Лосева Елизавета Юрьевна, группа 8-2

Лабораторная работа № 6

Вариант № 4

Распознавание образов на основе непараметрических алгоритмов оценивания плотности распределения случайной величины

Цель работы

Исследовать алгоритмы распознавания образов на основе оценивания плотности распределения случайных величин и случайных векторов при использовании методов Парзена и k ближайших соседей.

Задание

Используя код Вашей лабораторной №3, реализуйте алгоритм распознавания образов, применив оценивание по методу k ближайших соседей. Вычислите экспериментально вероятности ошибок распознавания. Сравните их с вероятностями ошибок (теоретическими или экспериментальными), полученными в ходе выполнения лабораторной №3. Отобразите поверхности плотностей распределения классов, задаваемых теоретически, и полученных в результате оценивания.

провести численный эксперимент или статистическое имитационное моделирование и представить соответствующие графики. Провести анализ полученных результатов и представить его в виде выводов по проделанной работе.

Код программы

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.patches import Ellipse
from scipy.stats import multivariate_normal
from sklearn.metrics import confusion_matrix
from sklearn.neighbors import KNeighborsClassifier
import seaborn as sns
import sys
import io

sys.stdout = io.TextIOWrapper(sys.stdout.buffer, encoding='utf-8')

# Флаги визуализации
PLOT_SURFACES = True  # 3D поверхности плотностей (теория vs k-NN)
PLOT_DECISION = True  # карта решений k-NN
PLOT_SWEEP = True     # график P(err) vs k
RANDOM_SEED = 42

# Параметры классов (из ЛР-3)
m1 = np.array([0, -1])
m2 = np.array([4, -2])
C1_base = np.array([[3, 1], [1, 3]])
C2_base = np.array([[4, -2], [-2, 4]])

# Используем коэффициент дисперсии = 1.0
factor = 1.0
```

```
C1 = C1_base * factor
C2 = C2_base * factor

p1 = 0.5
p2 = 0.5

rv1 = multivariate_normal(mean=m1, cov=C1)
rv2 = multivariate_normal(mean=m2, cov=C2)

rng = np.random.default_rng(RANDOM_SEED)

# Объемы выборок
N_train_per_class = 1000
N_test_per_class = 5000

# Вспомогательные функции
def add_cov_ellipse(ax, mean, cov, color, alpha=0.25):
    """Эллипс 95% (χ²_2(0.95)≈5.991) для наглядности."""
    vals, vecs = np.linalg.eigh(cov)
    width = 2*np.sqrt(5.991*vals[1])
    height = 2*np.sqrt(5.991*vals[0])
    angle = np.degrees(np.arctan2(vecs[1,1], vecs[0,1]))
    ax.add_patch(Ellipse(mean, width, height, angle=angle,
                         facecolor=color, edgecolor=color, alpha=alpha))

def classify_bayes_true(X, rv1, rv2, p1, p2):
    """Байес (истинные плотности)."""
    P1 = rv1.pdf(X) * p1
    P2 = rv2.pdf(X) * p2
    return np.where(P1 >= P2, 1, 2)

def compute_metrics(cm):
    """Из матрицы ошибок возвращает (acc, perr, alpha1, beta2)."""
    acc = np.trace(cm) / cm.sum()
    perr = 1.0 - acc
    alpha1 = cm[0,1] / cm[0].sum()  # P(реш.2 | ист.1)
    beta2 = cm[1,0] / cm[1].sum()   # P(реш.1 | ист.2)
    return acc, perr, alpha1, beta2

def knn_pdf_estimation(X_train, y_train, k, grid_points):
    """
    Оценка плотности методом k ближайших соседей.
    Возвращает оценки плотностей для каждого класса.
    """
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)

    # Для оценки плотности используем: p(x) = k / (n * V)
    # где V - объем гиперсферы, содержащей k ближайших соседей
    n1 = np.sum(y_train == 1)
    n2 = np.sum(y_train == 2)
```

```
    # Находим расстояния до k-го соседа для каждой точки сетки
    distances, indices = knn.kneighbors(grid_points, return_distance=True)
    radius = distances[:, -1]  # расстояние до k-го соседа

