Aflevering 1: Clustering

Sune Lauth Gadegaard

Efterår 2019

I denne opgave tages udgangspunkt i datasættet `USArrests.csv`. Heri er angivet statistik for antallet af arrestationer per 100.000 indbyggere for overfald, mord og voldtægt i hver af de 50 stater i USA i 1973. Vi vil her antage, at de amerikanske stater skal inddeles i 5 grupper baseret på disse data.

Det er nu op til jer at besvare de følgende opgaver:

Opgave 1: Vælg mindst én metrik (afstandsmål), som skal bruges til at måle hvor ens to stater er (I kan blandt andet se på Euklidisk afstand og/eller Taxi-cap normen). Brug metrikken/metrikkerne til at udregne afstandsmatricen/afstandsmatricerne c_{ij} (se evt. afsnittet om lokationsbaseret clustering i noterne).

Opgave 2: Opstil det lokationsbaserede IP med sum-objektfunktion fra Clustering-noten for datasættet `USArrests.csv`. Implementer modellen i `Python` og løs det vha. en solver.

Opgave 3: Visualiser løsning. Dette kan for eksempel gøres ved at farve staterne i en cluster samme farve (se evt. online værktøjet her eller lignende).

Opgave 4: Opstil nu min-max udgaven af det lokationsbaserede IP fra noterne omkring Clustering og løs det tilhørende program. Visualiser ligeledes denne løsning.

Opgave 5: Opstil nu en model, der minimerer der største diameter over alle clustre. Implementer modellen i Pyomo og løs vha. en solver. Visualiser din løsning

Opgave 6: Kommenter på ligheder og uoverensstemmelser mellem de tre foregående løsninger. Hvad er forskellene og hvorfor ser vi disse forskelle (vi er ikke ude efter en socioøkonomisk fortolkning, men nærmere en forklaring på hvorfor ændringen af modellen ledte til disse nye løsning.)?

Jeres besvarelse af ovenstående opgaver skal sammenfattes i en lille rapport, som uploades i Peergrade inden afleveringsfristen. Sørg for at være grundige i jeres beskrivelser og analyser.

1 Vejledende besvarelse

Opgave 1

På forelæsningerne er en funktion til udregning af såkaldte L^p normer. Hvis to punkter $x \in \mathbb{R}^n$ og $y \in \mathbb{R}^n$ er givne, så kan man for et positivt p udregne ℓ_p-afstanden mellem disse punkter som

$$dist^p(y, y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}}$$

I tilfældet hvor $p = 2$ haves den såkaldte Euklidiske afstand mellem x og y og hvis $p = 1$ haves Taxi cap afstanden/Manhattan afstanden mellem x og y. Det er denne afstand, der ønskes udregnet for alle par af stater i datasættet `USarrests`.

For at opnå dette beskrives her kort data-formatet: Dataet er gemt i en Json fil med seks keys, nemlig `"states"`, `"Murder"`, `"Assault"`, `"UrbanPop"`, `"Rape"` og `"k"`. De fem første gemmer en liste med det tilsvarende data fra `USarrests`-datasættet mens `"k"` gemmer antallet af clustre, der skal oprettes i datasættet. Nedenfor er et overblik over datafilen

```
{
  "States":["Alabama", "Alaska", "Arizona", "Arkansas",...
  "Murder": [0.000132,0.0001,0.000081,0.000088,0.00009,...
  "Assault": [ 0.00236,0.00263,0.00294,0.0019,0.00276,...
  "UrbanPop": [0.58,0.48,0.8,0.5,0.91,0.78,0.77,0.72,...
  "Rape": [0.000212,0.000445,0.00031,0.000195,0.000406,...
  "k":5
}
```

