Оценка качества классификации

При решении задачи кластеризации (классификации) для каждого объекта множества возможны 4 результата:

1. Объект отнесён к кластеру и действительно принадлежит этому кластеру - **True Positive.**
2. Объект отнесён к кластеру, но не принадлежит этому кластеру - **False Positive**.
3. Объект не отнесён к кластеру, но реально принадлежит этому кластеру - **True Negative.**
4. Объект не отнесён к кластеру и не принадлежит этому кластеру - **False Negative.**

Эти метрики для оценки качества кластеризации можно представить в виде таблицы:

|  |  |  |  |
| --- | --- | --- | --- |
| **Метрика** | | **Предсказанное состояние объекта** | |
| **в кластере (P)** | **не в кластере (N)** |
| **Реальное состояние объекта** | **в кластере (T)** | **True Positive** | **False Negative** |
| **не в кластере (N)** | **False Positive** | **True Negative** |

Для численных оценок качества системы обнаружения вторжений вводятся относительные величины:

***Sensitivity -***

- эта величина показывает отношение верно классифицированных объектов класса к общему числу элементов этого класса

***Specificity -***

- эта величина описывает ложные срабатывания системы; чем она меньше, тем лучше оценивается состояние системы.

Эти метрики для оценки качества кластеризации можно представить в виде таблицы:

|  |  |  |  |
| --- | --- | --- | --- |
| **Метрика** | | **Предсказанное состояние объекта** | |
| **в кластере** | **не в кластере** |
| **Реальное состояние объекта** | **в кластере** | **True Positive** | **False Negative** |
| **не в кластере** | **False Positive** | **True Negative** |

Идеальной системой считается та, для которой  и .

import pandas as pd

import numpy as np

from sklearn import preprocessing

import matplotlib.pyplot as plt

import random

from math import sqrt

#from math import sin

#from math import cos

iris = pd.read\_csv('../iris.csv')

print(iris.info(),"\n")

DIM\_N = 4

POINT\_N = iris.shape[0]

CLUST\_N = 3

targ = np.array(iris)[:, DIM\_N]

targ = preprocessing.LabelEncoder().fit\_transform(targ)

data = np.array(iris)[:, 0:DIM\_N]

"""

colors = ['blue', 'green', 'magenta', 'black']

fig, axes = plt.subplots(2, 3, figsize=(14, 8))

n = 0

for i in range(0, DIM\_N):

for j in range(i+1, DIM\_N):

ix = int(n/3)

iy = int(n%3)

for k in range(0, POINT\_N):

axes[ix][iy].scatter(data[k][i], data[k][j], c=colors[targ[k]], s=100)

axes[ix][iy].set\_xlabel('$Axis: ('+str(i)+', '+str(j)+')$', fontsize=10)

axes[ix][iy].set\_xticks([])

axes[ix][iy].set\_yticks([])

n += 1

fig.tight\_layout()

plt.show()

"""

class POINT:

def \_\_init\_\_(self, cl, x):

self.clust = cl

self.X = x

class CLUSTER(POINT):

def \_\_init\_\_(self, cl, x):

POINT.\_\_init\_\_(self, cl, x)

self.N = 0

def Dist(self, p):

d2 = 0.0

for i in range(DIM\_N):

d2 += (self.X[i] - p.X[i])\*\*2

return sqrt( d2 )

def Eval\_Center(self, P):

self.N = 0

self.X = [0.0 for i in range(DIM\_N)]

for p in P:

if p.clust == self.clust:

self.N += 1

for i in range (DIM\_N):

self.X[i] += p.X[i]

for i in range(DIM\_N):

self.X[i] /= self.N

#random.seed(5)

random.seed(0)

Cl = [CLUSTER(i, [4.0\*random.random()+2.0 for j in range(DIM\_N)]) for i in range(CLUST\_N)]

PP = [POINT(random.randint(0,2), data[i,:]) for i in range(POINT\_N)]

while (1):

CC = [POINT(0, Cl[i].X) for i in range(CLUST\_N)]

for cl in Cl:

cl.Eval\_Center(PP)

for p in PP:

dd = [Cl[i].Dist(p) for i in range(CLUST\_N)]

p.clust = np.argmin (dd)

"""

colors = ['blue', 'green', 'magenta', 'black']

for p in PP:

plt.scatter(p.X[1], p.X[2], c=colors[p.clust], s=20)

for cl in Cl:

plt.scatter(CC[cl.clust].X[1], CC[cl.clust].X[2], c='cyan', marker="\*")

plt.scatter(cl.X[1], cl.X[2], c='red', marker="\*")

plt.show()

