1. **Implementar um algoritmo de envoltória convexa visto em sala de aula -> Sequencia dos vertex**
2. **Implementar o algoritmo de varredura linear para detecção de interseções em conjuntos de segmentos -> Bool**
3. **Implementar o método para verificação de separabilidade linear conforme descrito acima -> (2)**
4. **Implementar o método para construir o modelo, caso os dados sejam linearmente separáveis ->**
5. **Implementar o classificador que recebe um conjunto de amostras desconhecidas e atribui rótulos a elas ->**
6. **Implementar o método para computar as métricas de classificação para os experimentos**
7. **Realizar os experimentos conforme a descrição acima**
8. **Redigir o relatório especificado acima**

1. Calculando a envoltoria convexa

Para caclular o envoltoria convexa usamos o algoritmo de Jarvis March.Começamos do ponto mais à esquerda (ou ponto com valor mínimo de coordenada x) e continuamos agrupando os pontos no sentido anti-horário.Para cada ponto p, usamos a função orientation, para descobrir o proximo ponto entrar na envoltoria.O próximo ponto é selecionado como o ponto que bate todos os outros pontos na orientação anti-horária, ou seja, o próximo ponto é q se para qualquer outro ponto r, tivermos "orientação (p, q, r) = anti-horário".

O algoritmo tem complexidade de tempo O(nh), onde n é o número de pontos e h é o número de pontos no casco convexo.

In [788]:

```
from scipy.spatial import ConvexHull, convex_hull_plot_2d
import numpy as np
rng = np.random.default_rng()
#geração dos pontos aleatorios
points = rng.random((30, 2))   # 30 random points in 2-D
#hull = ConvexHull(points)
```

In [789]:

```
import math

#calcula a orientacao dos pontos , horario, anti horario, colinear
def orientation(p1, p2, p3):
    x1, y1, x2, y2, x3, y3 = *p1, *p2, *p3
    d = (y3-y2)*(x2-x1) - (y2-y1)*(x3-x2)
    if d > 0:
        return 1
    elif d < 0:
        return -1
    else:
        return 0

#seleciona o ponto pivor, ponto mais a esquerda( menor coordenada x )
def left_most(points):
    min = 0
    for i in range(1,len(points)):
      if points[i][0] < points[min][0]:
        min = i
      elif points[i][0] == points[min][0]:
        if points[i][1] > points[min][1]:
          min = i

    return points[min]

#calcula a distancia de dois pontos
def dist(p1, p2):
    x1, y1, x2, y2 = *p1, *p2
    return math.sqrt((y2-y1)**2 + (x2-x1)**2)
```

```
#icalcula a envoltoria convexa
def gift_wrapping(points):

    on_hull = left_most(points)
    hull = []
    while True:
        hull.append(on_hull)
        next_point = points[0]
        for point in points:
            o = orientation(on_hull, next_point, point)
            if (next_point == on_hull).all() or o == 1 or (o == 0 and dist(on_hull, poin
t) > dist(on_hull, next_point)):
                next_point = point
        on_hull = next_point
        if (on_hull == hull[0]).all():
            break
    return hull
```

Comparando as envoltorias geradas

In [790]:

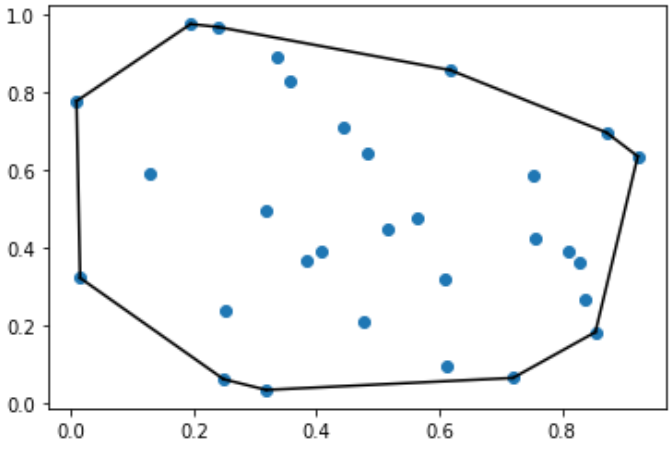
```
#calcula a envoltoria convexa pelo implementação do algoritmo
hull2 =gift_wrapping(points)
#calcula a envoltoria convexa pela biblioteca
hull = ConvexHull(points)
```

In [791]:

```
# plot da envoltoria convexa da implementação
import matplotlib.pyplot as plt

def scatter_plot(coords,convex_hull=None):
 xs,ys=zip(*coords)
 plt.scatter(xs,ys)

 if convex_hull!=None:
  for i in range(1,len(convex_hull)+1):
   if i==len(convex_hull): i=0
   c0=convex_hull[i-1]
   c1=convex_hull[i]
   plt.plot((c0[0],c1[0]),(c0[1],c1[1]),'r')
 plt.show()
```

In [792]:

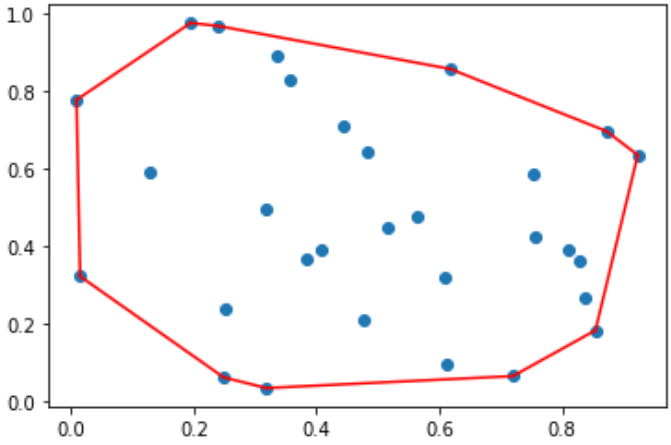
```
#plot da envoltoria da biblioteca
plt.plot(points[:,0], points[:,1], 'o')
for simplex in hull.simplices:
    plt.plot(points[simplex, 0], points[simplex, 1], 'k-')
```



In [793]:

```
#plot da envoltoria da implementação
```

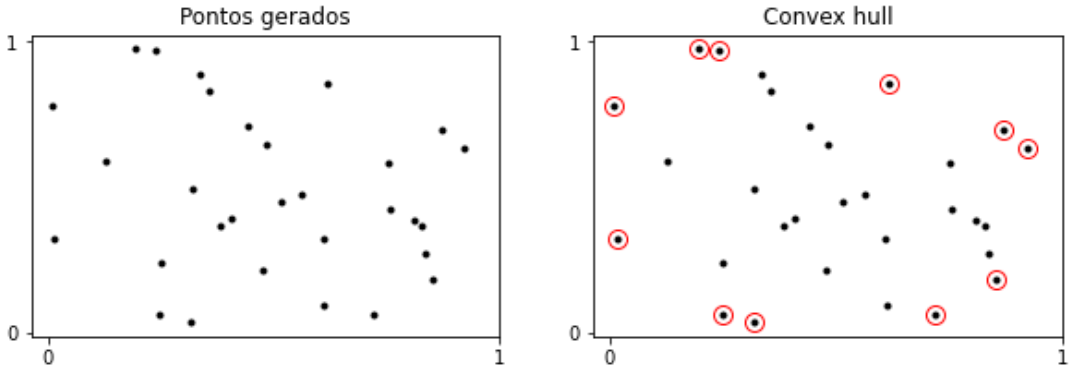
```
scatter_plot(points,hull2)
```



Plot dos pontos selecionados da envoltoria

In [794]:

```
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(10, 3))

for ax in (ax1, ax2):
    ax.plot(points[:, 0], points[:, 1], '.', color='k')
    if ax == ax1:
        ax.set_title('Pontos gerados')
    else:
        ax.set_title('Convex hull')
        for simplex in hull2:
            ax.plot(points[simplex.astype(int), 0], points[simplex.astype(int), 1], 'c')
        ax.plot(points[hull.vertices, 0], points[hull.vertices, 1], 'o', mec='r', color=
'none', lw=1, markersize=10)
    ax.set_xticks(range(2))
    ax.set_yticks(range(2))
plt.show()
```



2. Procurando interseções

Podemos encontrar o ponto mais a direita em tempo O(n)

In [795]:

```
# find the right most point index, assuming a convex hull that start from
# the left most point
def right_most(hull):
  for i in range(len(hull) - 1):
    if hull[i][0] > hull[i+1][0]:
      return i
  # last point is the right most
  return len(hull) - 1
```

Podemos checar se um ponto q está no quadrante delimitado pelos pontos p e r em tempo O(1). Caso q seja colinear a p, r, podemos também podemos concluir que q se encontra no segmento p, r

colinear a p-r, podemos também podemos concluir que q se encontra no segmento p-r.

In [796]:

```
# check if q is on the segment defined by p-r, assuming they are collinear
def on_segment(p, q, r):
  if q[0] <= max(p[0],r[0]) and q[0] >= min(p[0],r[0]):
    return True
  return False
```

Para verificar se duas arestas se interceptam, podemos concluir esse procedimento em tempo O(1), dado que não há iteração e usamos apenas metodos de tempo também constante.

In [797]:

```
# check if 2 edges intersect
def edge_intersection(p1, q1, p2, q2):
  o1 = orientation(p1,q1,p2)
  o2 = orientation(p1,q1,q2)
  o3 = orientation(p2,q2,p1)
  o4 = orientation(p2,q2,q1)

  # if they both see each point at a different orientation
  if o1 != o2 and o3 != o4:
    return True
  # if a point belongs to the segment
  if o1 == 0 and on_segment(p1, p2, q1):
    return True
  if o2 == 0 and on_segment(p1, q2, q1):
    return True
  if o3 == 0 and on_segment(p2, p1, q2):
    return True
  if o4 == 0 and on_segment(p2, q1, q2):
    return True
  return False
```

Agora, com os metodos auxiliares definidos e analisados, tratamos da varredura em si.

Como é impossivel que suas arestas se cruzem sem que interceptem o eixo x, podemos começar a analise nesse ponto e encerrá-la ao fim de qualquer uma das envoltórias. Como as aresta da mesma envoltória não se cruzam, basta iterarmos a aresta mais a esquerda (primeira a terminar na varredura) de cada uma envoltória com as duas da outra envoltória.

