1. **Задание**

Задание:

1) Смоделировать обучающие и тестовые выборки образцов согласно варианту (например, в варианте 1 образцом будет изображение). Для каждого класса моделировать по 10 обучающих и по 10 тестовых образцов.
2) По каждому образцу составить исходную последовательность наблюдений. Это можно реализовать как программно, так и вручную.
3) Ввести алфавит СММ, задать количество скрытых состояний *N*.
4) Преобразовать каждую исходную последовательность наблюдений согласно этому алфавиту в итоговую последовательность наблюдений (т.е. провести квантование, см [5])
5) Обучить каждую модель из варианта на своей обучающей выборке итоговых последовательностей наблюдений (см л/р №4).
6) Реализовать процедуру распознавания итоговых последовательностей наблюдений (см [4] стр 163-165, формула (6.21)).
7) Провести исследование по выбору числа скрытых состояний *N* на обучающей выборке итоговых последовательностей наблюдений. Результат для каждого взятого числа скрытых состояний *N* представить в таблице:

| число скрытых состояний *N=…* | | Номер класса, к которому отнесены обучающие итоговые последовательности наблюдений | | | |
|---|---|---|---|---|---|
| | | *1* | *2* | … | *Class* |
| Номер истинного класса, к которому принадлежат обучающие итоговые последовательности наблюдений | *1* | Кол-во обучающих итоговых последовательностей наблюдений, принадлежащих 1ому классу и отнесенных к 1ому классу | Кол-во обучающих итоговых последовательностей наблюдений, принадлежащих 1ому классу и отнесенных ко 2ому классу | … | Кол-во обучающих итоговых последовательностей наблюдений, принадлежащих 1ому классу и отнесенных к классу с номером *Class* |
| | *2* | Кол-во обучающих итоговых последовательностей наблюдений, принадлежащих 2ому классу и отнесенных к 1ому классу | Кол-во обучающих итоговых последовательностей наблюдений, принадлежащих 2ому классу и отнесенных ко 2ому классу | … | Кол-во обучающих итоговых последовательностей наблюдений, принадлежащих 2ому классу и отнесенных к классу с номером *Class* |
| | … | … | … | … | … |
| | *Class* | Кол-во обучающих итоговых последовательностей наблюдений, принадлежащих классу с номером *Class* и отнесенных к 1ому классу | Кол-во обучающих итоговых последовательностей наблюдений, принадлежащих классу с номером *Class* и отнесенных ко 2ому классу | … | Кол-во обучающих итоговых последовательностей наблюдений, принадлежащих классу с номером *Class* и отнесенных к классу с номером *Class* |

Выбрать число скрытых состояний, при котором обеспечивается наилучший процент распознавания на обучающей выборке.

8) Для наилучшего значения *N* привести оценки параметров модели.

9) Провести исследование по точности распознавания на тестовой выборке итоговых последовательностей наблюдений. Результат представить в виде таблицы:

| | | Номер класса, к которому отнесены тестовые итоговые последовательности наблюдений | | | |
|---|---|---|---|---|---|
| | | *1* | *2* | ... | *Class* |
| Номер истинного класса, к которому принадлежат тестовые итоговые последовательности наблюдений | *1* | Кол-во тестовых итоговых последовательностей наблюдений, принадлежащих 1ому классу и отнесенных к 1ому классу | Кол-во тестовых итоговых последовательностей наблюдений, принадлежащих 1ому классу и отнесенных ко 2ому классу | … | Кол-во тестовых итоговых последовательностей наблюдений, принадлежащих 1ому классу и отнесенных к классу с номером *Class* |
| | *2* | Кол-во тестовых итоговых последовательностей наблюдений, принадлежащих 2ому классу и отнесенных к 1ому классу | Кол-во тестовых итоговых последовательностей наблюдений, принадлежащих 2ому классу и отнесенных ко 2ому классу | ... | Кол-во тестовых итоговых последовательностей наблюдений, принадлежащих 2ому классу и отнесенных к классу с номером *Class* |
| | ... | ... | ... | ... | ... |
| | *Class* | Кол-во тестовых итоговых последовательностей наблюдений, принадлежащих классу с номером *Class* и отнесенных к 1ому классу | Кол-во тестовых итоговых последовательностей наблюдений, принадлежащих классу с номером *Class* и отнесенных ко 2ому классу | ... | Кол-во тестовых итоговых последовательностей наблюдений, принадлежащих классу с номером *Class* и отнесенных к классу с номером *Class* |