For at beregne en afstandmatrix mellem de 50 staters data, skal vi være i stand til at beregne afstande mellem punkter som eksisterer i et 4D rum, altså punkter, der har fire koordinater. For eksempel er staten "Alabama" repræsenteret ved punktet (0.000132, 0.00236, 0.58, 0.000212) og "Alaska' er repræsenteret ved punktet (0.0001, 0.00263, 0.48, 0.000445). Dermed bliver den Euklidiske afstand mellem disse to stater

$$\begin{aligned} &dist^2(''Alabama'',''Alaska'') \\ &= \sqrt{(0.000132 - 0.0001)^2 + \cdots + (0.000212 - 0.000445)^2} \\ &= 0.100000641 \end{aligned}$$

Koden fra forelæsningen rettes således til, at de punkter der udregnes afstande imellem, nu har fire koordinater i stedet for de to. Dermed kommer den endelige kode til at se ud som følger

```
def makeLpNormDistanceMatrix(data: dict, p: int) -> list:
  points = np.column_stack((data['Murder'], data['Assault'],
                data['UrbanPop'], data['Rape']))
  nrPoints = len(data['Murder'])
  dist = []
  for i in range(0, nrPoints):
    dist.append([])
      for j in range(0, nrPoints):
        dist[i].append(np.linalg.norm(points[i] - points[j], p))
  return dist
```

Denne funktion tager en dictionary `data` og et heltal `p` som argumenter og returnerer en afstandmatrix mellem indgangene i dataet. Som det ses, defineres de punkter der skal måles afstande imellem først som firedimensionale punkter, hvorefter afstandende udregnes ved hjælp af `linalg.norm` funktionen fra `numpy` biblioteket.

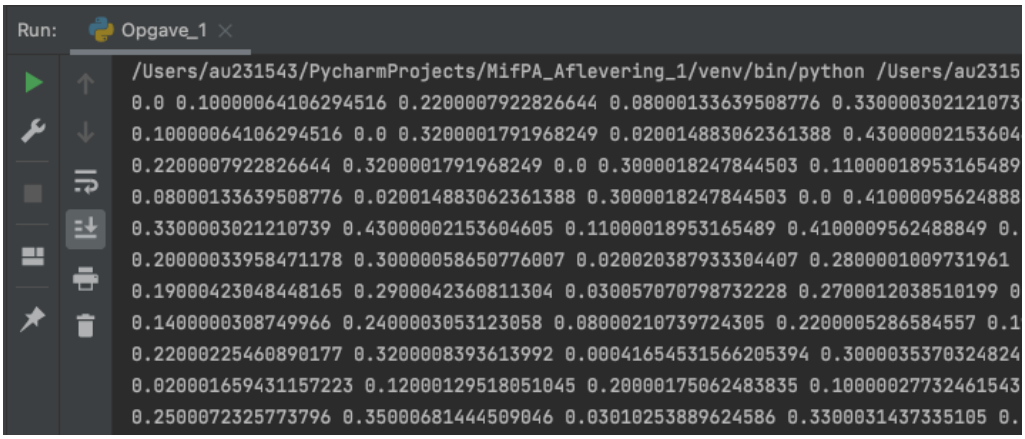
For at indlæse data og udregne afstandsmatricen implementeres en `readData` funktion som følger

```
def readData(filename: str) -> dict:
  # Read the data from a Json file
  data = rwJson.readJsonFileToDictionary(filename)
  # Print the keys available from the data
  print('Keys in data as read from file: ', rwJson.extractKeyNames(data))
  # Add a number to the dictionary holding the number of states
  # -> might come in handy later
  data['nrStates'] = len(data['States'])
  # Add a distance matrix to the dictionary.
  # Calculated using makeLpNormDistanceMatrix.
  data['dist'] = makeLpNormDistanceMatrix(data, 2)
  print('Keys in the data after adding:', rwJson.extractKeyNames(data))
  return data
```

Printes afstandmatricen hvor $p = 2$ til prompten fås et output ala det som er vist i Figur 1.