    # Объем гиперсферы в 2D: V = π * r²
    volumes = np.pi * radius**2

    # Оценка плотности для каждого класса
    pdf1_est = np.zeros(len(grid_points))
    pdf2_est = np.zeros(len(grid_points))

    for i, (dist, idx) in enumerate(zip(distances, indices)):
        neighbors_labels = y_train[idx]
        k1 = np.sum(neighbors_labels == 1)
        k2 = np.sum(neighbors_labels == 2)

        if volumes[i] > 0:
            pdf1_est[i] = k1 / (n1 * volumes[i])
            pdf2_est[i] = k2 / (n2 * volumes[i])

    return pdf1_est, pdf2_est

# Данные: train/test
X1_train = rv1.rvs(size=N_train_per_class, random_state=rng)
X2_train = rv2.rvs(size=N_train_per_class, random_state=rng)

X1_test = rv1.rvs(size=N_test_per_class, random_state=rng)
X2_test = rv2.rvs(size=N_test_per_class, random_state=rng)

X_train = np.vstack([X1_train, X2_train])
y_train = np.array([1]*N_train_per_class + [2]*N_train_per_class)

X_test = np.vstack([X1_test, X2_test])
y_true = np.array([1]*N_test_per_class + [2]*N_test_per_class)

# Базовое сравнение: Байес (истина) vs k-NN
# Байес (истинные)
y_bayes = classify_bayes_true(X_test, rv1, rv2, p1, p2)
cm_bayes = confusion_matrix(y_true, y_bayes, labels=[1,2])
acc_bayes, perr_bayes, a1_bayes, b2_bayes = compute_metrics(cm_bayes)

# k-NN: начальное значение k
k_initial = 50
knn = KNeighborsClassifier(n_neighbors=k_initial)
knn.fit(X_train, y_train)
y_knn = knn.predict(X_test)
cm_knn = confusion_matrix(y_true, y_knn, labels=[1,2])
acc_knn, perr_knn, a1_knn, b2_knn = compute_metrics(cm_knn)

print(" СРАВНЕНИЕ КЛАССИФИКАТОРОВ ")
```

```
print(f"[Байес (истинные)] acc={acc_bayes:.4f} P(err)={perr_bayes:.4f} "
      f"alpha1={a1_bayes:.4f} beta2={b2_bayes:.4f}")
print(f"[k-NN (k={k_initial})] acc={acc_knn:.4f} P(err)={perr_knn:.4f} "
      f"alpha1={a1_knn:.4f} beta2={b2_knn:.4f}")
print("\nМатрицы ошибок [истина по строкам 1/2, решение по столбцам 1/2]:")
print("Байес:\n", cm_bayes)
print(f"k-NN (k={k_initial}):\n", cm_knn)

# Сетка для поверхностей/карты решений
all_data = np.vstack([X1_train, X2_train, X_test])
x_min, x_max = all_data[:,0].min()-2.0, all_data[:,0].max()+2.0
y_min, y_max = all_data[:,1].min()-2.0, all_data[:,1].max()+2.0
xs = np.linspace(x_min, x_max, 80)  # уменьшим разрешение для скорости
ys = np.linspace(y_min, y_max, 80)
XX, YY = np.meshgrid(xs, ys)
GRID = np.stack([XX.ravel(), YY.ravel()], axis=1)

# теоретические плотности для поверхностей
f1_true = rv1.pdf(GRID).reshape(XX.shape)
f2_true = rv2.pdf(GRID).reshape(XX.shape)

# k-NN оценки плотностей (для визуализации используем большее k для сглаживания)
k_for_viz = 100
f1_knn, f2_knn = knn_pdf_estimation(X_train, y_train, k_for_viz, GRID)
f1_knn = f1_knn.reshape(XX.shape)
f2_knn = f2_knn.reshape(XX.shape)

# Нормализуем оценки плотностей k-NN для лучшей визуализации
f1_knn_norm = f1_knn / np.max(f1_knn) * np.max(f1_true)
f2_knn_norm = f2_knn / np.max(f2_knn) * np.max(f2_true)

