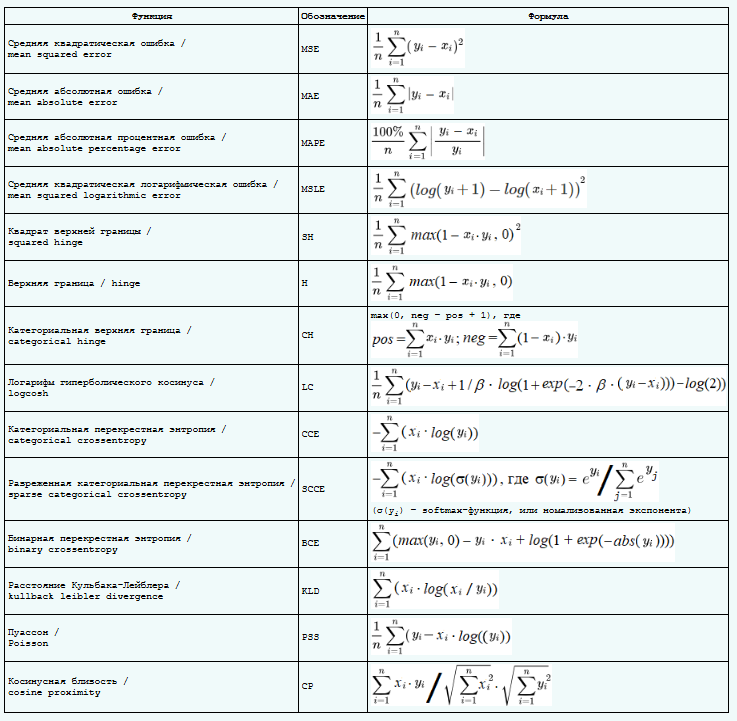
Кластеризация данных, целевая функция

***- Обычный подход***

Oбучение сети сводится к подбору параметров сети (весовых коэффициентов *w* и смещений *b* для нейронов сети). Параметры сети подбираются так, чтобы минимизировать отклонение результата работы сети от контрольного набора данных:

Функцию обычно называют функцией потерь или функцией ошибок. В качестве такой функции можно использовать сумму квадратичных отклонений

или какую либо другую функцию:



Существенным моментом здесь является использование известного контрольного набора данных. В этом случае задача обработки данных сводится к классификации данных (обучение с учителем в терминологии машинного обучения).

Кластеризации данных (обучение без учителя) не требуется контрольный набор. Кластеризация данных выявляет и использует внутреннюю структуру набора данных и на основе этого осуществляет фрагментацию данных.

Применение нейросетей и использование стандартных библиотек (Keras, tensorflow, …) для кластеризации данных можно свести к подбору соответствующей функции, минимизация которой приводит к кластеризации исходного набора данных.

функция ошибок, функция потерь, целевая функция

[NumPy ops](https://keras.io/api/ops/numpy/#numpy-ops) - набор функций для работы с тензорами в представлении tensorflow.

import keras.ops as ko

Пример предопределённой функции MSE:

def mse\_func(y\_true, y\_pred):

return ko.mean(ko.square(y\_pred-y\_true), axis = -1)

Использование центров кластеров – аналог K-means

здесь *n*-номер объекта во входном потоке данных, *i*-номер кластера.

Координаты центра кластера находим как среднее соотв. координат объектов образующих кластер:

Для этого случая:

Нейросеть:

model=Sequential()

model.add(Dense(30,input\_dim=DIM\_N,activation='relu'))

model.add(Dense(DIM\_N\*CLUST\_N,activation=None))

- выходные нейроны выдают координаты центров кластеров, для двумерного случая и трёх кластеров – массив *Cen* в программе:

Создаём представление входного набора данных data:

DataN = np.repeat(np.expand\_dims(data, axis=1), CLUST\_N, axis=1)

def my\_err\_func(y\_true, y\_pred):

global DataN

yy = ko.array(ko.split(ko.mean(y\_pred,axis=0),3))

z1 = ko.square(ko.subtract(DataN,yy))

z2 = ko.sum(z1,axis=2)

zz = ko.amin(z2,axis=1)

return zz

По функциям:

ko.mean(y\_pred,axis=0)

ko.split(ko.mean(y\_pred,axis=0),3)

yy = ko.array(ko.split(ko.mean(y\_pred,axis=0),3))

* Создаёт тензор в представлении tensorflow

z1 = ko.square(ko.subtract(DataN,yy))

z2 = ko.sum(z1,axis=2)

* перечень расстояний каждой точки до центров кластеров

zz = ko.amin(z2,axis=1)

- расстояния до центра ближайшего кластера для каждой точки

Последняя строка программы вычисляет значение искомой целевой функции:

Шаги итерации нейросети, аналогичные обучению при обычном подходе, подбирают значения , соответствуют поиску минимума целевой функции:

Полный текст соответствующей программы:

import numpy as np

import matplotlib.pyplot as plt

from sklearn import datasets

from sklearn import preprocessing

import random

POINT\_N = 300

DIM\_N = 2

CLUST\_N = 3

data, targ, Center = datasets.make\_blobs(n\_samples=POINT\_N, centers=CLUST\_N, n\_features=DIM\_N, center\_box=(-10,10), random\_state=0, return\_centers=True)

plt.figure(figsize=(7,7))

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

plt.scatter(Center[:,0], Center[:,1], c='red', marker="\*", s=100)

plt.show()

Изображение выглядит как снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

# ---------- Neural Network ----------

import warnings

warnings.filterwarnings("ignore")

import tensorflow as tf

import tensorflow.keras.backend as tfb

import keras

import keras.ops as ko

from keras.models import Sequential

from keras.layers import Dense

model=Sequential()

model.add(Dense(30,input\_dim=DIM\_N,activation='relu'))

model.add(Dense(DIM\_N\*CLUST\_N,activation=None))

DataN = np.repeat(np.expand\_dims(data, axis=1), CLUST\_N, axis=1)

def my\_err\_func(y\_true, y\_pred):

global DataN

yy = ko.array(ko.split(ko.mean(y\_pred,axis=0),3))

z1 = ko.square(ko.subtract(DataN,yy))

z2 = ko.sum(z1,axis=2)

zz = ko.amin(z2,axis=1)

return zz

model.compile(loss=my\_err\_func, optimizer='adam', metrics=['accuracy'], run\_eagerly=True)

model.summary()

Cen = np.array([data[random.randint(0,POINT\_N-1)] for i in range(CLUST\_N)])

out = np.zeros ((POINT\_N,CLUST\_N), dtype=int)

history = model.fit(data, out, batch\_size=data.shape[0], epochs=300, verbose=1)

Изображение выглядит как текст, снимок экрана, линия, График

Контент, сгенерированный ИИ, может содержать ошибки.

plt.plot(history.history['loss'], color='b', label="validation loss")

plt.xlabel("Number of Epochs")

plt.ylabel("Loss")

plt.legend()

plt.show()

pred=model.predict(data)

cen = np.array(np.split(np.mean(pred, axis=0), 3))

dd = np.sum(np.square(np.subtract(DataN,cen)), axis=2)

res = np.argmin(dd,axis=1)

colors = ['blue', 'green', 'purple', 'brown', 'black']

plt.figure(figsize=(7,7))

for i in range(POINT\_N):

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

plt.scatter(Center[:,0], Center[:,1], c='cyan', marker="\*", s=100)

plt.scatter(cen[:,0], cen[:,1], c='red', marker="\*", s=100)

plt.show()

Изображение выглядит как Красочность, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

- результат для 300 точек

Изображение выглядит как Красочность, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

- результат для 1000 точек