Opgave 2

I det følgende antages det, at $n = 50$ angiver antallet af stater i USA, og at $I = \{1, 2, \ldots, n\}$. Jævnfør forelæsningsnoterne kan et min-sum clustering problem over



Figur 1: Udsnit af afstandsmatricen når $p = 2$.

n punkter opskrives som følgende IP

$$
\begin{aligned}
\min \quad & \sum_{i \in I} \sum_{j \in I} c_{ij} x_{ij} && \\
\text{s.t.:} \quad & \sum_{i \in I} x_{ij} = 1, && \forall j \in I \\
& x_{ij} \leq y_i, && \forall i, j \in I \\
& \sum_{i \in I} y_i = p, && \\
& x_{ij} \in \{0, 1\}, && \forall i, j \in I \\
& y_i \in \{0, 1\}, && \forall i \in I
\end{aligned}
$$

Her er variablerne x_{ij} og y_i givet som

$$
y_i = \begin{cases} 1, & \text{hvis } i\text{'te stat repræsenterer et cluster} \\ 0, & \text{ellers} \end{cases}
$$

$$
x_{ij} = \begin{cases} 1, & \text{hvis den } j\text{'te stat er i et cluster, repræsenteret af den } i\text{'te stat} \\ 0, & \text{ellers} \end{cases}
$$

Objektfunktionskoefficienterne, c_{ij}, er et passende afstandsmål mellem staterne (se Opgave 1) og $k = 5$, da der skal etableres 5 grupper af staterne.

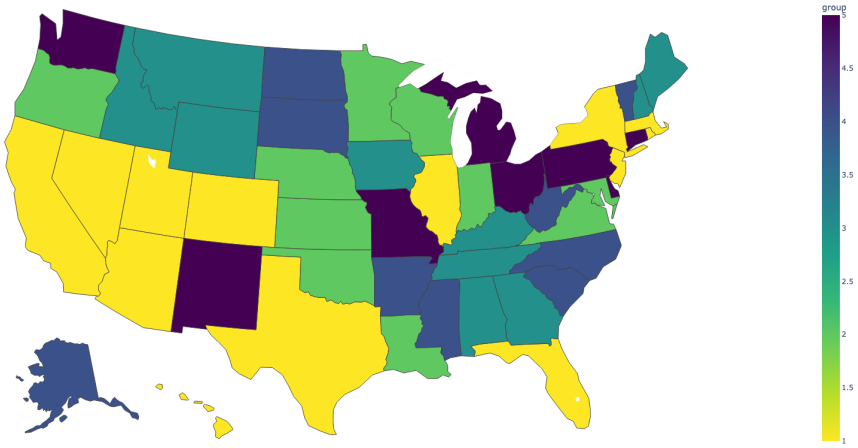
Implementeres denne model i `Python` og Pyomo kan det gøres på følgende måde:

```
def buildModel(data: dict) -> pyomo.ConcreteModel():
  # Define a model object
  model = pyomo.ConcreteModel()
  # Copy data to model object
  model.nrStates = data['nrStates']
  model.stateRange = range(0, model.nrStates)
  model.dist = data['dist']
  model.k = data['k']
  model.states = data['States']
  # Define variables
  model.y = pyomo.Var(model.stateRange, within=pyomo.Binary)
  model.x = pyomo.Var(model.stateRange, model.stateRange, within=pyomo.
      Binary)
  # Add the objective function
  model.obj = pyomo.Objective(expr=sum(model.dist[i][j]*model.x[i,j] for i
      in model.stateRange for j in model.stateRange))
  # Add "all represented" constraints
  model.allRep = pyomo.ConstraintList()
  for j in model.stateRange:
    model.allRep.add(expr=sum(model.x[i, j] for i in model.stateRange) ==
        1)
  # Add only represent if y[i]=1
  model.GUB = pyomo.ConstraintList()
  for i in model.stateRange:
    for j in model.stateRange:
      model.GUB.add(expr=model.x[i, j] <= model.y[i])
  # Add cardinality constraint on number of groups
  model.cardinality = pyomo.Constraint(expr=sum(model.y[i] for i in model.
      stateRange) == model.k)
  return model
```

Bruges Euklidisk afstand fås (der kan være alternative optimale løsninger) følgende løsning