# Визуализация поверхностей
if PLOT_SURFACES:
    fig = plt.figure(figsize=(14, 5))
    ax = fig.add_subplot(1, 2, 1, projection='3d')
    ax.plot_surface(XX, YY, f1_true, rstride=2, cstride=2, alpha=0.9,
cmap=cm.viridis)
    ax.set_title("Класс 1 --- теоретическая плотность")
    ax.set_xlabel("x1"); ax.set_ylabel("x2"); ax.set_zlabel("p(x|w1)")

    ax = fig.add_subplot(1, 2, 2, projection='3d')
    ax.plot_surface(XX, YY, f1_knn_norm, rstride=2, cstride=2, alpha=0.9,
cmap=cm.viridis)
    ax.set_title(f"Класс 1 --- k-NN оценка (k={k_for_viz})")
    ax.set_xlabel("x1"); ax.set_ylabel("x2"); ax.set_zlabel(r"$\hat p(x|w1)$")
    plt.tight_layout(); plt.show()

    fig = plt.figure(figsize=(14, 5))
    ax = fig.add_subplot(1, 2, 1, projection='3d')
    ax.plot_surface(XX, YY, f2_true, rstride=2, cstride=2, alpha=0.9,
cmap=cm.plasma)
```

```
    ax.set_title("Класс 2 --- теоретическая плотность")
    ax.set_xlabel("x1"); ax.set_ylabel("x2"); ax.set_zlabel("p(x|w2)")

    ax = fig.add_subplot(1, 2, 2, projection='3d')
    ax.plot_surface(XX, YY, f2_knn_norm, rstride=2, cstride=2, alpha=0.9,
cmap=cm.plasma)
    ax.set_title(f"Класс 2 --- k-NN оценка (k={k_for_viz})")
    ax.set_xlabel("x1"); ax.set_ylabel("x2"); ax.set_zlabel(r"$\hat p(x|w2)$")
    plt.tight_layout(); plt.show()

# Визуализация карты решений
if PLOT_DECISION:
    # Создаем классификатор для карты решений
    knn_viz = KNeighborsClassifier(n_neighbors=k_initial)
    knn_viz.fit(X_train, y_train)
    Z = knn_viz.predict(GRID).reshape(XX.shape)

    plt.figure(figsize=(7,6))
    plt.contourf(XX, YY, Z, levels=[0.5,1.5,2.5], colors=["#ffaaaa","#aaaaff"],
alpha=0.35)
    plt.contour(XX, YY, Z, levels=[1.5], colors='k', linestyles='--',
linewidths=2, alpha=0.9)
    plt.scatter(X1_train[:,0], X1_train[:,1], s=6, c='red', alpha=0.55,
label='train w1')
    plt.scatter(X2_train[:,0], X2_train[:,1], s=6, c='blue', alpha=0.55,
label='train w2')
    add_cov_ellipse(plt.gca(), m1, C1, 'red', 0.18)
    add_cov_ellipse(plt.gca(), m2, C2, 'blue', 0.18)
    plt.title(f"Карта решений k-NN (k={k_initial})")
    plt.xlabel("x1"); plt.ylabel("x2"); plt.legend(loc='upper right', fontsize=9)
    plt.grid(True, alpha=0.3)
    plt.tight_layout(); plt.show()

# Визуализация матрицы ошибок
fig, axes = plt.subplots(1, 2, figsize=(9,4))
for ax, cmx, ttl in zip(axes,
                        [cm_bayes, cm_knn],
                        [f"QDA (истинные)\nacc={acc_bayes:.3f},
P(err)={perr_bayes:.3f}",
                         f"k-NN (k={k_initial})\nacc={acc_knn:.3f},
P(err)={perr_knn:.3f}"]):
    sns.heatmap(cmx, annot=True, fmt="d", cmap="Blues", cbar=False, ax=ax)
    ax.set_xlabel("Предсказанный"); ax.set_ylabel("Истинный"); ax.set_title(ttl)
plt.tight_layout(); plt.show()

# Сканирование k -> P(err)
if PLOT_SWEEP:
    ks = np.arange(1, 201, 10)  # k от 1 до 200 с шагом 10
    results = []

    for k in ks:
```

```
        knn_temp = KNeighborsClassifier(n_neighbors=k)
        knn_temp.fit(X_train, y_train)
        y_hat = knn_temp.predict(X_test)
        cm = confusion_matrix(y_true, y_hat, labels=[1,2])
        acc, perr, a1, b2 = compute_metrics(cm)
        results.append((k, perr, acc, a1, b2))

