Лосева Елизавета Юрьевна, группа 8-2

Лабораторная работа № 4

Вариант № 3

Распознавание образов, описываемых бинарными признаками

Цель работы

Синтезировать алгоритмы распознавания образов, описываемых бинарными признаками. Исследовать синтезированные алгоритмы распознавания с точки зрения ожидаемых потерь и ошибок.

Задание

Получить матрицы ошибок (теоретическую и экспериментальную) распознавания трех образов (первых букв имени фамилии и отчества исполнителя). Использовать разделяющую функцию вида (5.36) (раздел учебника 5.3.3). Значение вероятности искажения символа pI = 0.25.

написать код, реализующий алгоритм распознавания образов, заданных бинарными изображениями. Проведите имитационное моделирование алгоритма, в ходе которого рассчитайте значения вероятности ошибок распознавания для трех различных случаев априорных вероятностей гипотез:

- $p(\omega1) > p(\omega2)$;
- $p(\omega1) = p(\omega2)$;
- $p(\omega1) < p(\omega2)$;

Сравните полученные вероятности ошибок их со значениями, вычисленными теоретически. Провести анализ полученных результатов и представить его в виде выводов по проделанной работе.

Код программы

```
import numpy as np
import matplotlib.pyplot as plt

# Параметры
pI = 0.0001 # вероятность инверсии
rng = np.random.default_rng(42) # генератор случайных чисел
N_THEOR = 200_000   # количество реализаций для "теоретич." оценки
N_EXP = 6_000     # количество реализаций для эксперимента

# Буквы L / E / U (8×8)

L_8x8 = np.array([
    [1, 0, 0, 0, 0, 0, 0, 0],
    [1, 0, 0, 0, 0, 0, 0, 0],
    [1, 0, 0, 0, 0, 0, 0, 0],
    [1, 0, 0, 0, 0, 0, 0, 0],
    [1, 0, 0, 0, 0, 0, 0, 0],
    [1, 0, 0, 0, 0, 0, 0, 0],
    [1, 0, 0, 0, 0, 0, 0, 0],
    [1, 1, 1, 1, 1, 1, 1, 1],
], dtype=int)
```

```
E_8x8 = np.array([
    [1, 1, 1, 1, 1, 1, 1, 1],
    [1, 0, 0, 0, 0, 0, 0, 0],
    [1, 0, 0, 0, 0, 0, 0, 0],
    [1, 1, 1, 1, 1, 1, 1, 1],
    [1, 0, 0, 0, 0, 0, 0, 0],
    [1, 0, 0, 0, 0, 0, 0, 0],
    [1, 0, 0, 0, 0, 0, 0, 0],
    [1, 1, 1, 1, 1, 1, 1, 1],
], dtype=int)

U_8x8 = np.array([
    [1, 0, 0, 0, 0, 0, 0, 1],
    [1, 0, 0, 0, 0, 0, 0, 1],
    [1, 0, 0, 0, 0, 0, 0, 1],
    [1, 0, 0, 0, 0, 0, 0, 1],
    [1, 0, 0, 0, 0, 0, 0, 1],
    [1, 0, 0, 0, 0, 0, 0, 1],
    [1, 0, 0, 0, 0, 0, 0, 1],
    [1, 1, 1, 1, 1, 1, 1, 1],
], dtype=int)

class_names = ["L", "E", "U"]
templates = [L_8x8, E_8x8, U_8x8]
T = np.stack([t.reshape(-1).astype(np.uint8) for t in templates], axis=0)  #
преобразуем 8×8 → вектор длиной 64
N_CLASSES, N_BITS = T.shape


def distort_bits(x_bits: np.ndarray, p: float, rng: np.random.Generator) ->
np.ndarray:
    # Искажение бита по модели:
    # x' = x с вероятностью (1-p), x' = 1-x с вероятностью p.
    flips = rng.random(x_bits.shape) < p
    return x_bits ^ flips.astype(np.uint8)  # XOR выполняет инверсию

def hamming_distance(A: np.ndarray, B: np.ndarray) -> np.ndarray:
    # Расстояние Хэмминга:
    # d(x,t) = сумма отличающихся битов
    xor = np.bitwise_xor(A[:, None, :], B[None, :, :])
    return xor.sum(axis=2)

def classify_map_classic(samples: np.ndarray, templates_vec: np.ndarray,
                         p: float, priors: np.ndarray) -> np.ndarray:
    # MAP-классификация:
    # выбираем класс k, максимизируя:
    # log P(x|w_k) + log P(w_k)
    p_eff = min(p, 1.0 - p)
    d = hamming_distance(samples.astype(np.uint8),
templates_vec.astype(np.uint8))  # расстояние Хэмминга d(x,t)
    # Лог-правдоподобие (по модели искажения):
```

```
    # log P(x|w_k) = (64-d)*log(1-p) + d*log(p)
    log1mp, logp = np.log(1.0 - p_eff), np.log(p_eff)
    loglik = (N_BITS - d) * log1mp + d * logp
    # Учет априорных вероятностей:
    score = loglik + np.log(priors.reshape(1, -1))
    return np.argmax(score, axis=1) # класс с максимальным значением

def confusion_mc(templates_vec: np.ndarray, p: float, priors: np.ndarray,
                 n_per_class: int, rng: np.random.Generator) -> np.ndarray:
    # Построение матрицы ошибок
    # CM_ij = вероятность классификации истинного класса i как j
    K = templates_vec.shape[0]
    cm = np.zeros((K, K), dtype=np.int64)
    for k in range(K):
        base = np.repeat(templates_vec[k][None, :], n_per_class, axis=0)
        X = distort_bits(base, p, rng)  # создаем искаженные примеры
        yhat = classify_map_classic(X, templates_vec, p, priors)
        for j in range(K):
            cm[k, j] += np.sum(yhat == j)
    cm = cm.astype(float)
    row_sums = cm.sum(axis=1, keepdims=True)
    row_sums[row_sums == 0] = 1.0
    return cm / row_sums

def annotate_heatmap(ax, cm, fmt="{:.4f}"):
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, fmt.format(cm[i, j]), ha="center", va="center",
color="w", fontsize=10)

def plot_cm_grid(cases, cms_theor, cms_exp, class_names, pI):
    fig, axes = plt.subplots(2, 3, figsize=(16, 8), constrained_layout=True)
    fig.suptitle(f"Влияние априорных вероятностей | pI = {pI}", fontsize=13,
y=0.98)


    for col, (title, pri) in enumerate(cases):
        # верх: теоретическая матрица
        ax_t = axes[0, col]
        im1 = ax_t.imshow(cms_theor[col], vmin=0.0, vmax=1.0, cmap="viridis")
        ax_t.set_title(f"{title}\npriors={np.round(pri, 3)}\nТеоретическая",
fontsize=11)
        ax_t.set_xticks(range(len(class_names)));
ax_t.set_yticks(range(len(class_names)))
        ax_t.set_xticklabels(class_names);        ax_t.set_yticklabels(class_name
s)
        annotate_heatmap(ax_t, cms_theor[col])

        # низ: экспериментальная матрица
        ax_e = axes[1, col]
        ax_e.imshow(cms_exp[col], vmin=0.0, vmax=1.0, cmap="viridis")
```

```
        ax_e.set_title(f"{title}\npriors={np.round(pri, 3)}\nЭкспериментальная",
fontsize=11)
        ax_e.set_xticks(range(len(class_names)));
ax_e.set_yticks(range(len(class_names)))
        ax_e.set_xticklabels(class_names);        ax_e.set_yticklabels(class_name
s)
        annotate_heatmap(ax_e, cms_exp[col])

    cbar = fig.colorbar(im1, ax=axes.ravel().tolist(), fraction=0.015, pad=0.02)
    cbar.ax.set_ylabel("Вероятность", rotation=90)
    return fig



# Основной блок выполнения
if __name__ == "__main__":
    np.set_printoptions(precision=3, suppress=True)

    cases = [
        ("Случай 1: p(w1)>p(w2)", np.array([0.7, 0.2, 0.1], dtype=float)),
        ("Случай 2: равные p(w)",  np.array([1/3, 1/3, 1/3], dtype=float)),
        ("Случай 3: p(w1)<p(w2)", np.array([0.1, 0.2, 0.7], dtype=float)),
    ]

    cms_theor, cms_exp = [], []
    for _, pri in cases:
        cms_theor.append(confusion_mc(T, pI, pri, N_THEOR, rng))
        cms_exp.append(  confusion_mc(T, pI, pri, N_EXP,   rng))

    plot_cm_grid(cases, cms_theor, cms_exp, class_names, pI)

    # 2) исходные буквы
    fig2, axes2 = plt.subplots(1, 3, figsize=(8, 3), constrained_layout=True)
    fig2.suptitle("Шаблоны 8×8 (оригинал)", fontsize=14)
    for ax, name, M in zip(axes2, class_names, templates):
        ax.imshow(M, cmap="gray_r", interpolation="nearest")
        ax.set_title(name); ax.set_xticks(range(8)); ax.set_yticks(range(8))
        ax.grid(False)

    # 3) по одному искажённому экземпляру на класс
    fig3, axes3 = plt.subplots(1, 3, figsize=(8, 3), constrained_layout=True)
    fig3.suptitle(f"Искажённые образы (одна итерация), pI={pI}", fontsize=14)
    for ax, name, vec in zip(axes3, class_names, T):
        distorted_vec = distort_bits(vec, pI, rng)
        ax.imshow(distorted_vec.reshape(8, 8), cmap="gray_r",
interpolation="nearest")
        ax.set_title(name); ax.set_xticks(range(8)); ax.set_yticks(range(8))
        ax.grid(False)

    plt.show()
```

Используемая формула

$$g''(x) = (L_{x,10} + P_{x,01}) \ln \frac{1 - p_I}{p_I} + (n_s - L_{x,10} - P_{x,01}) \ln \frac{p_I}{1 - p_I} \underset{\omega_2}{\overset{\omega_1}{\gtrless}} l'_0, \qquad (5.36)$$