Assim, itemos em tempo constante, no pior caso, cada aresta de A e B, o que nos dá um tempo O(m+n)

In [798]:

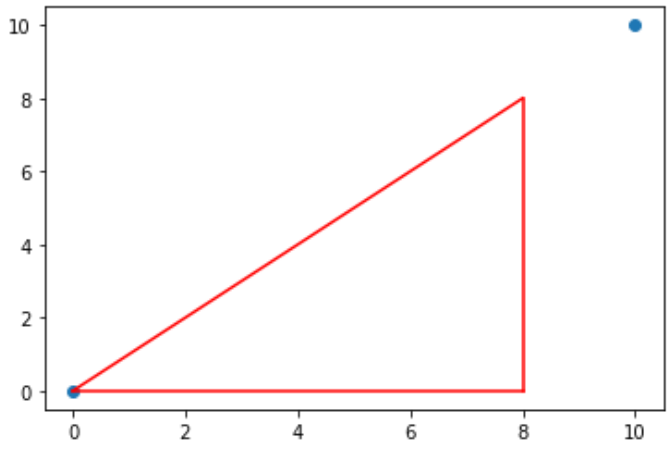
```
# check if the hull A intersect hull B
def intersect(A, B):
  # check if they intersect in x
  Aend = right_most(A)
  Bend = right_most(B)
  if A[0][0] > B[Bend][0] or B[0][0] > A[Aend][0]:
    return False

  # find start of x intersection
  Al=0
  Ar=-1
  Bl=0
  Br=-1
  if A[0][0] < B[0][0]:
    while A[Al+1][0] < B[0][0]:
      Al+=1
    while A[Ar][0] < B[0][0]:
      Ar-=1
  else:
    while B[Bl+1][0] < A[0][0]:
      Bl+=1
    while B[Br][0] < A[0][0]:
```

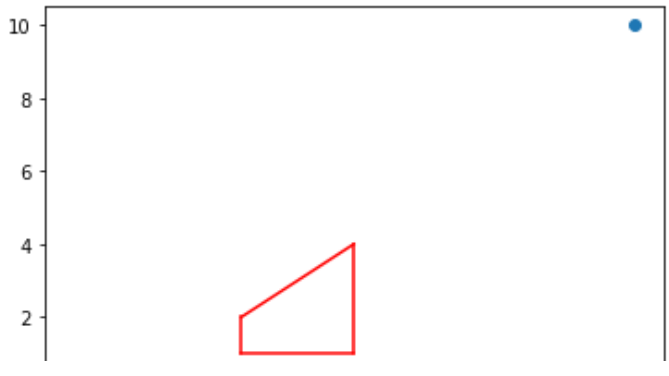
```
      Br-=1
  # start iterating over edges
  while (A[Al] != A[Ar] and B[Bl] != B[Br]):
    # find the next edge to iterate
    xmin = min(A[Al+1][0],A[Ar][0],B[Bl+1][0],B[Br][0])
    if A[Al+1][0] == xmin:
      edge = [A[Al],A[Al+1]]
      Al += 1
      choseA = True
    elif A[Ar][0] == xmin:
      edge = [A[Ar+1],A[Ar]]
      Ar -= 1
      choseA = True
    elif B[Bl+1][0] == xmin:
      edge = [B[Bl],B[Bl+1]]
      Bl += 1
      choseA = False
    else:
      edge = [B[Br+1],A[Br]]
      Br -= 1
      choseA = False
    # check for intersections with the current edges
    if choseA:
      if edge_intersection(edge[0], edge[1], B[Bl], B[Bl+1]) or edge_intersection(edge[0
], edge[1], B[Br+1], B[Br]):
        return True
    else:
      if edge_intersection(edge[0], edge[1], A[Al], A[Al+1]) or edge_intersection(edge[0
], edge[1], A[Ar+1], A[Ar]):
        return True
  # if no intersection was found
  return False
```

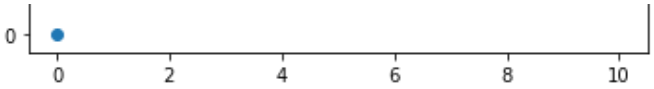
testing

In [799]:

```
A = [[0,0],[8,0],[8,8]]
B = [[3,1],[3,2],[5,4],[5,1]]
all = [[0,0],[10,10]]
scatter_plot(all,A)
scatter_plot(all,B)
intersect(A,B)
```







Out[799]:

```
False
```

3. Verificação de Separabilidade

Para que dois poligonos sejam separaveis, existem duas condições suficientes:

- **Não haverem interseções entre suas arestas**
- **Um não estar contido no outro**

Podemos verificar se um ponto pertence a uma envoltória verificando se ele possui a mesma orientação em relação a todas as suas arestas, o que ocorre em tempo O(n).

In [800]:

```
# check if point is contained in hull
def contained(hull, p):
  o = orientation(hull[-1],hull[0], p)
  for i in range(len(hull)-1):
    if o == orientation(hull[i],hull[i+1],p):
      return True
  return False
```

Agora basta verificar a separabilidade chamando os metodos já definidos, o que nos da um tempo O(m+n).

In [801]:

```
# check separability
def separable(A,B):
  # print(f"Intersect: {intersect(A,B)}")
  # print(f"Contained A,B[0]: {contained(A,B[0])}")
  # print(f"Contained B,A[0]: {contained(B,A[0])}")
  return (not intersect(A,B)) and (not contained(A,B[0])) and (not contained(B,A[0]))
```

test

In [802]:

```
separable(A,B)
```

Out[802]:

```
False
```

4. Construção do Modelo

In [803]:

```
## Imports
from matplotlib.patches import Polygon
from numpy import random
## Euclidean Distance
from scipy.spatial import distance
```

In [804]:

```
def printEverything(hull1, hull2, min_dist_points, ponto_medio, perpendicular_line, x_ran
ge=None, y_range=None, especifyRange=False):
  """
```

```
    Exibe as envoltórias convexas, segmento correspondente à menor distância
    entre as envoltórias e reta perpendicular.
  """
  y = hull1
  z = hull2

  p = Polygon(y, color="black", fill=False)
  q = Polygon(z, color="black", fill=False)
  # r = Polygon(sort_counterclockwise(a), color="black", fill=False)

  fig,ax = plt.subplots()

  ax.add_patch(p)
  ax.add_patch(q)

  # Segmento da menor distância
  x_values = [min_dist_points[0][0], min_dist_points[1][0]]
  y_values = [min_dist_points[0][1], min_dist_points[1][1]]

  # Segmento perpendicular
  x__values = [0, ponto_medio[0], 0 - perpendicular_line[1]/perpendicular_line[0]]
  y__values = [perpendicular_line[1], ponto_medio[0]*perpendicular_line[0] + perpendicul
ar_line[1], 0]

  plt.plot(x_values, y_values, color="red", linewidth=2)
  plt.plot(x__values, y__values, color="blue", linewidth=2)

  # passar ticks como parâmetro
  # plt.xticks(range(4, 8, 1))
  # plt.xticks(np.arange(min(x_values), max(y_values), 0.2))
  # plt.yticks(range(1, 7))
  if (especifyRange):
    plt.xlim([x_range[0], x_range[1]])
    plt.ylim([y_range[0], y_range[1]])

  # Exibindo ponto médio
  plt.plot(ponto_medio[0], ponto_medio[1], marker="o", markeredgecolor="black")
  plt.plot(x_values, y_values, color="red", linewidth=2)



  plt.show()
  return None
```

In [805]:

```
def findMinDistance(envolt1, envolt2):
  """
    Encontra o segmento correspondente à menor distância ("vertex distance") entre uma en
voltória
    P e outra Q.

    Retorno: (x, (y1, y2))
      x -> Distância mínima entre as duas envoltórias
      (y1, y2) -> Tupla com os pontos correspondente às extremidades do segmento de
      menor distância

    Complexidade do Algoritmo: O(n^2) ( Força Bruta )
  """

  ii32 = np.iinfo(np.int32)
  min = ii32.max ##max int 32 bits
  first_envolt_point = 0
  second_envolt_point = 0
  aux = []

  for y_point in envolt1:
    for z_point in envolt2:
      aux_dist = distance.euclidean(y_point, z_point)
      aux.append(aux_dist)
      if (aux_dist < min):
```

```
        min = aux_dist
        first_envolt_point = y_point
        second_envolt_point = z_point

  return (min, (first_envolt_point, second_envolt_point))
```

In [806]:

```
def findMidpoint(pontos):
  x_medio = np.array([pontos[1][0] + pontos[0][0]])/2
  y_medio = np.array([pontos[1][1] + pontos[0][1]])/2
  return (x_medio, y_medio)
```

In [807]:

```
def findPerpendicularLine(m, b, point):
  """
    Encontra um segmento perpendicular a uma reta da forma y = mx + b passada como
    argumento e um ponto sobre ela.

    OBS: Aparentemente há um problema onde a reta perpendicular apresenta um pe-
    queno deslocamento na exibição da mesma. Um outro aluno (de outro grupo)
    relatou o mesmo problema na aula do dia 18/10.
  """
  ## Inclinação da reta perpendicular
  m_ = -(1/m)
  b_ = point[1] - m_*point[0]

  return (m_, b_[0])
```

In [808]:

```
def getMinLine(min_dist_points, ponto_medio):
  m = (min_dist_points[1][1] - min_dist_points[0][1]) / (min_dist_points[1][0] - min_dis
t_points[0][0])
  b = ponto_medio[1] - m*ponto_medio[0]

  return (m, b)
```

Teste

In [809]:

```
## Auxiliary Test Method
def method2(size, init=0, final=99):
    ret = []
    while len(ret) < size:
        ret.append([random.randint(init, final), random.randint(init, final)])
    return ret
```

In [810]:

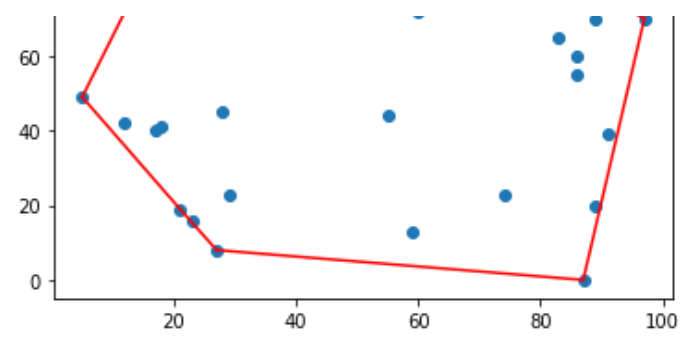
```
pontos1 = np.array(method2(30))
pontos2 = np.array(method2(30, init = 100, final = 200))
```

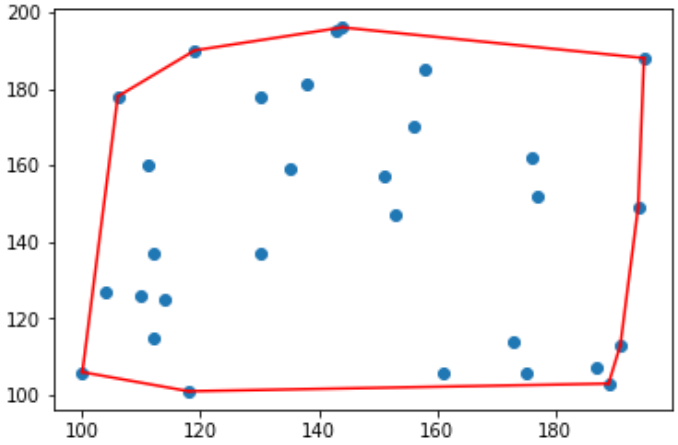
In [811]:

```
hull2 =gift_wrapping(pontos1)
hull3 =gift_wrapping(pontos2)
```

In [812]:

```
scatter_plot(pontos1,hull2)
scatter_plot(pontos2,hull3)
```





In [813]:

```
min_dist, min_dist_points = findMinDistance(hull2, hull3)
ponto_medio = findMidpoint(min_dist_points)
```

In [814]:

```
print(f"Distância Mínima: {min_dist}")
print(f"Extremidades do segmento de distância mínima: x1: {min_dist_points[0][0]} y1: {mi
n_dist_points[0][1]} -- x2: {min_dist_points[1][0]} y2: {min_dist_points[1][1]}")
```

```
Distância Mínima: 30.01666203960727
Extremidades do segmento de distância mínima: x1: 74 y1: 91 -- x2: 100 y2: 106
```

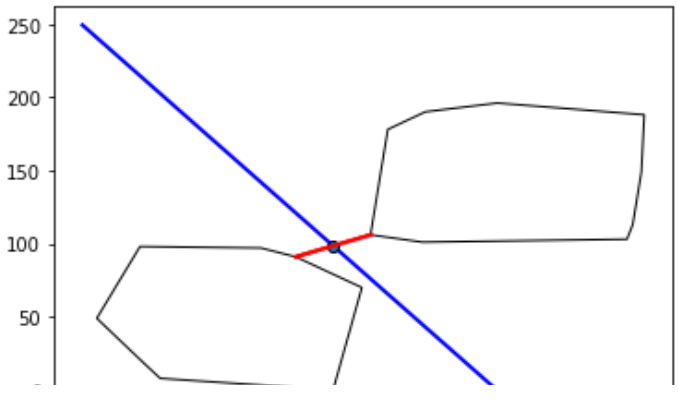
In [815]:

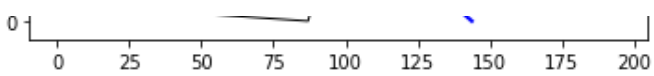
```
# Segmento da distância mínima
m, b = getMinLine(min_dist_points, ponto_medio)

# Reta perpendicular
perpendicular_line = findPerpendicularLine(m, b, ponto_medio)
```

In [816]:

```
printEverything(hull2, hull3, min_dist_points, ponto_medio, perpendicular_line)
```

```
/usr/local/lib/python3.7/dist-packages/numpy/core/shape_base.py:65: VisibleDeprecationWar
ning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists
-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to d
o this, you must specify 'dtype=object' when creating the ndarray.
  ary = asanyarray(ary)
```





5. Classificador

In [817]:

```
def classifier_predict(new_point, midpoint, line, class1, class2):
    #pontos a e b são pontos garantidamente que pertencem a reta perpenticular
    #ponto c para checar a orientação em relação a reta
    # aX = 0
    aX = line[0]
    aY = line[1]
    bX, bY, cX, cY = *midpoint, *new_point
    # print(bX) #ponto medio
    # print(bY) #ponto medio
    # print(cX)
    # print(cY)
    # print(f"new point: {new_point}")


    val = ((bX - aX)*(cY - aY) - (bY - aY)*(cX - aX))
    print(f"Produto Vetorial: {val}")

    if(val>0):
      return class1; #right: acima da reta perpenticular
    if(val<0):
      return class2 #left: abaixo da reta perpenticular

    return 0 #está sobre a reta perpenticular

def classifier(points, midpoint, line, class1, class2):
  valores_previstos = []

  for point in points:
    # print(f"ponto: {point}")
    classe = classifier_predict(point,midpoint,line, class1, class2)
    print('Predicted class of new data point {} is: {}\n'.format(point, classe))
    valores_previstos.append([point, classe])
  return valores_previstos
```

Análise: Base Iris

In [818]:

```
import pandas as pd
from google.colab import drive
import matplotlib.pyplot as plt
import itertools

arq2 =  pd.read_csv('/content/iris.csv')
arq2
```

Out[818]:

	SepalLength	SepalWidth	PetalLength	PetalWidth	Species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.6	3.1	1.5	0.2	Iris-setosa
3	5.0	3.6	1.4	0.2	Iris-setosa
4	5.4	3.9	1.7	0.4	Iris-setosa
...	...	...	...	...	...
145	6.5	3.0	5.5	1.8	Iris-virginica

	SepalLength	SepalWidth	PetalLength	PetalWidth	Species
146	7.7	2.6	6.9	2.3	Iris-virginica
147	6.0	2.2	5.0	1.5	Iris-virginica
148	6.9	3.2	5.7	2.3	Iris-virginica
149	6.2	2.8	4.8	1.8	Iris-virginica

150 rows × 5 columns

Vamos separar a base em dados de treino e teste

In [819]:

```
list(set(arq2["Species"].values))
```

Out[819]:

```
['Iris-virginica', 'Iris-setosa', 'Iris-versicolor']
```

Como vamos classificar uma planta entre duas classes, vamos remover uma das entradas da base

In [820]:

```
print(len(arq2[arq2["Species"] == "Iris-setosa"]))
print(len(arq2[arq2["Species"] == "Iris-virginica"]))
print(len(arq2[arq2["Species"] == "Iris-versicolor"]))
```

```
50
50
50
```

Inicialmente, optamos por remover as entradas que possuem classe Iris-versicolor, ou seja, gostaríamos de classificar novas entradas para as classes Iris-setosa e Iris-virginica

In [821]:

```
## Removendo as entradas de classe "Iris-versicolor"

arq2 = arq2[ (arq2["Species"] != "Iris-versicolor") ]
arq2 = arq2.reset_index(drop=True)
arq2
```

Out[821]:

	SepalLength	SepalWidth	PetalLength	PetalWidth	Species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.6	3.1	1.5	0.2	Iris-setosa
3	5.0	3.6	1.4	0.2	Iris-setosa
4	5.4	3.9	1.7	0.4	Iris-setosa
...	...	...	...	...	...
95	6.5	3.0	5.5	1.8	Iris-virginica
96	7.7	2.6	6.9	2.3	Iris-virginica
97	6.0	2.2	5.0	1.5	Iris-virginica
98	6.9	3.2	5.7	2.3	Iris-virginica
99	6.2	2.8	4.8	1.8	Iris-virginica

100 rows × 5 columns

Agora temos apenas as duas classes escolhidas:

In [822]:

In [822]:

```
list(set(arq2["Species"].values))
```

Out[822]:

```
['Iris-virginica', 'Iris-setosa']
```

Vamos escolher agora os atributos para que possamos representar a base de dados em duas dimensões, após conversarmos, achamos que seria interessante escolher os atributos SepalLength e SepalWidth para representar nossa base de dados.

In [823]:

```
## Diminuindo a dimensão da base de dados para 2-D
arq3 = arq2[["SepalLength", "SepalWidth", "Species"]]
arq3
```

Out[823]:

	SepalLength	SepalWidth	Species
0	5.1	3.5	Iris-setosa
1	4.9	3.0	Iris-setosa
2	4.6	3.1	Iris-setosa
3	5.0	3.6	Iris-setosa
4	5.4	3.9	Iris-setosa
...	...	...	...
95	6.5	3.0	Iris-virginica
96	7.7	2.6	Iris-virginica
97	6.0	2.2	Iris-virginica
98	6.9	3.2	Iris-virginica
99	6.2	2.8	Iris-virginica

100 rows × 3 columns

Para dividir a base entre dados de treino e teste, optamos pela divisão em 70-30, ou seja, 70% das instâncias foram utilizadas para treino, equanto que 30% foram utilizadas para teste.

In [824]:

```
training_data = arq3.sample(frac=0.7, random_state=25)
testing_data = arq3.drop(training_data.index)
```

In [825]:

```
print(f"No. of training examples: {training_data.shape[0]}")
print(f"No. of testing examples: {testing_data.shape[0]}")
```

```
No. of training examples: 70
No. of testing examples: 30
```

In [826]:

```
pontos_1 = [] #Iris-setosa
pontos_2 = [] #Iris-virginica

for line in training_data.values:
  if (line[2] == "Iris-setosa"):
    pontos_1.append([line[0], line[1]])

  if (line[2] == "Iris-virginica"):
    pontos_2.append([line[0], line[1]])
```

```
pontos_1 = np.array(pontos_1)
pontos_2 = np.array(pontos_2)
```

In [827]:

```
pontos_1
```

Out[827]:

```
array([[5.1, 3.8],
       [5.2, 4.1],
       [5.5, 3.5],
       [5. , 3.4],
       [5.1, 3.5],
       [4.9, 3.1],
       [5.2, 3.4],
       [4.8, 3.1],
       [5.4, 3.9],
       [4.8, 3.4],
       [5.2, 3.5],
       [5.7, 4.4],
       [4.4, 3. ],
       [4.4, 3.2],
       [4.4, 2.9],
       [5. , 3.5],
       [4.6, 3.2],
       [5.1, 3.5],
       [5.4, 3.4],
       [4.6, 3.6],
       [5.1, 3.3],
       [4.7, 3.2],
       [4.3, 3. ],
       [4.6, 3.4],
       [4.6, 3.1],
       [4.5, 2.3],
       [5.3, 3.7],
       [4.8, 3. ],
       [5. , 3.5],
       [5.5, 4.2],
       [4.9, 3.1],
       [5.1, 3.7],
       [4.8, 3. ],
       [5. , 3.2]])
```

In [828]:

```
separable(pontos_1,pontos_2)
```

Out[828]:

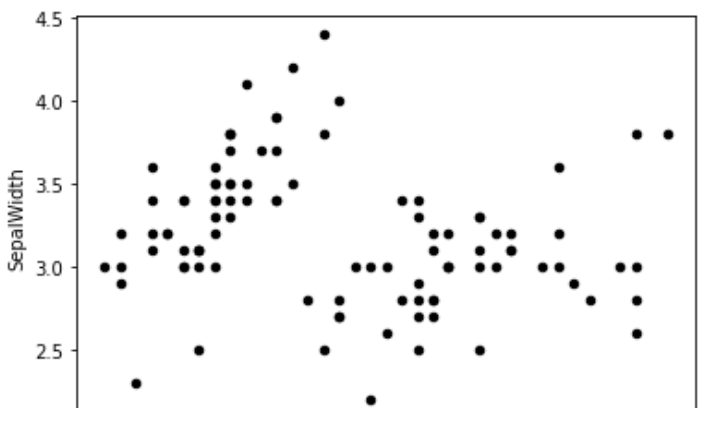
```
False
```

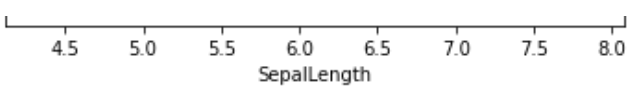
In [829]:

```
arq2.plot.scatter(x='SepalLength',y='SepalWidth',c='black')
```

Out[829]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f688e20d4d0>
```



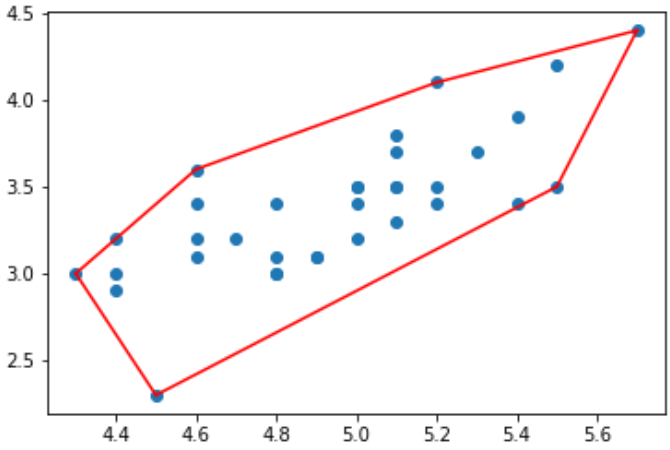


In [830]:

```
hull4 = gift_wrapping(pontos_1)
```

In [831]:

```
scatter_plot(pontos_1,hull4)
```

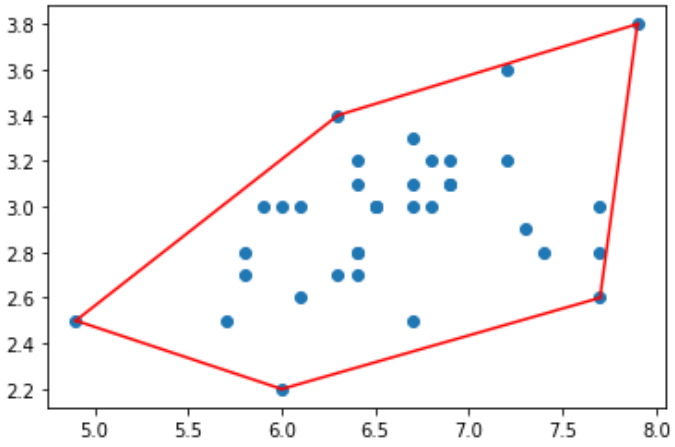


In [832]:

```
hull5 = gift_wrapping(pontos_2)
```

In [833]:

```
scatter_plot(pontos_2,hull5)
```



In [834]:

```
min_dist, min_dist_points = findMinDistance(hull4, hull5)
ponto_medio = findMidpoint(min_dist_points)
```

In [835]:

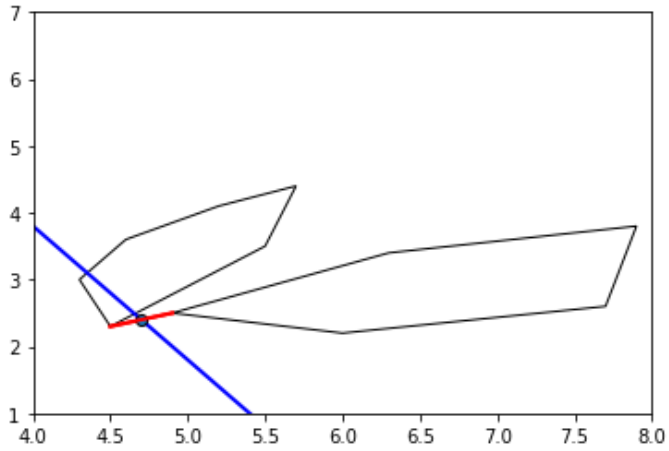
```
# Segmento da distância mínima
m, b = getMinLine(min_dist_points, ponto_medio)

# Reta perpendicular
perpendicular_line = findPerpendicularLine(m, b, ponto_medio)
```

In [836]:

```
printEverything(hull4, hull5, min_dist_points, ponto_medio, perpendicular_line, [4, 8],
[1, 7], especifyRange=True)
```

```
/usr/local/lib/python3.7/dist-packages/numpy/core/shape_base.py:65: VisibleDeprecationWar
ning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists
```

```
-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to d
o this, you must specify 'dtype=object' when creating the ndarray.
  ary = asanyarray(ary)
```



Note que a o conjunto de atributos utilizados para representar a base de dados, não formou "clusters" de pontos bem separados, entrando num caso indesejável para o nosso modelo. Vamos tentar uma outra combinação de atributos na expectativa de obter um resultado mais adequado para a aplicação do modelo.

Vamos escolher o par de atributos SepalLength e PentalLength.

In [837]:

```
## Diminuindo a dimensão da base de dados para 2-D
arq4 = arq2[["SepalLength", "PetalLength", "Species"]]
arq4
```

Out[837]:

	SepalLength	PetalLength	Species
0	5.1	1.4	Iris-setosa
1	4.9	1.4	Iris-setosa
2	4.6	1.5	Iris-setosa
3	5.0	1.4	Iris-setosa
4	5.4	1.7	Iris-setosa
...	...	...	...
95	6.5	5.5	Iris-virginica
96	7.7	6.9	Iris-virginica
97	6.0	5.0	Iris-virginica
98	6.9	5.7	Iris-virginica
99	6.2	4.8	Iris-virginica

100 rows × 3 columns

In [838]:

```
training_data = arq4.sample(frac=0.7, random_state=25)
testing_data = arq4.drop(training_data.index)
```

In [839]:

```
pontos_1 = [] #Iris-setosa
pontos_2 = [] #PetalLength

for line in training_data.values:
  if (line[2] == "Iris-setosa"):
    pontos_1.append([line[0], line[1]])
```

```
  if (line[2] == "Iris-virginica"):
    pontos_2.append([line[0], line[1]])

pontos_1 = np.array(pontos_1)
pontos_2 = np.array(pontos_2)
```

Explicar problema da separabilidade

In [840]:

```
separable(pontos_1,pontos_2)
```

Out[840]:

```
False
```

In [841]:

```
hull4 = gift_wrapping(pontos_1)
```

In [842]:

```
hull5 = gift_wrapping(pontos_2)
```

In [843]:

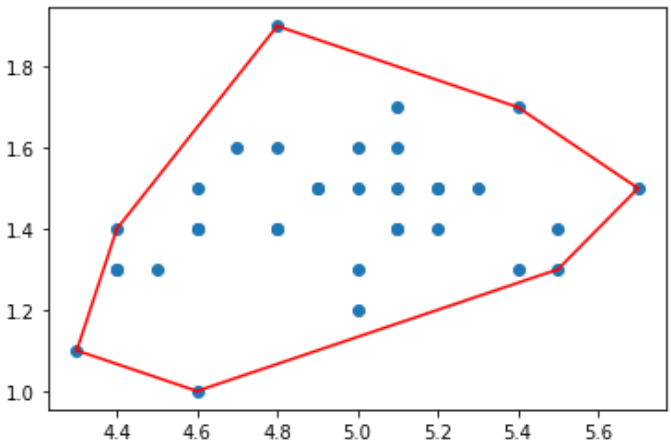
```
min_dist, min_dist_points = findMinDistance(hull4, hull5)
ponto_medio = findMidpoint(min_dist_points)
```

In [844]:

```
# Segmento da distância mínima
m, b = getMinLine(min_dist_points, ponto_medio)

# Reta perpendicular
perpendicular_line = findPerpendicularLine(m, b, ponto_medio)
```

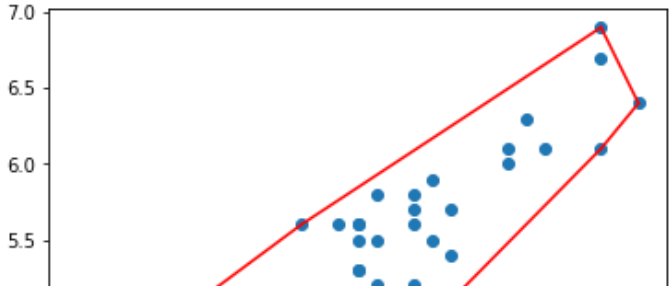
In [845]:

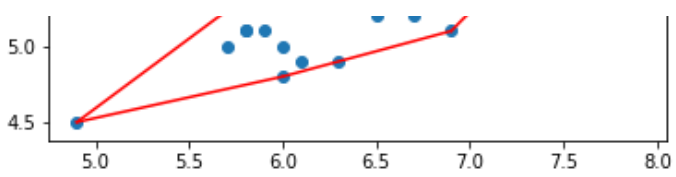
```
scatter_plot(pontos_1,hull4)
```



In [846]:

```
scatter_plot(pontos_2,hull5)
```

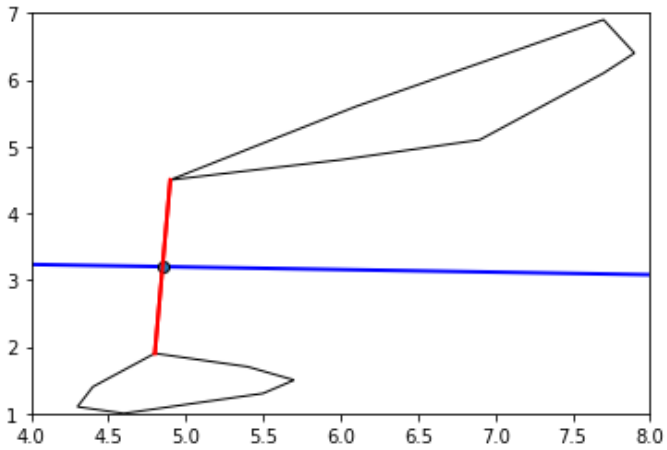




In [847]:

```
x_range = [4, 8]
y_range = [1, 7]
printEverything(hull4, hull5, min_dist_points, ponto_medio, perpendicular_line, x_range,
y_range, especifyRange=True)
```

```
/usr/local/lib/python3.7/dist-packages/numpy/core/shape_base.py:65: VisibleDeprecationWar
ning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists
-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to d
o this, you must specify 'dtype=object' when creating the ndarray.
  ary = asanyarray(ary)
```



Introduzindo o início da aplicação do modelo

In [848]:

```
pontos_1 = [] #Iris-setosa
pontos_2 = [] #PetalLength

for line in testing_data.values:
  if (line[2] == "Iris-setosa"):
    pontos_1.append([line[0], line[1]])

  if (line[2] == "Iris-virginica"):
    pontos_2.append([line[0], line[1]])

pontos_1 = np.array(pontos_1)
pontos_2 = np.array(pontos_2)
```

In [849]:

```
## Construindo o conjunto de pontos ( usando a base de treino )

Points = []
for x in pontos_1:
  Points.append(x)

for x in pontos_2:
  Points.append(x)

Points
```

Out[849]:

```
[array([4.9, 1.4]),
 array([5. , 1.4]),
 array([5.4, 1.7]),
 array([5.4, 1.5]),
```

```
 array([4.8, 1.6]),
 array([5.7, 1.7]),
 array([5.1, 1.5]),
 array([5. , 1.6]),
 array([5. , 1.6]),
 array([5.4, 1.5]),
 array([4.9, 1.5]),
 array([5.1, 1.5]),
 array([5.1, 1.9]),
 array([5. , 1.4]),
 array([4.7, 1.3]),
 array([5.8, 1.2]),
 array([6.3, 6. ]),
 array([7.1, 5.9]),
 array([6.3, 5.6]),
 array([7.6, 6.6]),
 array([6.5, 5.1]),
 array([7.7, 6.7]),
 array([5.6, 4.9]),
 array([7.2, 5.8]),
 array([6.3, 5.1]),
 array([5.8, 5.1]),
 array([6.7, 5.7]),
 array([6.3, 5. ]),
 array([6.2, 5.4]),
 array([6.2, 4.8])]
```

Vamos salvar os resultados da previsão do modelo na variável pontos_previstos

In [850]:

```
pontos_previstos = classifier(Points,ponto_medio,perpendicular_line, "Iris-virginica", "
Iris-setosa")
```

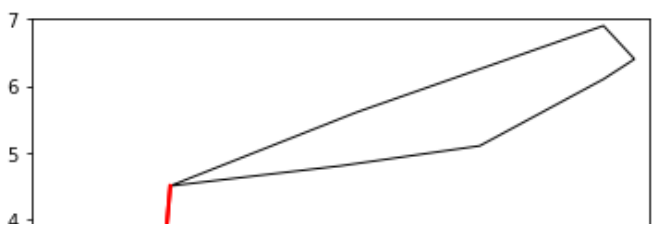
```
Produto Vetorial: [-8.78990385]
Predicted class of new data point [4.9 1.4] is: Iris-setosa

Produto Vetorial: [-8.77125]
Predicted class of new data point [5.  1.4] is: Iris-setosa

Produto Vetorial: [-7.23009615]
Predicted class of new data point [5.4 1.7] is: Iris-setosa

Produto Vetorial: [-8.20778846]
Predicted class of new data point [5.4 1.5] is: Iris-setosa

Produto Vetorial: [-7.83086538]
Predicted class of new data point [4.8 1.6] is: Iris-setosa

Produto Vetorial: [-7.17413462]
Predicted class of new data point [5.7 1.7] is: Iris-setosa

Produto Vetorial: [-8.26375]
Predicted class of new data point [5.1 1.5] is: Iris-setosa

Produto Vetorial: [-7.79355769]
Predicted class of new data point [5.  1.6] is: Iris-setosa

Produto Vetorial: [-7.79355769]
Predicted class of new data point [5.  1.6] is: Iris-setosa

Produto Vetorial: [-8.20778846]
Predicted class of new data point [5.4 1.5] is: Iris-setosa

Produto Vetorial: [-8.30105769]
Predicted class of new data point [4.9 1.5] is: Iris-setosa

Produto Vetorial: [-8.26375]
Predicted class of new data point [5.1 1.5] is: Iris-setosa

Produto Vetorial: [-6.30836538]
```

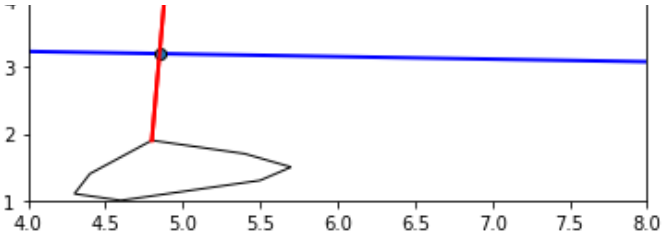
```
Predicted class of new data point [5.1 1.9] is: Iris-setosa

