**Laporan Tugas Kecil 2**

**Implementasi Convex Hull untuk Visualisasi Tes Linear Separability Dataset dengan Algoritma Divide and Conquer**

**IF2211 Strategi Algoritma**

Fawwaz Anugrah Wiradhika Dharmasatya – 13520086 – K02

1. Algoritma

Implementasi penyelesaian Convex Hull dengan algoritma Divide and Conquer ini diimplementasikan dalam bahasa Python. Program dimulai dengan memilih dan memuat dataset, baik dari dataset yang disediakan sklearn maupun dari file .csv di folder test.

Setelah data dimuat dan disimpan sebagai dataframe pandas, program akan membuat objek dari kelas MyConvexHull dengan parameter data tersebut. Kelas tersebut memiliki atribut, data yang dimasukkan, warna point dan garis bila diplotkan, dan list pasangan titik yang membentuk garis yang membentuk Convex Hull. Program lalu membentuk scatter plot dari data tersebut dan membentuk Convex Hull menggunakan list pasangan titik dari kelas tersebut.

Saat kelas MyConvexHull membuat list pasangan titik, kelas ini akan menggunakan metode dalam kelas tersebut untuk membentuk Convex Hull menggunakan algoritma Divide and Conquer.

Cara kerja algoritma pembentukan Convex Hull menggunakan algoritma Divide and Conquer adalah sebagai berikut:

1. Mengurutkan semua data pasangan titik berdasarkan sumbu-x. Jika ada yang sumbu-x nya sama, maka diurutkan menaik berdasarkan sumbu-y.
2. Memeriksa semua titik kecuali titik paling awal dan titik paling akhir menggunakan persamaan:

Dengan x1 dan y1 merupakan koordinat x dan y titik awal, x2 dan y2 merupakan koordinat x dan y titik akhir, serta x3 dan y3 merupakan koordinat x dan y titik yang diuji. Jika hasil persamaan bernilai positif, maka titik yang diuji berada di atas garis uji dan dimasukkan ke list titik yang berada di atas garis uji. Jika hasil persamaan bernilai negatif, maka titik yang diuji berada di bawah garis uji dan dimasukkan ke list titik yang berada di bawah garis uji.

1. Masukkan titik awal ke daftar titik yang membentuk Convex Hull
2. Periksa semua titik di upalarik bagian atas.
3. Cari titik terjauh di upalarik. Titik terjauh adalah titik dengan jarak terjauh dari garis yang dibentuk titik awal dan titik akhir. Jika ada yang jaraknya sama, pilih titik yang menyebabkan sudut antara titik awal, titik yang diperiksa, dan titik akhir yang lebih besar.
4. Periksa semua titik uji terhadap garis yang dibentuk titik awal dengan titik terjauh menggunakan persamaan nomor (1). Jika nilainya positif, maka titik tersebut berada di sebelah kiri garis dan dimasukkan ke sebuah upalarik. Jika nilainya negatif atau nol maka diabaikan.
5. Periksa semua titik uji terhadap garis yang dibentuk titik terjauh dengan titik akhir menggunakan persamaan nomor (1). Jika nilainya positif, maka titik tersebut berada di sebelah kanan garis dan dimasukkan ke sebuah upalarik. Jika nilainya negatif atau nol maka diabaikan.
6. Urut kedua upalarik, baik yang sebelah kiri atau kanan , secara menaik terbesar berdasarkan sumbu-x. Jika ada yang nilai-x nya sama, maka urutkan menaik berdasarkan sumbu-y.
7. Lakukan pemanggilan rekursif untuk memproses upalarik, baik yang kiri dan kanan. Langkah diulangi ke langkah nomor (5)
8. Masukkan titik terjauh, titik-titik pembentuk Convex Hull sebelah kiri, serta titik-tiitk pembentuk Convex Hull sebelah kanan ke daftar titik pembentuk Convex Hull sementara.
9. Setelah semua Convex Hull di bagian atas garis utama sudah diperoleh, urutkan menaik secara sumbu-x dan bila ada yang x nya sama maka urutkan berdasarkan sumbu-y secara menaik, lalu masukkan ke daftar titik pembentuk ConvexHull utama.
10. Tambahkan titik akhir garis utama ke daftar titik pembentuk ConvexHull.
11. Periksa semua titik di upalarik bagian bawah.
12. Cari titik terjauh di upalarik. Titik terjauh adalah titik dengan jarak terjauh dari garis yang dibentuk titik awal dan titik akhir. Jika ada yang jaraknya sama, pilih titik yang menyebabkan sudut antara titik awal, titik yang diperiksa, dan titik akhir yang lebih besar.
13. Periksa semua titik uji terhadap garis yang dibentuk titik awal dengan titik terjauh menggunakan persamaan nomor (1). Jika nilainya negatif, maka titik tersebut berada di sebelah kiri garis dan dimasukkan ke sebuah upalarik. Jika positif atau nol maka diabaikan.
14. Periksa semua titik uji terhadap garis yang dibentuk titik terjauh dengan titik akhir menggunakan persamaan nomor (1). Jika nilainya negatif, maka titik tersebut berada di sebelah kanan garis dan dimasukkan ke sebuah upalarik. Jika nilainya positif atau nol maka diabaikan.
15. Urut kedua upalarik, baik yang sebelah kiri atau kanan , secara menaik terbesar berdasarkan sumbu-x. Jika ada yang nilai-x nya sama, maka urutkan menaik berdasarkan sumbu-y.
16. Lakukan pemanggilan rekursif untuk memproses upalarik, baik yang kiri dan kanan. Langkah diulangi ke langkah nomor (14)
17. Masukkan titik terjauh, titik-titik pembentuk Convex Hull sebelah kiri, serta titik-tiitk pembentuk Convex Hull sebelah kanan ke daftar titik pembentuk Convex Hull sementara.
18. Setelah semua Convex Hull di bagian bawah garis utama sudah diperoleh, urutkan menurun secara sumbu-x dan bila ada yang x nya sama maka urutkan berdasarkan sumbu-y secara menurun, lalu masukkan ke daftar titik pembentuk ConvexHull utama.
19. Konversikan daftar titik tersebut menjadi sebuah list berisi pasangan titik yang bersebelahan di dalam ConvexHull.
20. Kode Program

