Министерство цифрового развития

Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

Кафедра прикладной математики и кибернетики

Отчёт

по лабораторной работе № 2 «Метод k-ближайших соседей»

Выполнил:

студент группы ИП-216

Русецкий А.С.
ФИО студента

Работу проверил: Преподаватель
должность преподавателя
Сороковых Д.А.
ФИО преподавателя

Новосибирск 2025 г.

Введение

Цель работы: разработка классификатора на основе метода k ближайших соседей.

Задание

1. Загрузите набор данных
2. Данные из файла необходимо разбить на две выборки
3. Проанализируйте обучающую выборку на возможность минимизировать разницу между количеством представленных в ней объектов разных классов.
4. На основе этих данных необходимо обучить разработанный классификатор

   Метод парзеновского окна с фиксированным h

   Функция ядра П - прямоугольное

Основная часть

1. Загрузка набора данных

```
df = pd.read_csv('data4.csv')
df['MrotInHour'] = df['MrotInHour'].astype(float)
df['Salary'] = df['Salary'].astype(float)
df['Class'] = df['Class'].astype(int)

print('Всего строк:', len(df))
print('Распределение классов:', dict(Counter(df['Class'])))
print(df.head())
```

```
Всего строк: 10000
Распределение классов: {1: 2015, 0: 7985}
   MrotInHour   Salary  Class
0         4.0  13270.0      1
1         3.0  24786.0      1
2         9.0   3651.0      0
3        10.0  10622.0      0
4         3.0  15634.0      1
```

2. Разбиение на выборки

```
def split_into_three_random(df, rng):
    df_sh = df.sample(frac=1, random_state=rng).reset_index(drop=True)
    n = len(df_sh)

    sizes = [n//3 + (1 if i < n%3 else 0) for i in range(3)]
    parts = []
    start = 0
    for s in sizes:
        parts.append(df_sh.iloc[start:start+s].reset_index(drop=True))
        start += s
    return parts

def make_train_test_from_parts(parts, test_part_index):
    test = parts[test_part_index].reset_index(drop=True)
    train = pd.concat([parts[i] for i in range(3) if i != test_part_index], ignore_index=True)
    return train, test
```

3. Обучаем подобранный классификатор

```python
def rectangular_kernel(r):
    """Прямоугольное ядро: K(r) = 1 если |r| <= 1, иначе 0"""
    r = np.asarray(r)
    return np.where(np.abs(r) <= 1, 1.0, 0.0)

def parzen_predict_point_fixed_h(x, X_train, y_train, h):
    dists = np.linalg.norm(X_train - x, axis=1)
    weights = rectangular_kernel(dists / h)

    if weights.sum() == 0:
        return int(y_train[np.argmin(dists)])

    s0 = weights[y_train == 0].sum()
    s1 = weights[y_train == 1].sum()

    return 1 if s1 > s0 else 0

def parzen_predict_batch_fixed_h(X_test, X_train, y_train, h):
    preds = []
    for i in range(len(X_test)):
        preds.append(parzen_predict_point_fixed_h(X_test[i], X_train, y_train, h))
    return np.array(preds)
```

Реализуем метод парзеновского окна с прямоугольным ядром

```python
def select_h_loo_fixed(X_train, y_train, h_grid, show_progress=True):
    n = len(X_train)
    best_h = None
    best_accuracy = 0.0

    for h_idx, h in enumerate(h_grid):
        correct_predictions = 0

        for i in range(n):
            mask = np.ones(n, dtype=bool)
            mask[i] = False
            X_tr = X_train[mask]
            y_tr = y_train[mask]

            y_pred = parzen_predict_point_fixed_h(X_train[i], X_tr, y_tr, h)

            if y_pred == y_train[i]:
                correct_predictions += 1

        accuracy = correct_predictions / n

        if accuracy > best_accuracy:
            best_accuracy = accuracy
            best_h = h

        if show_progress and (h_idx % max(1, len(h_grid)//10) == 0 or h_idx == len(h_grid)-1):
            print(f"h={h:.3f}, Точность LOO: {accuracy:.4f}")

    return best_h, 1 - best_accuracy
```

Подбираем h методом Leave-one-out

```python
def prepare_Xy(df):
    X = df[['MrotInHour','Salary']].values.astype(float)
    y = df['Class'].values.astype(int)
    return X, y

def balance_classes_undersampling(X, y):
    class_0_idx = np.where(y == 0)[0]
    class_1_idx = np.where(y == 1)[0]

    min_count = min(len(class_0_idx), len(class_1_idx))

    selected_0 = np.random.choice(class_0_idx, min_count, replace=False)
    selected_1 = np.random.choice(class_1_idx, min_count, replace=False)

    balanced_idx = np.concatenate([selected_0, selected_1])
    np.random.shuffle(balanced_idx)

    return X[balanced_idx], y[balanced_idx]
```

