento del conjunto de prueba, para lograr esto se deberá hacer uso de los histogramas.
- **Actividad 14:** Luego se debe mover el valor de Θ y clasificar cada elemento del conjunto de prueba, luego se tendrá que calcular la Tasa de verdaderos positivos y la tasa de falsos positivos, esto con la finalidad de generar la curva ROC del conjunto de prueba.
- **Actividad 15:** Finalmente se pide generar la curva de precisión-recall.

- **Parte 3: Clasificación usando gaussianas multivariantes**

  - **Actividad 16:** La primera actividad de la tercera parte de la tarea es similar a la actividad número 12 de la parte 2, pero en vez de entrenar al clasificador mediante histogramas se debe entrenar mediante un modelo de gaussiana multivariable, para esto se debe encontrar la media y la covarianza del conjunto de entrenamiento para cada clase. Para lograr lo previamente mencionado se hará uso de las funciones *mean()* y *cov()* de la librería *Numpy*.
  - **Actividad 17:** La siguiente actividad corresponde a encontrar las verosimilitudes de ambas clases para cada muestra del conjunto de prueba, para hacer esto se hará uso del vector de medias y la matriz de covarianza encontrados con anterioridad.
  - **Actividad 18:** Posteriormente se deberá mover el valor de Θ y calcular la tasa de verdaderos y falsos positivos.
  - **Actividad 19:** Finalmente, y al igual que en la actividad número 15, se deberá generar la curva de precisión-recall.

- **Parte 4: Comparación de resultados**

  - **Actividad 20:** En la actividad número 20 se deberán gráficar, ambas curvas ROC obtenidas anteriormente en un mismo gráfico.
  - **Actividad 21:** Para el caso de la actividad 21 se deberán graficar ambas curvas precisión-recall obtenidas previamente.
  - **Actividad 22:** Y finalmente se deberán comparar los tiempos requeridos para entrenar ambos clasificadores.

2. Desarrollo

Para la realización de la tarea se utilizó un archivo *.data* entregado por el cuerpo docente. En las siguientes subsecciones se detallarán las diferentes partes de la tarea, así como también se expondrán porciones del código implementado, esto se hará con la finalidad de clarificar el procedimiento realizado. Cabe destacar que durante esta sección solo se explicará el procedimiento con el que se resolvieron las diferentes actividades, no se explicará lo que se debe hacer, esto ya se realizó durante la introducción de este informe.

2.1. Parte 1: Visualización y análisis de datos

2.1.1. Actividad 1

En la primera parte de la tarea se debía dar lectura al archivo *.data* entregado por el cuerpo docente, en primer lugar se tuvo que subir el archivo previamente mencionado a la plataforma *Google Colaboratory*, para realizar esto se utilizó la porción de código que se puede apreciar en la figura 1.

```
from google.colab import files
uploaded = files.upload()
```

Figura 1: Código utilizado para subir los archivos.

Una vez subido el archivo se procedio a darle lectura, para realizar esto se utilizó la función *pd.read_csv()*, donde el primer parámetro de esta función corresponde al archivo que se quiere leer, en este caso el archivo correspondía al llamado *'magic04.data'* y como segundo parámetro se le entregó un vector que contiene los nombres de las columnas del archivo previamente mencionado, estos nombres son los siguientes: *'fLength', 'fWidth', 'fSize', 'fConc', 'fConc1','fAsym', 'fM3Long', 'fM3Trans', 'fAlpha', 'fDist', 'class'*, el archivo leído se nombró como *df*.

2.1.2. Actividad 2

Para separar los datos del archivo previamente subido en un conjunto que contenga a los datos de clase positiva (Hadrón) y otro que contenga a los de clase negativa (No-Hadrón) se utilizó la función *loc* de la librería *Pandas*. La implementación de esto se puede observar en la figura 2.

```
H = df.loc[df['class'] == 'h'] #Arreglo donde se guardaran los hadrones
NH = df.loc[df['class'] == 'g'] #Arreglo donde se guardaran los no-hadrones
```

Figura 2: Código utilizado para obtener los subconjuntos.

2.1.3. Actividad 3

Para generar los histogramas de cada característica, tanto para la clase positiva como para la negativa, se utilizó la función *hist* de la librería *Matplotlib*, función a la que se le entregaron como

parámetros: la columna de la característica que se buscaba dibujar, un vector que poseía los bordes de los bins que se estaban utilizando, un número que indicaba la transparencia de los histogramas y finalmente el nombre de la gráfica. El código implementado para obtener el histograma de la característica *'fLength'* se puede observar en la figura 3 y el histograma obtenido corresponde al de la figura 4, cabe destacar que el código utilizado para el resto de características es igual al mostrado en la figura 3 pero reemplazando el nombre de las características y dándole el vector de bins adecuado. El resto de histogramas obtenidos durante la actividad se pueden apreciar en las figuras 5, 6, 7, 8, 9, 10, 11, 12 y 13.

```
bins1 = np.linspace(min(H['fLength'].min(),NH['fLength'].min()),max(H['fLength'].max(),NH['fLength'].max()),100)
plt.hist(H['fLength'],bins1,alpha=0.8,label='Hadron')
plt.hist(NH['fLength'],bins1,alpha=0.8,label='No-Hadron')
plt.legend(loc='upper right')
plt.title('flength')
plt.show()
```

Figura 3: Código utilizado para el histograma de la característica 'fLength'.

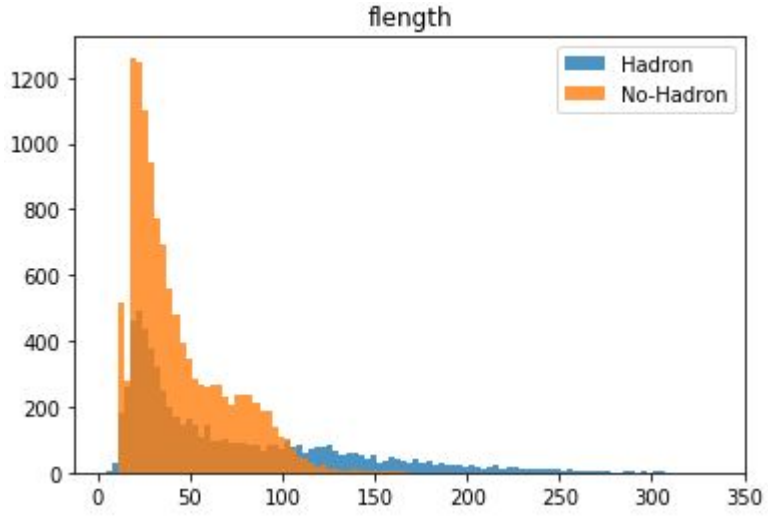

Figura 4: Histograma de la característica 'fLength'.

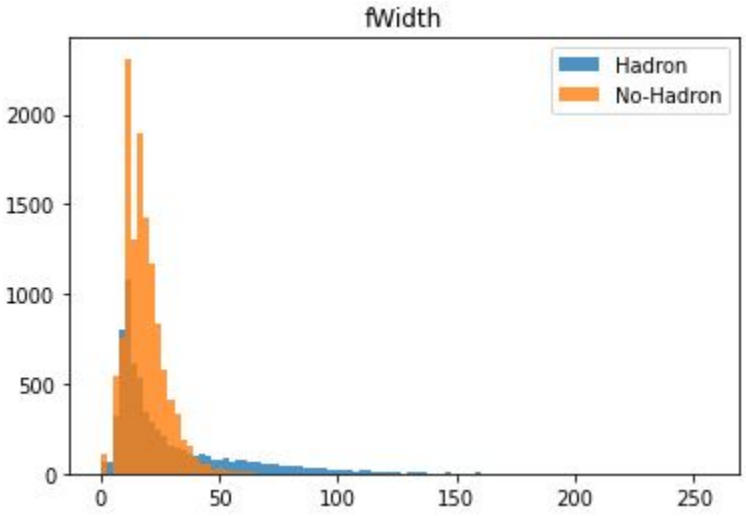

Figura 5: Histograma de la característica 'fWidth'.