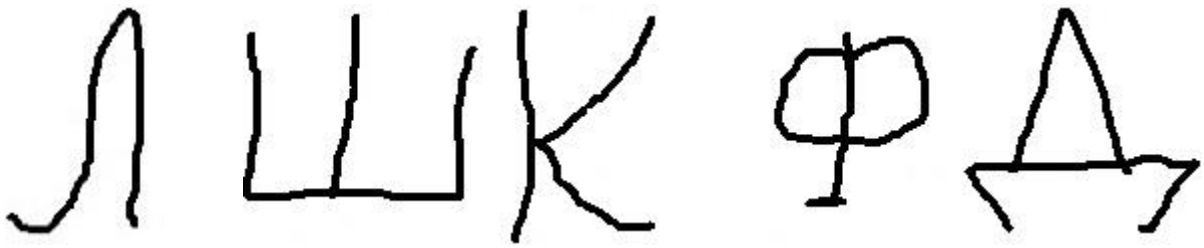
**2. Вариант задания**

В качестве признаков для построения исходных последовательностей наблюдений используется значение яркости центрального пикселя сканирующего окна, продвигающегося из верхнего левого угла изображения. Размер сканирующего окна $X \times Y$ пикселей, перекрытие между сканирующими окнами по горизонтали $Hx$ пикселей, по вертикали $Hy$ пикселей.

Пример изображения: см вариант 1 .

В л/р использовать 5 любых классов.

5 букв (классов)



## 3. Исследования

***Исследование по выбору числа скрытых состояний N на обучающей выборке итоговых последовательностей наблюдений.***

| число скрытых состояний *N=2* | | Номер класса, к которому отнесены обучающие итоговые последовательности наблюдений | | | | |
|---|---|---|---|---|---|---|
| | | *1* | *2* | *3* | *4* | *5* |
| Номер истинного класса, к которому принадлежат обучающие итоговые последовательности наблюдений | *1* | 40% | 20% | 10% | 0% | 30% |
| | *2* | 0% | 100% | 0% | 0% | 0% |
| | *3* | 20% | 0% | 80% | 0% | 0% |
| | *4* | 0% | 0% | 10% | 70% | 20% |
| | *5* | 0% | 20% | 0% | 20% | 60% |
| число скрытых состояний *N=3* | | Номер класса, к которому отнесены обучающие итоговые последовательности наблюдений | | | | |
| | | *1* | *2* | *3* | *4* | *5* |
| Номер истинного класса, к которому принадлежат обучающие итоговые последовательности наблюдений | *1* | 80% | 20% | 0% | 0% | 0% |
| | *2* | 0% | 100% | 0% | 0% | 0% |
| | *3* | 10% | 0% | 90% | 0% | 0% |
| | *4* | 0% | 0% | 0% | 80% | 20% |
| | *5* | 0% | 10% | 0% | 20% | 70% |
| число скрытых состояний *N=4* | | Номер класса, к которому отнесены обучающие итоговые последовательности наблюдений | | | | |
| | | *1* | *2* | *3* | *4* | *5* |
| Номер истинного класса, к которому принадлежат обучающие итоговые последовательности наблюдений | *1* | 80% | 0% | 10% | 0% | 10% |
| | *2* | 0% | 90% | 0% | 0% | 10% |
| | *3* | 20% | 0% | 40% | 0% | 40% |
| | *4* | 0% | 0% | 0% | 60% | 40% |
| | *5* | 10% | 0% | 0% | 0% | 90% |