Tabel 2.1. *Source Code* file main.py

|  |
| --- |
| #PROGRAM UTAMA  if \_\_name\_\_ == '\_\_main\_\_':      def getAxis(data):          #Mendapatkan index sumbu yang akan diproses          print("Daftar Kolom di Dataset")          for i in range(len(df.columns)):              print(i+1,".",df.columns[i])          while(True):              sumbu\_x = int(input("Pilih nomor kolom yang ingin menjadi sumbu-x: "))              if(sumbu\_x>=1 and sumbu\_x<=len(df.columns)):                  break              else:                  print("Pilihan tidak valid!")          while(True):              sumbu\_y = int(input("Pilih nomor kolom yang ingin menjadi sumbu-y: "))              if(sumbu\_y>=1 and sumbu\_y<=len(df.columns)and sumbu\_x!=sumbu\_y):                  break              else:                  print("Pilihan tidak valid!")          return (sumbu\_x-1,sumbu\_y-1)      import pandas as pd      import matplotlib.pyplot as plt      import myConvexHull as my      from sklearn import datasets      valid = False      data = None      print("Pilih sumber dari data yang ingin Anda masukkan(dalam angka): ")      print("1.Dataset sklearn")      print("2.CSV")      sumber\_data = int(input("Masukkan pilihan Anda: "))      if(sumber\_data==1):          while(not valid):              print("Pilih dataset yang diinginkan: ")              print("1.iris")              print("2.digits")              print("3.wine")              print("4.breast cancer")              pilihan = int(input("Masukkan pilihan Anda(dalam angka): "))              if(pilihan==1):                  data = datasets.load\_iris()                  valid = True              elif(pilihan==2):                  data = datasets.load\_digits()                  valid = True              elif(pilihan==3):                  data = datasets.load\_wine()                  valid = True              elif(pilihan==4):                  data = datasets.load\_breast\_cancer()                  valid = True              else:                  print("Pilihan Tidak Valid!")        elif(sumber\_data==2):          nama = input("Masukkan nama file(tanpa perlu menuliskan .csv) yang berada di folder test: ")          data = pd.read\_csv("../test/"+nama+".csv")          valid = True      else:          print("Pilihan tidak valid!\nKeluar program...")      if(valid):          df = pd.DataFrame(data.data, columns=data.feature\_names)          df['Target'] = pd.DataFrame(data.target)          print("Dimensi data: ",df.shape)          df.head()          #visualisasi hasil ConvexHull          plt.figure(figsize = (10, 6))          sumbu\_x,sumbu\_y = getAxis(df)          columns\_list = df.columns.values.tolist()          plt.title(columns\_list[sumbu\_y]+" vs "+columns\_list[sumbu\_x])          plt.xlabel(columns\_list[sumbu\_x])          plt.ylabel(columns\_list[sumbu\_y])          #Asumsi ada atribut target\_names          for i in range(len(data.target\_names)):              bucket = df[df['Target'] == i]              bucket = bucket.iloc[:,[sumbu\_x,sumbu\_y]].values              hull = my.MyConvexHull(bucket)              plt.scatter(bucket[:, 0], bucket[:, 1], label=data.target\_names[i],c=hull.color)              for simplex in hull.simplices:                  plt.plot(bucket[simplex, 0], bucket[simplex, 1], hull.color)          plt.legend()          plt.show() |

