Кулиев Фёдор Русланович, группа 20-1

Лабораторная работа № 4

**Цель работы**

Выбрать и решить задачу классификации изображений при помощи KNN. Выбрать необходимое количество классов для вашей задачи. Точность модели на тестовых данных не должна быть ниже 0.75.

**Задание**

1. Решите вашу задачу, используя обычный KNN. Подберите:
   1. Оптимальное количество k-соседей (параметр n\_neighbors).
   2. Оптимальную функцию вычисления расстояния между соседями (параметр metric).
   3. Оптимальную функцию для взвешивания расстояний между соседями (или может взвешивание вообще не нужно) (параметр weights).
2. Используйте «Метод главных компонент» (Principal Component Analysis или PCA) и подберите оптимальное число главных компонент, которые дадут в итоге наибольший прирост в точности вашей модели.
3. Решить вашу задачу, используя предварительно обученную свёрточную нейронную сеть совместно с KNN. При необходимости к выходным признакам свёрточной нейронной сети можете дополнительно применить метод PCA.

**Код программы / ответ на вопрос и т.п.**

# Подключение необходимых библиотек

import numpy as np

import torch

import torchvision

from torchvision import datasets

from torchvision import transforms

from tqdm import tqdm

import torch.nn as nn

from torch.utils.data import DataLoader

import matplotlib.pyplot as plt

from sklearn.metrics import classification\_report, accuracy\_score

from sklearn.neighbors import KNeighborsClassifier

from sklearn.decomposition import PCA

# Переключение на GPU, если это возможно

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

# Создание и нормализация обучающей и тестовой выборки

batch\_size = 32

transform = transforms.Compose([transforms.ToTensor(),

                                transforms.Resize((128, 128)),

                                transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

train\_path = "Faces/train"

train\_ds = datasets.ImageFolder(root=train\_path, transform=transform)

train\_dataloader = DataLoader(train\_ds, batch\_size=batch\_size, shuffle=True)

test\_path = "Faces/test"

test\_ds = datasets.ImageFolder(root=test\_path, transform=transform)

test\_dataloader = DataLoader(test\_ds, batch\_size=batch\_size, shuffle=True)

# Просматриваем изображения

def imshow(img):

    img = img / 2 + 0.5

    npimg = img.numpy()

    plt.imshow(np.transpose(npimg, (1, 2, 0)))

    plt.show()

train\_features, train\_labels = next(iter(train\_dataloader))

print(f"Feature batch shape: {train\_features.size()}")

print(f"Labels batch shape: {train\_labels.size()}")

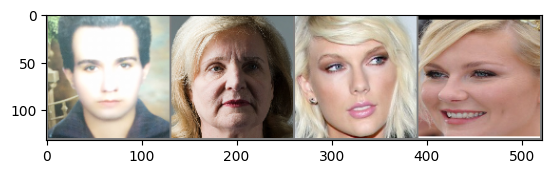
images = []

for i in range(4):

    images.append(train\_features[i])

imshow(torchvision.utils.make\_grid(images)) # show images

print(' '.join(f'{train\_ds.classes[train\_labels[j]]}\t' for j in range(4))) # print labels



Neutral Sad Neutral Happy

# Функция формирования набора данных (картинки и метки)

def extract\_dataset\_images(dataloader):

  labels = []

  images = []

  for batch, batch\_labels in tqdm(dataloader):

    images.append(batch.detach().numpy().reshape((batch.shape[0], -1))) # Делаем картинки в виде векторов

    labels.append(batch\_labels.detach().numpy())

  return np.concatenate(images), np.concatenate(labels)

x\_train, y\_train = extract\_dataset\_images(train\_dataloader)

x\_test, y\_test = extract\_dataset\_images(test\_dataloader)

# Функция обучения и тестирования модели KNN

def train\_and\_test\_KNN(x\_train, y\_train, x\_test, y\_test, n\_neighbors, metric, weights):

  # Обучение KNN на картинках

  knn\_classifier = KNeighborsClassifier(n\_neighbors=n\_neighbors, metric = metric, weights = weights)

  knn\_classifier.fit(x\_train, y\_train)

  # Тестирование

  # Классификация тестовых изображений с использованием KNN

  y\_pred = knn\_classifier.predict(x\_test)

  accuracy = accuracy\_score(y\_test, y\_pred)

  return accuracy

def inverse\_squared\_distance(dist):

    arr = []

    for d in dist:

        arr.append(1 / (d\*\*2))

    return arr

# Параметры KNN

metrics\_array = ["minkowski", "euclidean", "cosine"]

weights\_array = ["uniform", "distance", inverse\_squared\_distance]

n\_neighbors = [3, 5, 10, 20]

# Функция обучения и тестирования модели KNN и PCA

def train\_and\_test\_PCA\_KNN(x\_train, y\_train, x\_test, y\_test, n\_components, knn\_classifier):

  pca = PCA(n\_components=n\_components)

  x\_train\_pca = pca.fit\_transform(x\_train)

  knn\_classifier.fit(x\_train\_pca, y\_train)

  # Тестирование

  # Применение созданного ранее PCA для тестовых данных

  x\_test\_pca = pca.transform(x\_test)

  # Классификация тестовых изображений с использованием KNN

  y\_pred = knn\_classifier.predict(x\_test\_pca)

  accuracy = accuracy\_score(y\_test, y\_pred)

  return accuracy

# Параметры PCA

n\_components = [10, 50, 100, 200, 300, 400]

# Скачивание предобученной модели VGG16

from torchvision.models import vgg16

model = vgg16(pretrained=True)

model.classifier = nn.Flatten(start\_dim= 1)

model.to(device)