Выбираем из таблицы нужные столбы – признаки и классы

Берём по одинаковому количеству классов

```python
random_seeds = list(range(100, 110))
results = []

print("10 РАЗБИЕНИЙ")
print("=" * 60)

for idx, seed in enumerate(random_seeds, start=1):
    print(f"\n--- Разбиение #{idx} (seed={seed}) ---")

    # Разделение данных
    parts = split_into_three_random(df, rng=seed)
    test_part_index = (idx-1) % 3
    train_df, test_df = make_train_test_from_parts(parts, test_part_index=test_part_index)

    print(f"Обучающая выборка: {dict(Counter(train_df['Class']))}")
    print(f"Тестовая выборка: {dict(Counter(test_df['Class']))}")

    X_tr_raw, y_tr = prepare_Xy(train_df)
    X_te_raw, y_te = prepare_Xy(test_df)

    scaler = StandardScaler().fit(X_tr_raw)
    X_tr = scaler.transform(X_tr_raw)
    X_te = scaler.transform(X_te_raw)

    X_tr_balanced, y_tr_balanced = balance_classes_undersampling(X_tr, y_tr)
    print(f"После балансировки: 0:{sum(y_tr_balanced == 0)}, 1:{sum(y_tr_balanced == 1)}")

    pairwise_dists = np.linalg.norm(X_tr_balanced[:,None,:] - X_tr_balanced[None,:,:], axis=2)
    median_dist = np.median(pairwise_dists[np.triu_indices_from(pairwise_dists, k=1)])

    h_grid = np.concatenate((
        np.linspace(max(0.05, median_dist*0.1), median_dist*0.5, 10),
        np.linspace(median_dist*0.6, median_dist*2.0, 20)
    ))
    h_grid = np.unique(np.clip(h_grid, 0.05, 5.0))
```

```python
# Подбор оптимального h
best_h, best_err = select_h_loo_fixed(X_tr_balanced, y_tr_balanced, h_grid, show_progress=False)

y_pred_test = parzen_predict_batch_fixed_h(X_te, X_tr_balanced, y_tr_balanced, best_h)

# Метрики
acc = accuracy_score(y_te, y_pred_test)
prec = precision_score(y_te, y_pred_test, zero_division=0)
rec = recall_score(y_te, y_pred_test, zero_division=0)
f1 = f1_score(y_te, y_pred_test, zero_division=0)
cm = confusion_matrix(y_te, y_pred_test).tolist()

results.append({
    'split_id': idx,
    'seed': seed,
    'test_part_index': test_part_index,
    'train_counts': dict(Counter(train_df['Class'])),
    'test_counts': dict(Counter(test_df['Class'])),
    'selected_h': float(best_h),
    'loo_train_error': float(best_err),
    'accuracy': float(acc),
    'precision': float(prec),
    'recall': float(rec),
    'f1': float(f1),
    'confusion_matrix': cm
})

print(f"Результаты: h={best_h:.3f}, Accuracy={acc:.4f}, F1={f1:.4f}")
```

Основной блок выполнения разбиений и их вывод

```
--- Разбиение #1 (seed=100) ---
Обучающая выборка: {0: 5338, 1: 1328}
Тестовая выборка: {0: 2647, 1: 687}
После балансировки: 0:1328, 1:1328
Результаты: h=0.169, Accuracy=0.9862, F1=0.9676

--- Разбиение #2 (seed=101) ---
Обучающая выборка: {0: 5314, 1: 1353}
Тестовая выборка: {0: 2671, 1: 662}
После балансировки: 0:1353, 1:1353
Результаты: h=0.169, Accuracy=0.9883, F1=0.9714

--- Разбиение #3 (seed=102) ---
Обучающая выборка: {0: 5300, 1: 1367}
Тестовая выборка: {0: 2685, 1: 648}
После балансировки: 0:1367, 1:1367
Результаты: h=0.168, Accuracy=0.9853, F1=0.9636

--- Разбиение #4 (seed=103) ---
Обучающая выборка: {0: 5274, 1: 1392}
Тестовая выборка: {1: 623, 0: 2711}
После балансировки: 0:1392, 1:1392
Результаты: h=0.168, Accuracy=0.9877, F1=0.9681

--- Разбиение #5 (seed=104) ---
Обучающая выборка: {0: 5317, 1: 1350}
Тестовая выборка: {1: 665, 0: 2668}
После балансировки: 0:1350, 1:1350
Результаты: h=0.168, Accuracy=0.9892, F1=0.9736

--- Разбиение #6 (seed=105) ---
Обучающая выборка: {0: 5291, 1: 1376}
Тестовая выборка: {0: 2694, 1: 639}
После балансировки: 0:1376, 1:1376
Результаты: h=0.169, Accuracy=0.9841, F1=0.9602

--- Разбиение #7 (seed=106) ---
Обучающая выборка: {0: 5339, 1: 1327}
Тестовая выборка: {0: 2646, 1: 688}
После балансировки: 0:1327, 1:1327
Результаты: h=0.169, Accuracy=0.9877, F1=0.9711
```

```
--- Разбиение #8 (seed=107) ---
Обучающая выборка: {0: 5300, 1: 1367}
Тестовая выборка: {1: 648, 0: 2685}
После балансировки: 0:1367, 1:1367
Результаты: h=0.170, Accuracy=0.9883, F1=0.9708

--- Разбиение #9 (seed=108) ---
Обучающая выборка: {0: 5318, 1: 1349}
Тестовая выборка: {0: 2667, 1: 666}
После балансировки: 0:1349, 1:1349
Результаты: h=0.170, Accuracy=0.9892, F1=0.9737

--- Разбиение #10 (seed=109) ---
Обучающая выборка: {0: 5335, 1: 1331}
Тестовая выборка: {0: 2650, 1: 684}
После балансировки: 0:1331, 1:1331
Результаты: h=0.169, Accuracy=0.9874, F1=0.9702
```

