Saviorganizuojantys neuroniniai tinklai

Pijus Petkevičius

2023 m. lapkričio 27 d.

Turinys

1 Įvadas

1.1 Tikslas

Užduoties tikslas - suprogramuoti saviorganizuojančio neuroninio tinklo (žemėlapio, SOM) mokymo algoritmą, apmokyti jį naudojant irisų duomenis, gauti mokymo aibės klasterizavimo rezultatą.

1.2 Uždaviniai

- Normalizuoti irisų duomenų aibės įeitis.
- Rasti SOM programinį kodą, jį pakeisti, kad tiktų irisų duomenims, apskaičiuotų kvantavimo paklaidą.
- Apmokyti SOM irisų duomenimis su skirtingais žemėlapiais: 5x5, 10x10.
- Gauti irisų duomenų klasterizavimo ir kvantavimo paklaidos rezultatus su skirtingais žemėlapio dydžiais: 5x5, 10x10.
- Klasterizavimo rezultatus atvaizduoti lentelėje.

2 Eksperimentų vykdymas

2.1 Duomenys

Šiam darbui naudoti irisų duomenys. Šiame rinkinyje yra 150 įrašų, viso yra 3 klasės - 50 įrašų kiekvienoje klasėje.

Šio tyrimo metu visi 150 įrašų buvo panaudoti mokymui, jie taip buvo panaudoti duomenų klasterizavimui.

2.2 Programos kodo šaltinis

Programinis kodas perrašytas pagal šį straipsnį(lengviau išmokti perrašant), jam buvo atlikti šie pakeitimai:

- Norėta turėti galimybę įvesti epochų skaičiaus parametrą, duomenų aibės elementai naudojami iš eilės, vienos epochos metu;
- Pridėtas kvantavimo paklaidos skaičiavimast;
- Atlikti pakeitimai, kad SOM atvaizdavimas turėtų irisų klases legendoje;
- Duomenų aibė nebeskaidoma į mokymo ir testavimo aibes.

2.3 Parametrai

2.3.1 Hiperparametrai

- žemėlapio dydis - 5x5 arba 10x10;
- epochų skaičius - 400;
- pradinis mokymo parametras, $\alpha(t) = 0{,}2$;
- pradinis Manheteno atstumas - 4.

3 Programos kodas

```
import numpy as np
from math import ceil
import matplotlib.pyplot as plt
from matplotlib import colors

numRows = 5                  # SOM lentelės eilučių skaičius
numCols = 5                 # SOM lentelės stulpelių skaičius
maxLearningRate = 0.2       # Pradinis mokymo parametras
epochs = 400                # Epochų skaičius
maxManhatanDistance = 4            # Pradinis Manheteno atstumas
fileName = 'iris.data'        # Duomenų failo pavadinimas
labels = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'] # Klasių pavadinimai

# Irisų duomenų aibėje, duomenų reikšmėms priskiriamas numeris,
# po to duomenys sumaišomi, paskutinis stulpelis(klasės reikšmė) perkeliama į atskirą masyva) ir išmetama.
def prepareIrisData(inputFile: str) -> (np.ndarray, [int]):
    outputFile= inputFile.replace('.data', '2.csv')
    with open(inputFile, 'r') as fileInput:
        with open(outputFile, 'w') as file:
            for row in fileInput:
                modifiedString = row
                for i in range(len(labels)):
                    modifiedString = modifiedString.replace(labels[i], str(i))
                file.write(modifiedString)
    inputData = readDataFromFile(outputFile, 0)
    np.random.shuffle(inputData)
    output = inputData[:, -1]
    inputData = np.delete(inputData, -1, axis=1)
    return inputData, output