Результаты выполнения задания

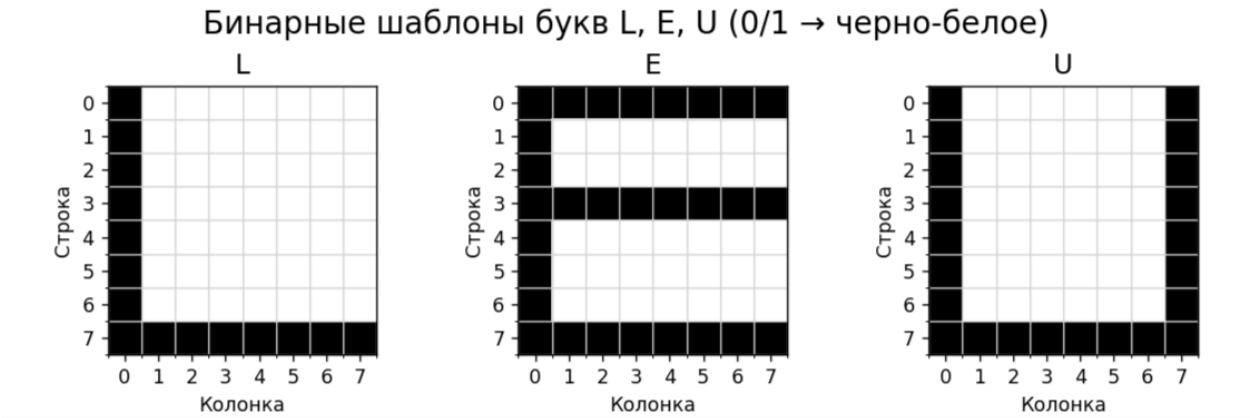

Рисунок 1.

Рисунок 2.

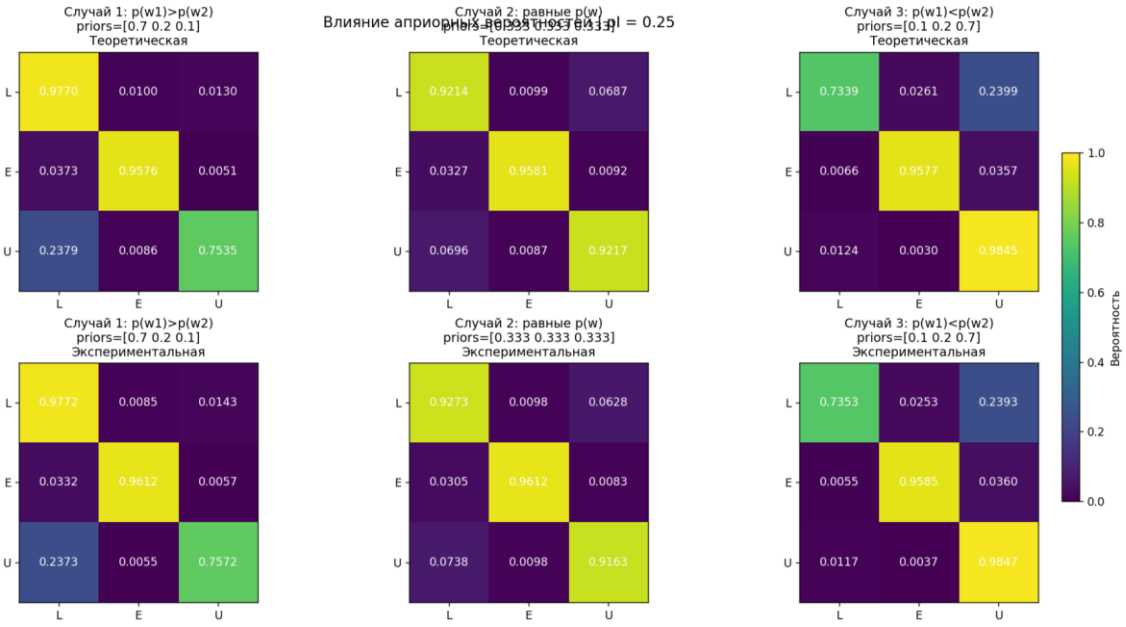

Рисунок 3.

Ответы на контрольные вопросы

1. Симметрия объясняется тем, что при pI и 1-pI система получает одинаковый объём информации об образе: при полном или нулевом искажении ошибка равна нулю, а при pI=0.5 - максимальна, так как признаки становятся случайными.

Выводы

В ходе лабораторной работы исследован процесс распознавания бинарных образов по байесовскому правилу с учётом вероятности искажения символов. Построены теоретические и экспериментальные матрицы ошибок для трёх классов (букв L, E, U) при различных априорных вероятностях. Получено, что при вероятность искажения pI = 0.25 все три класса демонстрируют невысокую точность распознавания, что свидетельствует о схожести их бинарных шаблонов. Наблюдается сильная перекрестная путаница: буква L преимущественно ошибочно классифицируется как E, а буквы E и U взаимно путаются друг с другом. Теоретические и экспериментальные результаты хорошо согласуются, подтверждая конкретность реализации алгоритма.