"""

d = 0.0

for i in range(CLUST\_N):

d += Cl[i].Dist(CC[i])

if d < 0.1\*CLUST\_N:

break

TP = 0

FP = 0

TN = 0

FN = 0

# Cluster number 1

for i in range(POINT\_N):

if PP[i].clust == 1:

if targ[i] == 1:

TP += 1

else:

FP +=1

else:

if targ[i] == 1:

FN += 1

else:

TN +=1

print ("TP=",TP,"FP=",FP,"FN=",FN,"TN=",TN)

print ("TPR=",TP/(TP+FN),"FNR=",FN/(TN+FP))

Модифицируем алгоритм K-means. Введём радиус (границу) кластера, объект принадлежит кластеру с ближайшим центром (как было ранее) и расстояние до центра меньше радиуса кластера (объект находится внутри кластера относительно его границы).

Центр кластера определяется как среднее арифметическое координат входящих в него точек:

здесь s обозначает координату в пространстве характеристик

"Радиус" кластера может быть определён как разброс точек от центра кластера:

n=1 -> 68%

n=2 -> 95%

n=3 -> 98.5%

Для учёта разброса характеристик относительно центра кластера удобно использовать ковариационную матрицу:

Обратная матрица

используется при определении расстояния до центра кластера в метрике Махаланобиса:

Метрика Махаланобиса может быть использована для кластеров вытянутой, эллипсоидной формы.

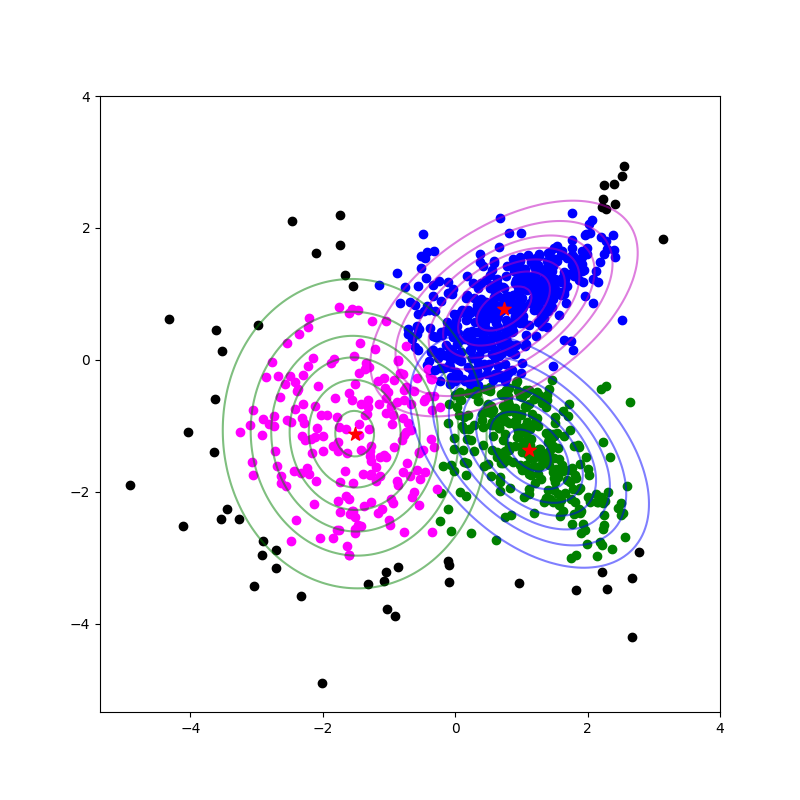
Ковариационная матрица *S* и обратная матрица *R* рассчитываются для конкретного кластера. Условие

определяет границу кластера на расстоянии *n* стандартных отклонений от его центра в предположении что кластер имеет форму эллипсоида.

Условием вхождения точки в кластер будет её локализация в указанной границе кластера. В случае нормального распределения около 68% точек находятся в пределах одного стандартного отклонения (n=1) от центра кластера, примерно 95 % точек лежат расстоянии не более двух стандартных отклонений (т=2); и 99,7 % не более трёх (n=3).

Изображение выглядит как диаграмма

Автоматически созданное описание



Изображение выглядит как диаграмма

Автоматически созданное описание

На рисунках приведены результаты кластеризации для одного, двух и трёх стандартных отклонений.