```

```
31 def readDataFromFile(filename: str, skipRows: int = 1) ->
   ↪  np.ndarray:
32     return np.loadtxt(filename, delimiter=',',skiprows=skipRows)
33
34 def euclideanDistance(x: np.array, y: np.array) -> float:
35     return np.sqrt(np.sum(np.power(x - y, 2)))
36
37 def manhattanDistance(x: np.array, y: np.array) -> float:
38     return np.sum(np.abs(x - y))
39
40 def normalizeData(data: np.ndarray) -> np.ndarray:
41     return (data - np.min(data, axis=0) - np.max(data,axis=0)) /
       ↪  (np.max(data, axis=0) - np.min(data, axis=0))
42
43 # Neurono-nugalėtojo paieška
44 def winningNeuron(data, t, som, numRows, numCols):
45   winner = [0,0]
46   shortestDistance = np.sqrt(data.shape[1])  # Kintamasis
     ↪  pradžioje inicializuojamas didžiausiu įmanomu atstumu
47   for row in range(numRows):
48     for col in range(numCols):
49       distance = euclideanDistance(som[row][col], data[t])
50       if distance < shortestDistance:
51         shortestDistance = distance
52         winner = [row,col]
53   return winner
54
55 # Didėjant epochos skaičiui apskaičiuojamas vis mažesnis mokymo
   ↪  parametras ir
56 # didžiausias leistinas Manheteno atstumas, kuriuo kaimynai
   ↪  nutolę nuo
57 # neurono-nugalėtojo
58 def decay(step, epochs,learningRate,maxManhatanDistance):
59   coefficient = 1.0 - (np.float64(step)/epochs)
60   learningRate = coefficient * learningRate
61   neighbourhoodRange = ceil(coefficient * maxManhatanDistance)
62   return learningRate, neighbourhoodRange
63
64 def SOMTraining(irisData, epochs:int, maxManhatanDistance:int,
   ↪  numRows: int, numCols: int):
65   # Inicializuojama matrica su atsitikinėmis vektorių reikšmėmis
66   numDimensions = irisData.shape[1]
67   np.random.seed(40)
68   som = np.random.random_sample(size=(numRows, numCols,
     ↪  numDimensions))
69
```

```
70    for epoch in range(epochs):
71      learningRate, neighbourhoodRange = decay(epoch, epochs,
        ↪  maxLearningRate, maxManhatanDistance)
72      for t in range(irisData.shape[0]):
73        winner = np.array(winningNeuron(irisData, t, som, numRows,
          ↪  numCols)) # Randamas neuronas-nugalėtojas duotam
          ↪  vektoriui
74
75        # Atnaujinami neuronų svoriai
76        # Naudojama burbuliuko kaimynystės funkcija, kur
          ↪  atnaujinami tik į
77        # neurono-nugalėtojo kaimynus patenkančių vektorių svoriai
78        for row in range(numRows):
79          for col in range(numCols):
80            if manhattanDistance(np.array([row,col]),winner) <=
              ↪  neighbourhoodRange:
81              som[row][col] += learningRate * (irisData[t] -
                ↪  som[row][col])
82
83    return som
84
85  def drawMap(map, numRows, numCols):
86    # Kiekviename langelyje parenkama vaizduoti ta klasė, kuri
      ↪  pasikartoja
87    # dažniausiai tame langelyje
88    labelMap = np.zeros(shape=(numRows, numCols), dtype=np.int16)
89    for row in range(numRows):
90      for col in range(numCols):
91        labelList = map[row][col]
92        if len(labelList) == 0:
93          label = 3
94        else:
95          label = max(labelList, key=labelList.count)
96        labelMap[row][col] = label
97
98    cmap = colors.ListedColormap(['tab:blue', 'tab:green',
      ↪  'tab:red', 'w'])
99    colorMap = plt.imshow(labelMap, cmap=cmap)
100   dataLabels = labels.copy()
101   dataLabels.insert(len(dataLabels),'No class')
102   cbar = plt.colorbar(colorMap, ticks=range(4))
103   cbar.ax.set_yticklabels(dataLabels)
104
105   plt.show()
106
107 def getMapAndQuantization(irisData, irisOutput, som):
```

```
    # Apmokytas SOM priskiria kiekvieno įeities vektoriaus klasę žemėlapio vietai,
    # kurioje yra neuronas-nugalėtojas

  map = np.empty(shape=(numRows, numCols), dtype=object)
  quantizationError = 0

  for row in range(numRows):
    for col in range(numCols):
      map[row][col] = []

  for t in range(irisData.shape[0]):
    winner = winningNeuron(irisData, t, som, numRows, numCols)
    map[winner[0]][winner[1]].append(irisOutput[t])
    quantizationError += euclideanDistance(irisData[t], som[winner[0]][winner[1]])
  return map, quantizationError/irisData.shape[0]


irisData, irisOutput = prepareIrisData('iris.data')
irisData = normalizeData(irisData)
irisData = np.array(irisData)
irisOutput = np.array(irisOutput)
print (irisOutput)

som = SOMTraining(irisData, epochs, maxManhatanDistance, numRows, numCols)

# Apskaičiuojama kvantavimo paklaida
map, quantizationError = getMapAndQuantization(irisData, irisOutput, som)
print("Quantization error: ", quantizationError)

drawMap(map, numRows, numCols)
```