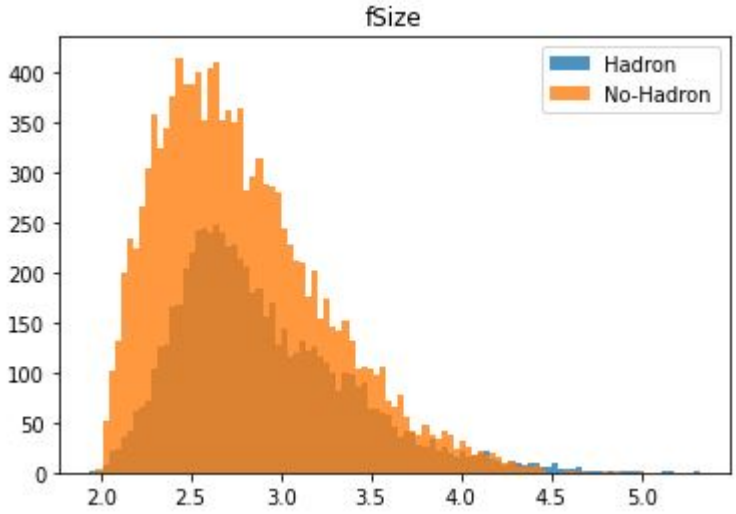

Figura 6: Histograma de la característica 'fSize'.

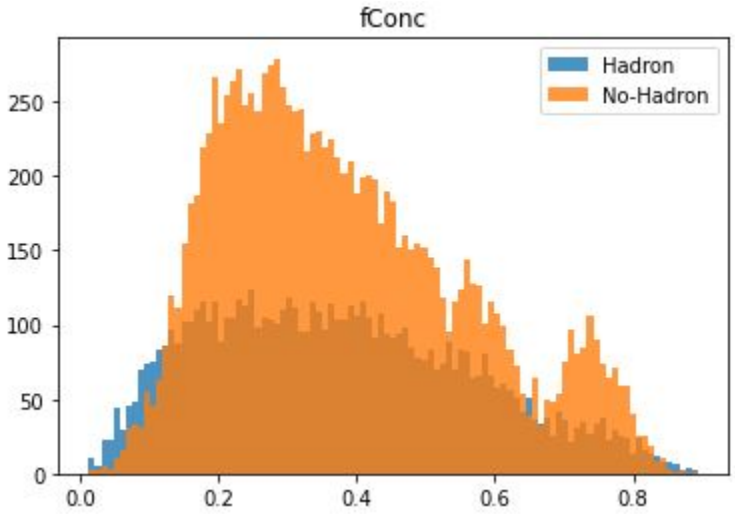

Figura 7: Histograma de la característica 'fConc'.

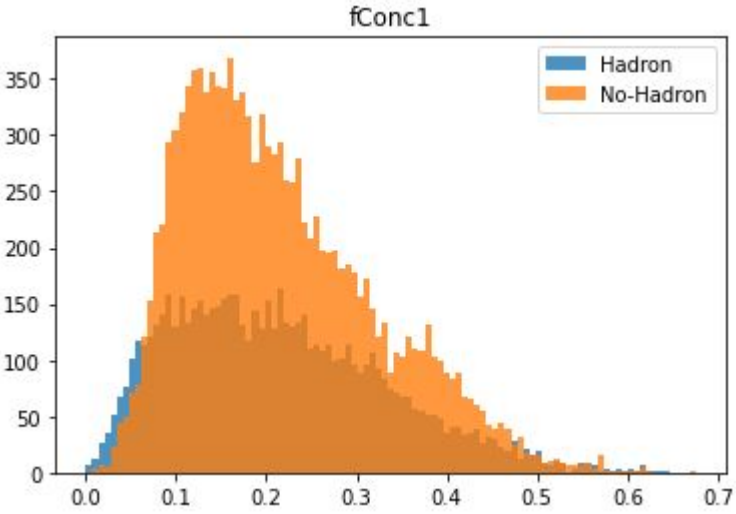

Figura 8: Histograma de la característica 'fConc1'.

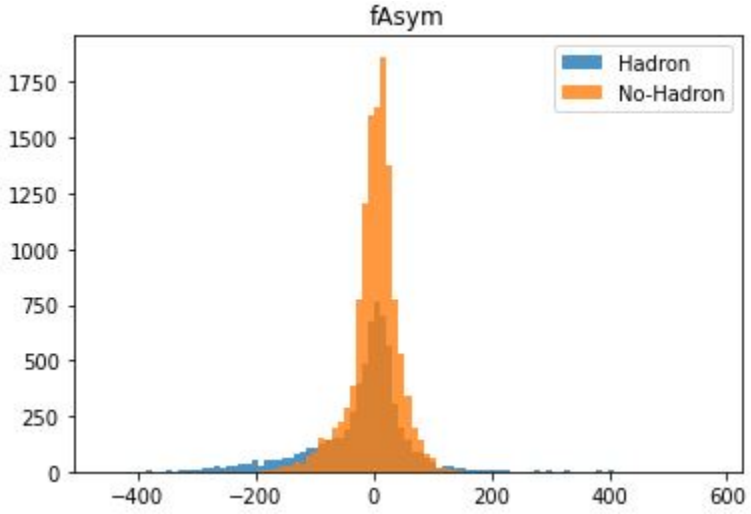

Figura 9: Histograma de la característica 'fAsym'.

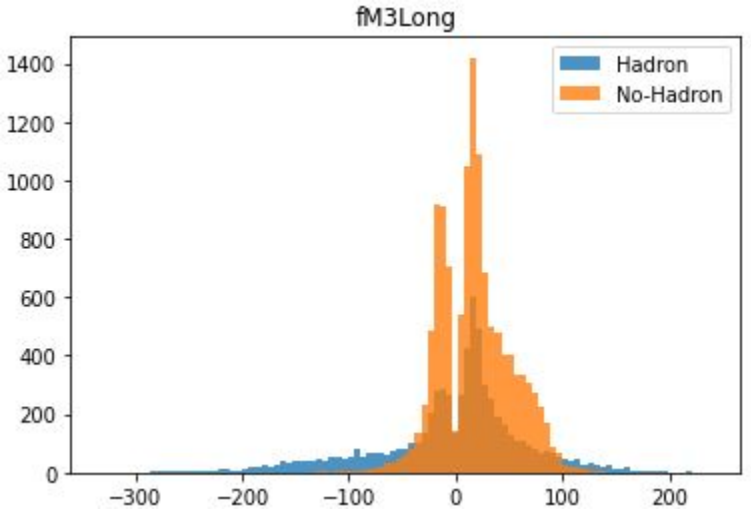

Figura 10: Histograma de la característica 'fM3Long'.

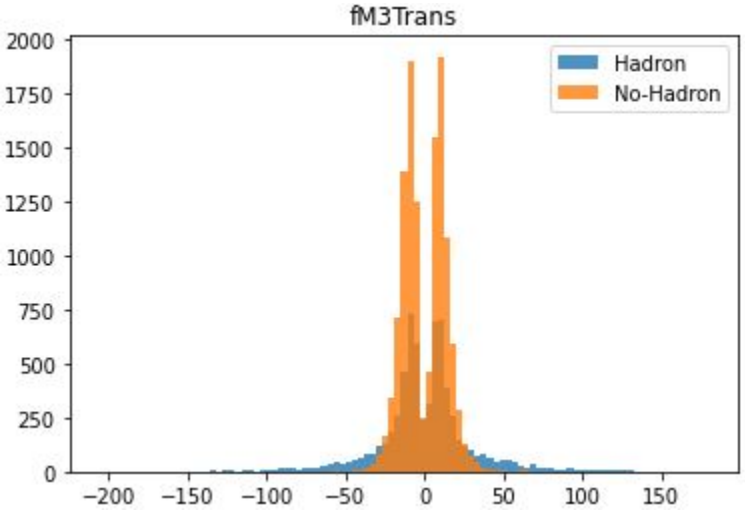

Figura 11: Histograma de la característica 'fM3Trans'.