Produto Vetorial: [-8.77125]
Predicted class of new data point [5.  1.4] is: Iris-setosa

Produto Vetorial: [-9.31605769]
Predicted class of new data point [4.7 1.3] is: Iris-setosa

Produto Vetorial: [-9.59971154]
Predicted class of new data point [5.8 1.2] is: Iris-setosa

Produto Vetorial: [13.95817308]
Predicted class of new data point [6.3 6. ] is: Iris-virginica

Produto Vetorial: [13.61855769]
Predicted class of new data point [7.1 5.9] is: Iris-virginica

Produto Vetorial: [12.00278846]
Predicted class of new data point [6.3 5.6] is: Iris-virginica

Produto Vetorial: [17.13375]
Predicted class of new data point [7.6 6.6] is: Iris-virginica

Produto Vetorial: [9.59586538]
Predicted class of new data point [6.5 5.1] is: Iris-virginica

Produto Vetorial: [17.64125]
Predicted class of new data point [7.7 6.7] is: Iris-virginica

Produto Vetorial: [8.45028846]
Predicted class of new data point [5.6 4.9] is: Iris-virginica

Produto Vetorial: [13.14836538]
Predicted class of new data point [7.2 5.8] is: Iris-virginica

Produto Vetorial: [9.55855769]
Predicted class of new data point [6.3 5.1] is: Iris-virginica

Produto Vetorial: [9.46528846]
Predicted class of new data point [5.8 5.1] is: Iris-virginica

Produto Vetorial: [12.56625]
Predicted class of new data point [6.7 5.7] is: Iris-virginica

Produto Vetorial: [9.06971154]
Predicted class of new data point [6.3 5. ] is: Iris-virginica

Produto Vetorial: [11.00644231]
Predicted class of new data point [6.2 5.4] is: Iris-virginica

Produto Vetorial: [8.07336538]
Predicted class of new data point [6.2 4.8] is: Iris-virginica
```

In [851]:

```
printEverything(hull4, hull5, min_dist_points, ponto_medio, perpendicular_line, x_range,
y_range, especifyRange=True)
```

```
/usr/local/lib/python3.7/dist-packages/numpy/core/shape_base.py:65: VisibleDeprecationWar
ning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists
-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to d
o this, you must specify 'dtype=object' when creating the ndarray.
  ary = asanyarray(ary)
```





In [851]:

```
```

Computando métricas para o modelo de classificação para a base em questão

In [852]:

```
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
```

In [853]:

```
dados_teste = testing_data["Species"].values
```

In [854]:

```
dados_previstos = [classe for [point, classe] in pontos_previstos]
dados_previstos = np.array(dados_previstos)
```

In [855]:

```
# Computando a revogação
recall = recall_score(dados_teste, dados_previstos, average='macro')
print('Recall: %f' % recall)
```

```
Recall: 0.866071
```

In [856]:

```
# Computando o F1-Score
f1 = f1_score(dados_teste, dados_previstos, average='macro')
print('F1-Score: %f' % f1)
```

```
F1-Score: 0.866071
```

In [857]:

```
# Computando a precisão
precision_score(dados_teste, dados_previstos, average='macro')
print('Precisão: %f' % f1)
```

```
Precisão: 0.866071
```

Análise: Base Fonema

In [858]:

```
import pandas as pd
from google.colab import drive
import matplotlib.pyplot as plt
import itertools

arq2 =  pd.read_csv('./phoneme.csv')
arq2
```

Out[858]:

	Aa	Ao	Dcl	Iy	Sh	Class
0	1.240	0.875	-0.205	-0.078	0.067	0
1	0.268	1.352	1.035	-0.332	0.217	0
2	1.567	0.867	1.300	1.041	0.559	0
3	0.279	0.990	2.555	-0.738	0.000	0
4	0.307	1.272	2.656	-0.946	-0.467	0
...	...	...	...	...	...	...
5399	0.254	2.392	0.689	1.828	-0.544	0
5400	0.781	1.250	0.793	0.383	0.816	1
5401	1.031	0.584	1.866	1.532	-0.671	1
5402	0.150	0.933	2.363	-0.742	-0.617	0
5403	0.137	0.714	1.350	0.972	-0.630	1

5404 rows × 6 columns

Novamente vamos separar a base em dados de treino e teste. As classes nesse exemplo são dadas por: 1 -> Nasal 2 -> Oral

In [859]:

```
list(set(arq2["Class"].values))
```

Out[859]:

```
[0, 1]
```

Diferentemente da última vez a quantidade de entradas da base de dados sitribuídas por classe é diferente para cada classe. E, como dessa vez temos apenas duas classes, não será necessário remover nenhuma classe da base original.

In [860]:

```
print(len(arq2[arq2["Class"] == 0]))
print(len(arq2[arq2["Class"] == 1]))
```

```
3818
1586
```

Vamos escolher agora os atributos para que possamos representar a base de dados em duas dimensões, dessa vez, achamos que seria interessante escolher os atributos, inicialmente, por Aa e Ao para representar nossa base de dados.

In [861]:

```
## Diminuindo a dimensão da base de dados para 2-D
arq3 = arq2[["Aa", "Ao", "Class"]]
arq3
```

Out[861]:

	Aa	Ao	Class
0	1.240	0.875	0
1	0.268	1.352	0
2	1.567	0.867	0
3	0.279	0.990	0
4	0.307	1.272	0

	...	...	...
	Aa	Ao	Class
5399	0.254	2.392	0
5400	0.781	1.250	1
5401	1.031	0.584	1
5402	0.150	0.933	0
5403	0.137	0.714	1

5404 rows × 3 columns

Vamos separar os dados entre conjunto de treino e conjunto de testes. Novamente, a divisão será feita em 70-30, ou seja, 70% das instâncias foram utilizadas para treino, equanto que 30% foram utilizadas para teste.

In [862]:

```
training_data = arq3.sample(frac=0.7, random_state=25)
testing_data = arq3.drop(training_data.index)
```

In [863]:

```
print(f"No. of training examples: {training_data.shape[0]}")
print(f"No. of testing examples: {testing_data.shape[0]}")
```

```
No. of training examples: 3783
No. of testing examples: 1621
```

In [863]:

```

```

In [866]:

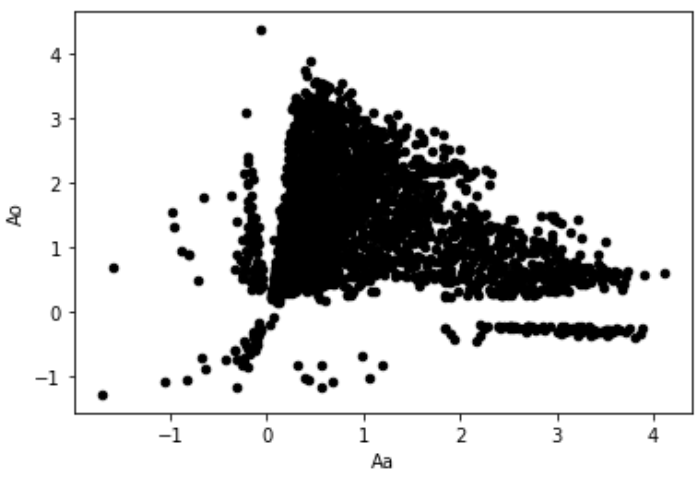
```
pontos_1 = [] # 0
pontos_2 = [] # 1

for line in training_data.values:
  if (line[2] == 0):
    pontos_1.append([line[0], line[1]])

  if (line[2] == 1):
    pontos_2.append([line[0], line[1]])

pontos_1 = np.array(pontos_1)
pontos_2 = np.array(pontos_2)
```

In [867]:

```
training_data.plot.scatter(x='Aa',y='Ao',c='black')
```

Out[867]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f68976a7890>
```

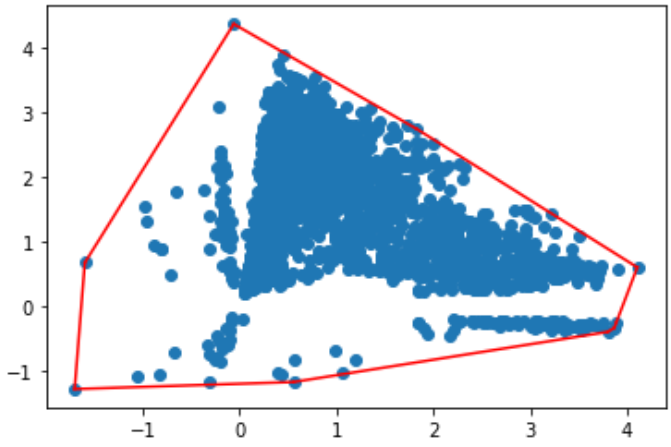


In [868]:

```
hull4 = gift_wrapping(pontos_1)
```

In [869]:

```
scatter_plot(pontos_1,hull4)
```

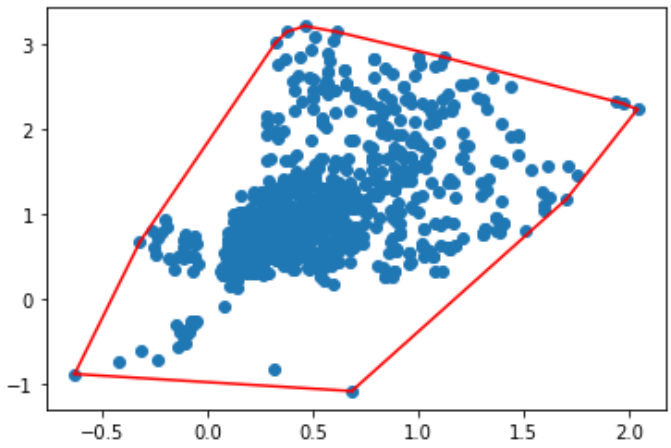


In [870]:

```
hull5 = gift_wrapping(pontos_2)
```

In [871]:

```
scatter_plot(pontos_2,hull5)
```



In [872]:

```
min_dist, min_dist_points = findMinDistance(hull4, hull5)
ponto_medio = findMidpoint(min_dist_points)
```

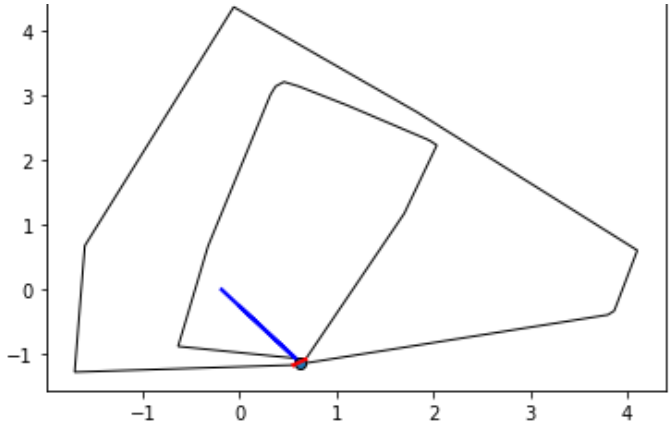
In [873]:

```
# Segmento da distância mínima
m, b = getMinLine(min_dist_points, ponto_medio)

# Reta perpendicular
perpendicular_line = findPerpendicularLine(m, b, ponto_medio)
```

In [874]:

```
printEverything(hull4, hull5, min_dist_points, ponto_medio, perpendicular_line)
```

```
/usr/local/lib/python3.7/dist-packages/numpy/core/shape_base.py:65: VisibleDeprecationWar
ning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists
-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to d
o this, you must specify 'dtype=object' when creating the ndarray.
  ary = asanyarray(ary)
```