Перейдём к оценке результатов кластеризации. Выберем один из кластеров (далее, для определённости это будет 1-й кластер, на рисунке его точки обозначены фиолетовым цветом). Подсчитаем число точек, действительно принадлежащих 1-му кластеру, которые мы идентифицировали как точки 1-го кластера. Эта величина называется True Positive (TP). Отношение TP к полному числу точек, принадлежащих 1-му кластеру

называется True Positive Rate (TPR). Иногда величину TPR называют Sensitivity.

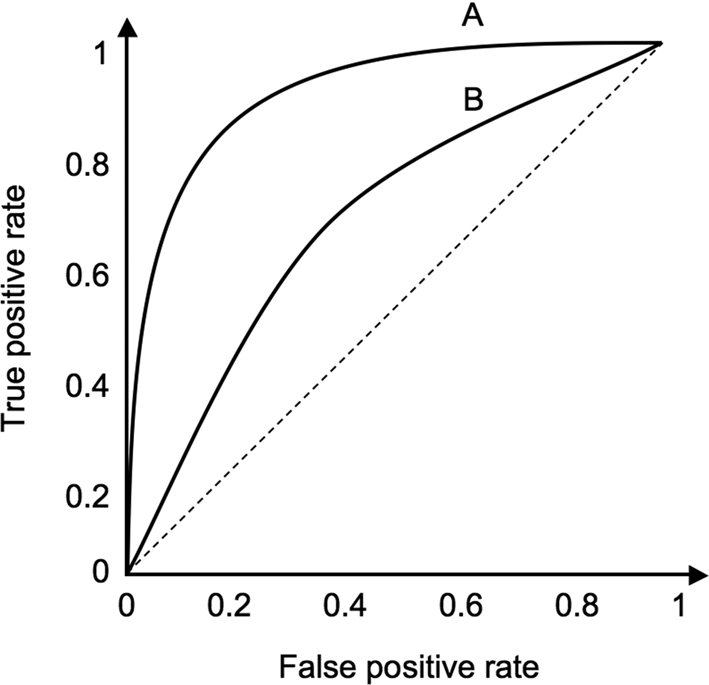
Аналогично, подсчитаем число точек, которые мы идентифицировали как точки 1-го кластера, но на самом деле не принадлежащих 1-му кластеру, (ошибочная идентификация). Эта величина называется False Positive (FP). Отношение FP к числу точек, не принадлежащих 1-му кластеру

называется False Positive Rate (FPR). FPR связана с понятием Specificity: Specificity=1 - FPR.

Величины TPR и FPR – взаимосвязаны. Увеличивая доверительный интервал (число стандартных отклонений) при определении размера кластера, мы увеличиваем TP – число точек, отнесённых к этому кластеру, тем самым увеличивается величина TPR. Но при этом увеличивается FP – количество точек, ошибочно отнесённых к кластеру и, как следствие, растёт FPR.

Число стандартных отклонений задаёт порог, согласно которому точка может быть включена или не включена в кластер. Для алгоритмов кластеризации отличных от K-means пороговое значение может определяться другим параметром кластеризации. (пример Fuzzy clustering).

Рассмотрим совместное изменение FPR и TPR на графике в соответствии с изменением параметра порога (доверительного интервала в рассматриваемом примере):



Этот график называется Receiver Operating Characteristic или ROC-кривые и позволяет судить о качестве метода классификации.

(Пунктирная линия – случайный выбор результата и равное число точек в принадлежащих и не принадлежащих кластеру)

Пример программной реализации:

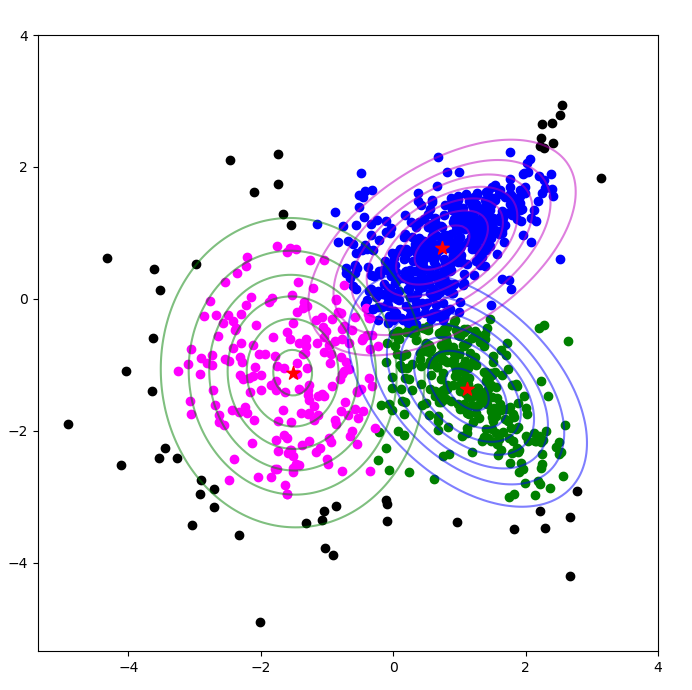
sklearn

* генерация набора данных: sklearn.datasets.make\_classification()
* тренировочный и тестовый набор данных: sklearn.model\_selection.train\_test\_split()
* выбор модели и результат, например:  
  model = KNeighborsClassifier(n\_neighbors=5)  
  model.fit (data\_train, targ\_train)

ИсходныйИзображение выглядит как Красочность, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

*Исходный набор данных.*



«Доверительные границы» кластеров

Изображение выглядит как диаграмма, линия, График

Контент, сгенерированный ИИ, может содержать ошибки.

Красный – метод ближайших соседей (sklearn)

Зелёный – Логистическая регрессия (sklearn)

Красный – K-means (собственная реализация)

import numpy as np

import matplotlib.pyplot as plt

from sklearn import datasets

from sklearn.model\_selection import train\_test\_split

POINT\_N = 1000

DIM\_N = 2

CLUST\_N = 3

COLORS = ['blue', 'green', 'magenta', 'black']

D = 2.0 # D\*sigma = Confidence interval

data, targ = datasets.make\_classification (n\_samples= POINT\_N,

n\_features=DIM\_N,

n\_informative=DIM\_N,

n\_redundant=0,

n\_classes=CLUST\_N, n\_clusters\_per\_class=1,

class\_sep = 1.0,

random\_state = 1 )

plt.figure(figsize=(8,8)) # creating the figure and assigning the size

for i in range(POINT\_N):

if targ[i] == 2:

plt.scatter(data[i,0], data[i,1], c='magenta', marker='o')

else:

plt.scatter(data[i,0], data[i,1], c='black', marker='o')

plt.show()

class CLUST:

def \_\_init\_\_(self, x, y, cl):

self.clust = cl

self.N = 0

self.X = x

self.Y = y

self.Sxx = 1.0

self.Syy = 1.0

self.Sxy = 0.0

self.Rxx = 1.0

self.Ryy = 1.0

self.Rxy = 0.0

def Dist2(self, x, y):

return self.Rxx\*(self.X - x)\*\*2 + self.Ryy\*(self.Y - y)\*\*2 + 2.0 \* self.Rxy\*(self.X - x)\*(self.Y - y)

def Eval(self, T):

self.N = 0

self.X = 0.0

self.Y = 0.0

self.Sxx = 0.0

self.Syy = 0.0

self.Sxy = 0.0

for i in range(0, POINT\_N):

if T[i] == self.clust:

self.N += 1

self.X += data[i,0]

self.Y += data[i,1]

self.Sxx += data[i,0]\*data[i,0]

self.Syy += data[i,1]\*data[i,1]

self.Sxy += data[i,0]\*data[i,1]

self.X /= self.N

self.Y /= self.N

if self.N > 2:

self.Sxx = self.Sxx/self.N - self.X\*self.X

self.Syy = self.Syy/self.N - self.Y\*self.Y

self.Sxy = self.Sxy/self.N - self.X\*self.Y

d = self.Sxx\*self.Syy - self.Sxy\*self.Sxy

self.RSxx = self.Syy/d

self.RSyy = self.Sxx/d

self.RSxy = -self.Sxy/d

else:

self.Sxx = 1.0

self.Syy = 1.0

self.Sxy = 0.0

self.Rxx = 1.0

self.Ryy = 1.0

self.Rxy = 0.0

Clust = [CLUST(-4.0, 0.0, 0), CLUST(2.0, 2.0, 1), CLUST(2.0, -2.0, 2)]

for cl in Clust:

cl.Eval(targ)

pred = [CLUST\_N for i in range(0, POINT\_N)] # first step

for i in range(0, POINT\_N):

c0 = POINT\_N

d0 = 1.0e+100

for c in Clust:

d = c.Dist2(data[i][0], data[i][1])

if d < d0:

d0 = d

pred[i] = c.clust

def Pred(K):

z0 = 1.0e+100

while True:

CC = [[Clust[i].X, Clust[i].Y] for i in range(0, CLUST\_N)]

for cl in Clust:

cl.Eval(pred)

for i in range(0, POINT\_N):

pred[i] = CLUST\_N

c0 = POINT\_N

d0 = 1.0e+100

for c in Clust:

d = c.Dist2(data[i][0], data[i][1])

if d < K\*K and d < d0:

d0 = d

pred[i] = c.clust

z0 = 0.0

for i in range(0, CLUST\_N):

z0 += Clust[i].Dist2(CC[i][0], CC[i][1])

if z0 < 1.0e-4:

break

Pred(D)

# Show results

np\_df = np.array(data)

m1 = [Clust[0].X, Clust[0].Y]

m2 = [Clust[1].X, Clust[1].Y]

m3 = [Clust[2].X, Clust[2].Y]

cov1 = [[D\*Clust[0].Sxx, D\*Clust[0].Sxy], [D\*Clust[0].Sxy, D\*Clust[0].Syy]]

cov2 = [[D\*Clust[1].Sxx, D\*Clust[1].Sxy], [D\*Clust[1].Sxy, D\*Clust[1].Syy]]

cov3 = [[D\*Clust[2].Sxx, D\*Clust[2].Sxy], [D\*Clust[2].Sxy, D\*Clust[2].Syy]]

x1 = np.linspace(-4,4,150)

x2 = np.linspace(-4,4,150)

X, Y = np.meshgrid(x1,x2)

from scipy.stats import multivariate\_normal

Z1 = multivariate\_normal(m1, cov1)

Z2 = multivariate\_normal(m2, cov2)

Z3 = multivariate\_normal(m3, cov3)

pos = np.empty(X.shape + (2,)) # a new array of given shape and type, without initializing entries

pos[:, :, 0] = X; pos[:, :, 1] = Y

plt.figure(figsize=(8,8))

for i in range(0, POINT\_N):

plt.scatter(data[i,0], data[i,1], c=COLORS[pred[i]], marker='o')

for cl in Clust:

plt.scatter(cl.X, cl.Y, c='red', marker="\*", s=100)

plt.contour(X, Y, Z1.pdf(pos), colors="m" ,alpha = 0.5)

plt.contour(X, Y, Z2.pdf(pos), colors="b" ,alpha = 0.5)

plt.contour(X, Y, Z3.pdf(pos), colors="g" ,alpha = 0.5)

plt.axis('equal') # making both the axis equal

plt.show()

#----------------------------------------------------------

# Receiver operating characteristic - ROC (BLUE)

cl = 0

pos = sum(targ==cl)

r = [[0.0, 0.0]]

tpr = [0.0]

fpr = [0.0]

for k in [0.3, 0.6, 1.0, 1.5, 2.0]:

Pred(k)

t = 0

f = 0

for i in range(0, POINT\_N):

t += (pred[i]==cl and targ[i]==cl)

f += (pred[i]==cl and targ[i]!=cl)

tpr.append(float(t)/pos)

fpr.append(float(f)/(POINT\_N-pos))

tpr.append(1.0)

fpr.append(1.0)

plt.figure(figsize=(8,8))

plt.plot(fpr, tpr)

plt.xlabel('fpr', fontsize=16) # X-Axis

plt.ylabel('tpr', fontsize=16)

#plt.show()

#--------------------------------------------------------------

# ROC for K-nearest (RED)

from sklearn.neighbors import KNeighborsClassifier

from sklearn import metrics

for i in range(0, POINT\_N):

targ[i] = (targ[i] if targ[i]<2 else 1)

data\_train, data\_test, targ\_train, targ\_test = train\_test\_split(data, targ, test\_size=.3 ,random\_state=0)

model = KNeighborsClassifier(n\_neighbors=5)

model.fit (data\_train, targ\_train)

targ\_pred = model.predict\_proba(data\_test)[:, 1]

model.score(data\_test,targ\_test)

fpr, tpr, thr = metrics.roc\_curve (targ\_test, targ\_pred)

plt.plot (fpr,tpr)

#plt.show()

#-------------------------------------------------------------------

# ROC for LogisticRegression (GREEN)

from sklearn. linear\_model import LogisticRegression

from sklearn import metrics

from sklearn. ensemble import GradientBoostingClassifier

model = LogisticRegression()

model.fit (data\_train, targ\_train)

targ\_pred = model.predict\_proba(data\_test)[:, 1]

fpr, tpr, thr = metrics.roc\_curve (targ\_test, targ\_pred)

plt.plot (fpr,tpr)

plt.show()