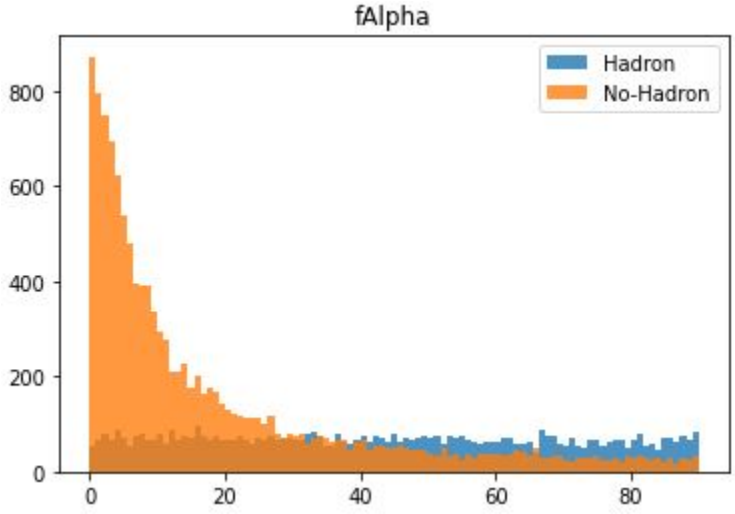

Figura 12: Histograma de la característica 'fAlpha'.

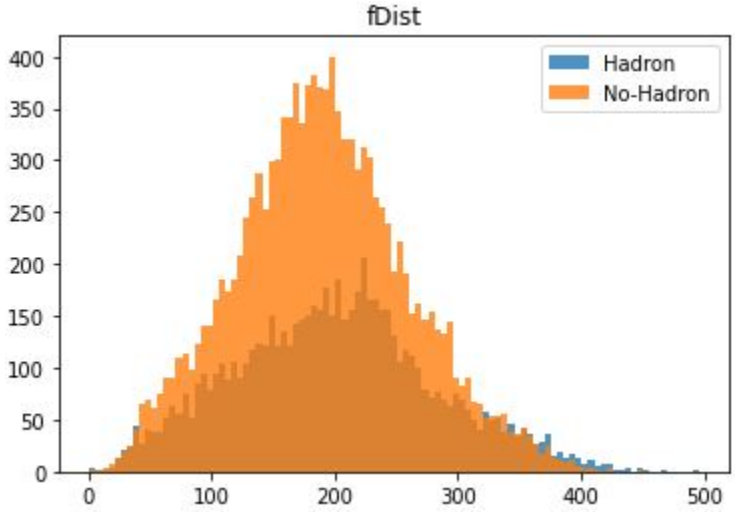

Figura 13: Histograma de la característica 'fDist'.

2.1.4. Actividad 4

Para lograr obtener la matriz de correlación de los datos primero se hizo una copía del dataframe original llamado *df*, copia que fue llamada *df_1*, para realizar esto se utilizó la función *copy()* de la librería *Pandas*, luego se utilizó a función *replace* de la misma librería para lograr que los datos que eran No-Hadrones fuesen identificados con un 0 y los Hadrones con un 1, este reemplazo se hizo en sus respectivas columnas *'class'*. Una vez realizado lo anterior se utilizó la función *corr()* de *Pandas* para obtener la matriz de correlación y finalmente se usó la función *abs()* para obtener el valor absoluto de las componentes de dicha matriz. Lo previamente mencionado se logra apreciar en la figura 14.

```
df_1 = df.copy() #Hacemos una copia del dataframe original

#Aca cambiamos la columna class como: 0 si es no-hadron y 1 si es hadron
df_1['class'] = df_1['class'].replace({'g' : 0, 'h' : 1})

#Obtenemos la matriz de correlacion
Correla = df_1.corr()

#Aplicamos valor absoluto a las celdas de la matriz de correlacion
Correla_abs = abs(Correla)
```

Figura 14: Código para obtener la matriz de correlación.

2.1.5. Actividad 5

Para obtener el orden de las caracteristicas más correlacionadas con la clase se obtuvo la columna de las matriz de correlación que corresponde a la clase, para luego ordenar de mayor a menor los valores de esta utilizando la función *sort_values(ascending=False)*. El resultado obtenido es el correspondiente a la figura 15.

```
Las caracteristicas mas correlacionadas con la clase
ordenadas de mayor a menor son:
class       1.000000
fAlpha      0.460979
fLength     0.307572
fWidth      0.265596
fM3Long     0.193409
fAsym       0.173587
fSize       0.117795
fDist       0.065203
fConc       0.024615
fConc1      0.004797
fM3Trans    0.003837
```

Figura 15: Características más correlacionadas con la clase en orden descendiente.

2.1.6. Actividad 6

Para lograr obtener los 5 pares de características más correlacionados entre sí se utilizó la función *unstack()*, luego se utilizó la función *sort_values(ascending=False)* para ordenar los pares de caracteristicas de mayor a menor, posteriormente se omitieron aquellos pares de características con correlación 1 (que corresponden a los casos donde se observó la correlación de una característica consigo misma) y finalmente se omitieron casos donde se tienen dos pares de características idénticos pero en distinto orden, es decir, casos como por ejemplo: 'fLength'-'fSize' y 'fSize'-'fLength'. Lo antes mencionado se puede observar en la figura 16 y el orden solicitado corresponde al de la figura 17.

```
Correla_abs_lista = Correla_abs.unstack()

#Ordenamos lo anterior de mayor a menor
Correla_orden = Correla_abs_lista.sort_values(ascending = False)
#Dado que las caracteristicas tienen correlacion igual a 1 consigo mismo elimino
#Esos casos
Correla_orden2 = Correla_orden[Correla_orden<1]

#Para mostrar los 5 pares de caracteristicas que mas estan correlacionadas
#se eliminaran el resto de pares
elim = list(range(0,10,2))+list(range(10,Correla_orden2.size))
mayores = Correla_orden2.drop(Correla_orden2.index[elim])
print('Los 5 pares de caracteristicas mas correlacionadas son:')
print(mayores)
```

Figura 16: Código utilizado para ordenar los 5 pares de características más correlacionados.

```
Los 5 pares de caracteristicas mas correlacionadas son:
fConc   fConc1     0.976412
fSize   fConc      0.850850
fConc1  fSize      0.808835
fWidth  fLength    0.770512
        fSize      0.717517
```

Figura 17: Pares de características más correlacionados entre sí.

2.1.7. Actividad 7

Con la finalidad de obtener las dos características distintas más correlacionadas se accedió a los primeros elementos de la lista que contenía los pares de características más correlacionadas y se utilizó la función *scatter()* de la librería *Matplotlib*. Una vez realizado esto se obtuvo el gráfico de la figura 18.

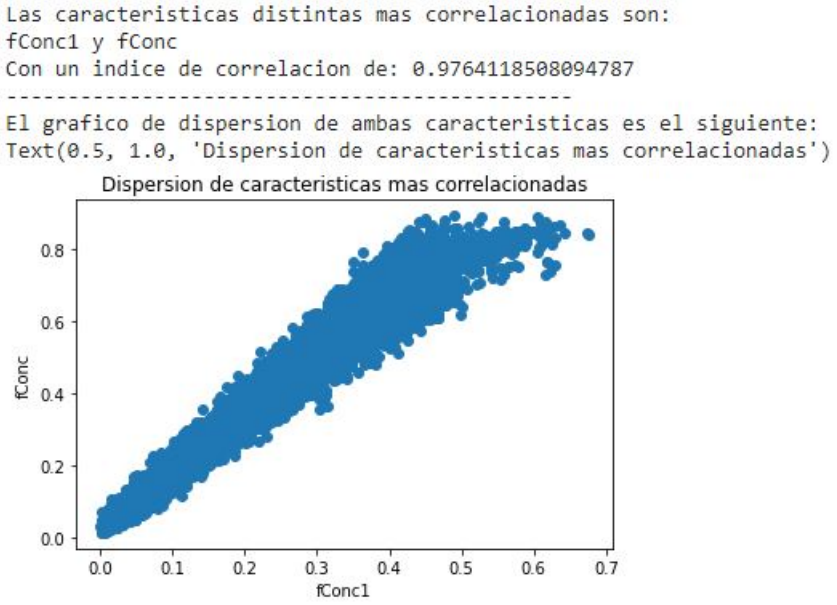

Figura 18: Gráfico de dispersión de las dos características más correlacionadas.