In [875]:

```
## Diminuindo a dimensão da base de dados para 2-D
arq4 = arq2[["Aa", "Sh", "Class"]]
arq4
```

Out[875]:

	Aa	Sh	Class
0	1.240	0.067	0
1	0.268	0.217	0
2	1.567	0.559	0
3	0.279	0.000	0
4	0.307	-0.467	0
...	...	...	...
5399	0.254	-0.544	0
5400	0.781	0.816	1
5401	1.031	-0.671	1
5402	0.150	-0.617	0
5403	0.137	-0.630	1

5404 rows × 3 columns

In [876]:

```
training_data = arq4.sample(frac=0.7, random_state=25)
testing_data = arq4.drop(training_data.index)
```

In [877]:

```
print(f"No. of training examples: {training_data.shape[0]}")
print(f"No. of testing examples: {testing_data.shape[0]}")
```

```
No. of training examples: 3783
No. of testing examples: 1621
```

In [878]:

```
pontos_1 = [] # 0
pontos_2 = [] # 1

for line in training_data.values:
  if (line[2] == 0):
    pontos_1.append([line[0], line[1]])

  if (line[2] == 1):
    pontos_2.append([line[0], line[1]])

pontos_1 = np.array(pontos_1)
```

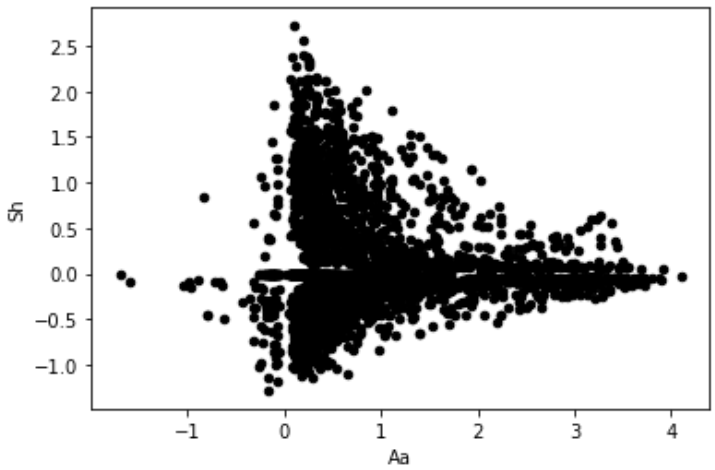
```
pontos_2 = np.array(pontos_2)
```

In [879]:

```
training_data.plot.scatter(x='Aa',y='Sh',c='black')
```

Out[879]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f68963062d0>
```

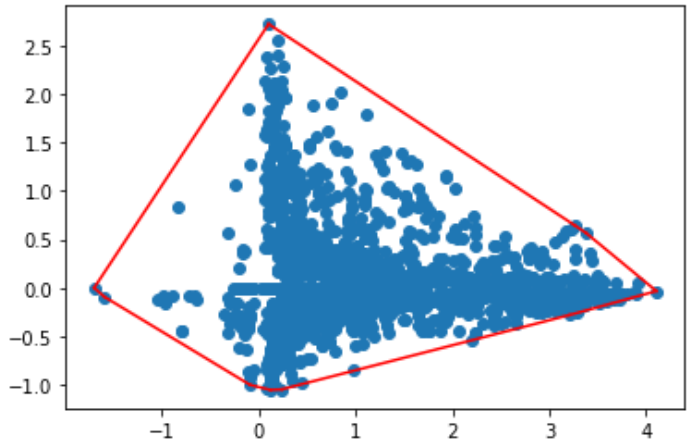


In [880]:

```
hull4 = gift_wrapping(pontos_1)
```

In [881]:

```
scatter_plot(pontos_1,hull4)
```

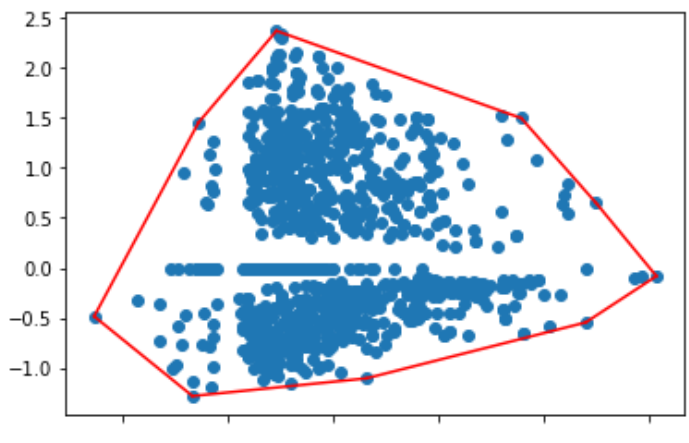


In [882]:

```
hull5 = gift_wrapping(pontos_2)
```

In [883]:

```
scatter_plot(pontos_2,hull5)
```



−0.5 0.0 0.5 1.0 1.5 2.0

In [884]:

```
min_dist, min_dist_points = findMinDistance(hull4, hull5)
ponto_medio = findMidpoint(min_dist_points)
```

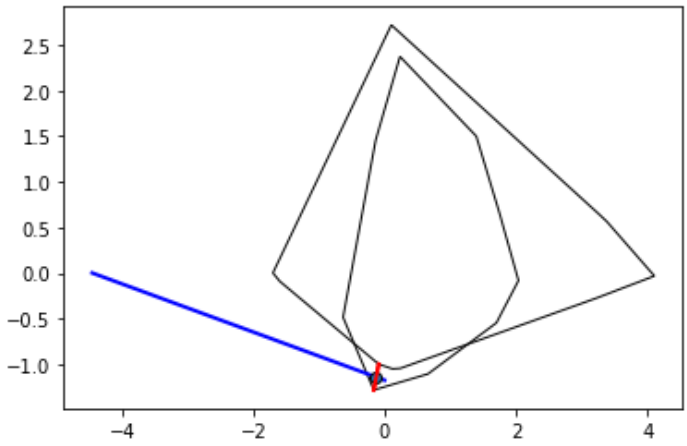
In [885]:

```
# Segmento da distância mínima
m, b = getMinLine(min_dist_points, ponto_medio)

# Reta perpendicular
perpendicular_line = findPerpendicularLine(m, b, ponto_medio)
```

In [886]:

```
printEverything(hull4, hull5, min_dist_points, ponto_medio, perpendicular_line)
```

```
/usr/local/lib/python3.7/dist-packages/numpy/core/shape_base.py:65: VisibleDeprecationWar
ning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists
-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to d
o this, you must specify 'dtype=object' when creating the ndarray.
  ary = asanyarray(ary)
```



In [886]:

```
```

Análise: Base BUPA

In [887]:

```
import pandas as pd
from google.colab import drive
import matplotlib.pyplot as plt
import itertools

arq2 =  pd.read_csv('./bupa.csv')
arq2
```

Out[887]:

	Mcv	Alkphos	Sgpt	Sgot	Gammagt	Drinks	Selector
0	85.0	92.0	45.0	27.0	31.0	0.0	1
1	85.0	64.0	59.0	32.0	23.0	0.0	2
2	86.0	54.0	33.0	16.0	54.0	0.0	2
3	91.0	78.0	34.0	24.0	36.0	0.0	2
4	98.0	55.0	13.0	17.0	17.0	0.0	2

...	Mcv	Alkphos	Sgpt	Sgot	Gammagt	Drinks	Selector
340	91.0	54.0	25.0	22.0	35.0	4.0	1
341	89.0	48.0	32.0	22.0	14.0	4.0	2
342	85.0	52.0	22.0	23.0	34.0	4.0	2
343	95.0	93.0	21.0	27.0	47.0	6.0	2
344	91.0	93.0	35.0	34.0	37.0	10.0	2

345 rows × 7 columns

Para essa base, poderíamos classificar as entradas entre a classe 1 que representa a pessoa que não sofre de alcolismo e a classe 2 que representa uma pessoa que sofre de alcolismo.

In [888]:

```
list(set(arq2["Selector"].values))
```

Out[888]:

```
[1, 2]
```

In [889]:

```
print(len(arq2[arq2["Selector"] == 1]))
print(len(arq2[arq2["Selector"] == 2]))
```

```
145
200
```

Tentamos utilizar de diversas combinações de atributos para representar as bases, mas aconteceu que nenhuma das várias combinações utilizadas gerou clusters de pontos os quais poderíamos dar seguimento para a construção do modelo. Segue abaixo uma tentativa de sequência dos próximos passos para um dos pares que utilizamos:

In [890]:

```
## Diminuindo a dimensão da base de dados para 2-D
arq3 = arq2[["Sgpt", "Gammagt", "Selector"]]
arq3
```

Out[890]:

	Sgpt	Gammagt	Selector
0	45.0	31.0	1
1	59.0	23.0	2
2	33.0	54.0	2
3	34.0	36.0	2
4	13.0	17.0	2
...	...	...	...
340	25.0	35.0	1
341	32.0	14.0	2
342	22.0	34.0	2
343	21.0	47.0	2
344	35.0	37.0	2

345 rows × 3 columns

In [891]:

```
training data = arq3.sample(frac=0.7, random state=25)
```

```
testing_data = arq3.drop(training_data.index)
```

In [892]:

```
print(f"No. of training examples: {training_data.shape[0]}")
print(f"No. of testing examples: {testing_data.shape[0]}")
```

```
No. of training examples: 241
No. of testing examples: 104
```

In [893]:

```
pontos_1 = [] # 1
pontos_2 = [] # 2

for line in training_data.values:
  if (line[2] == 1):
    pontos_1.append([line[0], line[1]])

  if (line[2] == 2):
    pontos_2.append([line[0], line[1]])

pontos_1 = np.array(pontos_1)
pontos_2 = np.array(pontos_2)
```

In [894]:

```
separable(pontos_1,pontos_2)
```

Out[894]:

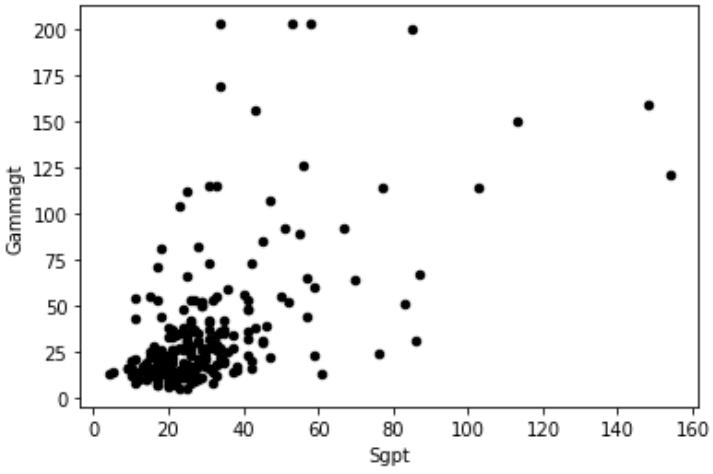
```
False
```

In [895]:

```
training_data.plot.scatter(x='Sgpt',y='Gammagt',c='black')
```

Out[895]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f688debc290>
```