    print("\n СКАНИРОВАНИЕ k (метод k ближайших соседей) ")
    best = None
    for k, perr, acc, a1, b2 in results:
        print(f"k={k:3d} P(err)={perr:.4f} acc={acc:.4f} alpha1={a1:.4f}
beta2={b2:.4f}")
        if best is None or perr < best[1]:
            best = (k, perr, acc, a1, b2)
    print(f"Лучшее: k={best[0]:3d} P(err)={best[1]:.4f} acc={best[2]:.4f} "
          f"alpha1={best[3]:.4f} beta2={best[4]:.4f}")

    plt.figure(figsize=(7,4))
    plt.plot([r[0] for r in results], [r[1] for r in results], marker='o')
    plt.xlabel("k"); plt.ylabel("P(err)")
    plt.title("Сканирование k: P(err) vs k")
    plt.grid(True, alpha=0.3)
    plt.tight_layout(); plt.show()

# Сравнение расстояний (метрик) для k-NN
print("\n СРАВНЕНИЕ МЕТРИК РАССТОЯНИЯ (при k=50) ")
metrics = ['euclidean', 'manhattan', 'chebyshev']
for metric in metrics:
    knn_metric = KNeighborsClassifier(n_neighbors=50, metric=metric)
    knn_metric.fit(X_train, y_train)
    y_hat = knn_metric.predict(X_test)
    cm = confusion_matrix(y_true, y_hat, labels=[1,2])
    acc, perr, a1, b2 = compute_metrics(cm)