***Для наилучшего значения N=3 полученные оценки параметров модели для каждого класса.***

```
[array([  0.00000000e+000,   7.47732992e-162,   1.00000000e+000]),
array([[ 0.88359071,  0.05265751,  0.06375178],
       [ 0.        ,  0.34967077,  0.65032923],
       [ 0.03401485,  0.        ,  0.96598515]]),
array([[ 0.02945389,  0.97054611],
       [ 0.99508191,  0.00491809],
       [ 0.05418625,  0.94581375]])]
```

```
[array([ 0.,  0.,  1.]),
array([[  8.46593164e-01,   3.26939352e-03,   1.50137443e-01],
       [  0.00000000e+00,   9.22498818e-01,   7.75011817e-02],
       [  2.95011827e-01,   1.07621485e-04,   7.04880552e-01]]),
array([[  7.97623932e-02,   9.20237607e-01],
       [  1.00000000e+00,   9.67144731e-29],
       [  6.70122976e-02,   9.32987702e-01]])]

[array([  0.00000000e+000,   6.46350382e-113,   1.00000000e+000]),
array([[  9.30833052e-01,   6.91481087e-02,   1.88391926e-05],
       [  0.00000000e+00,   3.60601868e-01,   6.39398132e-01],
       [  8.90347688e-02,   0.00000000e+00,   9.10965231e-01]]),
array([[  4.35744238e-02,   9.56425576e-01],
       [  9.16947946e-01,   8.30520542e-02],
       [  1.87764514e-06,   9.99998122e-01]])]

[array([ 0.,  0.,  1.]),
array([[ 0.82759918,  0.03936688,  0.13303394],
       [ 0.        ,  0.5943502 ,  0.4056498 ],
       [ 0.32353696,  0.00106787,  0.67539516]]),
array([[ 0.06289352,  0.93710648],
       [ 0.91682078,  0.08317922],
       [ 0.02975908,  0.97024092]])]

[array([  0.00000000e+00,   9.99989813e-01,   1.01874562e-05]),
array([[  8.10806262e-01,   1.88858497e-01,   3.35240973e-04],
       [  0.00000000e+00,   5.25210859e-01,   4.74789141e-01],
       [  7.81228789e-03,   9.25267553e-02,   8.99660957e-01]]),
array([[  1.00000000e+00,   3.99319085e-12],
       [  2.44725103e-04,   9.99755275e-01],
       [  7.03446286e-02,   9.29655371e-01]])]
```

***Исследование по точности распознавания на тестовой выборке итоговых последовательностей наблюдений.***

| число скрытых состояний *N=* | | Номер класса, к которому отнесены обучающие итоговые последовательности наблюдений | | | | |
|---|---|---|---|---|---|---|
| | | *1* | *2* | *3* | *4* | *5* |
| Номер истинного класса, к которому принадлежат обучающие итоговые последовательности наблюдений | *1* | 90% | 0% | 10% | 0% | 0% |
| | *2* | 0% | 50% | 10% | 0% | 40% |
| | *3* | 10% | 0% | 90% | 0% | 0% |
| | *4* | 0% | 0% | 0% | 40% | 60% |
| | *5* | 10% | 10% | 0% | 0% | 80% |