2.1.8. Actividad 8

La actividad 8 es prácticamente idéntica a la actividad número 6, solo que en esta se necesitaba encontrar los 5 pares de características menos correlacionados, debido a esto es que el procedimiento utilizado es análogo al anterior pero en vez de utilizar la función *sort_values(ascending=False)* se utilizó *sort_values(ascending=True)*. Al realizar esto se logró obtener que las características menos correlacionadas corresponden a las exhibidas en la figura 19.

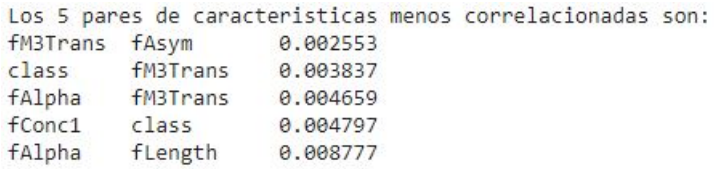

Figura 19: Pares de características menos correlacionadas.

2.1.9. Actividad 9

Al igual que en la actividad anterior, la actividad 9 es análoga a la actividad numero 7. La gráfica de dispersión obtenida para el par de características menos correlacionadas es el que corresponde a la figura 20.

```
Las caracteristicas distintas menos correlacionadas son:
fAsym y fM3Trans
Con un indice de correlacion de: 0.0025528373198593904
------------------------------------------------
El grafico de dispersion de ambas caracteristicas es el siguiente:
Text(0.5, 1.0, 'Dispersion de caracteristicas menos correlacionadas')
```

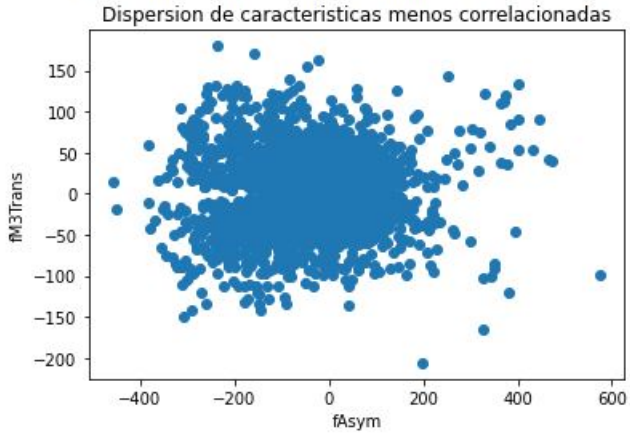

Figura 20: Gráfico de dispersión de las dos características menos correlacionadas.

2.1.10. Actividad 10

Para obtener la matriz de correlación se utilizó la función *heatmap* de la librería *Seaborn*, dicha matriz puede observarse en la figura 21.

```
La matriz de correlacion es:
<matplotlib.axes._subplots.AxesSubplot at 0x7fea2ddeffd0>
```

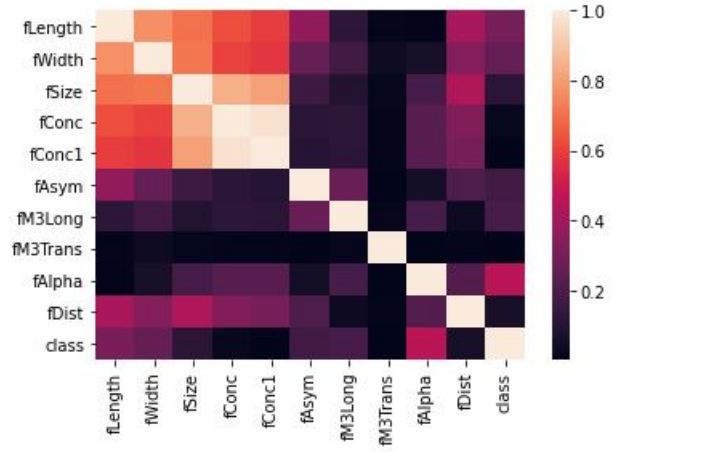

Figura 21: Matriz de correlación.

2.2. Parte 2: Clasificación usando Naive Bayes e histogramas

2.2.1. Actividad 11

Con la finalidad de obtener los conjuntos de entrenamiento y de prueba primero se hizo una copia del dataframe original, para esto se utilizó la función *copy()* de *Pandas*. Una vez realizado esto se obtuvo el punto de corte, con esto se hace alusión al punto en donde se dividirá el dataframe obtenido de utilizar la función *copy()* para obtener los dos conjuntos buscados, una vez realizado lo anterior se definió un margen de tolerancia, este margen es el que le da "flexibilidad" a la proporcionalidad de clases en ambos conjuntos. Lo recién comentado se puede observar en la figura 22, donde se puede observar que se definió una tolerancia de 5, cabe destacar que esta tolerancia se puede variar.

```
df_2 = df.copy() #Copia del dataframe
corte = int(len(df_2.index)*0.8) #aca se debe hacer la separacion de datos
tole = 5 #Aca definiremos la tolerancia para considerar representativo
         #o no a los conjuntos, no puede ser 0
```

Figura 22: Código previo a la obtención del conjunto de entrenamiento y prueba.

Con lo anterior previamente configurado se procedió a obtener los dos conjuntos buscados, para ello se hizo, en primera instancia, un desordenamiento de las filas del dataframe, para esto se utilizó la función *sample()* de *Pandas* y se le añadió la función *reset_index(drop=True)*, esta última función buscaba de que al momento de que se realizara el desordenamiento de las filas estas no quedasen con los índices que tenían previamente. Una vez realizado el proceso de desordenamiento se obtuvieron los conjuntos de entrenamiento y prueba, donde del 100 % de datos que se tenían el primer 80 % pertenecería al conjunto de entrenamiento y el 20 % restante pertenecería al conjunto de prueba, para ello se utilizó la función *iloc()* de *Pandas* así como también la función *reset_index(drop=True)*. Con lo anterior terminado se dio paso a la verificación de la representatividad de los conjuntos, para ello se utilizó un bucle sobre el conjunto de entrenamiento, donde se contaron todos los datos de clase 'g', es decir, de clase No-Hadrón, con este conteo realizado se verificó si el 80 % del total de elementos No-Hadrón del dataframe original (que poseía 12332 ejemplares del tipo No-Hadrón) menos el conteo realizado anteriormente era menor o igual a la tolerancia, con esto se daba un margen para que se definiesen los conjuntos, es decir, si se tenía una tolerancia de 5 y se tenía que el conjunto de prueba tenía $12338 * 0.8 \pm 5$ elementos tipo No-Hadrón los conjuntos obtenidos eran representativos.

Todo lo mencionado en el párrafo anterior estaba introducido en un *while* con condición de iteración $0 == 0$, esta iteración se repetiría infinitamente hasta que se logre la representatividad. El código encargado de realizar lo anterior se puede observar en la figura 23.

```
while 0==0:
  #Para obtener los conjuntos desordenamos el dataframe df_2
  df_2_des = df_2.sample(frac=1).reset_index(drop=True)

  #El conjunto de entrenamiento tiene que tener el 80% de los datos iniciales
  #Y el conjunto de prueba debe tener el 20% restante
  Entrena = df_2_des.iloc[:corte,:].reset_index(drop=True)
  Prueba = df_2_des.iloc[corte:,:].reset_index(drop=True)

  #Tengo 12332 'g' y 6688 'h', tengo que ver que en proporcion
  #el conjunto Entrena tenga 80% de datos 'g' y 20% de datos 'h'
  #y que el conjunto Prueba tenga 20% de datos de 'g' y 20% de datos 'h'
  #aproximadamente
  cont = 0
  for i in range(Entrena.shape[0]):
    if Entrena['class'][i] == 'g':
      cont = cont+1

  if 12332*0.8-cont<=tole:
    print('Se obtuvo representatividad')
    break
  print('No se ha logrado representatividad, por favor espere')
```

Figura 23: Obtención de conjuntos y verificación de representatividad.