Результаты разбиений

4. Формируем таблицу с результатами тестирования

```
results_df = pd.DataFrame(results)

print("\n" + "="*80)
print("ИТОГОВАЯ ТАБЛИЦА РЕЗУЛЬТАТОВ")
print("="*80)

display_cols = ['split_id', 'seed', 'test_part_index', 'selected_h',
                'loo_train_error', 'accuracy', 'precision', 'recall', 'f1']
print(results_df[display_cols].round(4))

print("\nСТАТИСТИКА ПО ВСЕМ РАЗБИЕНИЯМ:")
print("="*50)
for metric in ['accuracy', 'precision', 'recall', 'f1', 'selected_h']:
    values = results_df[metric]
    print(f"{metric:12}: mean = {values.mean():.4f} ± {values.std():.4f} "
          f"(min = {values.min():.4f}, max = {values.max():.4f})")
```

```
================================================================================
ИТОГОВАЯ ТАБЛИЦА РЕЗУЛЬТАТОВ
================================================================================
   split_id  seed  test_part_index  selected_h  loo_train_error  accuracy  \
0         1   100                0      0.1692           0.0064    0.9862
1         2   101                1      0.1690           0.0070    0.9883
2         3   102                2      0.1682           0.0069    0.9853
3         4   103                0      0.1684           0.0072    0.9877
4         5   104                1      0.1681           0.0111    0.9892
5         6   105                2      0.1693           0.0058    0.9841
6         7   106                0      0.1688           0.0072    0.9877
7         8   107                1      0.1698           0.0073    0.9883
8         9   108                2      0.1699           0.0067    0.9892
9        10   109                0      0.1685           0.0060    0.9874

   precision  recall      f1
0     0.9372     1.0  0.9676
1     0.9444     1.0  0.9714
2     0.9297     1.0  0.9636
3     0.9383     1.0  0.9681
4     0.9486     1.0  0.9736
5     0.9234     1.0  0.9602
6     0.9438     1.0  0.9711
7     0.9432     1.0  0.9708
8     0.9487     1.0  0.9737
9     0.9421     1.0  0.9702

СТАТИСТИКА ПО ВСЕМ РАЗБИЕНИЯМ:
==================================================
accuracy    : mean = 0.9873 ± 0.0017 (min = 0.9841, max = 0.9892)
precision   : mean = 0.9399 ± 0.0081 (min = 0.9234, max = 0.9487)
recall      : mean = 1.0000 ± 0.0000 (min = 1.0000, max = 1.0000)
f1          : mean = 0.9690 ± 0.0043 (min = 0.9602, max = 0.9737)
selected_h  : mean = 0.1689 ± 0.0006 (min = 0.1681, max = 0.1699)
```

Заключение

Был проведён эксперимент по классификации объектов методом парзеновского окна с фиксированным h. Для классификации применялось прямоугольное ядро. Данные были разделены на обучающую и тестовую выборки. Для оценки устойчивости классификатора проведено 10 разбиений данных с различными начальными значениями.

Краткие выводы:

- Во всех 10 экспериментах результаты оказались похожими, что говорит о стабильной работе метода.
- Метод Парзеновского окна показал высокую точность и стабильность на всех разбиениях.

Ссылка на google colab:

https://colab.research.google.com/drive/1hx684jen_FP9erGDCKti6oPWN7UTfjqF?usp=sharing