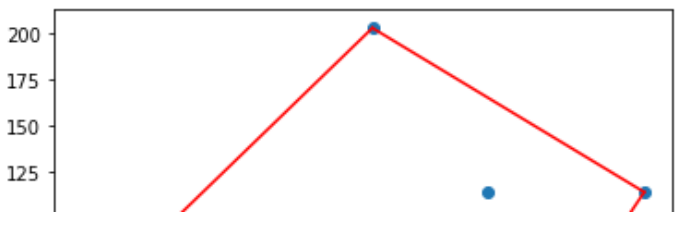


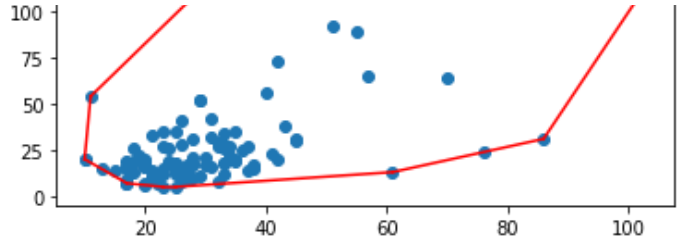
In [896]:

```
hull4 = gift_wrapping(pontos_1)
```

In [897]:

```
scatter_plot(pontos_1,hull4)
```



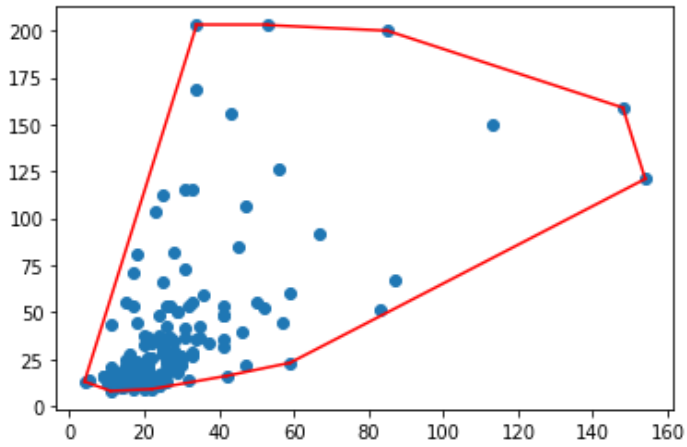


In [898]:

```
hull5 = gift_wrapping(pontos_2)
```

In [899]:

```
scatter_plot(pontos_2,hull5)
```



In [900]:

```
min_dist, min_dist_points = findMinDistance(hull4, hull5)
ponto_medio = findMidpoint(min_dist_points)
```

In [901]:

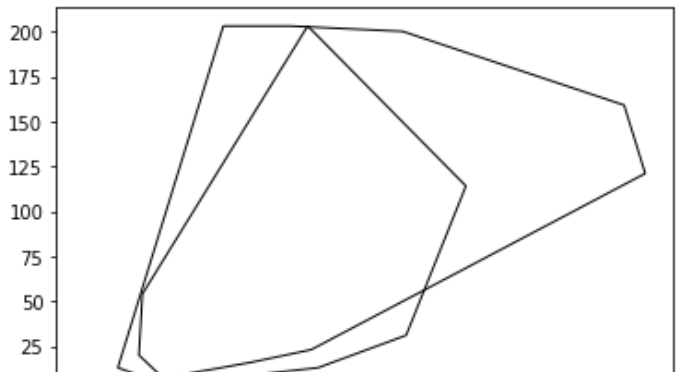
```
# Segmento da distância mínima
m, b = getMinLine(min_dist_points, ponto_medio)

# Reta perpendicular
perpendicular_line = findPerpendicularLine(m, b, ponto_medio)
```

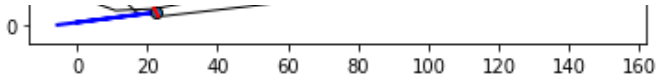
Como é possível ver na figura abaixo, uma das envoltórias está praticamente dentro da outra o que é um caso indesejado para a construção do nosso modelo classificador.

In [902]:

```
printEverything(hull4, hull5, min_dist_points, ponto_medio, perpendicular_line)
```

```
/usr/local/lib/python3.7/dist-packages/numpy/core/shape_base.py:65: VisibleDeprecationWar
ning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists
-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to d
o this, you must specify 'dtype=object' when creating the ndarray.
  ary = asanyarray(ary)
```





In [902]:

```
```

In [902]:

```
```

Análise: Base saheart

In [903]:

```
import pandas as pd
from google.colab import drive
import matplotlib.pyplot as plt
import itertools

arq2 =  pd.read_csv('./saheart.csv')
arq2
```

Out[903]:

	Sbp	Tobacco	Ldl	Adiposity	Famhist	Typea	Obesity	Alcohol	Age
160	12.00	5.73	23.11	Present	49	25.30	97.20	52	1
144	0.01	4.41	28.61	Absent	55	28.87	2.06	63	1
118	0.08	3.48	32.28	Present	52	29.14	3.81	46	0
170	7.50	6.41	38.03	Present	51	31.99	24.26	58	1
134	13.60	3.50	27.78	Present	60	25.99	57.34	49	1
...	...	...	...	...	...	...	...	...	...
214	0.40	5.98	31.72	Absent	64	28.45	0.00	58	0
182	4.20	4.41	32.10	Absent	52	28.61	18.72	52	1
108	3.00	1.59	15.23	Absent	40	20.09	26.64	55	0
118	5.40	11.61	30.79	Absent	64	27.35	23.97	40	0
132	0.00	4.82	33.41	Present	62	14.70	0.00	46	1

462 rows × 9 columns

Vamos separar a base em dados de treino e teste. O rótulo da classe indica se a pessoa tem doença cardíaca coronária: negativo (0) ou positivo (1).:

1. **Item da lista**
2. **Item da lista**

1 -> 0

2 -> 1

In [904]:

```
list(set(arq2["Age"].values))
```

Out[904]:

```
[0, 1]
```

Vamos estabelecer a analise para as duas classes disponiveis na base, indicando a presença da doença ou não.

In [905]:

```
print(len(arq2[arq2["Age"] == 0]))
print(len(arq2[arq2["Age"] == 1]))
```

```
302
160
```

Vamos escolher agora os atributos para que possamos representar a base de dados em duas dimensões, Analisando a base, decidimos escolher os seguintes atributos por observar uma certa relação.

Atributos representativos: Sbp e Tobacco

Ratificando que outras combinações de atributos foram testadas e os atributos selecionados foram os mais convinientes para seguir com os testes.

In [906]:

```
## Diminuindo a dimensão da base de dados para 2-D
arq3 = arq2[["Sbp", "Tobacco", "Age"]]
arq3
```

Out[906]:

	Sbp	Tobacco	Age
160	12.00	5.73	1
144	0.01	4.41	1
118	0.08	3.48	0
170	7.50	6.41	1
134	13.60	3.50	1
...	...	...	...
214	0.40	5.98	0
182	4.20	4.41	1
108	3.00	1.59	0
118	5.40	11.61	0
132	0.00	4.82	1

462 rows × 3 columns

Separando os dados entre conjunto de treino e conjunto de testes.Para todos os casos teste, vamos assumir a divisão 70-30, ou seja, 70% das instâncias foram utilizadas para treino, equanto que 30% foram utilizadas para teste.

In [907]:

```
training_data = arq3.sample(frac=0.7, random_state=25)
testing_data = arq3.drop(training_data.index)
```

Tamanho dos Dados de treino e teste

In [908]:

```
print(f"No. of training examples: {training_data.shape[0]}")
print(f"No. of testing examples: {testing_data.shape[0]}")
```

```
No. of training examples: 323
No. of testing examples: 8
```

In [909]:

```
pontos_1 = [] # 0
```

```
pontos_2 = [] # 1

for line in training_data.values:
  if (line[2] == 0):
    pontos_1.append([line[0], line[1]])

  if (line[2] == 1):
    pontos_2.append([line[0], line[1]])

pontos_1 = np.array(pontos_1)
pontos_2 = np.array(pontos_2)
```

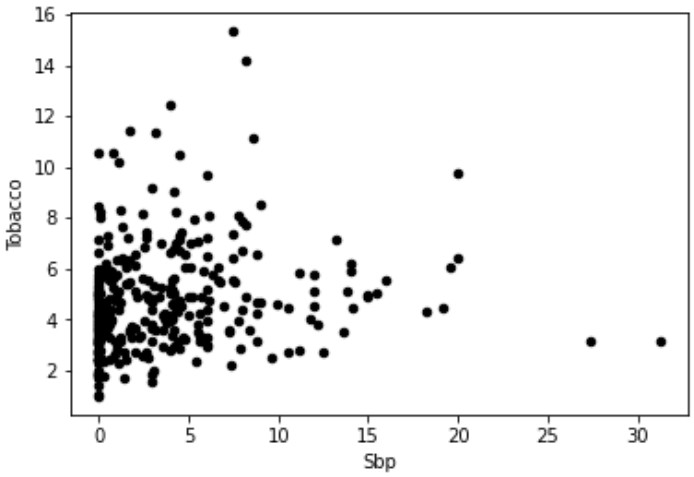
Plot dos pontos em relação aos atributos selecionados

In [910]:

```
training_data.plot.scatter(x='Sbp',y='Tobacco',c='black')
```

Out[910]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6896311890>
```

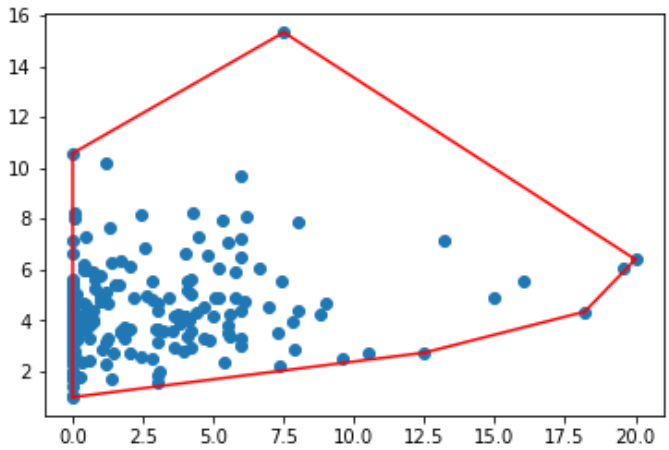


In [911]:

```
hull4 = gift_wrapping(pontos_1)
```

In [912]:

```
scatter_plot(pontos_1,hull4)
```

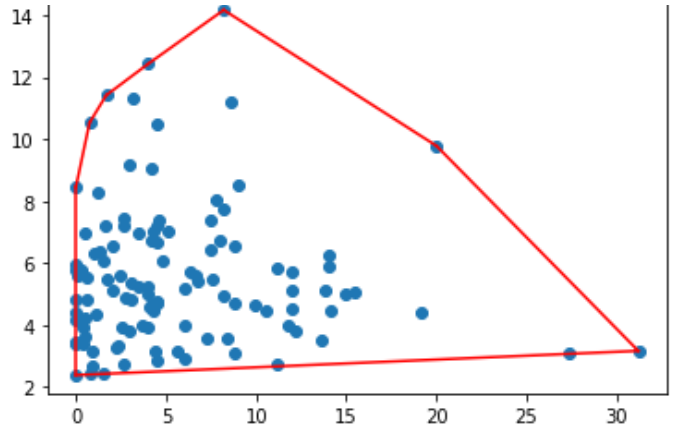


In [913]:

```
hull5 = gift_wrapping(pontos_2)
```

In [914]:

```
scatter_plot(pontos_2,hull5)
```



In [915]:

```
min_dist, min_dist_points = findMinDistance(hull4, hull5)
ponto_medio = findMidpoint(min_dist_points)
```

In [916]:

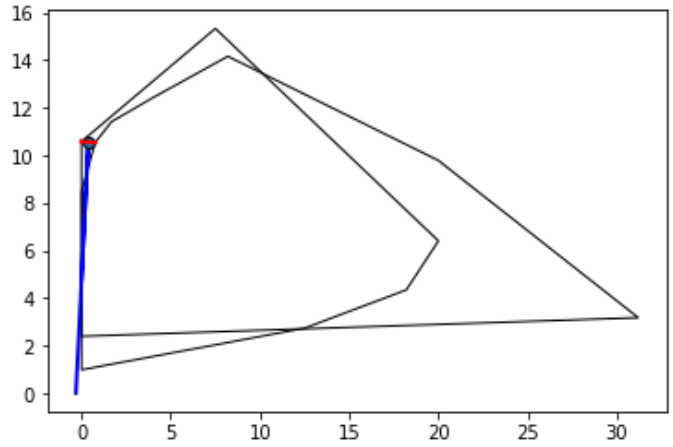
```
# Segmento da distância mínima
m, b = getMinLine(min_dist_points, ponto_medio)