2.2.2. Actividad 12

Para la obtención de los histogramas primero se necesitaba separar el conjunto de entrenamiento en dos subconjuntos, uno que poseyera a los datos de clase Hadrón y otro que tuviese a los de clase No-Hadrón, para ello se utilizó la porción de código que se observa en la figura 24, donde se observa que los histogramas poseerán 20 bins, cabe destacar que este parámetro se puede variar y es más, más adelante en este informe se variará este parámetro para obtener distintos resultados de curvas ROC y de precisión-recall.

```
#Separamos el conjunto de entrenamiento en hadron y no hadron
Entrena_H = Entrena[Entrena['class'] == 'h'].reset_index(drop=True) #Arreglo donde se guardaran los hadrones
Entrena_NH = Entrena[Entrena['class'] == 'g'].reset_index(drop=True) #Arreglo donde se guardaran los no-hadrones
bins = 20 #Bins a utilizar
```

Figura 24: Subconjuntos del conjunto de entrenamiento según su clase.

Con lo anterior configurado se procedió a obtener la cantidad de filas que poseía el subconjunto de entrenamiento que contenía a los elementos tipo Hadrón, esto se hizo con la finalidad de posteriormente normalizar los histogramas. Luego, se utilizó la función *histogram()* de *Numpy* para obtener el histograma, luego los valores obtenidos del histograma fueron divididos por el total de filas obtenidas con anterioridad, con esto se logró normalizar el histograma. Una vez realizado lo anterior se pasó a dar revisión a los valores obtenidos en el histograma, donde aquellos bins que tuviesen probabilidad de 0 se les otorgó una probabilidad de 0.0000001, esto se hizo para evitar futuras divisiones por 0. Posteriormente se implementó una forma de ver el histograma obtenido, para ello se utilizó la función *bar()* de *Matplotlib*, función a la que se le entregó un vector tal que el eje x de la gráfica a obtener sea el punto medio de cada bin, y para el caso del valor en el eje y sería la probabilidad de que un dato "caiga" en ese bin. Lo anterior se puede observar en la figura 25, donde se obtuvo el histograma

de la característica 'fLength' para los datos de tipo Hadrón del conjunto de entrenamiento, cuyo histograma puede observarse en la figura 26. Cabe mencionar que el procedimiento realizado para obtener el resto de histogramas es análogo, pero en el caso de los histogramas obtenidos para el subconjuntos de clase No-Hadrón la normalización se obtiene dividiendo por la cantidad de filas que poseía dicho subconjunto.

```
#Primero para los hadrones
filas_H = Entrena_H.shape[0]

#------------------------Caracteristica 1
hist1, binsedges1 = np.histogram(Entrena_H['fLength'],bins=bins) #Obtenemos el histograma
hist1 = hist1/filas_H #Lo normalizamos dividiendo por la cantidad de filas

#En este for nos encargamos de que los datos del histograma que tienen probabilidad
#0 pasen a tener una probabilidad muy pequeña, esto lo hacemos para evitar futuras
#divisiones por 0
for i in range(len(hist1)):
  if hist1[i] == 0:
    hist1[i] = 0.0000001
#Las siguientes 4 lineas de codigo sirven para obtener un grafico del histograma
#obtenido, donde para localizar cada uno de los valores del histograma se esta
#ocupando el punto medio entre los bordes de cada bin
medio1 = np.zeros(bins)
for i in range(len(binsedges1)-1):
  m = (binsedges1[i+1]+binsedges1[i])/2
  medio1[i] = m
#Esto solo sirve para apreciar mejor el plot
plt.bar(medio1,hist1,width=5,label='Hadron',ec='black')
plt.legend(loc='upper right')
plt.title('fLength del conjunto de entrenamiento')
plt.show()
```

Figura 25: Obtención de histogramas.

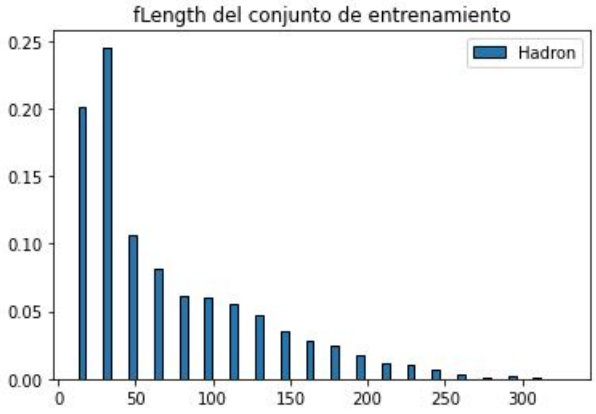

Figura 26: Histograma para caracteristica 'fLength' del conjunto de Hadrones.

Una vez obtenidos los histogramas estos se guardan en arreglos. Un arreglo posee a los histogramas obtenidos de la clase Hadrón y otro que posee a los de clase No-Hadrón, esto se hizo para facilitar

la implementación de la actividad siguiente. Además se definió el tiempo que tarda en ejecutarse el entrenamiento del clasificador utilizando la función *time()* de la librería *time*. Lo anteriormente mencionado se puede observar en la figura 27.

```
#Arreglos donde guardo los histogramas segun su clase, lo usare para que la actividad 13
#pueda hacerse mas facil
histograma_H = [hist1,hist2,hist3,hist4,hist5,hist6,hist7,hist8,hist9,hist10]
histograma_NH = [hist11,hist12,hist13,hist14,hist15,hist16,hist17,hist18,hist19,hist20]

#Tiempo de ejecucion
tiempo_hist = time.time()
```

Figura 27: Configuraciones finales.

2.2.3. Actividad 13

En la presente actividad se debía obtener la verosimilitud de cada elemento del conjunto de prueba, esto mediante el uso de los histogramas obtenidos en la actividad anterior. Para esto se definió, en primera instancia, un vector donde se guardarían las verosimilitudes de cada elemento, luego se procedió a iterar sobre cada uno de los elementos del conjunto de prueba. Cabe destacar que se expondrá el procedimiento realizado para obtener las verosimilitudes al considerar a los elementos del conjunto de prueba como hadrones, para el caso de los no-hadrones es análogo.

Al iterar se realizó una doble iteración, una sobre los elementos del conjunto de prueba y la otra sobre las características de estos, la lógica detrás de las iteraciones es: para el primer elemento, definir dos arreglos de largo 10, uno donde se guardaran las características del elemento en cuestión y otro donde se guardaran las probabilidades de ocurrencia de estos, probabilidades obtenidas de los histogramas, luego al iterar sobre las características se obtiene la primera de ellas y se observa, en su respectivo histograma, específicamente se observa en que bin "cae". Una vez definido este bin se debe observar si la característica que se está observando es menor a la menor característica que posee el conjunto de entrenamiento o si es mayor, si esto ocurre quiere decir que la característica observada cae fuera del histograma, ante esta situación se le define una probabilidad pequeña de 0.0000001, caso contrario, si lo anterior no ocurre quiere decir que la característica si cae dentro del histograma y por tanto tiene una probabilidad asociada. Para obtener dicha probabilidad se evalúa el bin obtenido anteriormente en el histograma y dicha probabilidad se guarda en uno de los arreglos creados. Una vez se itera sobre todas las características se procede a calcular la verosimilitud, esta verosimilitud corresponde a multiplicar todos los valores guardados en el arreglo donde se iban a guardar las probabilidades, para que finalmente el valor de verosimilitud obtenido se guarde en el arreglo defino fuera de los bucles, arreglo en donde se guardarían las verosimilitudes de los diferentes datos. Lo antes mencionado se puede observar en la figura 28

```
#Vector donde se guardaran las verosimilitudes obtenidas de los histogramas
#de la clase Hadron
Vero_Prueba_H = np.zeros(Prueba.shape[0])

#Calculamos la verosimilitud de cada dato perteneciente al dataframe Prueba
#Primero viendolo como si fuesen hadrones
for i in range(Prueba.shape[0]):
  caracteristicas = np.zeros(10)
  probabilidad = np.zeros(10)
  for j in range(10):
    #Obtenemos la caracteristica
    caracteristicas[j] = Prueba[names[j]][i]
    #Obtenemos en el que cae la caracteristica
    b = math.floor(bins * (caracteristicas[j] - Entrena_H[names[j]].min())/(Entrena_H[names[j]].max() - Entrena_H[names[j]].min()))
    if (caracteristicas[j]<=Entrena_H[names[j]].min()) or ((caracteristicas[j]>=Entrena_H[names[j]].max())):
      probabilidad[j] = 0.0000001
    else:
      probabilidad[j] = histograma_H[j][b]
  vero = np.prod(probabilidad)
  Vero_Prueba_H[i] = vero
```

Figura 28: Cálculo de verosimilitudes para el conjunto de hadrones.