    print(f"{metric:10s} P(err)={perr:.4f} acc={acc:.4f} alpha1={a1:.4f}
beta2={b2:.4f}")
```

Результаты выполнения задания

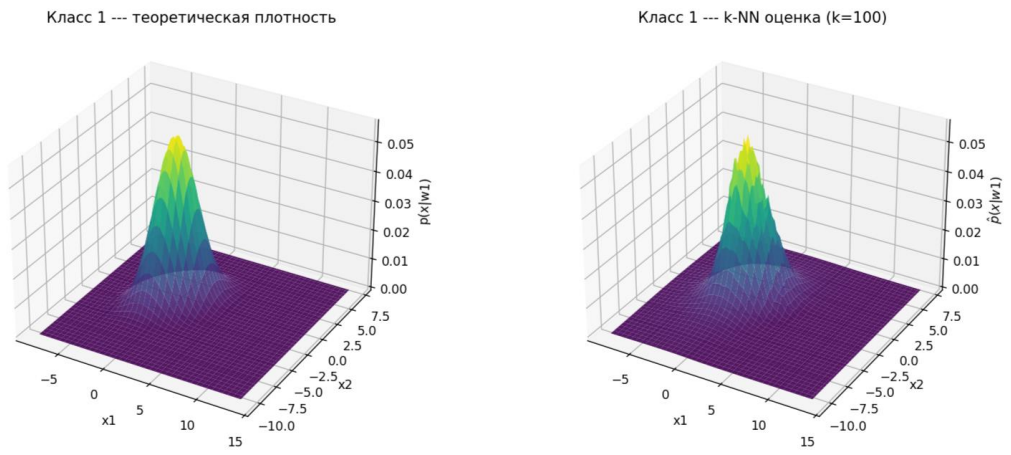


Рисунок 1.

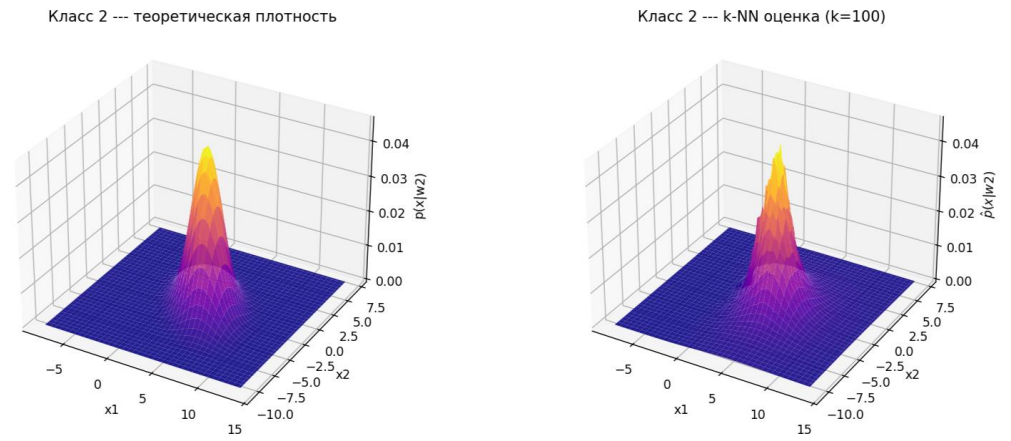


Рисунок 2.

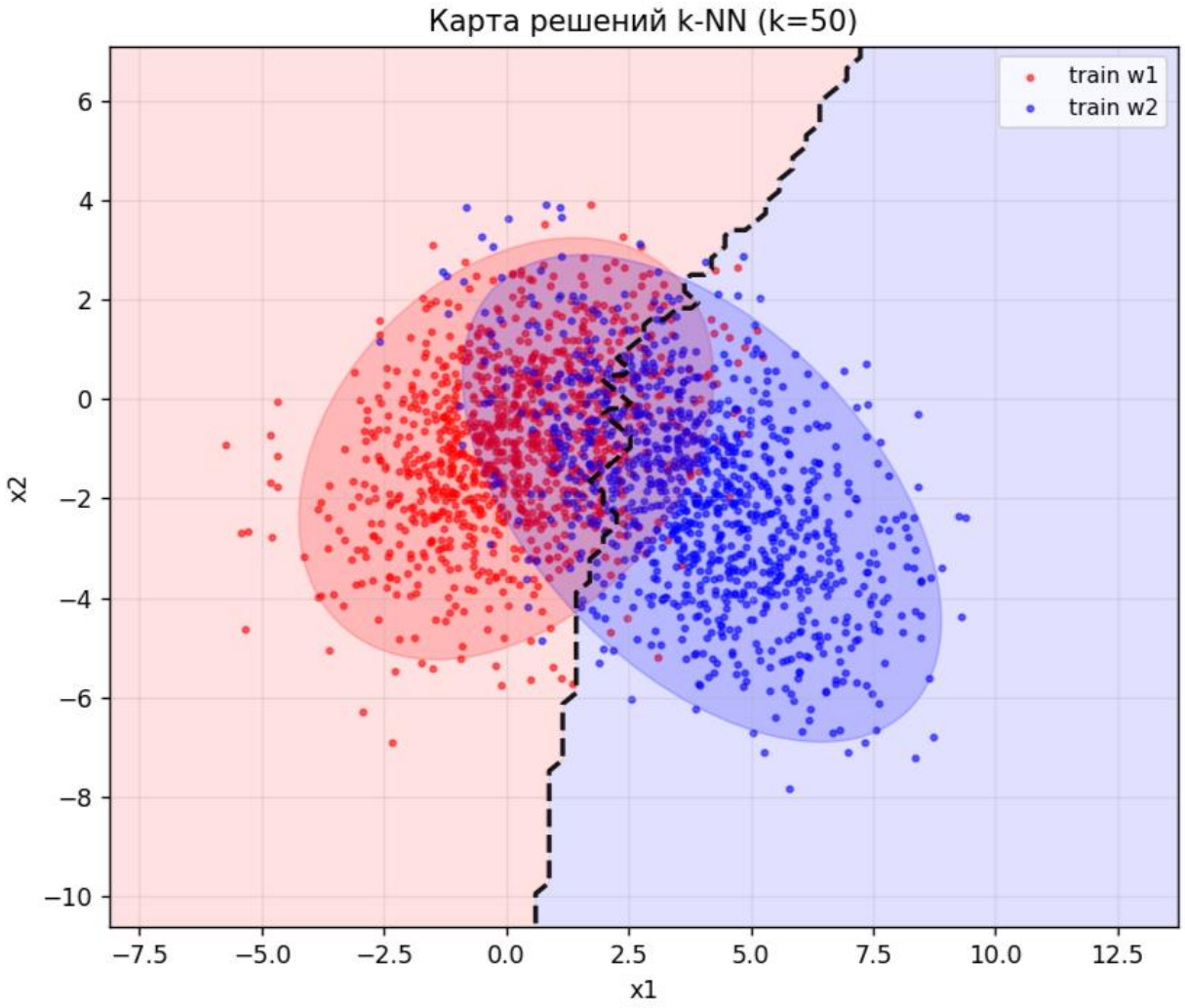


Рисунок 3.

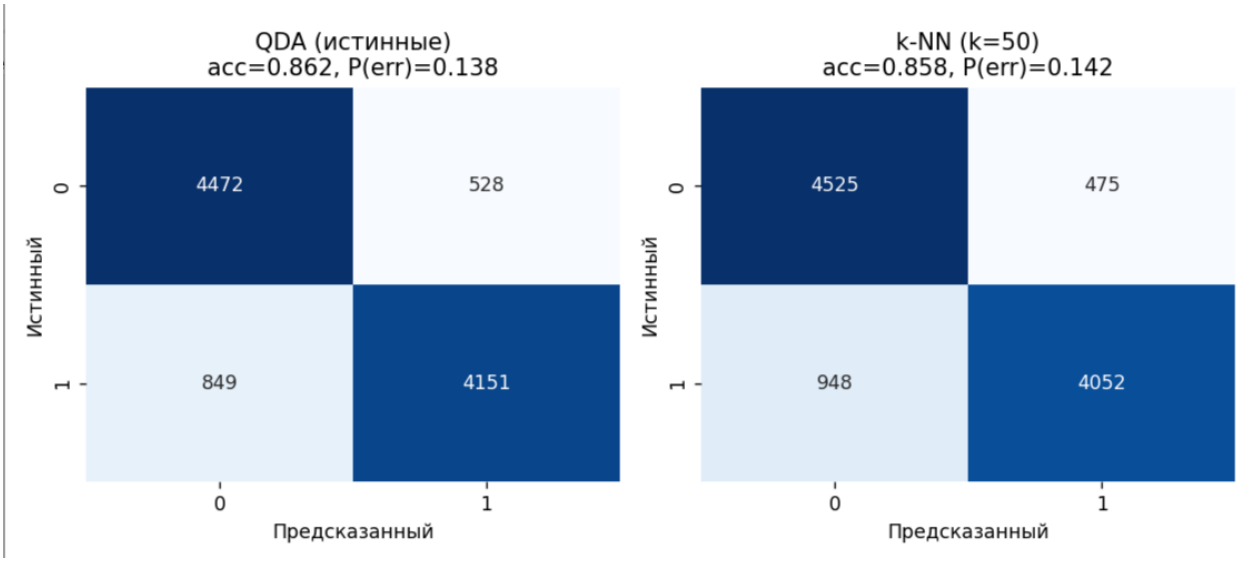
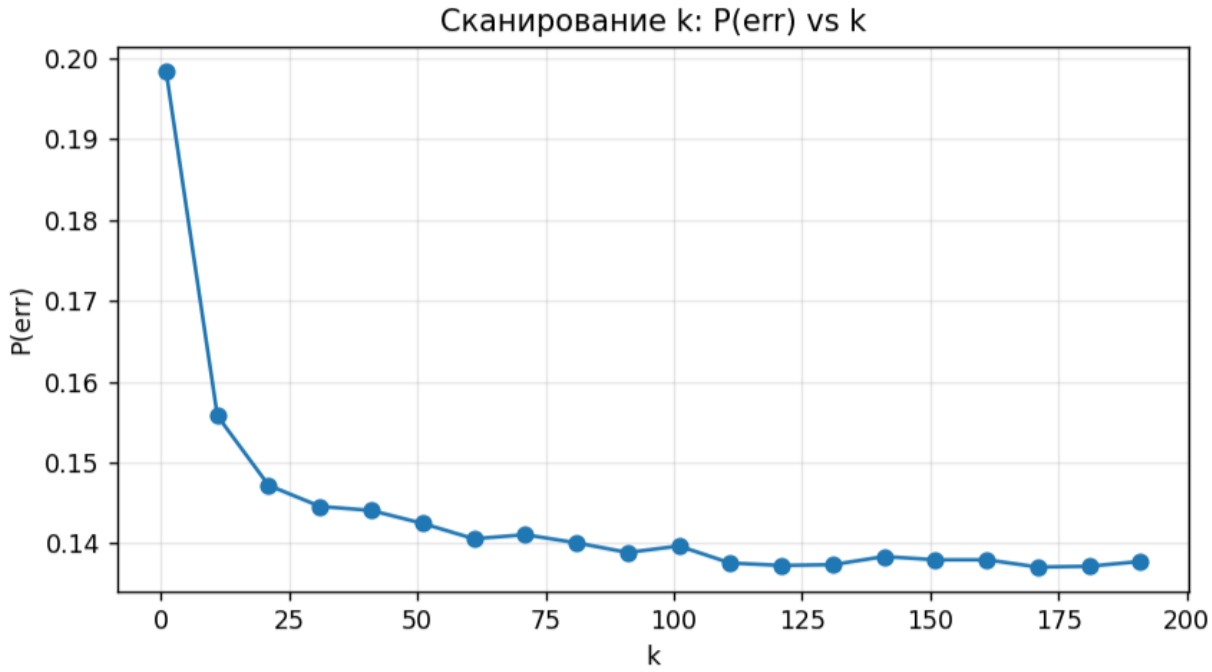


Рисунок 4.

Ответы на контрольные вопросы

1. Наилучшее качество оценивания достигается при k=171, где вероятность ошибки распознавания составляет P(err) = 0.1371, а точность классификации acc = 0.8629