Repræsentant	Resten
Kansas	Indiana, Louisiana, Maryland, Minnesota, Nebraska, Oklahoma, Oregon, Virginia, Wisconsin
New Hampshire	Alabama, Georgia, Idaho, Iowa, Kentucky, Maine, Montana, Tennessee, Wyoming
South Dakota	Alaska, Arkansas, Mississippi, North Carolina, North Dakota, South Carolina, Vermont, West Virginia
Washington	Connecticut, Delaware, Michigan, Missouri, New Mexico, Ohio, Pennsylvania
Illinois	Arizona, California, Colorado, Florida, Hawaii, Massachusetts, Nevada, New Jersey, New York, Rhode Island, Texas, Utah

Den optimale objektfunktionsværdi er $\approx 1.2978318537453606$ og den største dataafstand fra en stat til den stat som repræsenterer den er $\approx 0.1300005573526514$. Dette sker for Vermont som er repræsenteret af South Dakota.



Figur 2: Illustration af løsningen til min-sum clustering problemet.

Opgave 3

Løsningen kunne se ud som i Figur 2 Denne figur er produceret vha. et `Python` bibliotek kaldet `plotly-express`. En tutorial til at farve staterne i USA kan findes via dette link. Dette er dog ikke nødvendigt da man blot kan gøre det manuelt via det online værktøj, der er linket til i opgaven.

Opgave 4

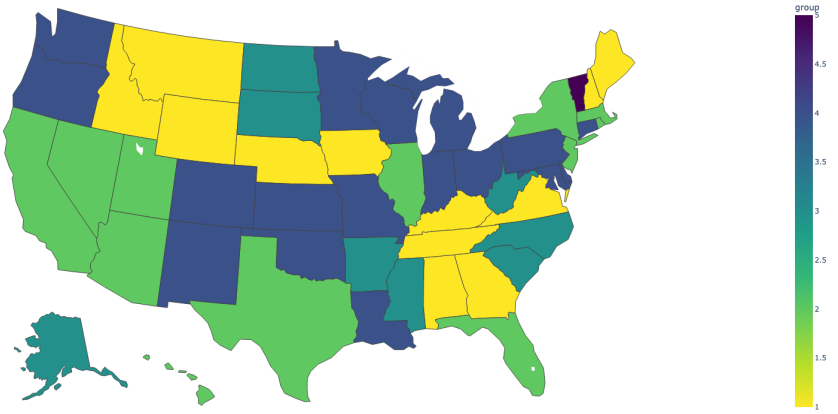
For at opstille min-max udgaven af problemet introduceres endnu en variabel, $\rho^{\max}$, som angiver den største afstand, der er mellem staterne og den stat de er repræsenteret af. Det nye min-max program bliver således

$$
\begin{aligned}
&\min\ \rho^{\max} \\
&\text{s.t.:}\ \sum_{i \in I} x_{ij} = 1, && \forall j \in I \\
&\quad x_{ij} \leq y_i, && \forall i, j \in I \\
&\quad \sum_{i \in I} y_i = p, \\
&\quad \sum_{i \in I} c_{ij} x_{ij} \leq \rho^{\max}, && \forall j \in I \\
&\quad x_{ij} \in \{0, 1\}, && \forall i, j \in I \\
&\quad y_i \in \{0, 1\}, && \forall i \in I
\end{aligned}
$$