Tabel 2.2. *Source Code* file myConvexHull.py

|  |
| --- |
| import util as ut  import numpy as np  class MyConvexHull:      color\_idx = 0      colors = ['b','r','g','c','m','y','k','#7b3f00']##7b3f00->chocolate      def \_\_init\_\_(self,coordinates):          self.coordinates = coordinates          self.simplices = self.getConvexHull(coordinates)          self.color = MyConvexHull.colors[(MyConvexHull.color\_idx)%len(MyConvexHull.colors)]          MyConvexHull.color\_idx += 1      def sort(self,coordinates,axis=0):#0:x,1:y          #pake quicksort          if len(coordinates)>1:                  k,coordinates = self.partisi(coordinates)                  left = self.sort(coordinates[0:k+1])                  right = self.sort(coordinates[k+1:len(coordinates)])                  coordinates = np.concatenate((left,right))          return coordinates      def sorty(self,coordinates):          #sort berdasarkan y agar terurut membesar          if(len(coordinates)>1):              final\_array= []              subarr =[coordinates[0]]              for i in range(1,len(coordinates)):                  if(coordinates[i][0]==subarr[0][0]):                      #x nya sama                      subarr.append(coordinates[i])                  else:                      final\_array = final\_array +  self.sub\_sorty(subarr)                      subarr = [coordinates[i]]              if(subarr):                      final\_array = final\_array + self.sub\_sorty(subarr)              return final\_array          elif len(coordinates)==1:              return coordinates          else:              return []      def sub\_sorty(self,coordinates):          #sub fungsi untuk solve sort sumbu y          if len(coordinates)>1:                  k,coordinates = self.partisi(np.array(coordinates),1)                  coordinates = list(coordinates)                  left = self.sub\_sorty(coordinates[0:k+1])                  right = self.sub\_sorty(coordinates[k+1:len(coordinates)])                  coordinates = left +right          return coordinates      def partisi(self,coordinates,axis=0):#0->x,1->Y          #buat partisi larik          #sort dari x nya dulu          pivot\_idx = len(coordinates)//2          pivot = coordinates[pivot\_idx]          p = 0          q = len(coordinates)-1          while p<=q:              while p<len(coordinates):                  if(coordinates[p][axis] >= pivot[axis]):                      break                  p += 1              while q>=0:                  if coordinates[q][axis] <= pivot[axis]:                      break                  q -=1              if p<=q:                  coordinates[[p,q]] = coordinates[[q,p]]                  p+=1                  q-=1          return (q,coordinates)      def getConvexHull(self,coordinates):          #dapetin list convexhull          temp = np.array([[coordinates[i][0],coordinates[i][1],i] for i in range(len(coordinates))])          temp1 = self.sort(temp)          sorted\_coordinates = list(self.sorty(temp1))          start = sorted\_coordinates[0]          end = sorted\_coordinates[-1]          left = []          right = []          for i in range(1,len(sorted\_coordinates)-1):              check\_position = ut.checkPosition(start,end,sorted\_coordinates[i])              if(check\_position>0):                  #kalo positif berarti di kiri                  left.append(sorted\_coordinates[i])              elif(check\_position<0):                  #kalo negatif berarti di kanan                  right.append(sorted\_coordinates[i])              #kalo misal 0 berarti bisa diabaikan(karena segaris)          simplices  = [start]          simplices = simplices + list(self.sorty(self.sort(np.array(self.searchConvexHullUpper(left,start,end)))))          simplices.append(end)          convex\_kanan = self.searchConvexHullLower(right,start,end)          for i in range(len(convex\_kanan)-1,-1,-1):              simplices.append(convex\_kanan[i])          #simplices = simplices + convex\_kanan          simplices\_map = []          simplices = [list(simplice) for simplice in simplices]          for i in range(len(simplices)):              n = i              n1 = (i+1) % len(simplices)              map1  = [int(simplices[n][2]),int(simplices[n1][2])]              simplices\_map.append(map1)          simplices\_map = np.array(simplices\_map)          return simplices\_map      def searchConvexHullUpper(self,coordinates,start,end):          if(len(coordinates)>1):              #buat nyari di bagian atas              start\_point = start              end\_point = end              convex\_simplices = []              left = []              right = []              #cari titik terjauh              if(len(coordinates)>1 and not(start[0]==end[0] and start[1]!=end[1])):                  farthet\_point,idx = ut.getFarthestPoint(coordinates,start,end)                  convex\_simplices.append(farthet\_point)                  coordinates.pop(idx)                  for j in range(len(coordinates)):                      if(ut.checkPosition(start\_point,farthet\_point,coordinates[j])>0):                          left.append(coordinates[j])                      elif(ut.checkPosition(farthet\_point,end\_point,coordinates[j])>0):                          right.append(coordinates[j])                  left = list(self.sorty(self.sort(np.array(left))))                  right = list(self.sorty(self.sort(np.array(right))))                  left\_simplices = []                  right\_simplices = []                  if(len(left)>0):                      left\_simplices = self.searchConvexHullUpper(left,start,farthet\_point)                  if(len(right)>0):                      right\_simplices = self.searchConvexHullUpper(right,farthet\_point,end)                  convex\_simplices = convex\_simplices + left\_simplices + right\_simplices              else:                  convex\_simplices = convex\_simplices + coordinates              return convex\_simplices          elif(len(coordinates)==1):              return coordinates          else:              return []      def searchConvexHullLower(self,coordinates,start,end):          if(len(coordinates)>1):              #buat nyari di bagian bawah              start\_point = start              end\_point = end              convex\_simplices = []              left = []              right = []              #cari titik terjauh              if(len(coordinates)>1 and not(start[0]==end[0] and start[1]==end[1])):                  farthet\_point,idx = ut.getFarthestPoint(coordinates,start,end)                  convex\_simplices.append(farthet\_point)                  coordinates.pop(idx)                  for j in range(len(coordinates)):                      if(ut.checkPosition(start\_point,farthet\_point,coordinates[j])<0):                          left.append(coordinates[j])                      elif(ut.checkPosition(farthet\_point,end\_point,coordinates[j])<0):                          right.append(coordinates[j])                  left = list(self.sorty(self.sort(np.array(left))))                  right = list(self.sorty(self.sort(np.array(right))))                  left\_simplices = []                  right\_simplices = []                  if(len(left)>0):                      left\_simplices = self.searchConvexHullLower(left,start,farthet\_point)                  if(len(right)>0):                      right\_simplices = self.searchConvexHullLower(right,farthet\_point,end)                  convex\_simplices = convex\_simplices + left\_simplices + right\_simplices              else:                  convex\_simplices = convex\_simplices + coordinates              return list(self.sort(np.array(convex\_simplices)))          elif(len(coordinates)==1):              return coordinates          else:              return [] |