2.2.4. Actividad 14

En la presente actividad se procedió a clasificar los diferentes elementos del conjunto de prueba así como también se le dio cálculo a las tasas de verdaderos y falsos positivos. En primer lugar se calcularon las divisiones entre las verosimilitudes para cada elemento del conjunto de prueba y se guardaron en un arreglo, luego se definieron los valores de Θ a utilizar así como también un conjunto de vectores que poseerán los verdaderos positivos, falsos positivos, verdaderos negativos y falsos negativos. Lo anterior se puede apreciar en la figura 29.

```
#Arreglo donde guardo las divisiones de verosimilitudes
clas = np.zeros(Vero_Prueba_H.shape[0])

#En el arreglo clas calculo la division de verosimilitudes al considerar el dato
#del conjunto de prueba como caso hadron y clase no-Hadron
for i in range(Vero_Prueba_H.shape[0]):
  clas[i] = (Vero_Prueba_H[i])/(Vero_Prueba_NH[i])

#Creamos un vector con posibles valores de Theta
Theta = np.linspace(0,9.9,100)

#Vector donde se guardaran los verdaderos positivos (VP: Cuando digo es hadron
# y efectivamente era hadron)
VP = np.zeros(len(Theta))

#Vector donde se guaradaran los falsos positivos (FP: Cuando digo es hadron
#y era no-hadron)
FP = np.zeros(len(Theta))

#Vector donde se guardaran los verdaderos negativos (VN: Cuando digo es no-hadron
#y efectivamente era no-hadron)
VN = np.zeros(len(Theta))

#Vector donde se guardaran los falsos negativos (FN: Cuando digo es no-hadron
#y es hadron)
FN = np.zeros(len(Theta))
```

Figura 29: Clasificador de Bayes.

Para realizar el calculo de las diferentes tasas se hizo una doble iteración, una sobre cada posible valor de Θ y la otra sobre cada elemento del conjunto de prueba. La lógica a seguir en estas iteraciones es: para el primer valor de Θ se configuran en 0 los valores de verdaderos y falsos positivos así como también los valores de verdaderos y falsos negativos, luego se itera sobre cada elemento del conjunto de prueba, esta iteración busca verificar si se acertó en la clasificación de los elementos de prueba, donde si por ejemplo, se dice que el elemento es un Hadrón se observa si la clase del dato efectivamente es un hadrón, si lo es se le suma uno al valor de verdadero positivo y si no es un hadrón se le suma 1 al valor de falso positivo. Para el caso contrario, si se dice que el elemento es un No-Hadrón, se observa si la clase del elemento de prueba es un hadron, si no lo es significa que falló la clasificación y se le suma uno a los falsos negativos y si acerté en la clasificación se le suma uno a los verdaderos negativos, una vez se realiza esto para todos los elementos del conjunto de prueba se añaden los valores de verdaderos positivos, falsos positivos, verdaderos negativos y falsos negativos a los arreglos definidos fuera de los bucles y se procede a repetir el proceso con el otro valor de Θ. El código encargado de realizar lo previamente mencionado se puede observar en la figura 30.

```
for i in range(len(Theta)):
  V_P = 0
  F_P = 0
  V_N = 0
  F_N = 0
  for j in range(Prueba.shape[0]):
    #Digo que es hadron
    if (clas[j] >= Theta[i]):
      clase = Prueba['class'][j]
      #Acerte
      if (clase == 'h'):
        V_P = V_P+1
      #No acerte
      elif (clase == 'g'):
        F_P = F_P+1
    #Digo que es no-hadron
    elif (clas[j] < Theta[i]):
      clase = Prueba['class'][j]
      #No acerte
      if (clase == 'h'):
        F_N = F_N+1
      #acerte
      elif (clase == 'g'):
        V_N = V_N+1
  #Añado lo obtenido a los vectores VP, FP, VN y FN
  VP[i] = V_P
  FP[i] = F_P
  VN[i] = V_N
  FN[i] = F_N
```

Figura 30: Cálculo de verdaderos y falsos para cada elemento.

Luego simplemente se realizó el cálculo de las tasas de verdaderos positivos y falsos positivos así como también el cálculo del precisión y el recall, finalmente solo se gráfico la tasa de verdaderos y falsos positivos para lograr obtener la curva ROC. El código encargado de hacer los cálculos de las tasas se observa en la figura 31 y la curva ROC obtenida se observa en la figura 32.

```
#Calculamos la taza de verdaderos positivos y falsos positivos para cada theta
TVP = np.zeros(len(Theta))
TFP = np.zeros(len(Theta))
#Ademas se calculara el precission y el recall, datos a utilizar en la siguiente
#actividad
P = np.zeros(len(Theta))
R = np.zeros(len(Theta))
for i in range(len(Theta)):
  Verd_Pos = VP[i]
  Fals_Neg = FN[i]
  Fals_Pos = FP[i]
  Verd_Neg = VN[i]
  TVP[i] = Verd_Pos/(Verd_Pos+Fals_Neg)
  TFP[i] = Fals_Pos/(Fals_Pos+Verd_Neg)
  P[i] = Verd_Pos/(Verd_Pos+Fals_Pos)
  R[i] = Verd_Pos/(Verd_Pos+Fals_Neg)

#Obtenemos la curva ROC
cosa1 = [0,1]
cosa2 = [0,1]
plt.plot(cosa1,cosa2,color='red')
plt.plot(TFP,TVP,color='blue')
plt.xlabel('Tasa falsos positivos')
plt.ylabel('Tasa verdaderos positivos')
plt.title('Curva ROC')
plt.show()
```

Figura 31: Cálculo de tasas.

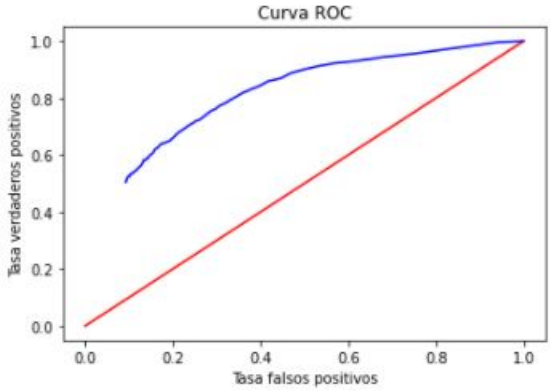

Figura 32: Curva ROC obtenida mediante los histogramas.

2.2.5. Actividad 15

Para el cálculo de la curva de Precisión-Recall simplemente se graficaron los valores de Precisión y Recall obtenidos en la actividad anterior, ante tal situación se obtuvo la curva de Precisión-Recall de la figura 33.

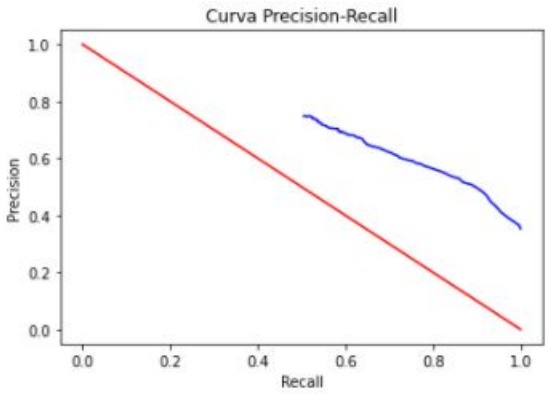

Figura 33: Curva Precisión-Recall obtenida mediante los histogramas.