4 Rezultatai

4.1 Kvantavimo paklaida

1 lentelėje 10x10 žemėlapio kvantavimo paklaida 2 kartus mažesnė lyginant su 5x5 žemėlapiu. Galime teigti, kad didesniame, 10x10 žemėlapyje neuronai buvo geriau prisitaikę prie duomenų įeičių nei mažesniame, 5x5 žemėlapyje.

1 lentelė: Kvantavimo paklaida skirtingo dydžio žemėlapiuose

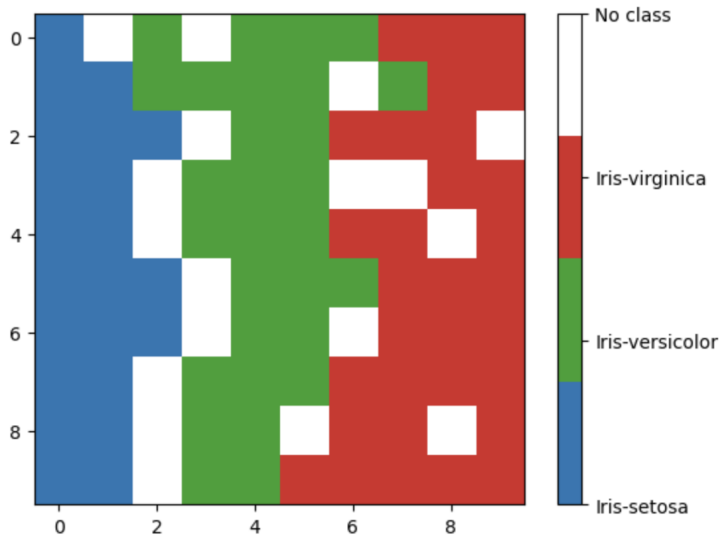
Žemėlapio dydis	**Kvantavimo paklaida**
5x5	0,1163
10x10	0,0653

4.2 SOM žemėlapiai

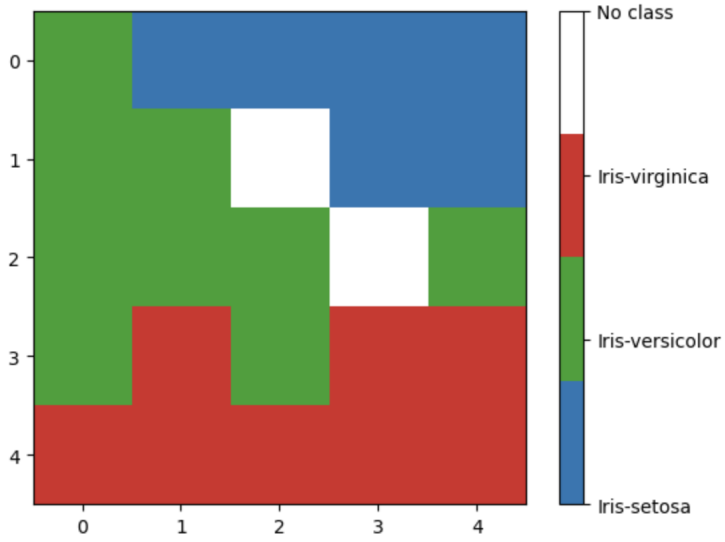
Žemėlapių langeliuose pasirinkta atvaizduoti tą klasę, kurios įeitys buvo dažniausiai priskirtos tam langeliui.