# Использование модели VGG19 для создания обучающей и тестовой выборки

def extract\_features(dataset, model):

    features = []

    labels = []

    for batch\_images, batch\_labels in tqdm(dataset):

        batch\_features = model(batch\_images.to(device))

        labels.append(batch\_labels.detach().numpy())

        features.append(batch\_features.cpu().detach().numpy())

    return np.concatenate(features), np.concatenate(labels)

x\_train\_features, y\_train\_labels = extract\_features(train\_dataloader, model)

x\_test\_features, y\_test\_labels = extract\_features(test\_dataloader, model)

**Результаты выполнения**

1. Решите вашу задачу, используя обычный KNN

# Поиск оптимальных параметров KNN

parameters = 0, "", ""

accuracy = 0

for metric, weights, n in itertools.product(metrics\_array, weights\_array, n\_neighbors):

    temp\_accuracy = train\_and\_test\_KNN(x\_train, y\_train, x\_test, y\_test, n, metric, weights)

    if accuracy < temp\_accuracy:

        parameters = n, metric, weights

        accuracy = temp\_accuracy

print("Best KNN accuracy:")

print(f"n\_neighbors = {parameters[0]} \nmetric = {parameters[1]} \nweights = {parameters[2]} \naccuracy = {accuracy}")

100%|██████████| 36/36 [10:48<00:00, 18.00s/it]

Best KNN accuracy:

n\_neighbors = 10

metric = minkowski

weights = inverse\_squared\_distance

accuracy = 0.4264285714

1. Используйте «Метод главных компонент» (Principal Component Analysis или PCA) и подберите оптимальное число главных компонент, которые дадут в итоге наибольший прирост в точности вашей модели.

#KNN + PCA accuracy

parameters = 0, "", "", 0

accuracy = 0

for metric, weights, n\_neigh, n\_comp in itertools.product(metrics\_array, weights\_array, n\_neighbors, n\_components):

    knn\_classifier = KNeighborsClassifier(n\_neighbors=n\_neigh, metric = metric, weights = weights, n\_jobs=-1)

    temp\_accuracy = train\_and\_test\_PCA\_KNN(x\_train, y\_train, x\_test, y\_test, n\_comp, knn\_classifier)

    if accuracy < temp\_accuracy:

        parameters = n\_neigh, metric, weights, n\_comp

        accuracy = temp\_accuracy

print("Best KNN accuracy:")

print(f"n\_neighbors = {parameters[0]} \nmetric = {parameters[1]} \nweights = {parameters[2]} \nn\_components = {parameters[3]} \naccuracy = {accuracy}")

100%|██████████| 180/180 [47:48<00:00, 15.94s/it]

Best KNN accuracy:

n\_neighbors = 20

metric = minkowski

weights = inverse\_squared\_distance

n\_components = 50

accuracy = 0.5264548

1. Решить вашу задачу, используя предварительно обученную свёрточную нейронную сеть совместно с KNN. При необходимости к выходным признакам свёрточной нейронной сети можете дополнительно применить метод PCA.

#KNN + PCA + CNN accuracy

parameters = 0, "", "", 0

accuracy = 0

for metric, weights, n\_neigh, n\_comp in itertools.product(metrics\_array, weights\_array, n\_neighbors, n\_components):

    knn\_classifier = KNeighborsClassifier(n\_neighbors=n\_neigh, metric = metric, weights = weights)

    temp\_accuracy = train\_and\_test\_PCA\_KNN(x\_train\_features, y\_train\_labels, x\_test\_features, y\_test\_labels, n\_comp, knn\_classifier)

    if accuracy < temp\_accuracy:

        parameters = n\_neigh, metric, weights, n\_comp

        accuracy = temp\_accuracy

print("Best KNN accuracy:")

print(f"n\_neighbors = {parameters[0]} \nmetric = {parameters[1]} \nweights = {parameters[2]} \nn\_components = {parameters[3]} \naccuracy = {accuracy}")

KNN + PCA + CNN accuracy comparison:

100%|██████████| 180/180 [47:48<00:00, 15.94s/it]

Best KNN accuracy:

n\_neighbors = 10

metric = cosine

weights = inverse\_squared\_distance

n\_components = 200

accuracy = 0.769846

# Точность модели

knn\_classifier = KNeighborsClassifier(n\_neighbors=best\_n\_neighbors, metric=best\_metric, weights=best\_weights)

pca = PCA(n\_components=best\_n\_components)

x\_train\_pca = pca.fit\_transform(x\_train\_features)

knn\_classifier.fit(x\_train\_pca, y\_train\_labels)

x\_test\_pca = pca.transform(x\_test\_features)

y\_pred = knn\_classifier.predict(x\_test\_pca)

print(classification\_report(y\_test\_labels, y\_pred, target\_names=test\_ds.classes))

precision recall f1-score support

Angry 0.82 0.79 0.80 200

Happy 0.83 0.88 0.87 200

Neutral 0.73 0.69 0.72 200

Sad 0.72 0.77 0.76 200

Surprise 0.69 0.75 0.77 200

accuracy 0.77 1000

macro avg 0.74 0.75 0.74 1000

weighted avg 0.76 0.76 0.75 1000

**Выводы**

В результате выполнения работы была решена задача классификации изображений при помощи KNN.

Были протестированы модели с различными параметрами, а также комбинации моделей – KNN, PCA+KNN, CNN+PCA+KNN.

Для KNN лучший результат показали следующие параметры:

*n\_neighbors = 10*

*metric = minkowski*

*weights = inverse\_squared\_distance*

Точность модели 53%

Для PCA + KNN:

n\_neighbors = 20

metric = *minkowski*

weights = inverse\_squared\_distance

n\_components = 50

Точность модели 39%

Для CNN + PCA + KNN:

n\_neighbors = 10

metric = cosine

weights = inverse\_squared\_distance

n\_components = 200

Точность модели 77%