```
k= 1 P(err)=0.1984 acc=0.8016 alpha1=0.1914 beta2=0.2054
k= 11 P(err)=0.1558 acc=0.8442 alpha1=0.1318 beta2=0.1798
k= 21 P(err)=0.1472 acc=0.8528 alpha1=0.1166 beta2=0.1778
k= 31 P(err)=0.1446 acc=0.8554 alpha1=0.1066 beta2=0.1826
k= 41 P(err)=0.1441 acc=0.8559 alpha1=0.1056 beta2=0.1826
k= 51 P(err)=0.1425 acc=0.8575 alpha1=0.1018 beta2=0.1832
k= 61 P(err)=0.1406 acc=0.8594 alpha1=0.0990 beta2=0.1822
k= 71 P(err)=0.1411 acc=0.8589 alpha1=0.0996 beta2=0.1826
k= 81 P(err)=0.1401 acc=0.8599 alpha1=0.0980 beta2=0.1822
k= 91 P(err)=0.1389 acc=0.8611 alpha1=0.0958 beta2=0.1820
k=101 P(err)=0.1397 acc=0.8603 alpha1=0.0986 beta2=0.1808
k=111 P(err)=0.1376 acc=0.8624 alpha1=0.0972 beta2=0.1780
k=121 P(err)=0.1373 acc=0.8627 alpha1=0.0990 beta2=0.1756
k=131 P(err)=0.1374 acc=0.8626 alpha1=0.0990 beta2=0.1758
k=141 P(err)=0.1384 acc=0.8616 alpha1=0.0984 beta2=0.1784
k=151 P(err)=0.1380 acc=0.8620 alpha1=0.0986 beta2=0.1774
k=161 P(err)=0.1380 acc=0.8620 alpha1=0.0986 beta2=0.1774
k=171 P(err)=0.1371 acc=0.8629 alpha1=0.0980 beta2=0.1762
k=181 P(err)=0.1372 acc=0.8628 alpha1=0.0970 beta2=0.1774
k=191 P(err)=0.1378 acc=0.8622 alpha1=0.0986 beta2=0.1770
Лучшее: k=171 P(err)=0.1371 acc=0.8629 alpha1=0.0980 beta2=0.1762
```



2. Все три рассмотренные метрики расстояния показывают близкие результаты, но евклидова и махэттенская метрики демонстрируют одинаково наилучшее качество P(err) = 0/1423

```
 СРАВНЕНИЕ МЕТРИК РАССТОЯНИЯ (при k=50)
euclidean  P(err)=0.1423 acc=0.8577 alpha1=0.0950 beta2=0.1896
manhattan  P(err)=0.1423 acc=0.8577 alpha1=0.0932 beta2=0.1914
chebyshev  P(err)=0.1419 acc=0.8581 alpha1=0.0968 beta2=0.1870
```

Выводы

В данной лабораторной работе реализован метод k ближайших соседей для распознавания образов на основе данных из ЛР-3.

Были построены теоретические и оценённые поверхности плотностей, которые показали хорошее соответствие, особенно в областях с высокой плотностью данных.

Экспериментально определено, что наилучшее качество распознавания достигается при k=171, где вероятность ошибки минимальна и составляет P(err) = 0.1371.

При малых значениях k наблюдается переобучение (шумная оценка плотности), а при больших k – чрезмерное сглаживание и потеря деталей с распределении.

Сравнение различных метрик расстояния показало, что евклидова метрика обеспечивает наиболее сбалансированное качество классификации с точностью 85.77%

Метод k ближайших соседей позволяет успешно оценивать плотности распределения и обеспечивает точность, сравнимую с байесовским классификатором.