Tabel 2.3. *Source Code* util.py

|  |
| --- |
| from cmath import sqrt  import numpy as np  def checkPosition(p1,p2,p3):      #p1->titik paling kiri      #p2->titik paling kanan      #p3->titik yang ingin diuji      #[0]->x      #[1]->y      return p1[0]\*p2[1]+p3[0]\*p1[1]+p2[0]\*p3[1]-p3[0]\*p2[1]-p2[0]\*p1[1]-p1[0]\*p3[1]  def getDistance(p1,p2):      #mengembalikan jarak antara 2 titik      return sqrt((p1[0]-p2[0])\*\*2+(p1[1]-p2[1])\*\*2).real  def getDegree(p1,p2,pmain):      #mendapatkan sudut p1pmainp2 menggunakan aturan cosinus      c = getDistance(p1,p2)      b = getDistance(p1,pmain)      a = getDistance(p2,pmain)      cosx = (a\*\*2-b\*\*2-c\*\*2)/(-2\*b\*c)      return np.arccos(cosx)  def distanceFromLine(start\_point,end\_point,check\_point):      #menghitung jarak suatu titik dari garis yg dibentuk oleh start\_point dan end\_point      A = end\_point[1]-start\_point[1]      B = -(end\_point[0]-start\_point[0])      C = start\_point[1]\*(end\_point[0]-start\_point[0])-start\_point[0]\*(end\_point[1]-start\_point[1])      return abs(A\*check\_point[0]+B\*check\_point[1]+C)/(sqrt(A\*\*2+B\*\*2))  def getFarthestPoint(coordinates,start,end):          #mendapatkan titik terjauh          idx = 0          farthest\_point = coordinates[0]          farthest\_distance = distanceFromLine(start,end,coordinates[0])          biggest\_degree = getDegree(start,end,coordinates[0])          for i in range(len(coordinates)):              distance = distanceFromLine(start,end,coordinates[i])              degree=getDegree(start,end,coordinates[i])              if(distance>farthest\_distance):                  farthest\_point = coordinates[i]                  farthest\_distance = distance                  biggest\_degree = degree                  idx = i              elif(distance==farthest\_distance and degree>biggest\_degree):                  farthest\_point = coordinates[i]                  farthest\_distance = distance                  biggest\_degree = degree                  idx = i          return (farthest\_point,idx) |

1. *Screenshot Input-Output*
2. Alamat Kode Program