og implementeringen i Python kan se ud som følger

```
def buildModel(data: dict) -> pyomo.ConcreteModel():
  # Define a model object
  model = pyomo.ConcreteModel()
  # Copy data to model object
  model.nrStates = data['nrStates']
  model.stateRange = range(0, model.nrStates)
  model.dist = data['dist']
  model.k = data['k']
  model.states = data['States']
  # Define variables
  model.y = pyomo.Var(model.stateRange, within=pyomo.Binary)
  model.x = pyomo.Var(model.stateRange, model.stateRange, within=pyomo.
      Binary)
  model.rhoMax = pyomo.Var(within=pyomo.NonNegativeReals)
  # Add the objective function
  model.obj = pyomo.Objective(expr=model.rhoMax)
  # Add "all represented" constraints
  model.allRep = pyomo.ConstraintList()
  for j in model.stateRange:
    model.allRep.add(expr=sum(model.x[i, j] for i in model.stateRange)==1)
  # Add "only represent if y[i]=1"
  model.GUB = pyomo.ConstraintList()
  for i in model.stateRange:
    for j in model.stateRange:
      model.GUB.add(expr=model.x[i, j] <= model.y[i])
  # Add the definition of the rhoMax variable
  model.rhoMaxDefinition = pyomo.ConstraintList()
  for j in model.stateRange:
    model.rhoMaxDefinition.add(expr=sum(model.dist[i][j]*model.x[i,j] for i
        in model.stateRange) <= model.rhoMax)
  # Ensure, that x[i][i]==y[i] in an optimal solution
  model.fixXandY = pyomo.ConstraintList()
  for i in model.stateRange:
    model.fixXandY.add(expr=model.x[i, i] == model.y[i])
  # Add cardinality constraint on number of groups
  model.cardinality = pyomo.Constraint(expr=sum(model.y[i] for i in model.
      stateRange) == model.k)
  return model
```

Bruges igen Euklidisk afstand fås (der kan være alternative optimale løsninger) følgende løsning



Figur 3: Illustration af løsningen til min-max clustering problemet.

Repræsentant	Resten
Iowa	Alabama, Georgia, Idaho, Kentucky, Maine, Montana, Nebraska, New Hampshire, Tennessee, Virginia, Wyoming
Massachusetts	Arizona, California, Florida, Hawaii, Illinois, Nevada, New Jersey, New York, Rhode Island, Texas, Utah
Mississippi	Alaska, Arkansas, North Carolina, North Dakota, South Carolina, South Dakota, West Virginia
Pennsylvania	Colorado, Connecticut, Delaware, Indiana, Kansas, Louisiana, Maryland, Michigan, Minnesota, Missouri, New Mexico, Ohio, Oklahoma, Oregon, Washington, Wisconsin
Vermont	Repræsenterer ikke andre end sig selv

Den største afstand mellem en stat og dens stat der repræsenterer den er nu nede på $\approx 0.0700000621571152$ (før $\approx 0.1300005573526514$) hvilket næsten er en halvering. Den totale distance internt i grupperne er steget til $\approx 1.6848968182342834$ (den var før $\approx 1.2978318537453606$) hvilket svarer til en stigning på cirka 30%. Læg også mærke til, at gruppernes størrelser nu er markant forskellige; den ene gruppe består kun af Vermont og ikke andre stater mens gruppen repræsenteret af Pennsylvania består af 17 stater.

En visualisering kan ses i Figur 3.

Opgave 5

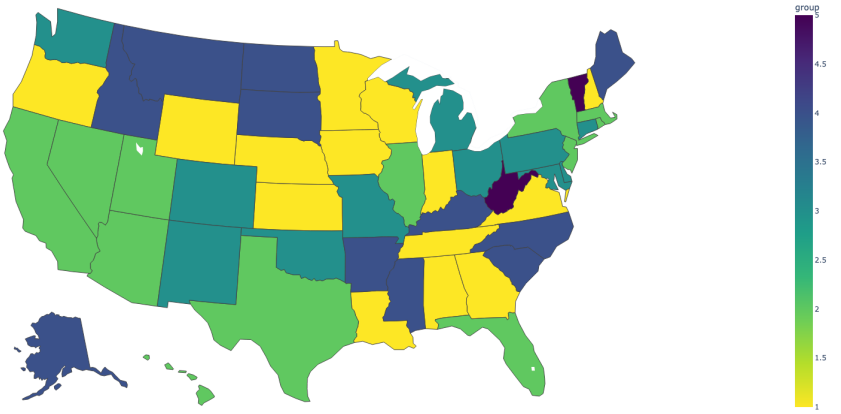
I denne opgave opstilles et clustering problem, der fokuserer på at minimere den største diameter over alle 5 clustre. Dette problem kan, jf. forelæsningsnoterne, opstilles som