# Reta perpendicular
perpendicular_line = findPerpendicularLine(m, b, ponto_medio)
```

Partindo do principio que não obtivermos uma separabilidade plena dos dados em relação aos atributos selecionados. Podemos afirmar que novamente, os pontos não comfiguram uma relação muito diferente do que fora estipulado, possuindo uma atribuição comum em entre os dados.Dessa forma, não conseguimos uma analise satisfatoria, em relação a classificação.

In [917]:

```
printEverything(hull4, hull5, min_dist_points, ponto_medio, perpendicular_line)
```

```
/usr/local/lib/python3.7/dist-packages/numpy/core/shape_base.py:65: VisibleDeprecationWar
ning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists
-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to d
o this, you must specify 'dtype=object' when creating the ndarray.
  ary = asanyarray(ary)
```



In [917]:

```

```

Análise: Base Ecoli

In [918]:

```
import pandas as pd
```

```
from google.colab import drive
import matplotlib.pyplot as plt
import itertools

arq2 =  pd.read_csv('./ecoli.csv')
arq2
```

Out[918]:

	Mcg	Gvh	Lip	Chg	Aac	Alm1	Alm2
49.0	29.0	48.0	5.0	56.0	24.0	35.0	cp
7.0	4.0	48.0	5.0	54.0	35.0	44.0	cp
56.0	4.0	48.0	5.0	49.0	37.0	46.0	cp
59.0	49.0	48.0	5.0	52.0	45.0	36.0	cp
23.0	32.0	48.0	5.0	55.0	25.0	35.0	cp
...	...	...	...	...	...	...	...
74.0	56.0	48.0	5.0	47.0	68.0	3.0	pp
71.0	57.0	48.0	5.0	48.0	35.0	32.0	pp
61.0	6.0	48.0	5.0	44.0	39.0	38.0	pp
59.0	61.0	48.0	5.0	42.0	42.0	37.0	pp
74.0	74.0	48.0	5.0	31.0	53.0	52.0	pp

336 rows × 7 columns

Vamos separar a base em dados de treino e teste. As classes nesse exemplo são dadas por:

1. **Item da lista**
2. **Item da lista**

1 -> cp

2 -> pp

In [919]:

```
list(set(arq2["Alm2"].values))
```

Out[919]:

```
['imL', 'om', 'pp', 'im', 'cp', 'omL', 'imU', 'imS']
```

Vamos estabelecer a analise para as duas classes disponiveis na base.

In [920]:

```
print(len(arq2[arq2["Alm2"] == 'cp']))
print(len(arq2[arq2["Alm2"] == 'pp']))
```

```
143
52
```

Vamos escolher agora os atributos para que possamos representar a base de dados em duas dimensões, Analisando a base, decidimos escolher os seguintes atributos por observar uma certa relação.

Atributos representativos: Mcg e Alm1

In [921]:

```
## Diminuindo a dimensão da base de dados para 2-D
arq3 = arq2[["Mcg", "Alm1", "Alm2"]]
arq3
```

Out[921]:

	Mcg	Alm1	Alm2	
49.0	29.0	35.0	cp
7.0	4.0	44.0	cp
56.0	4.0	46.0	cp
59.0	49.0	36.0	cp
23.0	32.0	35.0	cp
...	...	...	...
74.0	56.0	3.0	pp
71.0	57.0	32.0	pp
61.0	6.0	38.0	pp
59.0	61.0	37.0	pp
74.0	74.0	52.0	pp

336 rows × 3 columns

Separando os dados entre conjunto de treino e conjunto de testes.Para todos os casos teste, vamos assumir a divisão 70-30, ou seja, 70% das instâncias foram utilizadas para treino, equanto que 30% foram utilizadas para teste.

In [922]:

```
training_data = arq3.sample(frac=0.7, random_state=25)
testing_data = arq3.drop(training_data.index)
```

Tamanho dos Dados de treino e teste

In [923]:

```
print(f"No. of training examples: {training_data.shape[0]}")
print(f"No. of testing examples: {testing_data.shape[0]}")
```

```
No. of training examples: 235
No. of testing examples: 2
```

In [924]:

```
pontos_1 = [] # 0
pontos_2 = [] # 1

for line in training_data.values:
  if (line[2] == 'cp'):
    pontos_1.append([line[0], line[1]])

  if (line[2] == 'pp'):
    pontos_2.append([line[0], line[1]])

pontos_1 = np.array(pontos_1)
pontos_2 = np.array(pontos_2)
```

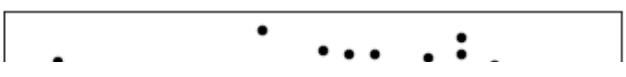
Plot dos pontos em relação aos atributos selecionados

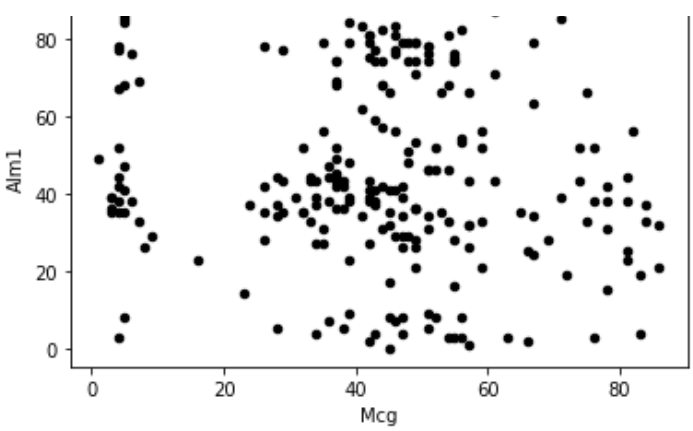
In [925]:

```
training_data.plot.scatter(x='Mcg',y='Alm1',c='black')
```

Out[925]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f68966b94d0>
```



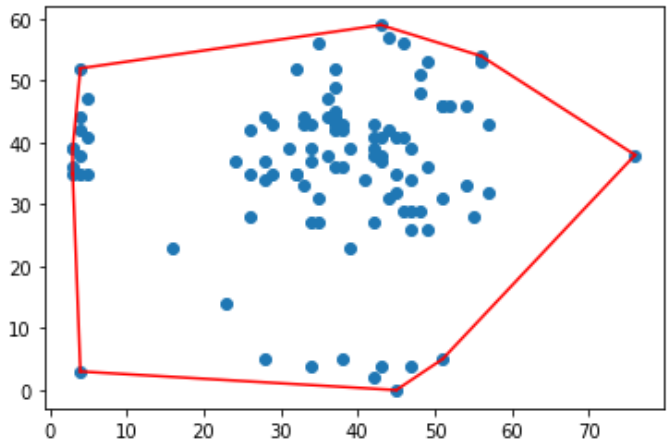


In [926]:

```
hull4 = gift_wrapping(pontos_1)
```

In [927]:

```
scatter_plot(pontos_1,hull4)
```

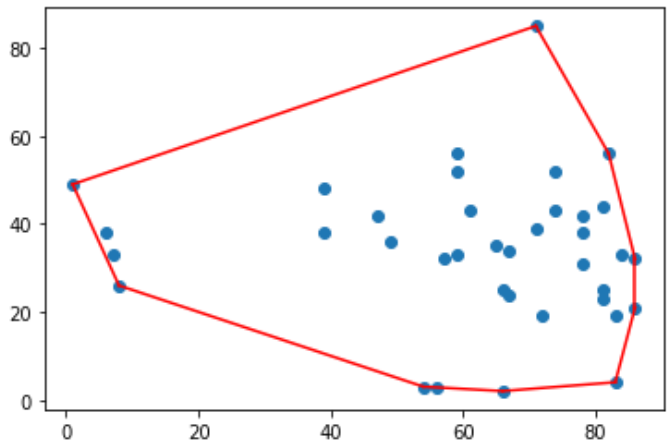


In [928]:

```
hull5 = gift_wrapping(pontos_2)
```

In [929]:

```
scatter_plot(pontos_2,hull5)
```



In [930]:

```
min_dist, min_dist_points = findMinDistance(hull4, hull5)
ponto_medio = findMidpoint(min_dist_points)
```

In [931]:

```
# Segmento da distância mínima
m, b = getMinLine(min_dist_points, ponto_medio)
```

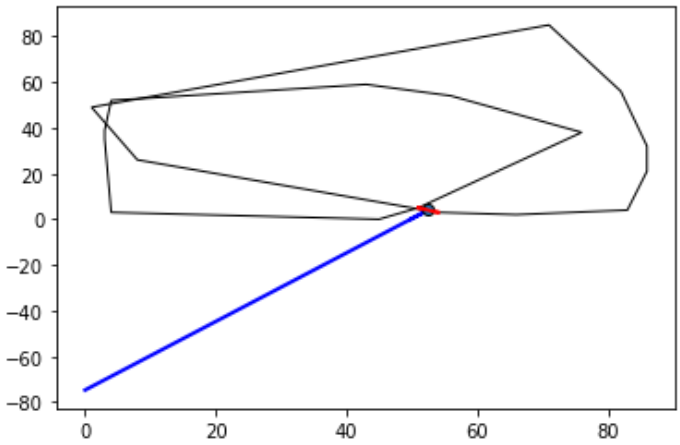
```
# Reta perpendicular
perpendicular_line = findPerpendicularLine(m, b, ponto_medio)
```

Podemos observar que os pontos que formam as duas involtorias formam um conjunto em comum de atribuições em relação aos atributos selecionados, uma vez que a reta perpenticular que traça a distancia media entre elas se perde um pouco no plot abaixo.

In [932]:

```
printEverything(hull4, hull5, min_dist_points, ponto_medio, perpendicular_line)
```

```
/usr/local/lib/python3.7/dist-packages/numpy/core/shape_base.py:65: VisibleDeprecationWar
ning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists
-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to d
o this, you must specify 'dtype=object' when creating the ndarray.
  ary = asanyarray(ary)
```



Conclusão

Este foi o modelo no qual obtivemos o meior sucesso. Sabemos que ele é um conjunto de dados canônico para estudos em machine learning, em especial, no contexto de classificação. Utilizando dessa base, conseguimos implementar um modelo bem interessante no qual conseguimos prever valores com um taxa considerável de acerto, claro, guardado o fato de que utilizamos de uma simplificação da base, afinal, nem todas as features foram utilizadas para descrever uma entrada para o problema, mas apenas duas. Ainda assim, este foi o modelo em que conseguimos o maior sucesso e que todo o "pipeline" de desenvolvimento do trabalho pode ser observado.

Para outras base de dados, não obtivemos o mesmo sucesso por diversos fatores, principalmente envolvendo a dificuldade em termos da separabilidade dos dados.

Realizamos testes com dezenas de bases de dados (e diversas combinações de atributos em cada uma), mas nenhuma delas apresentou desempenho tão bom quando o que pode ser observado com a base Iris. Neste trabalho se encontram nosso processo de criação do classificador para 5 bases, sendo que nas 4 últimos não conseguimos concluir o modelo pelos motivos citados à pouco. Resolvemos não completar a documentação com as 5 análises restantes visando não ser redundante, reiterando que, realizamos diversos testes sem sucesso com diferentes bases de dados.

In [932]:

```
```