## 4. Текст программы

```
from PIL import Image
import numpy as np
import scipy as sp
from functools import reduce
import matplotlib.pyplot as plt
import os
def get_obs(path, K):
    pix_bit = []
    step = 5
    image_paths = [os.path.join(path, f) for f in os.listdir(path) if f.endswith('.jpg')]
    im = [Image.open(imp).convert('1') for imp in image_paths]
    for k in range(K):
        W = im[k].width
        H = im[k].height
        pix = []
        for i in range(step, H, step - 1):
            for j in range(0, W, step - 1):
                if j + step < W:
                    GP = im[k].crop((j, i - step, j + step,i))
                    x1 = GP._ImageCrop__crop[0]
                    x3 = GP._ImageCrop__crop[2]
                    y1 = GP._ImageCrop__crop[1]
                    y3 = GP._ImageCrop__crop[3]
                    x = (x1 + x3) / 2
                    y = (y1 + y3) / 2
                    pix.append(im[k].getpixel((x,y)))
        pix_bit.append(list(map(lambda x: 1 if x == 255 else x, pix)))
    return np.array(pix_bit)
def get_data(fname, type):
    O = np.array([[i for i in line.split()] for line in open(fname, encoding="utf-8")],
dtype=type)
    return O

def get_data1(fname, type):
    O = np.array([i for i in open(fname, encoding="utf-8").readline().split()], dtype=type)
    return O

def WritingInFile(names, sequences, fileName):
    with open(fileName, "w") as file:
        for line in sequences:
            print(line, file=file)

def forward_path(O, pi, A, B, T, N, K):
    alpha_k = []
    for k in range(K):
        alpha = np.zeros((T, N))
        alpha[0, :] = pi * B[:, O[k, 0]]
```

```
        for t in range(1, T):
            for j in range(N):
                tmp = np.zeros(N)
                for i in range(N):
                    tmp[i] = alpha[t - 1, i] * A[i, j]
                alpha[t, j] = tmp.sum() * B[j, O[k, t]]
        alpha_k.append(alpha)
    return np.array(alpha_k)

def backward_path(O, pi, A, B, T, N, K):
    beta_k = []
    for k in range(K):
        beta = np.zeros((T, N))
        beta[T - 1, :] = 1
        for t in range(T - 2, -1, -1):
            for i in range(N):
                tmp = np.zeros(N)
                for j in range(N):
                    tmp[j] = beta[t + 1, j] * A[i, j] * B[j, O[k, t + 1]]
                beta[t, i] = tmp.sum()
        beta_k.append(beta)
    return np.array(beta_k)

def calculate_gamma(alpha, beta, T, N, K):
    gamma_k = []
    for k in range(K):
        gamma = np.zeros((T, N))
        for t in range(T):
            for i in range(N):
                gamma[t, i] = alpha[k, t, i] * beta[k, t, i]
            sum_all = gamma[t, :].sum()
            gamma[t, :] = gamma[t, :] / sum_all
        gamma_k.append(gamma)
    return np.array(gamma_k)

def calculate_ksi(O, alpha, beta, A, B, T, N, K):
    ksi_k = []
    for k in range(K):
        ksi = np.zeros((T, N, N))
        for t in range(T - 1):
            for i in range(N):
                for j in range(N):
                    ksi[t, i, j] = alpha[k, t, i] * A[i, j] * beta[k, t + 1, j] * B[j, O[k, t +
1]]
            sum_all = ksi[t, :, :].sum()
            ksi[t, :, :] = ksi[t, :, :] / sum_all
        ksi_k.append(ksi)
    return np.array(ksi_k)

def estimate_parameter(O, pi_0, A_0, B_0, T, N, M, K):
    alp = forward_path(O, pi_0, A_0, B_0, T, N, K)
    bet = backward_path(O, pi_0, A_0, B_0, T, N, K)
    gam = calculate_gamma(alp, bet, T, N, K)
    ksi = calculate_ksi(O, alp, bet, A_0, B_0, T, N, K)

    est_pi = np.sum(gam[:, 0, :], axis=0) / K

    est_A_k = np.zeros((K, N, N))
    for k in range(K):
        for i in range(N):
            denom = gam[k, :-1, i].sum()
            for j in range(N):
                est_A_k[k, i, j] = ksi[k, :-1, i, j].sum() / denom

    est_A = np.sum(est_A_k, axis=0) / K
    est_B_k = np.zeros((K, N, M))
    for k in range(K):
        for i in range(N):
```

```
            denom = gam[k, :, i].sum()
            for j in range(M):
                numer = gam[k, :, i][O[k] == j].sum()
                est_B_k[k, i, j] = numer / denom
    est_B = np.sum(est_B_k, axis=0) / K