2.3. Parte 3: Clasificación usando gaussianas multivariantes

2.3.1. Actividad 16

Para la primera actividad asociada a la clasificación usando gaussianas multivariantes se definieron 2 vectores, uno donde se guardarán las medias por característica de los elementos del conjunto de entrenamiento de clase hadrón y otro que cumplirá la misma función pero para los No-hadrones dichas medias se obtuvieron mediante el uso de la función *mean()* de *Numpy*. Una vez calculadas las medias se procedió a calcular las matrices de covarianza de los subconjuntos del conjunto de entrenamiento, para esto se eliminó la columna de los subconjuntos que tenía a la clase y luego se traspuso el dataframe, con esto se logró que las características fuesen las filas y los datos las columnas. Posteriormente se necesitaba utilizar la función *cov()* de *Numpy* para calcular las matrices de covarianzas, sin embargo, esta función no acepta como parámetro a un dataframe, sino que acepta matrices, motivo por el cual se pasó el dataframe a matriz mediante el uso de la función *to_numpy()*. Una vez calculadas las matrices de covarianza para cada subconjunto del conjunto de entrenamiento se procedió a obtener el determinante de estas y la inversa, para ello se utilizaron las funciones *linalg.det()* y *linalg.inv()* de *Numpy*, respectivamente, además se obtuvo el tiempo de ejecución del entrenamiento mediante el uso de la función *time()*. Lo anterior se puede apreciar en la figura 34.

```
#Obtenemos los promedios del conjunto de entrenamiento para cada caracteristica y clase
Prom_H = np.zeros(10)
Prom_NH = np.zeros(10)

for i in range(10):
  Prom_H[i] = np.mean(Entrena_H[names[i]])
  Prom_NH[i] = np.mean(Entrena_NH[names[i]])

#Con la finalidad de obtener la matriz de covarianza para el conjunto de entrenamiento
#de los hadrones y no hadrones, primero quitamos la columna 'class' de los conjuntos
#Y luego trasponemos el dataframe, de tal forma que las caracteristicas sean las filas
#y los datos las columnas
Entrena_H_SC = Entrena_H.iloc[:,:10].transpose()
Entrena_NH_SC = Entrena_NH.iloc[:,:10].transpose()

#Para obtener la matriz de covarianza pasamos los dataframes recien obtenidos
#a arrays (a matrices)
M_Entrena_H_SC = Entrena_H_SC.to_numpy()
M_Entrena_NH_SC = Entrena_NH_SC.to_numpy()

#Obtenemos las matrices de covarianza para cada clase
cov_H = np.cov(M_Entrena_H_SC)
cov_NH = np.cov(M_Entrena_NH_SC)

#Obtenemos las inversas de estas matrices
I_cov_H = np.linalg.inv(cov_H)
I_cov_NH = np.linalg.inv(cov_NH)

#Obtenemos los determinantes de las matrices de covarianza
det_cov_H = np.linalg.det(cov_H)
det_cov_NH = np.linalg.det(cov_NH)

#Tiempo de ejecucion
tiempo_gauss = time.time()
```

Figura 34: Entrenamiento del clasificador mediante gaussianas multivariantes.

2.3.2. Actividad 17

Para obtener la verosimilitud de los elementos del conjunto de prueba se definieron 2 arreglos, uno donde se guardarán los valores de las gaussianas de los diferentes datos del conjunto de prueba al considerarlos como hadrones y otro donde se guardarán los valores de las gaussianas al considerarlas no-hadrones. Posteriormente se procedió a realizar un proceso análogo al realizado para obtener la verosimilitud de los datos del conjunto de prueba mediante histogramas, es decir, se hizo una doble iteración una sobre cada elemento del conjunto de prueba y la otra sobre cada característica, donde se obtenían las diferentes características de cada elemento y se guardaban en un vector, luego se definía a la gaussiana multivariante utilizando los valores de medias y matriz de covarianza correspondientes y el valor obtenido se guardaba en los arreglos definidos anteriormente. Lo antes mencionado se puede observar en la figura 35, donde se expone el cálculo de las verosimilitudes al considerar los elementos del conjunto de prueba como hadrones.

```
#Definimos vectores en los que se guardaran los datos de las gaussianas obtenidas
gauss_H = np.zeros(Prueba.shape[0])
gauss_NH = np.zeros(Prueba.shape[0])

#Calculamos la verosimilitud de los elementos del conjunto de prueba
#considerando como que son hadrones
for i in range(Prueba.shape[0]):
  caracteristicas = np.zeros(10)
  for j in range(10):
    #Obtenemos la caracteristica
    caracteristicas[j] = Prueba[names[j]][i]
  dato = caracteristicas - Prom_H #resta entre el vector de caracteristicas y vector de promedios
  tras = np.transpose(dato) #vector dato transpuesto
  p = np.dot(tras,I_cov_H) #calculamos la multiplicacion entre tras y la inversa de la matriz de covarianza
  ps = np.dot(p,dato) #calculamos la multiplicacion entre p y dato
  pi = math.pi #definicion del numero pi
  guassilla = (math.exp((-1/2)*ps)) / (math.sqrt(((2*pi)**(10))*det_cov_H))
  gauss_H[i] = guassilla
```

Figura 35: Cálculo de verosimilitudes mediante gaussianas multivariantes.

2.3.3. Actividad 18

La actividad número 18 es idéntica a la actividad número 16, solo se hicieron renombres de algunas variables. La curva ROC obtenida es la apreciable en la figura 36.

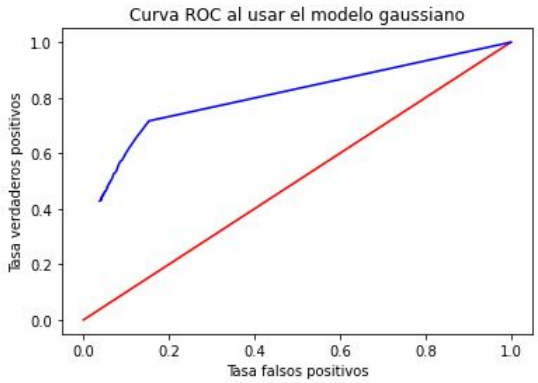

Figura 36: Curva ROC obtenida mediante gaussianas multivariantes.

2.3.4. Actividad 19

La curva de precisión-recall obtenida mediante el uso de gaussianas multivariantes se observa en la figura 37.

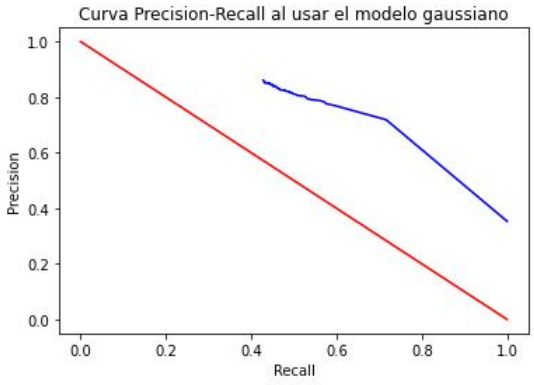

Figura 37: Curva de Precisión-Recall obtenida mediante gaussianas multivariantes.

2.4. Parte 4: Comparación de resultados

2.4.1. Actividad 20

El graficar la curva ROC obtenida mediante el uso de histogramas y la obtenida mediante el uso de gaussianas multivariantes se obtiene la gráfica de la figura 38.

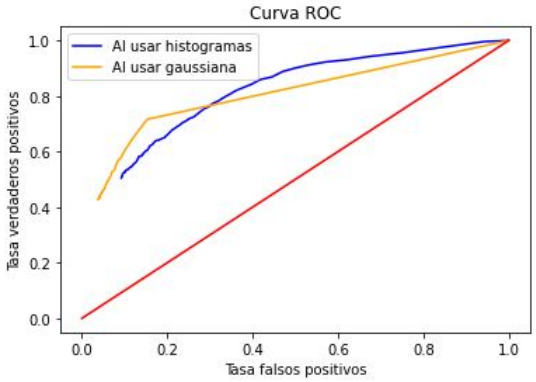

Figura 38: Curvas ROC.