følgende MILP

$$
\begin{aligned}
\min\ & D^{\max} \\
\text{s.t. } & D^{\max} \geq D_l, && \forall l = 1, \ldots, k \\
& D_l \geq c_{ij}\left(x_{il} + x_{jl} - 1\right), && \forall l = 1, \ldots, k,\ i, j = 1, \ldots, n : i \neq j \\
& D_l \geq 0, && \forall l = 1, \ldots, k \\
& \sum_{l=1}^{k} x_{il} = 1, && \forall i = 1, \ldots, n \\
& x_{il} \in \{0, 1\}, && \forall i = 1, \ldots, n,\ l = 1, \ldots, k
\end{aligned}
$$

Her er de binære variabler givet ved $x_{il} = 1$ hvis den i'te stat er i gruppe l og nul ellers. Variablen D_l angiver en øvre grænse for diameteren i de l'te cluster mens $D^{\max}$ antager værdien af den største diameter i en optimal løsning. Implementeringen af denne model i Pyomo kan se ud som følger

```
def buildModel(data: dict) -> pyomo.ConcreteModel():
  # Define a model object
  model = pyomo.ConcreteModel()
  # Copy data to model object
  model.nrStates = data['nrStates']
  model.stateRange = range(0, model.nrStates)
  model.dist = data['dist']
  model.k = data['k']
  model.groupRange = range(0, model.k)
  model.states = data['States']
  # Define variables
  model.D = pyomo.Var(model.groupRange, within=pyomo.NonNegativeReals)
  model.x = pyomo.Var(model.stateRange, model.groupRange, within=pyomo.
      Binary)
  model.Dmax = pyomo.Var(within=pyomo.NonNegativeReals)
  # Add the objective function
  model.obj = pyomo.Objective(expr=model.Dmax)
  # Define the Dmax variables correct value
  model.DmaxDefinition = pyomo.ConstraintList()
  for l in model.groupRange:
    model.DmaxDefinition.add(expr=model.D[l] <= model.Dmax)
  # Define the correct value for the D-variables
  model.DVarDefinition = pyomo.ConstraintList()
  for i in model.stateRange:
    for j in model.stateRange:
      for l in model.groupRange:
        model.DVarDefinition.add(
          expr=model.D[l]>=model.dist[i][j]*(model.x[i,l]+model.x[j,l] - 1)
          )
  # Make sure all states are in a group
  model.sumToOne = pyomo.ConstraintList()
  for i in model.stateRange:
    model.sumToOne.add(expr=sum(model.x[i,l] for l in model.groupRange)==1)
  return model
```



Figur 4: Illustration af løsningen fundet ved minimering af den største cluster diameter.

Bruges igen Euklidisk afstand fås (der kan være alternative optimale løsninger) følgende grupper

Gruppe	Resten
1	Alabama, Georgia, Indiana, Iowa, Kansas, Louisiana, Minnesota, Nebraska, New Hampshire, Oregon, Tennessee, Virginia, Wisconsin, Wyoming
2	Arizona, California, Florida, Hawaii, Illinois, Massachusetts, Nevada, New Jersey, New York, Rhode Island, Texas, Utah
3	Colorado, Connecticut, Delaware, Maryland, Michigan, Missouri, New Mexico, Ohio, Oklahoma, Pennsylvania, Washington
4	Alaska, Arkansas, Idaho, Kentucky, Maine, Mississippi, Montana, North Carolina, North Dakota, South Carolina, South Dakota
5	Vermont, West Virginia

Den største diameter er givet ved $\approx 0.11001121894152432$ og dette er dimeteren i gruppe 3. En illustration af løsning til dette problem kan ses i Figur 4.

Opgave 6

Der er mere eller mindre fri leg til at kommentere her!