    return est_pi, est_A, est_B

def log_likelihood(O, pi, A, B, T, N, K):
    alp = forward_path(O, pi, A, B, T, N, K)
    L = []
    for k in range(K):
        l = np.zeros((N))
        for i in range(N):
            l[i] = alp[k, T - 1, i]
        sum_all = l[:].sum()
        L.append(sum_all)
    lnL = np.sum(np.log(L))
    return lnL

def forward_path1(O, pi, A, B, T, N, K):
    alpha = np.zeros((T, N))
    alpha[0, :] = pi * B[:, O[0]]
    for t in range(1, T):
        for j in range(N):
            tmp = np.zeros(N)
            for i in range(N):
                tmp[i] = alpha[t - 1, i] * A[i, j]
            alpha[t, j] = tmp.sum() * B[j, O[t]]
    return np.array(alpha)
def log_likelihood_for_learn_or_test(O, pi, A, B, T, N):
    alp = forward_path1(O, pi, A, B, T, N, 1)
    l = np.zeros((N))
    for i in range(N):
        l[i] = alp[T - 1, i]
    L = l[:].sum()
    lnL = np.sum(np.log(L))
    return lnL

def iter_exit(O, pi_old, A_old, B_old, pi_new, A_new, B_new, T, N, K):
    old = log_likelihood(O, pi_old, A_old, B_old, T, N, K)
    new = log_likelihood(O, pi_new, A_new, B_new, T, N, K)
    exit = abs(old - new)
    if exit > 1e-3:
        return False, exit
    else:
        return True, exit

def baum_welch(O, pi, A, B, T, N, M, K):
    iter = 0
    exit = False
    max_iter = 100
    ex = []
    temp = []
    temp.append(log_likelihood(O, pi, A, B, T, N, K))
    while exit == False:
        iter += 1
        new_pi, new_A, new_B = estimate_parameter(O, pi, A, B, T, N, M, K)
        exit, tmp = iter_exit(O, pi, A, B, new_pi, new_A, new_B, T, N, K)
        temp.append(log_likelihood(O, new_pi, new_A, new_B, T, N, K))
        if iter > max_iter:
            exit = True
        ex.append(tmp)
        pi, A, B = new_pi, new_A, new_B
    return pi, A, B, ex

def test_or_learn():
    K = 10      #количество картинок в каждом классе для обучения
    N = 2       #число скрытых состояний
```

```
    M = 2       #алфавит
    CL = 5      #число классов
    path = []
    #path_test = []
    path.append(os.path.abspath('./1/'))
    path.append(os.path.abspath('./2/'))
    path.append(os.path.abspath('./3/'))
    path.append(os.path.abspath('./4/'))
    path.append(os.path.abspath('./5/'))
    path_test.append(os.path.abspath('./6/'))
    path_test.append(os.path.abspath('./7/'))
    path_test.append(os.path.abspath('./8/'))
    path_test.append(os.path.abspath('./9/'))
    path_test.append(os.path.abspath('./10/'))
    A_path = os.path.abspath('./A/')
    B_path = os.path.abspath('./B/')
    pi_path = os.path.abspath('./PI/')
    A_arr = [os.path.join(A_path, f) for f in os.listdir(A_path) if f.endswith('.txt')]
    B_arr = [os.path.join(B_path, f) for f in os.listdir(B_path) if f.endswith('.txt')]
    pi_arr = [os.path.join(pi_path, f) for f in os.listdir(pi_path) if f.endswith('.txt')]
    A_0 = [get_data(a, np.double) for a in A_arr]
    B_0 = [get_data(b, np.double) for b in B_arr]
    pi_0 = [get_data(pi, np.double) for pi in pi_arr]
    O = [get_obs(el, K) for el in path]
    O_test = [get_obs(el, K) for el in path_test]
    #оценка параметров
    est_p = []
    for k in range(CL):#количество классов
        est = []
        lnL = []
        for i in range(5):#количество начальных приближений
            est_pi, est_A, est_B, ex = baum_welch(O[k], np.array(pi_0[i]), np.array(A_0[i]),
np.array(B_0[i]), O[k].shape[1], N, M, K)
            est.append([est_pi, est_A, est_B])
            lnL.append(log_likelihood(O[k], est_pi, est_A, est_B, O[k].shape[1], N, K))
        maxlnL = lnL.index(np.max(np.array(lnL)))
        est_p.append(est[maxlnL])
    WritingInFile(['est'], est_p, 'est.txt')

    test_t = []
    for c in range(5):#class
        test_t.append([])
        for i in range(K):
            test_t[c].append([])
            for j in range(5):#class
                test_t[c][i].append(log_likelihood_for_learn_or_test(O_test[c][i], est_p[j][0],
est_p[j][1], est_p[j][2], O_test[c][i].shape[0], N))
    res_class_t = np.argmax(test_t, axis = 2)
    WritingInFile(['rest'], res_class_t, 'rest.txt')
    return est_p

test_or_learn()
```