Tiek 1 pav., tiek 2 pav. Iris-versicolor ir Iris-virginica duomenys persikloja kur kas dažniau nei su Iris-setosa. 1 pav. galima aiškiau matyti nei 2 pav., kad kai kurios Iris-versicolor duomenų įeitys panašesnės į Iris-virginica nei į Iris-versicolor.

Tačiau Iris-versicolor klasė artima ir Iris-setosa, nuspalvinti langeliai arti vieni kitų. Labiausiai nutolusios yra Iris-setosa ir Iris-virginica klasės.



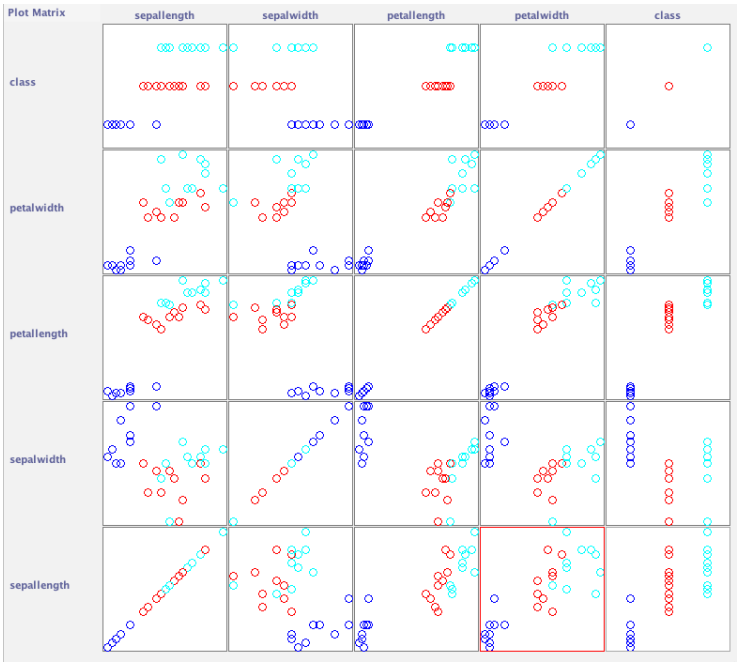
1 pav.: 10x10 žemėlapio klasterizavimo rezultatas



2 pav.: 5x5 žemėlapio klasterizavimo rezultatas

3 pav. pavaizduota mokymo ir testavimo aibės duomenų požymių porų vaizdai. Mėlyna spalva pažymėta Iris-setosa, raudona spalva Iris-versicolor ir žydra spalva Iris-Virginica.

3 pav. galime pastebėti, kad Iris-versicolor ir Iris-virginica požymių porų reikšmės ganėtinai artimos viena kitai, dažnai sutampa, o Iris-setosa klasės požymiai išsiskiria iš kitų dviejų klasių.



3 pav.: Irisų duomenų skirtingų klasių atributų reikšmės

5 Išvados

- 10x 10 žemelapis geriau išskiria irisų duomenis, lyginant su 5x5 žemėlapiu.
- Iris-virginica ir Iris-versicolor duomenys SOM išsidėsto, kartais net persikloja, nes jų duomenų atributai yra ganėtinai artimi.