2.4.2. Actividad 21

Al graficar las curvas de precisión-recall obtenidas anteriormente se obtiene la figura 39.

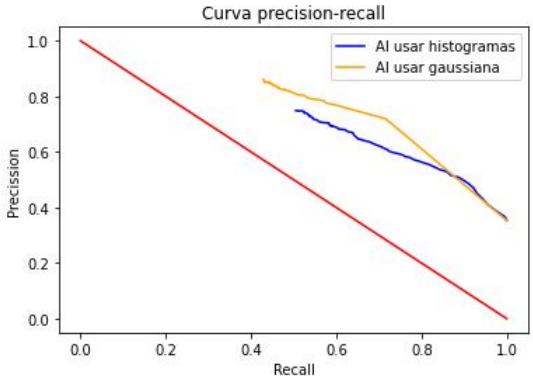

Figura 39: Curvas de Precisión-Recall.

2.4.3. Actividad 22

Finalmente se tiene que los tiempos requeridos para entrenar a ambos clasificadores son los de la figura 40.

```
Al usar histogramas el programa tarda 1617738581.6254563
y
Al usar el modelo gaussiano el programa tarda 1617743949.4561937
```

Figura 40: Tiempos de entrenamiento.

2.5. Resultados

A lo largo de las diferentes secciones previas se observaron las curvas ROC y de precisión-recall al considerar histogramas con 20 bins y 100 valores de Θ que iban entre 0 y 9.9. A continuación se mostraran las curvas ROC y de precisión-recall obtenidas ante diferentes configuraciones, así como también su tiempo de ejecución.

El primero de los casos consiste en aumentar la cantidad de bins a 50 y mantener la misma cantidad de Θ. De esta configuración se obtuvieron las curvas de las figuras 41 y 42, así como el tiempo de ejecución de la figura 43.

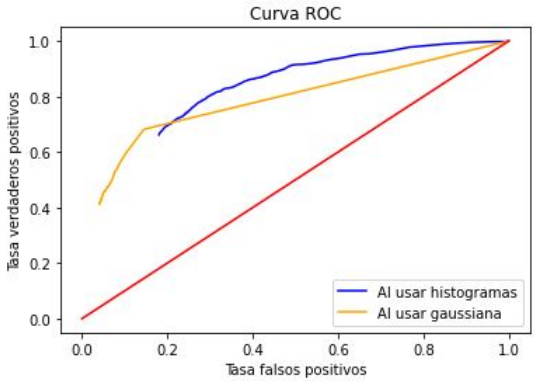

Figura 41: Curvas ROC de ambos clasificadores al considerar 50 bins.

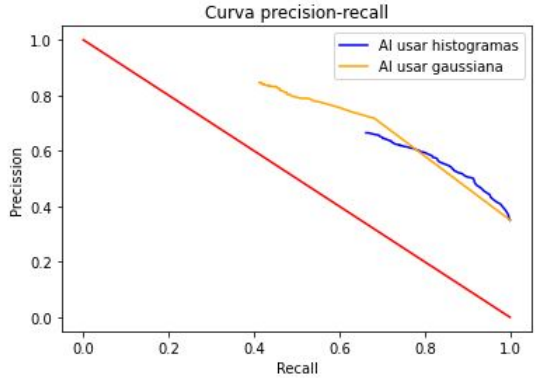

Figura 42: Curvas Precisión-Recall de ambos clasificadores al considerar 50 bins.

```
Al usar histogramas el programa tarda 1617821765.1763256
y
Al usar el modelo gaussiano el programa tarda 1617821825.0048873
```

Figura 43: Tiempos de ejecución al considerar 50 bins.

Luego, al mantener la cantidad de bins en 50 y aumentar la cantidad de valores de Θ a 500, con

valores entre 0 y 100, se obtuvieron las curvas de las figuras 44, 45 y el tiempo de ejecución de la figura 46.

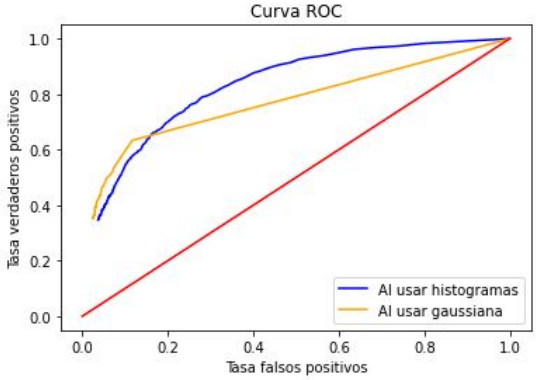

Figura 44: Curvas ROC al tener más valores de θ.

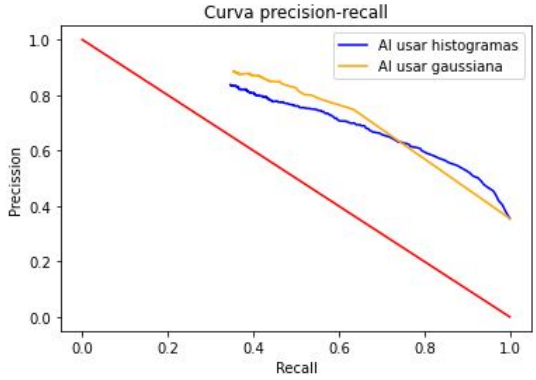

Figura 45: Curvas de Precisión-Recall al considerar más valores de Θ.

```
Al usar histogramas el programa tarda 1617822157.8688636
y
Al usar el modelo gaussiano el programa tarda 1617822286.4992175
```

Figura 46: Tiempos de ejecución al considerar más valores de Θ.

3. Conclusión

A modo de conclusión se puede afirmar el cumplimiento del objetivo propuesto en la tarea número 1 del curso EL4106 Inteligencia Computacional, donde se logró implementar un clasificador de Bayes mediante dos modelos diferentes, uno basado en el uso de histogramas y otro basado en el uso de Gaussianas multivariantes.

Dentro de los aprendizajes obtenidos se destaca fuertemente el hecho de implementar un clasificador de Bayes, pues en cursos anteriores se había visto la teoría detrás de estos clasificadores más nunca se llegó a implementar uno como tal, por lo cual esta tarea sirvió para llevar a la práctica los conocimientos aprendidos tanto en cursos anteriores así como también en este. Se destaca también el aprendizaje obtenido de utilizar las bibliotecas *Pandas*, *Numpy* y *Matplotlib* pertenecientes al lenguaje *Python*, sin embargo, el uso de estas bibliotecas supuso una dificultad considerable pues eran nuevas para el estudiante encargado de realizar la tarea, así como también que este mismo llevaba mucho tiempo sin programar en *Python*, motivo que retrasó el avance de la tarea en sus primeros instantes.

Dentro de los resultados obtenidos se logra apreciar que se tiene un comportamiento como el que indica la teoría, con esto se hace referencia a que el clasificador implementado no es perfecto y que presenta una cierta tasa de error, pero tampoco es un pésimo clasificador. De las curvas ROC obtenidas al considerar diferentes cantidades de bins y cantidades de Θ se logra observar la superioridad del clasificador que utiliza histogramas, por otro lado, de las curvas de Precisión-Recall se logra observar un comportamiento muy similar en clasificadores. En base a lo antes mencionado se puede afirmar, que en términos generales, el clasificador que opera en base a histogramas es superior al que opera en base a Gaussianas, esto tiene sentido, pues de los histogramas obtenidos en la actividad número 12 se logra observar que no todos tienen una distribución gaussiana o normal y por tanto asumir que todas las características se comportan como gaussianas no es una buena aproximación.

Una forma de mejorar los algoritmos implementados sería , por ejemplo, no hacer dobles iteraciones y buscar una manera de realizar la misma tarea pero realizando solo una iteración. Otra manera de optimizar el algoritmo implementado, principalmente en el segundo clasificador implementado, sería aproximar las diferentes características de formas diferentes, es decir, si una característica no tuviese una distribución gaussiana y tuviese una distribución uniforme (o pseudo-uniforme) sería bueno aproximar esa característica con dicha distribución, quizás con esto se lograría que el segundo clasificador supere al clasificador que funciona con histogramas.